

**МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ  
(ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)**

На правах рукописи

*Рахман Павел Азизурович*

**Разработка методики повышения эффективности  
использования вычислительных ресурсов при применении  
технологии виртуальных машин**

Специальность 05.13.13 –

“Телекоммуникационные системы и компьютерные сети”

**Диссертация на соискание ученой степени  
кандидата технических наук**

Научный руководитель:

к.т.н., профессор

Ладыгин И.И.

**Москва, 2005**

# Оглавление

Основные термины, обозначения и сокращения .....	4
Введение .....	6
Постановка задачи.....	14
1. Обзор существующих подходов к повышению эффективности использования ресурсов и моделей распределения ресурсов .....	15
1.1. Обзор существующих подходов к повышению эффективности использования вычислительных ресурсов .....	15
1.1.1. Использование ресурсов для внутренних задач.....	16
1.1.2. Использование ресурсов для задач сторонних организаций .....	18
1.1.3. Применение адекватных аппаратных решений .....	19
1.1.4. Объединение сервисов и снижение числа серверных ОС .....	21
1.1.5. Применение технологии виртуальных машин .....	24
1.2. Обзор моделей распределения ресурсов.....	28
1.2.1. Традиционные модели распределения ресурсов.....	28
1.2.2. Современные модели распределения ресурсов.....	30
Выводы по главе 1 .....	37
2. Разработка методики по реорганизации серверного парка.....	38
2.1. Общий подход .....	38
2.2. Подходы к сбору первичной информации.....	45
2.2.1. Выбор типов ресурсов и размерностей для оценки их уровней.....	47
2.2.2. Сбор информации по физическим компьютерам.....	48
2.2.3. Сбор информации по логическим серверам.....	50
2.2.4. Оценка требований базовой операционной системы .....	56
2.2.5. Требования надежности функционирования серверного парка.....	57
2.2.6. Выбор цели оптимизации .....	60
2.3. Поиск распределения логических серверов на компьютеры.....	61
2.4. Первичная и вторичная оценка распределения .....	62
2.5. Первичная реорганизация серверного парка .....	68
2.6. Оценка качества функционирования серверного парка .....	78

2.7. Поиск причин неудовлетворительного функционирования.....	81
2.8. Корректировочная реорганизация серверного парка.....	86
2.9. Возврат к исходному состоянию или компромиссы.....	89
Выводы по главе 2.....	91
3. Разработка математической модели и метода решения задачи поиска распределения логических серверов по компьютерам .....	93
3.1. Разработка математической модели .....	94
3.2. Поиск метода решения.....	101
3.2.1 Существующие методы решения поставленной задачи.....	101
3.2.2 Предлагаемый вариант решения с разбиением на подзадачи.....	106
3.2.3. Существующие приближенные методы решения подзадач.....	112
3.2.4. Предлагаемый вариант локального поиска с использованием мультистарта, функции штрафов и управляемого радиуса зон поиска.....	118
3.3. Анализ характеристик предложенных методов. ....	124
Выводы по главе 3.....	141
4. Программная реализация алгоритма поиска оптимального распределения. ....	143
4.1. Требования к программной реализации .....	143
4.2. Описание разработанного программного обеспечения .....	145
4.3. Экспериментальное исследование и внедрение .....	156
Выводы по главе 4.....	172
Заключение .....	173
Литература.....	176
<b>Приложение 1. Примеры использования разработанной методики. ....</b>	<b>181</b>
<b>Приложение 2. Руководство по переносу системы MS Windows 2000 с физической аппаратной платформы на виртуальную.....</b>	<b>206</b>
<b>Приложение 3. Обзор технологии виртуальных машин.....</b>	<b>231</b>
<b>Приложение 4. Руководство пользователя к программе DTSOLVEX.....</b>	<b>251</b>
<b>Приложение 5. Тексты исходного кода программы DTSOLVEX.....</b>	<b>271</b>
<b>Приложение 6. Акт о внедрении .....</b>	<b>400</b>

## Основные термины, обозначения и сокращения

**Физический компьютер** или просто **компьютер** – некоторая ЭВМ, состоящая из реально существующих компонент ее аппаратного обеспечения и предоставляющая некоторые вычислительные ресурсы.

**Виртуальная машина** – некоторая ЭВМ, полноценно эмулируемая некоторым специальным программным обеспечением и состоящая, как из полностью эмулируемых компонент, так и компонент, отображаемых на реально существующие компоненты некоторого физического компьютера. Виртуальная машина функционирует как процесс под управлением некоторой многозадачной операционной системы, называемой **базовой ОС**, на некотором физическом компьютере. Виртуальная машина предоставляет вычислительные ресурсы, которые в действительности являются частями ресурсов физического компьютера. На физическом компьютере могут параллельно (псевдопараллельно) функционировать несколько виртуальных машин.

**Технология виртуальных машин** – специальная технология, позволяющая эмулировать виртуальные машины на базе аппаратного обеспечения физического компьютера и обеспечивать их параллельное (псевдопараллельное) функционирование под управлением базовой ОС.

**Логический сервер** – некоторая серверная операционная система, со всеми встроенными и дополнительными программными компонентами, серверными и клиентскими службами, сконфигурированными на ней. В каждый момент времени на каком-либо физическом компьютере или виртуальной машине может функционировать только один логический сервер.

**Распределение логических серверов по физическим компьютерам** – распределение виртуальных машин с функционирующими на них логическими серверами по компьютерам при применении технологии виртуальных машин.

**Серверный парк** – совокупность физических компьютеров и логических серверов, функционирующих на них, связанных некоторой высокоскоростной и высоконадежной сетью передачи данных.

**Эффективность использования вычислительных ресурсов серверного парка** – отношение денежных средств, получаемых за счет решения серверным парком некоторого множества “полезных” задач к совокупной стоимости владения серверным парком. Под множеством “полезных” задач будем понимать множество задач, функций и сервисов, решение (выполнение, предоставление) которых либо приносят определенный доход, либо снижают определенные виды затрат. Совокупная стоимость владения включает в себя стоимость оборудования, стоимость программного обеспечения, стоимость первоначальной настройки, стоимость аренды помещений, затраты на потребляемую электроэнергию, оплату услуг обслуживающих серверный парк специалистов, затраты на ремонт и замену устаревших компонентов, и она напрямую зависит от объема эксплуатируемого оборудования. Очевидно, что эффективность использования ресурсов можно повысить, либо увеличив объем решаемых “полезных” задач, дающих доход или снижающих какие-либо затраты, либо уменьшив объем оборудования. В рамках данной диссертационной работы термин эффективность рассматривается только с точки зрения экономического определения.

**Реорганизация серверного парка** (при применении технологии виртуальных машин) – перенос логических серверов с аппаратной платформы физических компьютеров на платформы виртуальных машин, развертывание базовых ОС на физических компьютерах и размещение виртуальных машин на компьютерах в соответствии с некоторым вариантом распределения логических серверов по компьютерам.

**Корпоративная сеть** – сеть, основным назначением которой является поддержание работы конкретного предприятия, владеющего данной сетью.

**IT (Information Technology)** – информационные технологии.

**ИВЦ МЭИ (ТУ)** – Информационно – Вычислительный Центр Московского Энергетического Института (Технического Университета).

## Введение

**Актуальность.** В настоящее время информационные технологии проникли практически во все сферы деятельности человека, и он не представляет свою работу без использования компьютера и Интернет, а серьезная организация – без многофункциональной и хорошо защищенной корпоративной сети. Как человек, так и организации заинтересованы в надежности, безопасности хранения своих данных и высокой скорости их обработки. С другой стороны, как персональный компьютер, так и корпоративная сеть требует определенных затрат на приобретение, настройку и техническую поддержку в эксплуатации, и здесь, как и во всем остальном, человек стремится снизить затраты, не теряя при этом в качестве. Однако, если в случае отдельного персонального компьютера не так уж сложно найти компромисс между ценой и желаемыми техническими параметрами, то в случае корпоративной сети с тысячами рабочих мест, сотнями серверных систем для того, чтобы найти оптимальное решение требуется значительная аналитическая работа, а зачастую и серьезные вычислительные возможности.

На сегодняшний день существует большое количество организаций, имеющих корпоративную сеть, состоящую из множества конечных рабочих мест пользователей и некоторого, так называемого серверного парка, который предоставляет широкий спектр сервисов: доступ в Интернет, корпоративная почта, антивирусная защита, файловые ресурсы, службы печати и многое другое. Как правило, изначально корпоративная сеть грамотно проектируется специалистами с учетом надежности, безопасности и многофункциональности, и руководство организаций крайне отрицательно относятся к внесению значительных или даже небольших изменений в сеть, которая уже много лет исправно функционирует и удовлетворяет всем требованиям. Однако, тем не менее руководство всегда интересуют возможности снижения затрат на содержание сети и специалистов, обслуживающих ее, а также получения дополнительной прибыли с используемого технического оборудования.

Для конечных рабочих мест характерно то, что, как правило, для них выделяются компьютеры, которые в той или иной степени уступают по техническим параметрам компьютерам, используемым в парке серверов. Кроме того, пользователи используют множество приложений, которые могут на 100% загружать процессор, “съедать” всю оперативную память, помимо этого, пользователи часто размещают на своих персональных компьютерах данные, не относящиеся к работе (музыка, видео), которые могут занимать все дисковое пространство. Наконец, пользователи вправе считать и требовать, что все ресурсы их рабочего компьютера принадлежат задачам и приложениям, используемыми ими. В таких условиях, практически отсутствует возможность и, главное, целесообразность повышения загрузки ресурсов на рабочих компьютерах. Использование же серверов приложений, когда множество пользователей использует один мощный сетевой вычислительный ресурс для запуска своих приложений, частично решает проблему, но как, показывает практика, рабочие компьютеры все равно остаются достаточно сильно нагруженными. Наконец, по элементарным соображениям информационной безопасности недопустимо размещение каких-либо серверных функций или чужих приложений или данных на пользовательском компьютере.

Что же касается компьютеров серверного парка, то многолетняя практика эксплуатации серверных операционных систем и сетевых служб показала то, что на сегодняшний день большинство компьютеров серверного парка достаточно слабо загружены по ресурсам. Такая ситуация сложилась по следующей причине: рынок аппаратных решений развивается стремительно и производители оборудования очень быстро отказываются поддерживать старое оборудование, которое можно было бы эффективно использовать для размещения некоторых серверных служб. Так, например, один из важнейших элементов хорошо защищенной корпоративной сети – контроллер домена, по сегодняшним меркам крайне малотребователен к ресурсам: 2-3% среднесуточной загрузки процессора класса Pentium, сетевой трафик – несколько десятков или сотен килобайт данных, которые передаются не

постоянно, а однократно через определенные периоды (от 15 минут до 1 часа или даже реже), памяти требуется не более 64 МБ, дискового пространства требуется порядка 1-1.5 Гб. В тоже время, по соображениям безопасности, на контроллере домена крайне не рекомендуется размещать какие-либо иные серверные службы (файловые ресурсы, WEB-серверы и т.д.), то есть одному лишь контроллеру со столь низкими требованиями необходим целый компьютер. Конечно, можно было бы подобрать адекватную конфигурацию компьютера для такого малотребовательного сервера, однако, тот же жесткий диск емкостью 1.5-2 Гб на сегодняшний день найти достаточно сложно, а новые диски такой емкости не производятся уже давно. Использовать старые диски 10-летней давности крайне неразумно и чревато: они могут выйти из строя в любой момент, а гарантии и техническая поддержка на них отсутствует. Наконец, любой разумный и ответственный человек вряд ли захочет связываться со столь рискованным и устаревшим оборудованием. В таких условиях, руководство фирмы вынуждено приобретать для сервера современные диски емкостью от 40 Гб и выше, модули памяти емкостью от 256 МБ, современные процессоры, прекрасно осознавая, что большая часть каждого из ресурсов сервера будет безнадежно простаивать. Тем не менее, выход из данной ситуации есть – это применение технологии виртуальных машин, позволяющей функционировать нескольким операционным системам вместе со всеми ее службами – логическим серверам – на одном физическом компьютере. При этом с точки зрения безопасности в корпоративных сетях, обеспечивается максимальная изоляция каждой ОС вместе с ее службами от остальных ОС, они просто разделяют одни и те же аппаратные средства. Таким образом, серверный парк со слабой загрузкой ресурсов может быть существенно оптимизирован, что в конечном счете должно привести к уменьшению объема используемого оборудования и затрат на его поддержку. Кроме того, освободившееся оборудование может быть использовано для получения дополнительной прибыли. Данная диссертация посвящается разработке методики повышения эффективности использования ресурсов компьютеров.



**Научная новизна.** В рамках данной диссертации для решения новой проблемы повышения эффективности использования вычислительных ресурсов при условии применения технологии виртуальных машин разработана методика на основе новой математической модели, предложенной в диссертации. Разработана математическая модель и предложен метод решения задачи поиска оптимального распределения логических серверов по компьютерам при применении технологии виртуальных машин.

Сама по себе задача повышения эффективности использования ресурсов является достаточно старой и в каждой конкретной сфере деятельности человека решается своими методами. Однако, технология виртуальных машин в корпоративных сетях стала применяться относительно недавно и до сих пор пока не ставилась задача повышения эффективности использования ресурсов компьютеров серверного парка при условии применения этой технологии. Виртуальные машины на сегодняшний день чаще всего применяются для развертывания тестовых систем и для предварительной оценки качества IT-решений. Реже встречается применение этой технологии для пробного размещения 2-3 логических серверов на одном физическом компьютере, однако, такое применение еще довольно мало распространено и пока еще не существует какой-либо методики для реорганизации серверного парка с целью повышения эффективности использования ресурсов при условии применения данной технологии. Наконец, задача поиска оптимального распределения ресурсов, на решение которой должна опираться методика, применительно к рассматриваемой проблеме является новой и требует разработки специальной математической модели этой задачи и поиска метода ее решения.

**Практическая ценность.** В данной диссертации разработана методика по реорганизации серверного парка с целью повышения эффективности использования ресурсов компьютеров. Наиболее трудоемкая часть разработанной методики реализована в виде программного инструмента, с помощью которого специалист-аналитик на основе предварительного сбора информации о серверном парке корпоративной сети заказчика IT-услуг может

получить первичное распределение логических серверов по физическим компьютерам при применении технологии виртуальных машин и принять решение о целесообразности проведения реорганизации серверного парка. В положительном случае аналитик на основе полученного распределения может подготовить проектное решение о построении нового или реорганизации существующего серверного парка корпоративной сети. Это решение, безусловно требует определенных затрат на реализацию, но в будущем приносит существенную прибыль за счет снижения объема эксплуатируемого оборудования. Таким образом, диссертационная работа дает методику для снижения затрат организации и получения дополнительной прибыли от капиталовложений в информационные технологии, что, безусловно, имеет практическую ценность в современных условиях жесткой конкуренции.

**Результаты, полученные автором.** В рамках диссертационной работы автором были получены следующие основные результаты:

- Для поставленной проблемы повышения эффективности использования вычислительных ресурсов разработана методика реорганизации серверного парка, позволяющая решить эту проблему, на базе модели задачи поиска оптимального распределения логических серверов по компьютерам при применении технологии виртуальных машин.
- Разработана технология переноса операционных систем с физической аппаратной платформы на виртуальную платформу, которая делает методику реорганизации серверного парка при применении технологии виртуальных машин технически реализуемой.
- Разработана математическая модель задачи поиска оптимального распределения множества логических серверов на множество компьютеров при применении технологии виртуальных машин.
- Предложена схема разбиения, с использованием аппарата динамического программирования, исходной задачи на множество подзадач поиска распределения логических серверов на один компьютер, являющихся задачами условной псевдобулевой оптимизации. Для решения подзадач

предложен модифицированный метод локального поиска с использованием функции штрафов, управляемого радиуса зон поиска и множества случайных стартовых точек. Выполнены анализ сходимости, оценки объема перебора и качества решений для предложенного метода решения.

- Разработана программная реализация алгоритма решения задачи поиска оптимального распределения логических серверов по компьютерам.
- Проведено экспериментальное исследование на ряде модельных примеров и производственных задач при внедрении результатов диссертации с использованием разработанного программного обеспечения и предложенной методики.

**Область применения разработанной методики.** Следует отметить, что разработанная методика имеет определенные ограничения по применению:

- С точки зрения запасов по ресурсам серверного парка нецелесообразно применение методики в серверных парках, где все компьютеры постоянно около 100% загружены по какому-либо ресурсу или ресурсам (например, вычислительные кластеры, работающие с большим непрерывным потоком математических задач). Методика дает эффект лишь тогда, когда есть хотя бы какой-то запас по каждому из ресурсов на компьютерах рассматриваемого серверного парка. Это характерно для компьютеров с серверной ОС, под управлением которой выполняется некоторое множество сетевых служб, не слишком требовательных к ресурсам (например, контроллер домена, сервер сертификатов или DNS-сервер).
- С точки зрения времени отклика серверных приложений и служб не рекомендуется использовать методику в системах реального и жесткого реального времени, где нарушение крайних сроков недопустимо. Применение методики уместно только в случае, если допустимы некоторые задержки в разумных пределах. Например, в ситуации, когда не так уж критично то, что контроллеры одного домена синхронизируются через каждый час и может произойти задержка начала синхронизации от нескольких десятков миллисекунд до нескольких секунд.

- С точки зрения однородности сети передачи данных методику нежелательно применять для серверных парков, компьютеры которых связаны сложной сетью с разнородными по скорости и надежности участками, а также, если имеются отдельные компьютеры или группы, которые сильно удалены от остальных компьютеров. Благоприятный случай для использования методики – это серверный парк, компьютеры которого расположены в одном или нескольких помещениях (в относительно недалеко расположенных зданиях), компьютеры соединены друг с другом небольшим числом коммутаторов, однородных по функциональным возможностям и характеристикам, и некоторого множества линий связей, однородных по свойствам и характеристикам. Пример идеального случая – серверный парк, компьютеры которого находятся в одном помещении и подключены по витой паре одной категории к одному коммутатору.
- С точки зрения надежности функционирования применение методики не даст особого эффекта в серверном парке, в котором имеются серьезные проблемы с обеспечением надежности функционирования критически важных логических серверов (отсутствуют дополнительные логические серверы, дублирующие функции критически важных серверов). В такой ситуации, поскольку использование методики еще больше снижает надежность функционирования из-за размещения нескольких логических серверов на одном компьютере, то придется создавать дополнительные логические серверы, дублирующие функции критически важных серверов. Поскольку дополнительные логические серверы (их может оказаться немало, вплоть до удвоения общего числа логических серверов) требуют дополнительные ресурсы, то если бы при исходном количестве логических серверов использование методики могло освободить один или несколько компьютеров, дополнительные же логические серверы могли всех их занять. Поэтому использование методики даст лучший эффект для тех серверных парков, в которых для всех критически важных логических серверов уже имеются логические серверы, дублирующие их функции.

**Внедрение.** В рамках диссертационной работы разработана методика повышения эффективности использования вычислительных ресурсов при применении технологии виртуальных машин, а также программное обеспечение, реализующее предложенный в диссертации алгоритм решения задачи поиска распределения логических серверов по физическим компьютерам. Разработанная методика и программа внедрены в ИВЦ МЭИ (ТУ), г. Москва. Акт о внедрении находится в приложении 6.

**Публикации.** Опубликовано 4 статьи по теме диссертации:

- Рахман П.А. Подходы к повышению эффективности использования вычислительных ресурсов корпоративных сетей // Труды международной конференции “Информационные средства и технологии”. – М.: Янус-К, 2004. – Т. 3. – С. 120-121.
- Рахман П.А. Использование методов дискретной оптимизации для решения задач распределения ресурсов при применении технологии виртуальных машин в корпоративных сетях // Труды международной конференции “Информационные средства и технологии”. – М.: Янус-К, 2004. – Т. 3. – С. 122-123.
- Рахман П.А. Проблемы переноса современных операционных систем с реальной аппаратной платформы на виртуальную // Труды международной конференции “Информационные средства и технологии”. – М.: Янус-К, 2004. – Т. 3. – С. 124-125.
- Рахман П.А. Концептуальный подход к повышению эффективности использования вычислительных ресурсов корпоративных сетей при применении технологии виртуальных машин // Объединенный научный журнал. – М.: Тезарус, 2005. – № 2. – С. 59-67.

## Постановка задачи

Задано множество компьютеров  $\{H_k\}$ ,  $k=1..NH$  серверного парка. Каждый  $k$ -й компьютер предоставляет определенный набор ресурсов, базовые уровни которых заданы в виде множества  $\{R_{i,k}\}$ ,  $i=1..NC$ ,  $k=1..NH$ . Базовый уровень каждого  $i$ -го типа ресурса представляет собой ключевую техническую характеристику определенной аппаратной компоненты  $k$ -го компьютера, например,  $R_{1,k}$  – емкость оперативной памяти,  $R_{2,k}$  – емкость дисковой подсистемы,  $R_{3,k}$  – пропускная способность сетевой подсистемы,  $R_{4,k}$  – вычислительная мощность процессора и т.д. Задано множество логических серверов  $\{S_j\}$ ,  $j=1..NS$ , каждая из которых в любой момент времени функционирует на одном конкретном  $k$ -м компьютере. Каждый логический сервер для возможности его полноценного функционирования предъявляет определенный набор требований  $\{Q_{i,j}\}$ ,  $i=1..NC$ ,  $j=1..NS$ , к вышеприведенному множеству ресурсов.

Необходимо разработать методику реорганизации серверного парка с целью повышения эффективности использования вычислительных ресурсов серверного парка, учитывая то, что заказчиками ИТ-услуг, как правило, не допускается внесение изменений в конфигурацию физических компьютеров, в службы логических серверов и в настройки для рабочих мест конечных пользователей. Методика также должна включать в себя предварительную оценку целесообразности проведения реорганизации и в положительном случае предлагать решение, дающее более эффективное использование вычислительных ресурсов.

# 1. Обзор существующих подходов к повышению эффективности использования ресурсов и моделей распределения ресурсов

## 1.1. Обзор существующих подходов к повышению эффективности использования вычислительных ресурсов

На сегодняшний день существует большое количество организаций, имеющих более или менее сложную корпоративную сеть [3] и решающих самые различные виды производственных задач. В таких условиях, как правило, в каждой организации своя специфика и свои подходы к снижению затрат на обслуживание, развитию корпоративной сети и получению от нее дополнительной прибыли. Задача повышения эффективности использования ресурсов также в каждом конкретном случае решается с учетом особенностей и специфики, присущей организации. В рамках данной диссертации мы будем рассматривать эффективность только как отношение денежных средств, прямо или косвенно получаемых за счет решения задач серверным парком к совокупной стоимости владения серверным парком.

Помимо основного критерия – эффективности, при выборе каких-либо конкретных подходов к повышению эффективности использования ресурсов следует также учитывать еще 4 немаловажных критерия:

- Изоляция несовместимого программного обеспечения. Определенные виды программного обеспечения, сервисных функций в силу особенностей их разработки не могут функционировать в рамках одной операционной системы [4] либо могут, но вызывают серьезные сбои или крах системы.
- Безопасность корпоративной сети. Данные пользователей корпоративной сети должны быть защищены от несанкционированного доступа. Программное обеспечение должно быть размещено на серверах и сконфигурировано таким образом, чтобы обеспечить устойчивость к вирусным и сетевым атакам.

- Надежность аппаратного обеспечения (оборудования). Надежность физических компьютеров, коммуникационного оборудования и линий связи должна быть достаточно высокой, чтобы обеспечить бесперебойную работу серверного парка. Единичный сбой или даже крах на одном из компьютеров не должен приводить к фатальным последствиям: сбою работы всей сети, потере критических данных.
- “Прозрачность” для пользователей и обслуживающих специалистов: какие-либо преобразования в серверном парке не должны требовать внесения изменений на рабочих ПК пользователей, также такие преобразования, по возможности, не должны затрагивать логическую структуру сети с точки зрения привязки тех или иных служб к тем или иным логическим серверам. Например, если некий логический сервер MAILSRV ассоциируется у пользователей и специалистов как почтовый сервер, то после преобразований MAILSRV должен остаться почтовым сервером.

На сегодняшний день можно выделить несколько общеизвестных подходов к повышению эффективности использования ресурсов. Кратко рассмотрим их, анализируя их с точки зрения вышеприведенных критериев.

#### 1.1.1. Использование ресурсов для внутренних задач

При данном подходе [1, 11, 18] явным образом затраты на обслуживание не снижаются и не извлекается дополнительная прибыль, а увеличивается объем решаемых задач с целью повышения надежности хранения и обработки данных, что может в будущем предотвратить непредвиденные затраты в случае серьезных сбоев или потери данных. Свободные ресурсы привлекаются для дублирования каких-либо задач: например, на свободном дисковом пространстве размещаются резервные копии (или дополнительные резервные копии) данных, простаивающие вычислительные ресурсы используются для дублирующей обработки задач с целью снижения риска ошибок. Современные операционные ОС предоставляют богатые возможности для использования этого подхода: так, например, ОС MS Windows NT/2000 имеет встроенную



поддержку системы распределенного хранения данных в сети [48] – Distributed File System (DFS). Простаивающие ресурсы процессора и памяти можно задействовать для запуска дополнительных задач, таких как автоматическое резервное копирования данных, мониторинг состояния сети, антивирусные мониторы, слежение за внешними сетевыми атаками на корпоративную сеть, контроль состояния оборудования компьютера и т.д. В качестве примера можно привести серверный парк ИВЦ МЭИ (ТУ), в котором задействовано множество компьютеров в качестве серверов СУБД, на них имеется свободное дисковое пространство, и на каждом таком компьютере помимо основной БД, хранятся резервные копии БД других серверов СУБД.

Однако, зачастую даже введение дополнительных задач не всегда дает существенного повышения загрузки ресурсов серверного парка. Кроме того, отсутствие явной ощутимой выгоды также делает данный подход не слишком привлекательным. Наконец, самое главное то, что по соображениям информационной безопасности и устойчивости функционирования операционной системы с определенным набором служб, зачастую бывает недопустимо размещение дополнительных данных и задач. Каждая задача в данном случае – это некоторая программа (зачастую это сетевое приложение), которая имеет уязвимости, так называемые “дыры в безопасности”, соответственно, каждая дополнительная программа – это возможные дополнительные уязвимости для операционной системы [2]. Эти уязвимости могут успешно быть использованы злоумышленником для доступа к конфиденциальным данным, для атак типа “отказ в обслуживании” [17], когда приложение в силу внутренних ошибок зависает, при этом может “съесть” все ресурсы процессора, памяти и сетевой подсистемы, что немедленно отразится на функционировании основных серверных приложений. Наконец, в последнее время стали популярны атаки с использованием уязвимостей в приложениях, парализующие коммутационное оборудование. Таким образом, одно единственное “не на совесть сделанное” приложение, дополнительно запущенное на компьютере, вполне может привести к зависанию не только ОС

этого компьютера вместе со всеми ее службами, но и парализовать всю сеть. Поэтому размещение дополнительных программ и данных, особенно без изоляции их от основных программ и данных ОС конкретного компьютера может резко негативно отразиться на безопасности корпоративной сети.

Подводя итог, можно сказать, что рассмотренный подход может быть выгоден с точки зрения эффективности использования ресурсов при наличии дополнительных внутренних задач, решение которых дает ощутимую прибыль. С точки зрения надежности оборудования ничего не меняется, кроме того, появление дополнительных задач на логических серверах проходит прозрачно, как минимум, с точки зрения пользователей. С точки зрения безопасности сети, могут быть серьезные проблемы в плане устойчивости к сетевым и вирусным атакам. Также большой проблемой является отсутствие изоляции основных и дополнительных задач (программного обеспечения).

#### 1.1.2. Использование ресурсов для задач сторонних организаций

Данный подход [1, 11, 18] аналогичен предыдущему, за исключением того, что ресурсы предоставляются не для дополнительных внутренних задач, а сторонним организациям за определенную арендную плату, тем самым увеличивается объем решаемых задач с целью извлечения дополнительной прибыли. В данном случае к главному недостатку – угрозе безопасности функционирования корпоративной сети, добавляется серьезная угроза конфиденциальности данных организации, поскольку сторонняя организация вполне может использовать специальные программы для хищения информации. Эти программы, как правило, будут выполняться в одной ОС вместе с рабочими программами организации, сдающей свободные ресурсы в аренду. Несмотря на наличие явной прибыли, такой подход совершенно неприемлем с точки зрения безопасности данных пользователей [2, 17, 46].

Следует отметить, что существует множество организаций, для которых основной деятельностью является предоставление ресурсов своей сети другим организациям, например, размещение WEB-сайтов, FTP-сайтов и т.д. Но в этом

случае мы говорим не о дополнительных свободных ресурсах некоторой организации, для которой такого рода деятельность вполне может не являться основной. В данном случае, мы имеем дело с основными ресурсами, используемыми для решения основных задач. Организация, предоставляющая такого рода услуги, изначально проектирует свою сеть с учетом того, что она в основном будет использоваться для чужих задач и данных, и принимает все меры безопасности по их изоляции от своих внутренних информационных процессов. Для организации же типа банка, в которой, как правило, имеется хорошо защищенная и зачастую даже физически изолированная от Интернета сеть, попытка использования простаивающих ресурсов для размещения чужих WEB-сайтов, подключив при этом внутреннюю сеть к Интернет, закончится самым фатальным образом. Потери от кражи или несанкционированного доступа к критически важной информации может на порядки превышать доходы от решения “чужих” задач.

Подводя итог, можно сказать, что рассмотренный подход может быть выгоден с точки зрения эффективности использования ресурсов при наличии дополнительных задач внешних заказчиков, решение которых дает ощутимую прибыль. С точки зрения надежности оборудования ничего не меняется, кроме того, появление дополнительных задач на логических серверах проходит прозрачно, как минимум, с точки зрения пользователей. С точки зрения безопасности сети, могут быть серьезные проблемы в плане устойчивости к сетевым и вирусным атакам, а также защиты данных пользователей предприятия, сдающего свои вычислительные ресурсы в аренду. Также большой проблемой является отсутствие изоляции основных и “чужих” дополнительных задач (программного обеспечения).

### 1.1.3. Применение адекватных аппаратных решений

Данный подход [8, 11, 12] предполагает подбор аппаратных конфигураций по техническим параметрам соответствующих требованиям тем приложениям и операционной системе, которые на них будут выполняться.

Тем самым делается попытка снижения объема оборудования, требуемого для решения задач организации. Однако, этот подход больше годится на этапе проектирования серверного парка, а не тогда, когда все программное обеспечение уже размещено и функционирует. Иначе говоря, мы изначально пытаемся максимально сократить разрыв между базовыми уровнями ресурсов и требованиями по каждому типу ресурсов. Однако, на сегодняшний день рынок аппаратных решений таков, что крайне сложно придерживаться такого условия. Как упоминалось ранее, найти комплектующие части для компьютера с заданными точными требуемыми техническими характеристиками практически невозможно, рынок предлагает некоторый очень дискретный ряд параметров комплектующих. Более того, производители и дилеры очень быстро отказываются поддерживать старое оборудование и обслуживать их по гарантии, имеющие характеристики, которые как нельзя лучше подошли бы для решения многих серверных задач. Все это вынуждает организации приобретать современные комплектующие с высокими техническими параметрами. Серверная ОС для роли контроллера домена не использует даже 10% ресурсов (за исключением оперативной памяти) самого слабого компьютера, собранного из новых комплектующих. Разумеется, существуют специализированные серверные решения, но их технические параметры еще выше и цены на них, как правило, очень высоки и многим малым и даже средним масштабным организациям это не по карману. Кроме того, приобретать такой агрегат ради размещения того же самого контроллера домена, требующего для себя отдельный компьютер – это крайне дорогое удовольствие. Использование же старого оборудования, с возрастом зачастую от 5 лет и более, в крайней степени неразумно по очевидным причинам – риск выхода из строя в любой момент. В таких условиях адекватные аппаратные решения легче выбирать для игровых и графических приложений, нежели чем пытаться решать эту задачу для малотребовательных сетевых служб [4, 43, 44].

Подводя итог, можно сказать, что рассмотренный подход может быть выгоден с точки зрения эффективности использования ресурсов, за счет

снижения стоимости эксплуатируемого оборудования, но неизвестно, какое применение найдет (и будет ли оно достаточно выгодным) ранее использованное оборудование. Также может резко пострадать надежность оборудования при использовании устаревших компонентов. С точки зрения прозрачности для пользователей и обслуживающих специалистов, изоляции несовместимых служб и безопасности сети ничего не меняется.

#### 1.1.4. Объединение сервисов и снижение числа серверных ОС

При данном подходе [1, 11, 18] делается попытка снижения объема оборудования за счет снижения числа логических серверов путем объединения сервисов, предоставляемых различными логическими серверами.

На каждом компьютере серверного парка, как правило, функционирует некоторая операционная система, под управлением которой работает некоторое множество серверных приложений, предоставляющих определенные сервисы (службы). Службы распределяются между теми или иными компьютерами по соображениям безопасности, по рекомендациям разработчиков ОС и серверных приложений, с учетом специфики деятельности организации и многих других факторов. На рисунке 1.1 приводится пример корпоративной сети с относительно невысоким уровнем защищенности для небольшой организации.

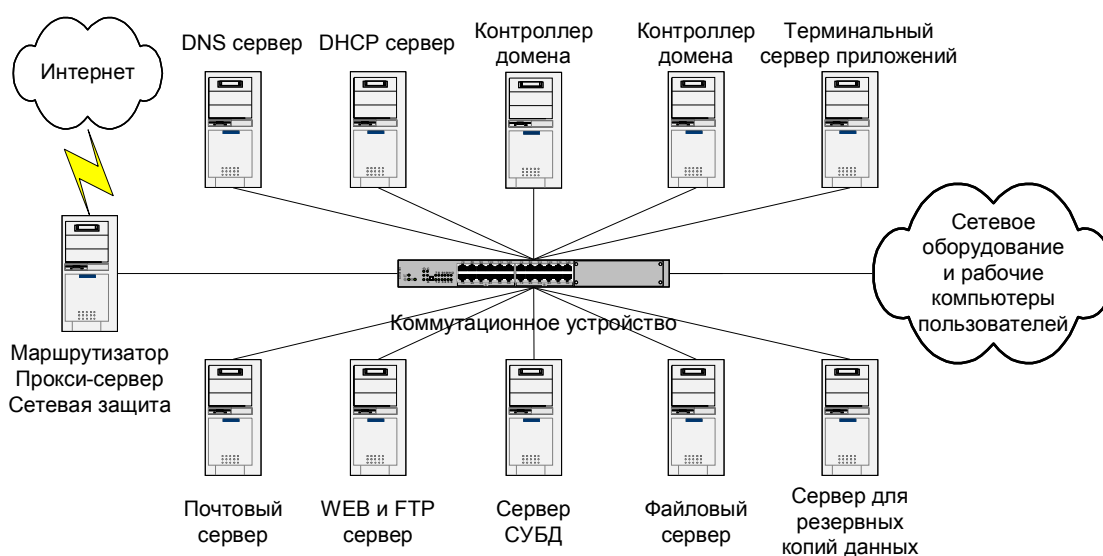


Рис. 1.1. Пример корпоративной сети

В приведенном примере имеется некоторое множество компьютеров, выполняющих серверные функции в сети, причем сетевые службы разнесены по отдельным компьютерам по техническим соображениям специалистов-проектировщиков, специалистов, ответственных за обслуживание сети, и специалистов по безопасности. Серверный парк этой сети можно без особого труда построить на базе компьютеров, использующих одну и ту же серверную ОС, например, MS Windows 2000 Server [1, 18, 20], в которую встроено большинство необходимых сетевых служб таких как: DNS (система доменных имен), DHCP (протокол динамического конфигурирования сетевых настроек компьютеров сети), IIS (сервисы для публикации ресурсов в Интернет, которая обеспечивает возможность развертывания собственных WEB, FTP-сайтов) и многое другое. Отсюда следует то, что в предельном случае теоретически можно собрать все службы под управление одной серверной ОС одного единственного компьютера, а остальные компьютеры освободить от их основных функций и использовать альтернативным способом.

Однако, в этом подходе есть несколько негативных моментов:

- Объединение служб всегда сказывается на безопасности, разные службы имеют разный уровень уязвимости, а разные компьютеры серверного парка – разный уровень важности для организации. Так, например, компьютер, выполняющий функцию контроллера домена, имеет наивысшую важность, поскольку он хранит сведения о пользователях и группах, корпоративной политики безопасности и многое критичного с точки зрения безопасности и его, как правило, максимально защищают и ведущие специалисты в области безопасности крайне не рекомендуют размещать на нем какие-либо иные службы. В то же время WEB, FTP-серверы, серверы СУБД на сегодняшний день являются одними из наиболее уязвимых мест для сетевых атак, а файловые серверы – для вирусных атак. В таких условиях, размещение их на компьютерах, выполняющих функции контроллеров доменов, DNS-серверов или серверов корпоративной почты, является крайне небезопасным. Иначе говоря, перенос более уязвимого серверного приложения с менее важного

компьютера на более важный компьютер делает последний более уязвимым, что не соответствует его уровню важности для организации.

- Существует такая извечная проблема как совместимость программного обеспечения, функций и ролей. Так, например, один компьютер не имеет возможности под управлением одной ОС выполнять функции нескольких контроллеров доменов в силу специфики самой функции и операционной системы. Кроме того, современные ОС и серверные приложения настолько усложнились, что зачастую содержат множество специфических ошибок, не позволяющих корректно работать под управлением одной ОС нескольким приложениям разных, а иногда даже и одного и того же производителя.
- Руководство и IT-специалисты организации крайне негативно относятся к перемещениям служб между компьютерами серверного парка, поскольку это зачастую приводит к необходимости изменения настроек на рабочих местах множества пользователей, внесения изменений в документацию, а иногда даже перераспределения обязанностей между ответственными за обслуживание сети. Иначе говоря, меняется некоторая логическая структура размещения сетевых служб, к которой пользователи, специалисты и руководство привыкают годами. Так, например, если в некоторой организации было принято, что SRV1 – это почтовый сервер, SRV2 – контроллер домена, SRV3 – файловый сервер, то перемещение функций файлового и почтового серверов на SRV2 с целью освободить оборудование SRV1 и SRV3 внесет очевидные изменения в логической структуре и повлечет за собой дополнительные работы.

Подводя итог, можно сказать, что рассмотренный подход может быть выгоден с точки зрения эффективности использования ресурсов, поскольку снижение числа серверных ОС снижает и объем эксплуатируемого оборудования (число компьютеров). С точки зрения надежности оборудования возникают проблемы, поскольку сбой или отказ конкретного компьютера приводит уже к нарушению большего числа служб, кроме того, перераспределение служб, очевидно, по определению не будет прозрачным для

пользователей и обслуживающих специалистов. С точки зрения безопасности сети, могут быть серьезные проблемы в плане устойчивости к сетевым и вирусным атакам из-за объединения служб. Также большой проблемой является отсутствие изоляции несовместимых служб.

#### 1.1.5. Применение технологии виртуальных машин

В рассмотренных выше подходах мы, так или иначе, сталкивались с серьезными проблемами информационной безопасности, совместимости программного обеспечения, нарушения некоторой логической структуры размещения служб и функций среди компьютеров сети, а также сложности подбора адекватных аппаратных конфигураций. Первые три проблемы возникают потому, что в серверную операционную систему некоторого компьютера к существующим приложениям добавляются дополнительные свои либо чужие приложения, либо службы от других серверов. Когда в пределах одной ОС оказываются множество работающих приложений, то вопросы безопасности ОС в целом и совместимости приложений неизбежны. Поскольку сетевые имена присваиваются не компьютерам, а операционным системам функционирующих на них, то перемещение служб с одного компьютера на другой меняют привязку служб от одного сетевого имени сервера к другому.

Здесь возникает закономерный вопрос: а нельзя ли перенести не просто службы с другого компьютера, а всю ОС вместе с ее службами и уникальным сетевым именем, при этом добиться максимальной изоляции операционных систем друг от друга? Технология виртуальных машин [5, 9, 10] отвечает на этот вопрос положительно. Действительно, эта технология позволяет на одном и том же физическом компьютере эмулировать работу множества виртуальных машин, на каждую из которых можно разместить ОС с ее уникальным сетевым именем и набором служб. Причем операционные системы с точки зрения безопасности и совместимости приложений изолируются настолько, что функционируют так, как будто находятся на различных физических компьютерах и могут взаимодействовать друг с другом не иначе как через сеть.



Таким образом, нам удастся решить проблемы безопасности серверной ОС, совместимости приложений и неизменности логической структуры, и повысить тем самым эффективность использования ресурсов компьютеров за счет размещения нескольких ОС на одном и том же оборудовании, что также отбросит необходимость подбора адекватных аппаратных решений.

Подводя итог, можно сказать, что рассмотренный подход выгоден с точки зрения эффективности использования ресурсов, поскольку снижается объем эксплуатируемого оборудования (число компьютеров). Кроме того, сохраняется изоляция несовместимых служб, перевод логических серверов на виртуальную платформу остается совершенно прозрачной для пользователей, никак не отражается на безопасности корпоративной сети с точки зрения устойчивости к вирусам, сетевым атакам и несанкционированному доступу к данным. Единственный недостаток – неизбежное снижение надежности из-за размещения нескольких логических серверов на одном компьютере. Тем не менее, из всех вышерассмотренных подходов применение технологии виртуальных машин является наиболее приемлемым подходом, а вопрос надежности вполне можно разрешить.

Рассмотрим более подробно те проблемы, которые могут возникнуть при использовании этого подхода:

- В первом приближении неясно, каким образом перераспределять и объединять логические серверы среди компьютеров. Очевидно, что необходимо оценивать требования логических серверов, технические характеристики компьютеров, после чего каким-то образом размещать логические серверы на компьютеры, чтобы добиться хорошей загрузки используемого оборудования, но отсутствует четкий алгоритм того, как это делать. Однако, эта проблема вполне решаема и здесь требуется постановка математической задачи поиска оптимального распределения логических серверов на физические компьютеры и разработка метода ее решения.
- Применение технологии виртуальных машин требует использования, так называемой базовой ОС, под управлением которой функционируют

виртуальные машины. Базовая ОС нуждается в определенных ресурсах компьютера, и здесь с этим приходится просто мириться.

- Следующая проблема – как известно, что любая операционная система, так или иначе, привязана к аппаратному оборудованию и содержит в себе некоторую аппаратно-зависимую часть. Как показала практика, некоторые ОС отказываются работать на другом оборудовании (жесткий диск переносится с одного компьютера на другой, сильно отличающийся по аппаратной конфигурации), обычно при попытке загрузки появляется неустраняемая ошибка на уровне ядра и дальнейшая работа невозможна. Очевидно, при переносе ОС с аппаратной платформы на виртуальную эта проблема будет стоять еще более остро. Тем не менее, выход из этой ситуации есть – это специально разработанные подготовительные процедуры, выполняемые в ОС, которую предстоит переносить. Так, например, за годы эксплуатации серверных систем, автором была разработана исчерпывающая технология переноса ОС семейства MS Windows 2000 с одной аппаратной платформы на другую аппаратную или виртуальную платформу (см. приложение 2).
- Следующая существенная проблема – надежность работы множества логических серверов на одном оборудовании. Очевидно, что при выходе из строя комплектующих частей компьютера, будет нарушена работа уже не одного, а нескольких логических серверов вместе со всеми их службами. Тем не менее, следует отметить, что на сегодняшний день большинство служб и функций могут дублироваться несколькими серверами, важно лишь то, чтобы серверы, на которых дублируется одна и та же функция были расположены на различных физических компьютерах. Этот момент несложно учесть: при постановке задачи поиска оптимального распределения заложить возможность задания и учета дополнительных ограничений-исключений, препятствующих размещению конкретных виртуальных машин на один и тот же физический компьютер.

- Наконец, последняя проблема – это, безусловно, некоторое увеличение времени обработки запросов и времени отклика серверных систем в случае одновременного поступления запросов к логическим серверам, работающих на одном и том же физическом компьютере. Однако, во введении были сделаны специальные оговорки о том, что мы не будем иметь дело с системами реального времени и с постоянной 100% загруженностью какого-либо из ресурсов, и будем считать, что задержки в обслуживании вполне допустимы в разумных пределах.

Более подробно технология виртуальных машин, ее особенности и основные программные реализации рассмотрены в приложении 3.

Таким образом, мы рассмотрели 5 основных подходов, используемых для повышения эффективности использования вычислительных ресурсов, установили, что подход с применением технологии виртуальных машин является наиболее приемлемым. Однако, на сегодняшний день отсутствует какая-либо методика по планированию нового или реорганизации существующего серверного парка с целью повышения эффективности использования вычислительных ресурсов при применении технологии виртуальных машин. Реорганизация проводится специалистами, как правило, на основе некоторого небольшого перебора вариантов (чаще всего вручную). Очевидно, что при большом количестве компьютеров и логических серверов перебор вариантов становится очень большим и полный перебор становится практически невозможным. Соответственно, возникает необходимость в разработке некоторой математической модели для задачи поиска распределения логических серверов по физическим компьютерам с учетом специфики, присущей технологии виртуальных машин, и поиске методов ее решения. Задача поиска распределения логических серверов по физическим компьютерам является разновидностью задач распределения ресурсов.

В рамках данной главы мы рассмотрим несколько существующих моделей распределения ресурсов.

## 1.2. Обзор моделей распределения ресурсов

На сегодняшний день проблемы эффективного распределения ресурсов (вычислительных, материальных, времени и т.п.) возникают и требуют решения в самых различных сферах деятельности человека. Соответственно, появляются новые задачи поиска оптимального распределения ресурсов, разрабатываются математические модели для них и разрабатываются новые либо усовершенствуются существующие методы решения.

Задача поиска оптимального распределения множества логических серверов по компьютерам при применении технологии виртуальных машин является новой и для нее не существует какой-либо готовой модели, учитывающей все ее особенности. Тем не менее, мы рассмотрим некоторые существующие модели ресурсов и покажем, что в чистом виде они не могут быть применены в силу специфики рассматриваемой задачи.

### 1.2.1. Традиционные модели распределения ресурсов

Поскольку задача поиска распределения логических серверов по компьютерам является задачей булевой оптимизации (конкретный логический сервер либо назначается, либо не назначается на конкретный компьютер), то в качестве одной из традиционных моделей распределения ресурсов можно рассмотреть модель задачи о рюкзаке [24, 36, 37, 40], которая принадлежит к классу задач условной псевдобулевой оптимизации.

#### **Модель задачи о рюкзаке**

Пусть имеется один поставщик, предоставляющий один тип ресурса с базовым уровнем  $b$ , и  $N$  потребителей с требованиями  $\{a_j\}$  ( $j=1..N$ ), предъявляемыми к ресурсу поставщика. Каждому  $j$ -му потребителю ресурс поставщика может быть предоставлен ( $x_j = 1$ ), либо нет ( $x_j = 0$ ),  $x_j$  – булева переменная назначения  $j$ -го потребителя на поставщика ресурса. Суммарные требования потребителей, которым предоставляются ресурсы поставщика, не могут превышать базовый уровень ресурса:

$$\begin{aligned} \sum_{j=1}^N a_j x_j &\leq b \\ \forall j \in [1, N]: x_j &\in [0, 1] \end{aligned} \quad (1.1a)$$

Пусть имеется некоторая целевая функция:

$$L = \sum_{j=1}^N c_j x_j \rightarrow \max \quad (1.1b)$$

где коэффициенты  $\{c_j\}$  отражают некоторую выгоду, получаемую при предоставлении  $j$ -му потребителю части имеющихся у поставщика ограниченных ресурсов.

Требуется отыскать такое распределение  $\{x_j\}$  потребителей на поставщика, при котором суммарная выгода (1.1б) достигает максимума, и соблюдаются ограничения (1.1a)

Нетрудно увидеть, что применительно к задаче поиска распределения логических серверов по компьютерам, модель (1.1), как минимум, не учитывает, что поставщиков (компьютеров) множество, типов ресурсов также множество. Таким образом, модель (1.1) в таком виде невозможно использовать, ее можно применить, только если существенно доработать так, чтобы она стала более или менее адекватной рассматриваемой проблеме поиска оптимального распределения логических серверов по компьютерам.

Существует еще ряд традиционных моделей распределения ресурсов, хорошо освещенных в литературе [13, 14, 15, 21-25, 36-40]. Однако, все эти модели, также как и рассмотренная выше модель задачи о рюкзаке, не учитывают те или иные особенности рассматриваемой задачи поиска оптимального распределения логических серверов по компьютерам, поэтому их рассмотрение мы оставим за рамками диссертации.

### 1.2.2. Современные модели распределения ресурсов.

В отличие от рассмотренных выше моделей, которые в большинстве своем являются стационарными, поскольку в них попросту отсутствует время ( $t$ ), и они по определению не способны учитывать какие-либо динамические изменения во времени, современные модели распределения ресурсов редко обходятся без такого важнейшего параметра как время.

В рамках данной главы мы кратко рассмотрим две модели: модель составления расписания, а также модель распределения ресурсов в системах жесткого реального времени.

#### **Модель составления расписания**

В моделях составления расписания [26, 27, 28, 29, 30, 31, 32, 33] делаются принципиальные допущения, основанные на том, что вся информация об операциях (элементарных процессах) заранее известна: 1) задано отношения предшествования операций, например в виде графа; 2) все операции должны быть выполнены; 3) известны длительность каждой операции и ее потребность в том или ином ресурсе; 4) на один экземпляр того или иного типа ресурса (например, на один процессор) в текущий момент времени может назначаться только одна операция.

Рассмотрим основные компоненты модели:

Пусть для множества  $U = \{u_1, \dots, u_N\}$  операций, связанных отношением предшествования, задан отрезок планирования  $[0, T]$ , где  $t \in [0, T]$  представляет собой независимую переменную времени. На множестве операций определено отношение предшествования  $\prec$  (например,  $A \prec B$ , означает, что  $A$  должно выполниться перед  $B$ ). Отношения предшествования представлены в виде, например, ориентированного бесконтурного графа  $(U, E)$ , с множеством  $U$  вершин, соответствующим операциям, и множеством дуг  $E \subseteq U \times U$ . При этом  $u_i \prec u_j \Leftrightarrow (u_i, u_j) \in E$ . Транзитивные дуги исключаются.

Пусть  $s_i, f_i$  – моменты начала и окончания операции  $u_i \in U, i = 1..N$ . Тогда в общем случае расписание  $Sh$  определяется как множество:

$$Sh = \{(s_i, f_i), i = 1, \dots, N, s_i, f_i \in [0, T]\} \quad (1.2a)$$

Пусть имеются  $J$  типов ресурсов. Заданы общие уровни наличия каждого типа ресурса во времени:

$$r_t = \{r_t^j, j = 1, \dots, J, t \in [0, T]\} \quad (1.2б)$$

В большинстве моделей полагается, что  $r_t^j$  не зависит от времени, соответственно, верхнюю границу ресурса  $j$ -го типа будем обозначать  $r_0^j$ .

Для  $i$ -й операции задается длительность  $\tau_{ij} > 0$  ее исполнения на  $j$ -м типе ресурса, причем, если  $i$ -я операция не может использовать  $j$ -й тип ресурса, то  $\tau_{ij} = \infty$ . Каждая операция характеризуется потребностью  $r_{ij}$  в ресурсах  $j$ -го типа, причем  $r_{ij} \leq r_0^j$ , для всех  $i$  и  $j$ .

Пусть задана удельная стоимость (или вес)  $w_i$  операции, которая чаще всего полагается постоянной во времени. Тогда стоимость завершения операции равна  $w_i * f_i$  и представляет собой неубывающую от срока завершения операции функцию.

Ограничения на составление расписаний могут быть заданы из имеющихся ресурсов и свойств системы операций. Для общего случая, когда возможны прерывания и назначения с задержкой, условия допустимости расписания выглядят следующим образом:

$$(\forall u_i \in U) f_i \leq T \quad (1.2в)$$

$$(\forall u_i, u_j \in U) u_i \prec u_j \Rightarrow f_i \leq s_j \quad (1.2г)$$

$$(\forall t \in [0, T], U_t = \{u_i \in U \mid s_i \leq t \leq f_i\}) r^J(t) = \sum_{u_i \in U_t} r_{ij} \leq r_t^j \quad (1.2д)$$

где,  $U_t$  – множество операций, выполняемых в момент времени  $t$ ;  $r^J(t)$  – потребность в ресурсе  $j$ -го типа, а  $r_t^j$  – уровень наличия ресурса  $j$ -го типа (1.2б) в момент времени  $t$ . Фиксация свойств архитектуры определяется неизменяемыми уровнями наличия каждого типа ресурса  $r_0^j$ , длительностью  $\tau_{ij}$ ,

потребностью в ресурсах  $g_{ij}$  и удельной стоимостью  $w_i$  каждой операции. Исходя из этого, формируется задача составления допустимого в соответствии с (1.2в), (1.2г) и (1.2д) расписания (1.2а).

Что же касается, эффективности расписания, то она, как правило, оценивается по одному критерию. Для наиболее общего случая построения расписания с прерываниями и назначениями с задержкой, в качестве основного критерия используется длина расписания  $Sh$ , определяемая как:

$$T(Sh) = \max_{1 \leq i \leq N} \{f_i\} \quad (1.2е)$$

В качестве дополнительных ограничений могут быть, например, заданы крайние сроки  $D_i \in (0, T]$ ,  $i = 1 \dots N$ , для окончания операций:

$$(\forall u_i \in U) f_i \leq D_i, D_i \in (0, T] \quad (1.2ж)$$

Общая постановка задачи составления расписания состоит в том, чтобы с помощью некоторого множества ресурсов выполнить фиксированную систему операций  $U$ . Цель – для определенных свойств операций и особенностей архитектуры, выраженных через доступные ресурсы (1.2б) и наложенные на них ограничения, построить оптимальное расписание (1.2а) при заданной мере эффективности (1.2е). При этом должны соблюдаться условия допустимости расписания (1.2в), (1.2г) и (1.2д).

### **Модель распределения ресурсов в системах жесткого реального времени**

Модель распределения ресурсов в системах жесткого реального времени [34] является одной из современных моделей, используемых в системах, где устанавливаются жесткие требования к гарантированному выполнению, как отдельных задач, так и некоторых составных действий (заданий, работ) до наступления заданных крайних сроков.

Прежде всего, отметим ключевое отличие рассматриваемой модели от модели составления расписаний: модель допускает изменение длительностей выполнения задач и их потребностей в ресурсах. Кроме того, ресурсы являются масштабируемыми. Соответственно, во-первых, вводится понятие базового



(основного) и вводимого (дополнительного ресурса), а во-вторых, в результате распределения получаются не длительности выполнения задач, а время, отведенное для их решения. Необходимость гарантированного выполнения работ до наступления крайних сроков требует учета динамики окружения системы реального времени: изменений параметров объекта управления, количества заявок на обслуживание, возможных отказов оборудования и т.п. Это приводит к тому, что в общем случае требуется не один, а множество вариантов распределения ресурсов, или *стратегия*.

Рассмотрим основные компоненты модели.

Пусть  $A = \{T_1, T_2, \dots, T_N\}$  – совокупность задач, которых должно быть завершено до наступления крайнего срока  $t^*$ . На  $A = P \cup D$  задано отношение частичного порядка  $\prec$  с помощью ориентированного бесконтурного графа  $(P, D)$ , множество  $P$  вершин которого соответствует задачам обработки и хранения информации, а множество  $D$  дуг – передаче данных. Заданием назовем последовательность  $(T_{i_1}, \dots, T_{i_k})$  информационно или логически связанных задач, таких, что  $T_{i_1} \prec \dots \prec T_{i_k}, 1 \leq i_1 \dots \leq i_k \leq n$ . Работа – набор заданий и отдельных задач.

Для краткости изложения будем считать, что ресурсы включают в себя процессоры и каналы обмена.

Для каждой задачи  $T_i \in A, i = 1 \dots n$ , имеются априорные оценки длительности ее выполнения  $\tau_{ij}^0$  на базовом ресурсе  $j \in \{1, \dots, J\}$ . Базовый ресурс  $j$ -го типа ограничен уровнем  $br_j$  – числом процессоров или каналов обмена данных. Задача  $T_i$  монопольно использует соответствующий ресурс в течение всего времени  $\tau_i$ , отведенного для ее решения. Если  $\tau_i < \tau_{ij}^0$ , то для решения задачи  $i$ -й задачи вводится процессор, по своим функциональным возможностям не уступающий базовому процессору, но более производительный. Вводимый ресурс будет обозначать индексами  $j^\circ$ . Вводимый ресурс может быть необходим в случае, если суммарный уровень потребности в ресурсе  $j$ -го типа для параллельно выполняемых задач

превышает уровень  $br_j$ . Соответственно, введем параметр  $\alpha_i$  – соответствует назначению  $i$ -й задачи на базовый ( $\alpha_i = j$ ) либо вводимый ( $\alpha_i = j^0$ ) ресурс.

Тогда можно определить распределения ресурсов между задачами  $\{T_1, T_2, \dots, T_N\}$  как множество:

$$R = \{(s_i, \tau_i, \alpha_i) \mid i = 1, \dots, n, \alpha_i = j \vee j^0, j = 1, \dots, J, s_i \in [0, t^*)\} \quad (1.3a)$$

где,  $s_i$  – момент запуска  $i$ -й задачи.

Необходимое условие введения более производительного процессора:

$$(\forall T_i \in P)(\tau_i < \tau_{ij}^0) \Rightarrow \alpha_i = j^0 \quad (1.3б)$$

Время, отводимое для обмена данными, не может быть меньше априорного, минимального значения, определяемого пропускной способностью канала соответствующего типа:

$$(\forall T_i \in D)(\tau_i < \tau_{ij}^0) \quad (1.3в)$$

Если последовательно запускаемые задачи обработки назначаются на один и тот же процессор, то соответствующее время обмена данными между ними и параметр назначения полагаются равными нулю.

Далее, пусть,  $A_t$  – множество задач, выполняемых в момент времени  $t \in [0, t^*]$ , а  $r_{ij}$  – потребность задачи  $T_i$  в  $j$ -м типе ресурса, причем  $r_{ij} = 1$ , если  $i$ -я задача назначается  $j$ -й ресурс,  $r_{ij} = 0$  в противном случае. Тогда необходимое условие разрешения конфликтов параллельных задач [35], конкурирующих за один и тот же ресурс  $j$ -го типа, можно записать следующим образом:

$$(\forall t \in [0, t^*]) \sum_{T_i \in A_t} r_{ij} > br_j \Rightarrow (\forall T_i \in A_t)(\exists j \in \{1, \dots, J\}) \alpha_i = j \vee j^0 \quad (1.3г)$$

Условие (1.3г) означает, что одна из двух конкурирующих задач должна быть назначена на вводимый ресурс.

Распределение  $R$  ресурсов (1.3а) считается допустимым при выполнении условий (1.3б), (1.3в) и (1.3г). Каждый из вариантов допустимого распределения можно представить вектором  $\rho = (\tau_1, \dots, \tau_n, a_1, \dots, a_n)$ , поскольку при заданном отношении предшествования и известном времени выполнения  $\tau_i$ ,  $i = 1, \dots, n$ , каждой из задач однозначно определяется момент ее запуска  $s_i$ .

Пусть на множестве задач  $\{T_1, T_2, \dots, T_n\}$  определено задание, состоящее из последовательно запускаемых логически связанных задач  $T_{i_k}$ ,  $k = 1, \dots, K$ .

Тогда в качестве критерия эффективности распределения ресурсов можно использовать аддитивно-сепарабельные критерий  $f(\rho)$ , отражающий эффективность выполнения всех задач  $T_i$ ,  $i=1, \dots, n$ , и  $f(\rho')$ , отражающий эффективность выполнения задач  $T_{i_k}$ ,  $k = 1, \dots, K$ , конкретного задания:

$$\begin{aligned} f(\rho) &= \sum_{i=1}^n f_i(\tau_i, \alpha_i); \rho = (\tau_1, \dots, \tau_n, \alpha_1, \dots, \alpha_n) \\ f(\rho') &= \sum_{k=1}^K f_{i_k}(\tau_{i_k}); \rho' = (\tau_{i_1}, \dots, \tau_{i_k}, \alpha_{i_1}, \dots, \alpha_{i_k}); 1 \leq i_1 \leq \dots \leq i_k \leq n \end{aligned} \quad (1.3д)$$

где  $f_i(\cdot)$  – некоторая частная функция эффективности выполнения  $i$ -й задачи, а  $f_{i_k}(\cdot)$  – некоторая частная функция выполнения  $i_k$ -й задачи, входящей в задание. Функция  $f(\rho')$  однозначно определяется переменными  $\tau_{i_k}$ ,  $k = 1, \dots, K$ , а назначения  $\alpha_{i_k}$  является функцией от  $\tau_{i_k}$ , поскольку задачи  $T_{i_1}, \dots, T_{i_k}$  не могут конкурировать между собой за использование ресурсов. Коллизии возможны между параллельными задачами различных заданий.

Наконец, помимо крайнего срока  $t^*$  завершения всей системы задач, заданы сроки  $t_g^*$  и  $t_h^*$ ,  $g, h \in \{1, \dots, n\}$ , завершения как отдельных, так и составных действий (заданий). Тогда добавляются еще одни ограничения:

$$t_g^* - \tau_g \geq 0, t_h^* - \sum_h \tau_h \geq 0; g, h \in \{1, \dots, n\} \quad (1.3е)$$

где  $\tau_g$  и  $\tau_h$  – время выполнения задач  $T_g, T_h \in A$ .

Постановка задачи распределения ресурсов в системе жесткого реального времени заключается в том, чтобы при заданном сроке  $t^*$  завершения всего комплекса работ и ограничениях (1.3е) из множества допустимых, согласно условиям (1.3б), (1.3в) и (1.3г), вариантов распределения ресурсов (1.3а), выделить  $\Phi$ -оптимальную стратегию распределения, где  $\Phi$  – бинарное отношение для сравнения вариантов распределения ресурсов, формируемое вектором  $F(\rho) = (f_1(\rho), \dots, f_L(\rho))$  аддитивно-сепарабельных критериев вида (1.3д).

Рассмотренные выше современные модели, очевидно, гораздо сложнее традиционных моделей, и каждая из них в соответствующих реальных проблемных ситуациях обеспечивает высокий уровень адекватности. Однако, при попытке использования их для задачи распределения логических серверов по физическим компьютерам при применении технологии виртуальных машин возникают определенные сложности:

Во-первых, логический сервер – сущность, сильно отличающаяся от операции или задачи. Для логических серверов также не может быть определено отношение предшествования (для отдельных серверных служб, находящихся на одном или разных логических серверах – можно задать такое отношение, но не для логических серверов). Логические серверы просто должны одновременно запускаться и потом работать параллельно (псевдопараллельно) и совместно обеспечивать те функции, которые на них возложены, и нет каких-то априорных сроков (в том числе и каких-либо крайних сроков), когда логические серверы прекратят свое существование.

Во-вторых, логический сервер – весьма “тяжеловесная” сущность: может занимать на диске от 1 до 100 Гб. Соответственно, масштабирование ресурсов (введение дополнительных компьютеров) вряд ли возможно в такой ситуации, поскольку перенос логического сервера на вводимый ресурс (перенос файлов, содержащих диски логического сервера, с одного компьютера на другой по сети) на современном коммуникационном оборудовании может продлиться до нескольких часов. Все это время сервер не сможет обеспечивать сервисы, от которых либо напрямую зависит работа пользователей, либо зависит работа каких-либо служб, находящихся на других логических серверах.

В-третьих, доступное на сегодняшний день программное обеспечение, реализующее технологию виртуальных машин, не имеет возможности встраивания собственных механизмов динамического управления логическими серверами и компьютерами. Запуск, останов, а также перенос логического сервера на другой компьютер может осуществляться только вручную.

Таким образом, очевидно, требуется разработка новой модели.

## Выводы по главе 1

В первой части главы проведен обзор существующих подходов к повышению эффективности использования ресурсов компьютеров. Помимо основного критерия – возможностей повышения эффективности использования ресурсов – сформулированы несколько немаловажных вторичных критериев: уровень изоляции несовместимого программного обеспечения, безопасность данных и программ, надежность функционирования серверного парка, “прозрачность” для пользователей. В соответствии с основным и вторичными критериями рассмотрены и критически проанализированы существующие подходы: расширение круга внутренних задач, предоставление ресурсов для задач сторонних организаций, подбор адекватных аппаратных конфигураций, объединение сервисов с целью снижения числа серверных ОС и применение технологии виртуальных машин. Результат критического анализа подходов показал, что наиболее приемлемым является применение технологии виртуальных машин.

Однако, для применения технологии виртуальных машин на сегодняшний день отсутствует какая-либо методика реорганизации серверного парка с целью повышения эффективности использования вычислительных ресурсов. Также отсутствует какой-либо подход к постановке и решению задачи поиска распределения логических серверов по физическим компьютерам при применении технологии виртуальных машин. Соответственно, требуется разработка методики реорганизации серверного парка.

Во второй части главы рассмотрены несколько существующих моделей распределения ресурсов, как традиционных, так и современных. Результат анализа этих моделей показал, что традиционные модели невозможно использовать из-за гораздо более высокой сложности поставленной задачи поиска распределения, а современные модели не учитывают ряд специфичных моментов, присущих технологии виртуальных машин. Соответственно, требуется разработка новой модели для задачи поиска распределения логических серверов по физическим компьютерам.

## 2. Разработка методики по реорганизации серверного парка

### 2.1. Общий подход

Реорганизация серверного парка с целью повышения эффективности использования ресурсов физических компьютеров – это сложная работа, требующая предварительного проектирования с использованием определенных подходов и методик. Подходы мы рассмотрели в предыдущей главе и выбрали технологию виртуальных машин как наиболее приемлемый подход, в данной же главе будет создаваться методика реорганизации серверного парка при применении технологии виртуальных машин.

В современных условиях жесткой конкуренции реорганизация серверного парка – это далеко не только сбор предварительной информации, постановка задачи реорганизации и ее решение одним из известных методов, но и максимальный учет требований и пожеланий заказчика IT-услуг. Под заказчиком мы будем понимать не только главу какой-либо коммерческой организации, но и, в общем случае, некоторых его IT-специалистов: системных администраторов, аналитиков, экспертов – на которых глава опирается в технических вопросах и которые, в конечном счете, будут оценивать качество выполненных работ и в дальнейшем иметь дело с реорганизованным серверным парком. Как упоминалось ранее, в любой коммерческой организации присутствует своя специфика, которую лучше всего может знать только заказчик, и только лишь тесно взаимодействуя с ним можно получить не только хорошее решение, но и решение, которое будет максимально отвечать интересам заказчика и оправдывать его ожидания.

Из вышесказанного следует, что необходимо предварительно оценивать целесообразность проведения реорганизации серверного парка, а заказчик, взвесив все “за” и “против” принимает окончательное решение о проведении работ по реорганизации. Тогда, очевидно, что задача реорганизации серверного парка должна разбиваться на два этапа:

- I) Анализ задачи, сбор первичной информации, разработка первичного проектного решения, оценка качества решения с учетом возможных непредвидимых ситуаций, анализ выгоды, которое дает это решение, и оценка целесообразности проведения реорганизации. Цель первого этапа – максимально обезопасить проектное решение от провала на втором этапе – этапе реализации проектного решения. На первом этапе не допускается внесение каких-либо изменений в серверный парк.
- II) Непосредственная реализация первичного решения, выявление негативных последствий реорганизации. В случае неудовлетворительного качества работы серверного парка – корректировка решения и реализация скорректированного решения, далее повторяется анализ качества. Если на каком-либо шаге корректировка невозможна в силу неприемлемого снижения коммерческой выгоды – то поиск компромиссных решений либо отказ от проекта с возвратом серверного парка в исходное состояние – возможность такого исхода зависит от качества анализа на первом этапе.

Рассмотрим подробнее оба этапа:

- I) На первом этапе выполняются следующие шаги:
- 1) Сбор первичной информации по серверному парку: получение данных по имеющимся ресурсам физических компьютеров и требованиям программного обеспечения, а также анализ загрузки ресурсов. Учет вопросов по надежности функционирования: выделение логических серверов, дублирующих функции друг друга. Выделение логических серверов, для которых нет дублирующих серверов и в силу их важности желательно создание таких дублей. Составление дополнительных ограничений для задачи распределения, исключающих в дальнейшем размещение виртуальных машин, содержащих логические серверы дублирующих функции друг друга, на один и тот же физический компьютер. Выбор базовой ОС и определение ее требований к ресурсам. Выбор тех типов ресурсов, по которым предпочтительно решение проблемы их неэффективного использования.

- 2) Получение с помощью программно-математического инструмента первичного распределения логических серверов по физическим компьютерам (поиск распределения по первичным исходным данным).
- 3) Первичный анализ полученного распределения: удаление из распределения “бесперспективных” физических компьютеров и логических серверов. “Бесперспективным” считаются пара “компьютер – логический сервер” в случае если:
  - Логический сервер в соответствии с полученным распределением размещен на том же компьютере, на котором он работал изначально, при этом на этот компьютер другие логические серверы согласно полученному распределению не размещаются.
  - Логический сервер в соответствии с полученным распределением никуда не размещен, а компьютер, на котором изначально работал этот логический сервер, оказывается незадействованным.
- 4) Вторичный анализ полученного распределения: прогноз возможных корректировок полученного распределения на случай если реорганизованный серверный парк не будет отвечать техническим требованиям и пожеланиям заказчика, прогноз количества освобождаемого оборудования с учетом возможных корректировок.
- 5) Предоставление результатов анализа и прогнозов заказчику для оценки выгоды и получение окончательного решения о целесообразности проведения работ по реорганизации.
- 6) Завершение первого этапа.

В случае если на первом этапе было принято положительное решение, то вступает в действие второй этап.

II) На втором этапе выполняются следующие шаги:

- 1) Реализация первичного распределения – первичная реорганизация серверного парка в соответствии с полученным на первом этапе распределением: проведение работ по резервному копированию данных для возможности “отката” к исходному состоянию в наихудшем случае



(когда, несмотря на самые “радужные” прогнозы на первом этапе, внедрение решения приводит к неудовлетворительной работе серверного парка и внесение всех возможных корректировок не исправляет ситуации и приводит к тому, что последняя корректировка требует столько же оборудования, сколько было до реорганизации или даже больше или количество освободившегося оборудования снижается настолько, что заказчику уже такой результат просто невыгоден). Подготовка логических серверов к переносу на виртуальную аппаратную платформу. Подготовка задействованных в распределении компьютеров: развертывание базовой ОС, подготовка виртуальных машин. Развертывание логических серверов на виртуальных машинах физических компьютеров в соответствии с первичным распределением.

- 2) Оценка качества работы реорганизованного серверного парка: оценка времени отклика серверов, скорости обработки запросов по ключевым сервисам (непосредственно используемым пользователями) и т.п. Если все сервисы функционируют и заказчика удовлетворяет время отклика серверных систем, время обработки запросов, то реорганизация считается успешно завершенной и переход к шагу 9. В противном случае переход к шагу 3.
- 3) Выполняется поиск логических серверов, функционирование которых не отвечают тем или иным требованиям. Анализ причин их неудовлетворительной работы и выделение тех типов ресурсов, имеющиеся уровни по которым недостаточны. Выделение логических серверов, для которых размещение на одном компьютере создает проблемы. Повышение требований “проблемных” логических серверов к ресурсам, внесение дополнительных ограничений на одновременное размещение логических серверов на физических компьютерах.
- 4) Получение с помощью программно-математического инструмента оптимального распределения логических серверов по физическим компьютерам по скорректированным исходным данным.

- 5) Первичный анализ скорректированного распределения: удаление из распределения “бесперспективных” физических компьютеров и логических серверов. Данная операция проводится совершенно аналогично соответствующей операции первого этапа реорганизации.
- 6) Предоставление результатов анализа заказчику для оценки выгоды и получение окончательного вывода о целесообразности реализации скорректированного варианта распределения.
- 7) Если заказчик принимает положительное решение, то проведение коррекционной реорганизации: перераспределение логических серверов с привлечением дополнительных (из числа освободившихся ранее) физических компьютеров в соответствии со скорректированным распределением и переход к шагу 2. Иначе – переход к шагу 8.
- 8) Поиск компромиссных вариантов с заказчиком, в худшем случае “откат” к исходному состоянию серверного парка и прекращение работ.
- 9) Завершение второго этапа.

Следует отметить, что, несмотря на такой худший случай, как возможный на втором этапе “откат” к исходному состоянию, то есть фактически признание проектного решения по реорганизации “провальным” после его реализации, реально на практике, при максимально ответственном подходе со стороны исполнителя и заказчика на первом этапе, такого не происходит, а, в крайнем случае, они останавливаются на каком-либо компромиссном решении. Заметим также, что окончательное решение о целесообразности проведения реорганизации на первом этапе принимает именно заказчик.

На рисунках 2.1 и 2.2 приведены схемы алгоритмов для первого и второго этапа реорганизации соответственно. Алгоритмы для первого и второго этапов реорганизации состоят из достаточно укрупненных блоков, которые требуют дальнейшей детализации и разработки конкретных обрабатывающих процедур для них. Мы выделим в алгоритмах по отдельности блоки, для которых требуется детализация, и рассмотрим их в последующих разделах главы.



Рис 2.1. Схема алгоритма для первого этапа реорганизации

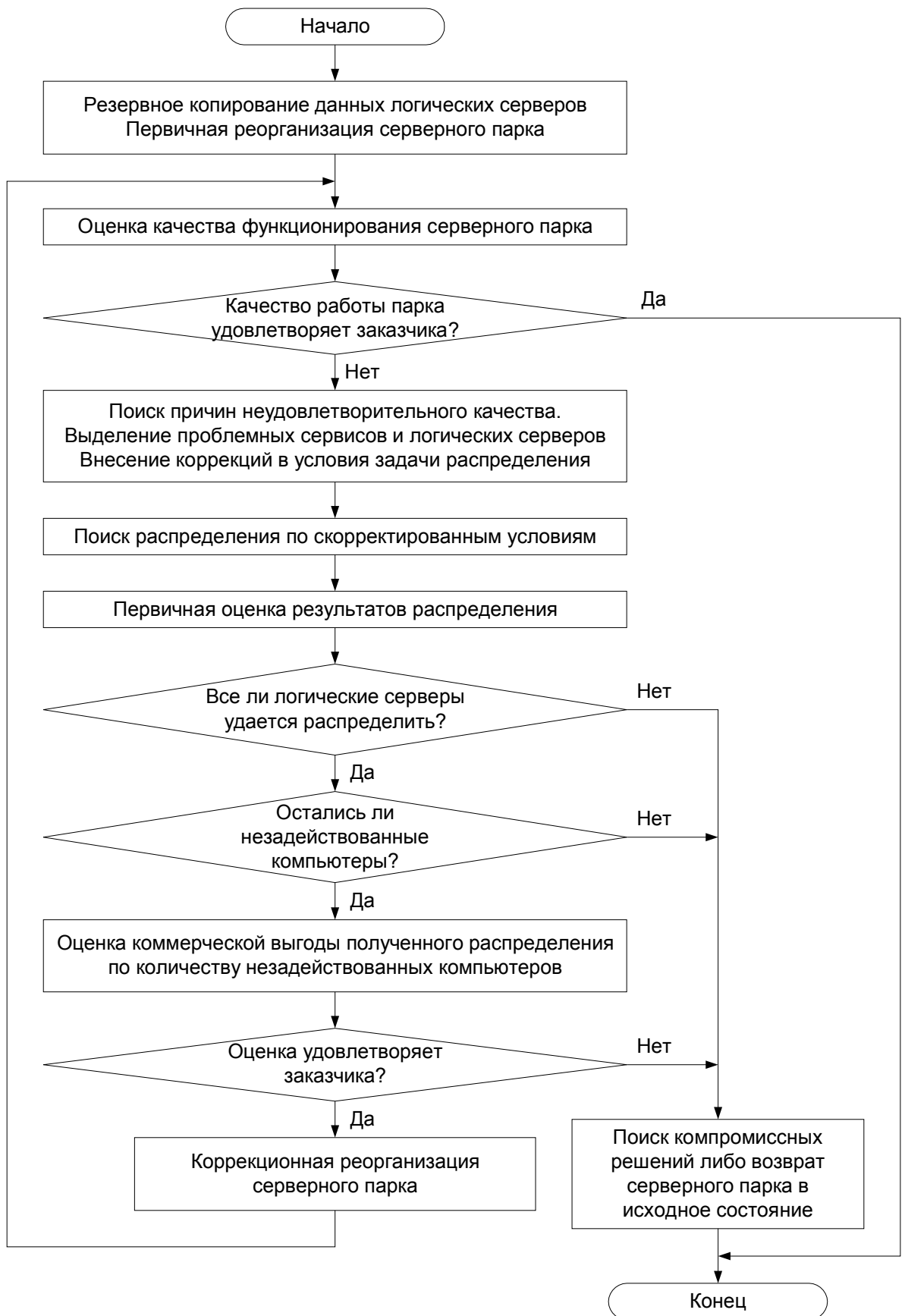


Рис 2.2. Схема алгоритма для второго этапа реорганизации

## 2.2. Подходы к сбору первичной информации

Для решения какой-либо задачи необходимы некоторые исходные данные. Алгоритм для первого этапа реорганизации (рис 2.1) начинается со сбора исходной информации по серверному парку. Чем более точные и объективные исходные данные будут получены, тем более качественным будет конечный результат. Серверный парк представляет собою:

- Некоторое множество физических компьютеров, объединенных некоторой высокоскоростной и надежной сетью передачи данных.
- Некоторое множество логических серверов, функционирующих на компьютерах и предоставляющих те или иные сервисы.
- Некоторую логическую структуру, которая помимо физической и логической топологии сети включает в себя: схему разбиения на маршрутизируемые подсети и распределение пространства сетевых адресов, систему именования узлов сети и распределение пространства имен, схему разбиения на домены и рабочие группы, схему разбиения на условные зоны с различными требованиями к уровню безопасности, схему привязок тех или иных сервисов к логическим серверам.

В рамках диссертационной работы не рассматривается логическая структура серверного парка, поскольку при применении технологии виртуальных машин реорганизация в большинстве случаев не затрагивает логическую структуру. В случае же реорганизации с введением дополнительных логических серверов с целью повышения надежности функционирования серверного парка, в логической структуре, безусловно, будут некоторые изменения, но все же это не какие-либо коренные изменения и они достаточно “прозрачны” для конечных пользователей.

Основная информация, которую требуется собирать – это информация по компьютерам и программному обеспечению, работающему на них. При сборе первичной информации необходимо выполнить следующее: выбрать типы ресурсов и меры для их измерения, оценить базовые уровни ресурсов компьютеров, оценить требования логических серверов к ресурсам, оценить

требования базовой ОС, учесть вопросы надежности и выбрать те типы ресурсов, по которым необходимо добиться более эффективного использования. Следует особо отметить, что большинство программных средств (за исключением задач с распределенным вычислением и распределенного хранения данных) предъявляют требования только к ресурсам того компьютера, на котором они работают. В то же время при рассмотрении такого устройства как сетевой адаптер, невозможно ограничиться только его характеристиками (в простейшем случае базовый уровень ресурса для сетевого адаптера – это его пропускная способность), поскольку на скорость передачи и время отклика логических серверов, безусловно, влияет также коммуникационное оборудование и физические линии связи. Поэтому, несмотря на то, что они явным образом не рассматриваются, косвенно их характеристики должны учитываться при определении базового уровня ресурса сетевой подсистемы компьютера.

Первичная информация необходима для возможности постановки и решения задачи поиска оптимального распределения логических серверов по физическим компьютерам. В следующих подразделах главы будет более подробно рассмотрено то, какая именно информация, в каком виде и каким образом собирается.

**Примечание.** В качестве дополнительной информации, но необязательной при первичном сборе информации, которая может косвенно использоваться на втором этапе реорганизации, может быть множество ключевых сервисов, необходимых пользователям для полноценной работы и заказчику для обеспечения функционирования всех бизнес-процессов организации. Множество ключевых сервисов – это далеко не простое объединение всех сервисов, предоставляемых всеми логическими серверами. Как известно, сервисы логических серверов могут образовывать сложные схемы зависимостей друг друга, как внутри логического сервера, так и среди множества логических серверов. Кроме того, пользователям, как правило, важен лишь некоторый небольшой набор ключевых сервисов, которыми они

пользуются повседневно: корпоративная почта, Интернет, файловый обмен, возможности печати, возможности сетевой и антивирусной защиты, возможности защиты данных от потерь и несанкционированного доступа и т.д. В такой ситуации полезно иметь информацию о множестве ключевых сервисов и допустимых рамок для времени обслуживания по каждому из них.

### 2.2.1. Выбор типов ресурсов и размерностей для оценки их уровней

Для того, чтобы начать сбор информации по базовым уровням ресурсов физических компьютеров и требованиям логических серверов, необходимо сначала выбрать те типы ресурсов, которые являются наиболее критичными и представляют интерес для оптимизации. В общем случае типов ресурсов может быть сколь угодно много, однако, на сегодняшний день в серверных системах используются в основном 4 ключевых типа: оперативная память, дисковая подсистема, сетевая подсистема и процессор. Причем как ограничивающие факторы оперативная память и дисковая подсистема являются более строгими, нежели чем процессор или сетевая подсистема. Действительно, в силу жестких ограничений мы физически не можем поместить количество виртуальных машин больше, чем позволяет емкость оперативной памяти компьютера и емкость дисковой подсистемы. Что же касается процессора и сетевого подсистемы, физически при достаточных ресурсах по памяти и дисковой подсистеме поместить можно сколь угодно много виртуальных машин, но при нехватке ресурсов по процессору и сетевой подсистеме будут наблюдаться сильные задержки по обработке запросов и передаче данных. Поэтому оперативная память и дисковая подсистема обязательно и в любом случае входит в список критичных ресурсов. Вычислительная мощность процессора и пропускная способность сетевой подсистемы также критичны, но все же в определенных условиях могут не учитываться — например, множество контроллеров доменов создают очень незначительную и эпизодическую нагрузку на процессор и сетевую подсистему.

Следующий важный момент – для возможности корректной постановки задачи оптимизации для уровней ресурсов компьютеров и требований логических серверов должны использоваться одни и те же размерности по каждому из типов ресурсов: для емкости оперативной памяти и дисковой подсистемы – мегабайты, для сетевой подсистемы – Мбайт/с (или Кбайт/с), для процессора – какие-либо известные размерности: MIPS, MFLOPS, MTOPS, CPU Mark Score и т.д. Для процессоров мы будем ориентироваться на MTOPS (Millions of Theoretical Operations Per Second), поскольку на сегодняшний день в корпоративных сетях наиболее часто встречаются процессоры фирмы Intel [6, 7, 8], и их официальным показателем быстродействия является Intel CTP (Composite Theoretical Performance), измеряемый в MTOPS.

### 2.2.2. Сбор информации по физическим компьютерам

Базовые уровни ресурсов  $\{R_{i,k}\}$  (технические характеристики аппаратных компонент интересующих специалиста) физических компьютеров – являются первой неотъемлемой частью исходных данных задачи оптимизации.

В любой коммерческой организации, а тем более в той, где имеется серьезный серверный парк и, возможно, не один, в той или иной форме оборудование документируется. Тем не менее, следует предусмотреть то, что документация быстро устаревает и не всегда актуальна целиком и полностью. Соответственно, необходимо также располагать собственными методами идентификации оборудования и оценки их характеристик. На сегодняшний день существует три основных способа идентификации:

- Информация, выводимая на экран при включении и перезагрузке компьютера, и информация, выводимая в настройках BIOS материнской платы [12]. Такой способ мало предпочтителен поскольку, как правило, требует перезагрузки работающего в сети компьютера.
- Специальные программные средства для идентификации аппаратных компонент компьютера. Наиболее популярные среди них – это AIDA32, Check It, SI Soft Sandra, Dr. Hardware, Norton Utilities System Information.



Такой способ более предпочтителен, однако, зачастую утилиты требуют предварительной установки, а также могут быть несовместимыми с определенным оборудованием или приложениями и вызвать сбой операционной системы.

- Средства операционной системы. Практически все современные ОС имеют встроенные утилиты для сбора информации по конфигурации компьютера. Так, например, при помощи встроенной утилиты System Information ОС MS Windows 2000 Advanced Server [1, 18] можно собрать исчерпывающую информацию по типам и базовым уровням ресурсов компьютера, драйверам устройств, по системным компонентам, основным настройкам и т.д. Такой способ является наиболее удобным и предпочтительным, поскольку не требует установки дополнительного программного обеспечения и прерывания работы логического сервера.

Следует особо обратить внимание на то, что сбор информации по таким типам ресурсов как оперативная память и дисковая подсистема не представляет особой сложности – это простое выяснение их емкостей. Что же касается процессора и сетевой подсистемы, то здесь ситуация немного сложнее:

- Очевидно, что частота процессора не является объективным показателем. На сегодняшний день существует большое множество различных интегральных оценок фирм-производителей: iCOMP 3.0, CPU Mark, Intel CTP и т.д., причем некоторые из них являются сильно усредненными, а некоторые ориентированы на определенный класс задач, но, тем не менее, они вполне могут служить в качестве исходных данных. Мы будем ориентироваться на Intel CTP (Composite Theoretical Performance).
- Для сетевой подсистемы важно указывать не просто ее пропускную способность, а минимум из возможностей сетевого адаптера, промежуточного коммутационного оборудования и физических линий связи – т.е. характеристику самого медленного участка в наихудшем возможном варианте соединения с другими компьютерами серверного парка.

### 2.2.3. Сбор информации по логическим серверам

Следующей частью исходных данных задачи оптимизации – это данные о требованиях  $\{Q_{i,j}\}$  логических серверов. Под требованием понимается некоторая величина, приведенная к тем же единицам измерения, что и базовые уровни ресурсов (для памяти, например, требование программного обеспечения – это требования к емкости памяти, измеряемой в мегабайтах). С одной стороны такие сведения можно собрать из документации программного обеспечения, суммируя требования операционной системы и всех серверных программ по каждому типу ресурсов. Так, например, согласно документации, минимальные требования ОС MS Windows 2000:

- Процессор: 133 МТОРs (соответствует процессору Intel Pentium 133 MHz).
- Оперативная память: 128 МБ (на практике ОС достаточно приемлемо функционирует и при 96 Мб при отсутствии каких-либо установленных серверных служб, требовательных к оперативной памяти)
- 850 Мб дискового пространства

С другой стороны для большей достоверности предпочтительнее было бы анализировать текущую загрузку ресурсов компьютера, на котором функционирует логический сервер – это дало бы более объективную картину. В современные серверные ОС встраиваются достаточно мощные средства по анализу загрузки ресурсов, как в целом, так и по отдельным приложениям. Эти средства также позволяют вести мониторинг в течение произвольного интервала времени, что дает возможность в дальнейшем проводить тонкую настройку для получения максимальной производительности для заданных задач на заданном оборудовании. Однако, при анализе загрузки можно зачастую получить слишком завышенные требования, поскольку специфика современного программного обеспечения такова, что если есть какие-то свободные ресурсы, то приложения и ОС рано или поздно их могут загрузить целиком, особенно это касается такого ресурса как оперативная память, которую без какой-либо экономии расходуют не очень качественные операционные системы. Соответственно, может оказаться так, что реально

логический сервер вполне приемлемо может функционировать при уровнях ресурсов меньших, чем показывают данные анализа загрузки ресурсов. Объективно оценить те уровни ресурсов, которые “действительно” необходимы для нормального функционирования логического сервера, может оценить только лишь специалист-эксперт, хорошо знающий операционные системы и серверное программное обеспечение, и имеющий достаточный опыт работы с ними. Все вышеприведенные рассуждения ведут к тому, что можно создавать сколь угодно сложные методы оценки требований логического сервера, но любой новичок, пользуясь этими методами, как правило, получит менее объективные данные, нежели чем специалист, который опирается лишь на свой богатый практический опыт.

Таким образом, проведя краткий обзор, мы приходим к тому, что есть три основных способа получения информации о требованиях логических серверов:

- Информация в документации к программному обеспечению. Суммирование требований ОС и серверного программного обеспечения. Такой способ практически лишен гибкости и не учитывает специфики задач конкретной коммерческой организации.
- Анализ загрузки ресурсов встроенными в ОС средствами мониторинга (в ОС MS Windows 2000 – это Performance Monitor). Более мощный, гибкий и предпочтительный способ, однако, может давать неверные результаты (особенно по загрузке оперативной памяти) в силу ошибок в ОС и серверном программном обеспечении (проблемы утечки памяти). Также можно ошибиться в данных о требуемом дисковом пространстве, в случае, когда помимо файлов операционной системы и серверного программного обеспечения на дисках хранятся какие-либо временные данные или установлены программы (исключение – антивирусные и прочие программы защиты), не имеющие отношения к выполнению серверных функций.
- Экспертная оценка на основе практического опыта и рекомендаций разработчиков ПО, хорошего знания ОС и программного обеспечения и их “реальных потребностей” с учетом результатов детального изучения задач,

решаемых в организации, оценки объемов обрабатываемых данных и объемов запросов приложений пользователей, а также анализа загрузки ресурсов и минимальных требований в документации. Данный подход объединяет в себе предыдущие два и значительно расширяет его.

До сих пор мы говорили о различных подходах по оценке требований логических серверов без уточнения того, как именно эти подходы реализуются. Остановимся на них подробнее.

### **Документация по программному обеспечению**

Что касается первого подхода, связанного с изучением документации программного обеспечения, то здесь вряд ли требуются какие-либо уточнения. Общие требования по каждому из ресурсов являются суммой требований серверных приложений (СП) и операционной системы (ОС), причем у каждого СП (и ОС) свои собственные независимые требования:

$$Q_{i,j} = q_{i,j,OC} + \sum_{f=1}^{F_j} q_{i,j,f} \quad (2.1)$$

где,

$q_{i,j,OC}$  – требование по  $i$ -му типу ресурса операционной системы, функционирующей на  $j$ -м логическом сервере,  $i = 1..NC$ ,  $j = 1..NS$ .

$q_{i,j,f}$  – требования по  $i$ -му типу ресурса  $f$ -го серверного программного обеспечения, функционирующей на  $j$ -м логическом сервере,  $f = 1..F_j$ .

$F_j$  – число серверных программ на  $j$ -м логическом сервере.

### **Анализ загрузки ресурсов компьютера**

В современных серверных ОС имеются мощные средства для анализа загрузки ресурсов и мониторинга производительности. Причем эти средства позволяют получать не только текущее, минимальное и максимальное значение того или иного показателя загрузки ресурсов, но и собирать статистику в течение сколь угодно долгого интервала времени. В качестве примера утилиты для быстрой оценки загрузки процессора и оперативной памяти можно

привести утилиту Task Manager встроенной в ОС MS Windows 2000. Исчерпывающую информацию по жестким дискам и свободном дисковом пространстве в ОС Windows 2000 можно получить при помощи встроенной утилиты Computer Management. Можно приводить еще множество различных встроенных возможностей ОС по первичному анализу загрузки ресурсов, но все они достаточно хорошо освещаются в документации по операционным системам, соответствующей литературе [1] и на WEB-сайте разработчиков [11]. Однако, нельзя обойти вниманием такое мощное средство ОС MS Windows 2000 как утилита “Performance” – системный монитор (рис. 2.3) и система ведения журналов для регистрации показаний различных счетчиков, отражающих состояние системных ресурсов и их загрузку. С помощью этой утилиты можно отслеживать как мгновенные значения, так и средние значения за некоторый период, причем как общие (суммарные) показания счетчиков, так и показания счетчиков для каждого приложения в отдельности.

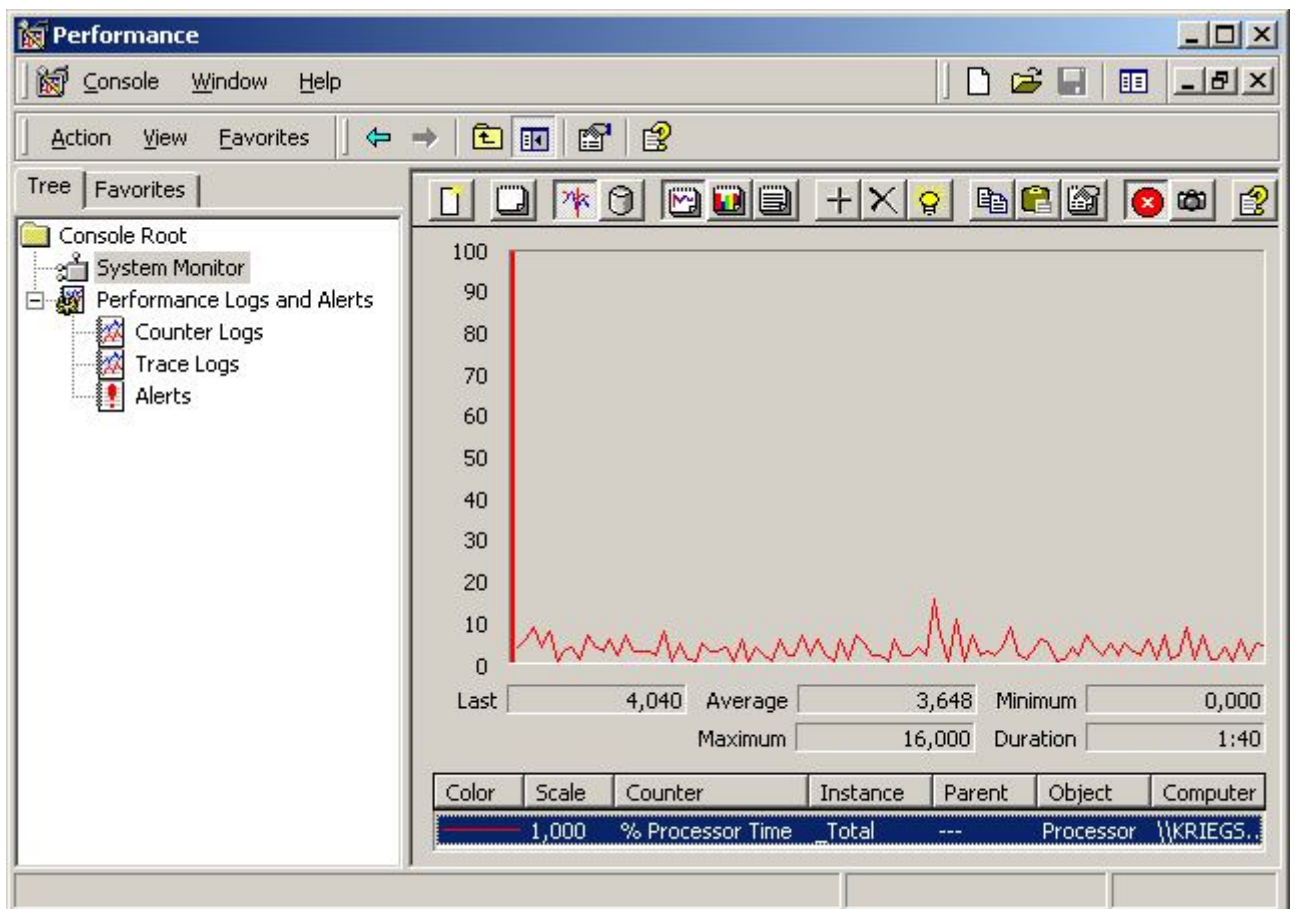


Рис. 2.3. Снимок окна утилиты Performance (ОС MS Windows 2000)

Мы не будем далее углубляться в тему анализа загрузки ресурсов и мониторинга, поскольку это тема для отдельного обсуждения и материал на многие сотни страниц [16]. Отметим лишь, что при сборе информации необходимо добиваться того, чтобы на компьютере функционировали только ОС вместе со службами и располагались только те данные, которые имеют непосредственное отношение к выполнению серверных функций. К сожалению, это не всегда возможно, поэтому специалисту зачастую приходится прибегать к экспертным оценкам. Проведя же анализ, специалист как минимум собирает минимальные, максимальные и средние за некоторый период значения загрузки по тем или иным ресурсам. Что именно из этого выбирать в качестве требований – не всегда можно однозначно определить. Взяв минимальное или даже среднее значение, можно ошибиться и в дальнейшем после реорганизации парка серверов получить неудовлетворительную работу логического сервера. Беря же всегда максимальное значение, можно сильно завysить требования и упустить возможность получения более выгодных решений. Кроме того, собирать статистику по загрузке ресурсов можно при различной нагрузке на логический сервер, что еще более усложняет задачу определения требований. В простейшем случае в качестве требования можно принять среднее за некоторый отрезок времени значение загрузки по конкретному ресурсу, но выраженное не в процентах, а в абсолютном значении, например, для оперативной памяти в мегабайтах используемого пространства:

$$Q_{i,j} = \frac{R_{i,j}}{100Y} \left( \sum_{y=1}^Y W_{i,j}(t_y) \right) \quad (2.2)$$

где,

$R_{i,j}$  – базовый уровень  $i$ -го ресурса физического компьютера, на котором функционирует  $j$ -й логический сервер,  $i = 1..NC$ ,  $j = 1..NS$

$W_{i,j}(t_y)$  – уровень загрузки (в процентах)  $i$ -го ресурса компьютера, на котором функционирует  $j$ -й логический сервер, в момент времени  $t_y$ ,  $y = 1..Y$ .

$Y$  – число выборок значений загрузки, которые делаются на отрезке времени  $[t_1, t_Y]$ . Длина отрезка и число выборок выбирается экспертным путем.

### Экспертная оценка требований

Экспертная оценка предполагает, что помимо двух вышеуказанных способов, будет использоваться также практический опыт специалиста, который, проанализировав всю специфику серверного парка, решаемых задач, объемов данных и запросов приложений пользователей с некоторой достоверностью установит некоторые экспертные значения требований  $\{Q_{i,j}\}$ . В рамках данной диссертации не будем рассматривать различные виды экспертных систем [19], их математические аппараты, а также способы определения степени достоверности экспертных оценок, поскольку это целая отдельная наука. Отметим лишь, что для формирования окончательных требований специалист также должен учесть результаты предыдущих двух способов (изучение документации и анализ загрузки ресурсов компьютеров) и применить некоторую функцию к трем показателям.

$$Q_{i,j} = F(Q_{i,j}^{ДП}, Q_{i,j}^{АЗ}, Q_{i,j}^{ЭО}) \quad (2.3)$$

где,

ДП – документация по программному обеспечению, АЗ – анализ загрузки, ЭО – экспертная оценка требований по каждому из ресурсов.

Функция  $F$  выбирается экспертом, это может быть функция максимума, минимума, среднего значения или что-то иное. С одной стороны специалист не должен занижать требования, так как это в дальнейшем может привести к неудовлетворительной работе логического сервера после реорганизации и потребовать множество корректировок и повторных перераспределений логических серверов, с другой стороны не должен слишком завышать требования, чтобы не упустить выгодных вариантов реорганизации.

Наконец, после выбора типов ресурсов, сбора информации по физическим компьютерам и логическим серверам, мы можем в математической форме записать основные ограничения (2.4) по ресурсам, справедливые для каждого физического компьютера:

Для  $\forall H_k, k = 1 \dots NH$ :

$$\begin{cases} Q_{1,1} X_{k,1} + \dots + Q_{1,NS} X_{k,NS} \leq R_{1,k} \\ \dots \\ Q_{NC,1} X_{k,1} + \dots + Q_{NC,NS} X_{k,NS} \leq R_{NC,k} \end{cases} \quad (2.4)$$

где,

$NC$  – число типов ресурсов.

$NH$  – число физических компьютеров.

$NS$  – число логических серверов (множество логических серверов может быть расширено при учете вопросов надежности (см. далее в этой главе).

$R_{i,k}$  – базовый уровень  $i$ -го ресурса  $k$ -го физического компьютера.

$Q_{i,j}$  – требование  $j$ -го логического сервера к  $i$ -му ресурсу.

$X_{k,j}$  – элемент булевой матрицы, равный “1”, если  $j$ -й логический сервер (виртуальная машина с логическим сервером) размещается на  $k$ -м физическом компьютере, “0” – в противном случае.

#### 2.2.4. Оценка требований базовой операционной системы

Как упоминалось ранее, при размещении на физическом компьютере множества виртуальных машин (на базе которых будут работать логические серверы) возникает необходимость в некотором базовом программном обеспечении, на платформе которого будут функционировать виртуальные машины. В нашем случае базовая ОС – это некоторая серверная операционная система, развернутая в минимальной конфигурации и с установленным специальным программным обеспечением. Базовая ОС также предъявляет некоторые свои требования  $\{V_1, \dots, V_{NC}\}$  к ресурсам. Очевидно то, что требования эти оценить куда проще, чем для логических серверов, хотя способы оценки одинаковые. В качестве примера, приведем требования базовой ОС MS Windows 2000 Advanced Server: оперативная память – 96 Мб, жесткий диск – 2 Гб, к процессору и сетевому адаптеру специальных требований нет: средняя загрузка процессора класса Pentium – 0.5%, собственный сетевой трафик базовой ОС практически отсутствует.



Базовая ОС в любом случае “занимает свое место” на физическом компьютере и поэтому не может участвовать в распределении – мы просто должны мириться с накладной потерей определенной части ресурсов каждого физического компьютера в размерах  $\{V_1, \dots, V_{NC}\}$ .

Тогда ограничения по ресурсам (2.4) принимают следующий вид.

Для  $\forall H_k, k = 1 \dots NH$ :

$$\begin{cases} Q_{1,1}X_{k,1} + \dots + Q_{1,NS}X_{k,NS} \leq R_{1,k} - V_1 \\ \dots \\ Q_{NC,1}X_{k,1} + \dots + Q_{NC,NS}X_{k,NS} \leq R_{NC,k} - V_{NC} \end{cases} \quad (2.5)$$

где,

$V_i$  – требование базовой ОС к  $i$ -му ресурсу,  $i = 1 \dots NC$

### 2.2.5. Требования надежности функционирования серверного парка

При использовании технологии виртуальных машин очевидным становится то, что при размещении нескольких логических серверов на одном физическом компьютере страдает надежность функционирования серверного парка – об этой проблеме было также упомянуто при рассмотрении недостатков технологии виртуальных машин. Здесь возможен выбор:

- Игнорирование вопросов надежности с целью получения максимального выигрыша при оптимизации. Крайне неразумный подход и серьезный дальновидный заказчик никогда на это не пойдет.
- Выделение в серверном парке логических серверов, дублирующих функции друг друга. Подготовка дополнительных логических серверов с целью дублирования функций тех критичных логических серверов, для которых отсутствуют дублирующие серверы – это также ведет к расширению множества логических серверов  $\{S_j\}$ . Составление дополнительных ограничений для задачи оптимизации для исключения возможности размещения логических серверов, дублирующих функции друг друга на один и тот же физический компьютер.

По элементарным соображениям надежности, будем опираться на второй подход, остановимся на нем подробнее. Для простоты рассмотрим простейший пример: задано 5 контроллеров доменов для 5 различных доменов. Нет смысла лишний раз подчеркивать важность информации, которая хранятся на них. Будем считать, что малоопытный заказчик с целью экономии при проектировании серверного парка не стал развертывать дублирующие контроллеры домена ни для одного из доменов. Однако, после постановки вопроса о целесообразности проведения реорганизации, исполнитель предупредил о потенциальной опасности, и заказчик поставил вопрос о повышении надежности функционирования серверного парка при его реорганизации. В данном случае, для исполнителя нет ничего сложного: он подготавливает 5 дополнительных логических серверов (на виртуальной платформе) в качестве дублирующих контроллеров доменов, затем формирует дополнительные ограничения на их размещение.

Рассмотрим теперь общий случай: задано множество  $\{S_j^0\}$ , где  $j = 1..NS_0$ , логических серверов, среди которых есть серверы, дублирующие друг друга по функциям. В общем случае среди логических серверов также могут присутствовать те серверы, которые нуждаются в дублирующих серверах, но не имеют их. Соответственно, как исходя из экспертных соображений специалиста, так и по требованиям заказчика, выделяется подмножество критичных серверов и для каждого из них обеспечивается как минимум одна дополнительная виртуальная машина с логическим сервером, дублирующим функции. В результате множество логических серверов расширяется до  $\{S_j\}$ , где  $j = 1..NS$ , причем  $NS > NS_0$ ,  $NS = NS_0 \cdot (1 - \Psi) + 2 \cdot NS_0 \cdot \Psi$ , где  $\Psi$  – доля критичных логических серверов исходного серверного парка, не имеющих дублирующих серверов. Далее, среди всего множества логических серверов выделяются все подмножества логических серверов, дублирующих функции друг друга, например,  $\{S_1, S_2\}$ ,  $\{S_7, S_{10}\}$  и так далее, и для них мы формулируем дополнительные ограничения (2.6), которые всегда должны выполняться при размещении логических серверов на любой из компьютеров.

Для  $\forall H_k, k = 1..NH$ :

$$\begin{cases} E_{1,1}X_{k,1} + \dots + E_{1,NS}X_{k,NS} \leq 1 \\ \dots \\ E_{NX,1}X_{k,1} + \dots + E_{NX,NS}X_{k,NS} \leq 1 \end{cases} \quad (2.6)$$

где,

$NX$  – число дополнительных ограничений (число подмножеств, в каждое из которых включаются все дублирующие друг друга логические серверы).

$E_{d,j}$  – элемент булевой матрицы, равный “1”, если  $j$ -й логический сервер присутствует в  $d$ -м ограничении, “0” – в противном случае.

$X_{k,j}$  – элемент булевой матрицы, равный “1”, если  $j$ -й логический сервер (виртуальная машина с логическим сервером) размещается на  $k$ -м физическом компьютере, “0” – в противном случае.

Таким образом, мы исключаем возможность размещения логических серверов, с дублирующими друг друга функциями, на один и тот же компьютер и учитываем вопросы надежности функционирования серверного парка. Для ясности дополнительные ограничения для различения их с основными ограничениями по ресурсам будем также называть ограничениями-исключениями или просто исключениями.

Следует особо отметить, что если множество логических серверов расширяется, то, безусловно, это отразится на системе ограничений по ресурсам (2.5). Однако, здесь нет особо сложности – увеличивается число логических серверов ( $NS$ ), а требования к ресурсам  $\{Q_{i,j}\}$  берутся такими же, как у тех логических серверов, чьи функции они будут дублировать. Соответственно, для дополнительных логических серверов не требуется рутинной работы по определению требований к ресурсам. Дополнительные логические серверы совершенно на равных условиях с другими логическими серверами участвуют в распределении. В целом, задача поиска оптимального распределения, безусловно, усложняется, как за счет дополнительных логических серверов, так и дополнительных ограничений.

### 2.2.6. Выбор цели оптимизации

После сбора информации по ресурсам компьютеров, требованиям логических серверов и базовой ОС, введения дополнительных ограничений для обеспечения надежности, для окончательной постановки математической задачи оптимизации, безусловно, необходима еще и некоторая цель. Основная цель – это снижение объема задействованного оборудования, однако в такой постановке разработать в дальнейшем адекватную математическую модель и тем более найти способ решения будет крайне затруднительно. С другой стороны, неоправданно большой объем задействованного оборудования – это всегда следствие неэффективного использования имеющихся ресурсов. На практике это выражается в виде низкого уровня загрузки вычислительных ресурсов компьютеров. Поэтому можно стремиться к повышению уровня загрузки по каждому типу ресурсов задействованных компьютеров путем перераспределения логических серверов по физическим компьютерам. При этом какие-то компьютеры будут освобождаться, а другие, наоборот, более сильно нагружаться и тем самым будет достигаться основная цель – множество логических серверов “разбросанных” по одному на физические компьютеры, будут сосредотачиваться на некотором подмножестве компьютеров, уровень загрузки которых, очевидно, окажется выше, чем до перераспределения логических серверов. Очевидно, можно добиваться повышения загрузки сразу по всем типам ресурсов, но для большей гибкости следует заложить возможность выбирать те типы ресурсов, по которым предпочтительно решить проблему их неэффективного использования. В общем случае это может быть любой набор типов ресурсов. Нетрудно заметить, что этот набор можно представить в следующем виде – булевым вектором  $\{O_1, \dots, O_{NC}\}$ , где  $O_i$  принимает значение “1”, если по соответствующему  $i$ -му типу ресурса предпочтительно повышение эффективности его использования, “0” – в противном случае. Вектор  $\{O_i\}$  будем называть *вектором маски*.

### 2.3. Поиск распределения логических серверов на компьютеры

Ключевой частью разрабатываемой методики является постановка и решение задачи поиска оптимального распределения логических серверов на компьютеры. Задача поиска оптимального распределения решается один раз на первом этапе и несколько раз в зависимости от результатов реорганизации серверного парка на втором этапе реорганизации.

Задача поиска оптимального распределения является задачей дискретной оптимизации. Входными данными задачи являются: множество типов ресурсов  $\{1, \dots, NC\}$ , множество компьютеров  $\{N_k\}$  и матрица базовых уровней их ресурсов  $\{R_{i,k}\}$ ,  $k = 1..NH$ ,  $i = 1..NC$ , множество логических серверов  $\{S_j\}$  и матрица их требований  $\{Q_{i,j}\}$ ,  $j = 1..NS$ ,  $i = 1..NC$ , базовая ОС и ее требования  $\{V_i\}$ ,  $i = 1..NC$ , дополнительные ограничения  $\{E_{d,j}\}$ ,  $d = 1..NX$ ,  $i = 1..NC$ , на одновременное размещение логических серверов на один и тот же компьютер, и, наконец, двоичный вектор  $\{O_i\}$ ,  $i = 1..NC$ , определяющий то, по каким именно типам ресурсов предпочтительно решение проблемы их неэффективного использования. Целью является повышение эффективности использования ресурсов. Выходными данными задачи (решением задачи) является матрица распределения  $\{X_{k,j}\}$ ,  $k = 1..NH$ ,  $j = 1..NS$ , логических серверов по физическим компьютерам.

Соответственно, для решения задачи оптимизации требуется разработка математической модели и метода решения задачи. Однако, ввиду исключительной сложности задачи оптимизации для избежания чрезмерного усложнения описания разрабатываемой методики в данной главе будем считать блок поиска распределения логических серверов на компьютеры “черным ящиком”, на вход которого подаются исходные данные, на выходе имеем результаты решения задачи оптимизации. Блок поиска распределения будет раскрыт в главе 3 с исчерпывающей детализацией: будет разработана математическая модель и метод решения задачи поиска оптимального распределения логических серверов по компьютерам.

## 2.4. Первичная и вторичная оценка распределения

После получения некоторого распределения, как на первом этапе (рис. 2.1), так и на втором этапе (рис. 2.2) реорганизации возникает необходимость в экспертной оценке полученного распределения. На первом этапе это необходимо для оценки целесообразности проведения работ по реорганизации серверного парка. На втором же этапе повторной поиск распределения производится только в том случае, если серверный парк после первичной реорганизации или очередных корректировок по каким-либо показателям не удовлетворяет требованиям заказчика. В таком случае ищутся причины проблем, выделяются узкие места, модифицируются требования логических серверов и исключения, ставится новая задача поиска распределения и полученное распределение используется для оценки целесообразности внесения очередных корректировок в реорганизованный парк.

После решения задачи поиска распределения логических серверов по компьютерам возможны следующие варианты:

- Множество индексов оставшихся (незадействованных) физических компьютеров  $\{K(T_{FIN})\}$  пусто, где  $T_{FIN}$  – последний шаг алгоритма решения задачи поиска распределения, а множество индексов логических серверов, оставшихся нераспределенными,  $\{J(T_{FIN})\}$  не пусто. В таком случае, имеет место быть нехватка ресурсов либо при заданных условиях, несмотря даже на достаточные суммарные ресурсы, эти ресурсы распределены среди физических компьютеров таким образом, что невозможно распределить все без исключения логические серверы. Если такой результат получен на первом этапе реорганизации, то принимается решение о нецелесообразности проведения реорганизации, либо заказчик может принять решение о привлечении дополнительного оборудования, после чего может быть поставлена новая задача поиска распределения. Если же такой результат был получен на втором этапе, то необходимо искать какие-либо компромиссы либо вернуть серверный парк в исходное состояние (при ответственном подходе к первому этапу реорганизации, вероятность такого исхода мала).

- Множество индексов оставшихся (незадействованных) физических компьютеров  $\{K(T_{FIN})\}$  пусто и множество индексов логических серверов, оставшихся нераспределенными,  $\{J(T_{FIN})\}$  пусто. В таком случае логические серверы все распределены, однако, освободившихся физических компьютеров нет, соответственно, отсутствует какая-либо выгода от такого результата. Этот случай с точки зрения дальнейших действий на первом или втором этапах реорганизации не отличается от первого варианта распределения логических серверов на физические компьютеры.
- Множество индексов оставшихся (незадействованных) физических компьютеров  $\{K(T_{FIN})\}$  не пусто и множество индексов логических серверов, оставшихся нераспределенными,  $\{J(T_{FIN})\}$  не пусто. В таком случае, несмотря на имеющиеся свободные физические компьютеры, оставшиеся логические серверы не могут быть распределены на них. Этот случай с точки зрения дальнейших действий на первом или втором этапах реорганизации не отличается от первого варианта распределения логических серверов на физические компьютеры.
- Множество индексов оставшихся (незадействованных) физических компьютеров  $\{K(T_{FIN})\}$  не пусто, а множество индексов логических серверов, оставшихся нераспределенными,  $\{J(T_{FIN})\}$  пусто. Это самый удачный вариант – вариант, к которому мы и стремимся. В таком случае все логические серверы распределены и осталось некоторое множество освободившихся физических компьютеров.

Рассмотрим подробнее дальнейшие действия на первом и втором этапах реорганизации для последнего варианта распределения:

- Несмотря на очевидную выгоду в последнем варианте распределения, на первом этапе реорганизации при оценке целесообразности проведения реорганизации серверного парка следует застраховаться и сделать некоторые прогнозы относительно возможных проблем (неудовлетворительная работа каких-либо логических серверов после реорганизации), которые могут вызвать необходимость внесения

корректировок в распределение, и среди освободившихся физических компьютеров оставить некоторый “запас”. Вопрос о том, сколько и какие именно физические компьютеры остаются про запас – решается достаточно несложно: для самых критичных по важности логических серверов завышаются требования по ресурсам (насколько именно – выбирает технический специалист, полагаясь на свой опыт – завышение, как правило, не более чем на 5-15%) и задача поиска оптимального распределения решается вторично. В результате поиска оптимального распределения по завышенным требованиям логических серверов, получаем некоторые модифицированные множества индексов  $\{K^*(T_{FIN})\}$  и  $\{J^*(T_{FIN})\}$  оставшихся физических компьютеров и логических серверов. Эти множества, как и в случае поиска первичного распределения, могут иметь 4 ключевых вариантов и только в случае, если  $\{J^*(T_{FIN})\}$  пусто, а  $\{K^*(T_{FIN})\}$  – нет, мы можем говорить о каком-либо запасе  $\{K(T_{FIN})\} \setminus \{K^*(T_{FIN})\}$  по физическим компьютерам. Для максимально осторожного и грамотного подхода к оценке целесообразности проведения реорганизации серверного парка, при оценке коммерческой выгоды заказчику необходимо предоставлять именно множество  $\{K^*(T_{FIN})\}$ . В случае если после реорганизации запас окажется невостребованным или не полностью израсходованным, для заказчика всегда будет возможность получения дополнительной прибыли, что, несомненно, повысит степень его удовлетворенности выполненными работами. На первом этапе реорганизации именно на основе оценки выгоды по  $\{K^*(T_{FIN})\}$  (при условии  $\{J^*(T_{FIN})\} = \emptyset$ ), принимается решение о целесообразности проведения реорганизации серверного парка. В случае положительного решения – выполняется переход ко второму этапу реорганизации.

- На втором этапе в случае, если на очередной  $\lambda$ -й итерации алгоритма второго этапа  $\{J_\lambda(T_{FIN})\}$  пусто, а  $\{K_\lambda(T_{FIN})\}$  – нет, заказчик оценивает выгоду именно по множеству  $\{K_\lambda(T_{FIN})\}$  и принимает решение о целесообразности проведения дальнейших корректировок реорганизованного парка серверов.



**Особые случаи.** Как в первичном распределении, так и в последующих модифицированных и скорректированных вариантах распределения могут встретиться следующие ситуации:

- 1) Согласно полученному распределению на каком-либо из физических компьютеров размещается один единственный логический сервер, причем тот же самый, который и функционировал на данном компьютере. В таком случае, очевидно, что нет смысла рассматривать этот компьютер и логический сервер – их необходимо исключить из распределения, чтобы в дальнейшем не выполнять лишнюю работу по переносу логического сервера на виртуальную платформу и размещению его на тот же самый физический компьютер без возможности размещения других виртуальных машин.
- 2) Нередко встречаются случаи, когда какой-либо из физических компьютеров остается незадействованным, а логический сервер, работавший на нем, никуда не размещается. Возникает особая ситуация: изначально логический сервер работал на физическом компьютере, но в силу дополнительных требований базовой ОС, он не может быть размещен на виртуальной платформе на этом же компьютере, поскольку на нем недостаточно ресурсов для размещения базовой ОС вместе с соответствующей виртуальной машиной. Очевидно, что в такой ситуации подобный физический компьютер вместе с его логическим сервером необходимо исключить из распределения, чтобы попусту не увеличивать число нераспределенных логических серверов и тем самым улучшить окончательный результат распределения.

Соответственно, на обоих этапах реорганизации при получении оптимального распределения результат должен быть проверен на вышеуказанные две особые ситуации, и при необходимости из него должны быть исключены “бесперспективные” с точки зрения реорганизации физические компьютеры вместе с работающими на них логическими серверами. В конечном счете, реорганизация должна проводиться для тех физических компьютеров, на которые размещаются как минимум два логических сервера.

С учетом всего вышесказанного в данном подразделе, можно детализировать блоки, связанные с оценкой результатов распределения в алгоритмах первого и второго этапов реорганизации (рис. 2.1 и рис. 2.2),

- Блок первичной оценки оптимального распределения алгоритмов для обоих этапов реорганизации детализируется следующим образом (рис. 2.4):

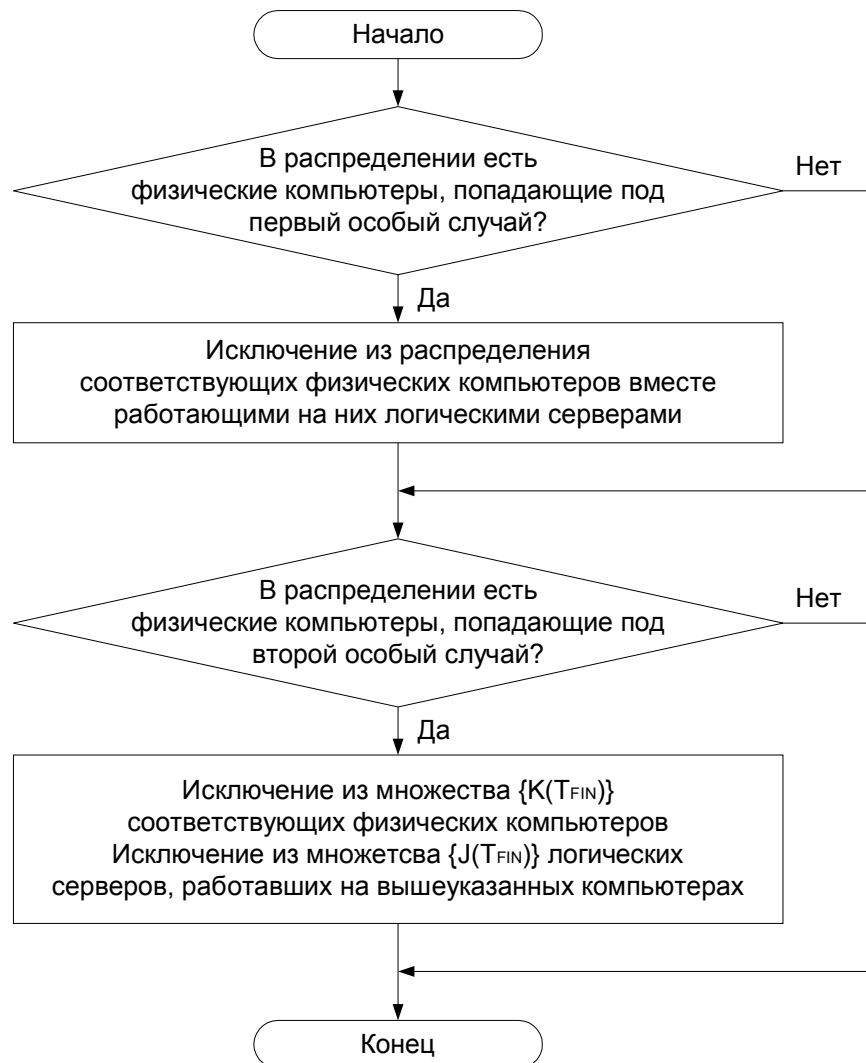


Рис 2.4. Схема алгоритма первичной оценки распределения

- Блок вторичной оценки: прогноз возможных корректировок, которые могут потребоваться после реорганизации серверного парка, и оценки остатка – количества незадействованных физических компьютеров и суммарной стоимости их ресурсов. Блок присутствует только в алгоритме для первого этапа реорганизации и детализируется следующим образом (рис. 2.5):

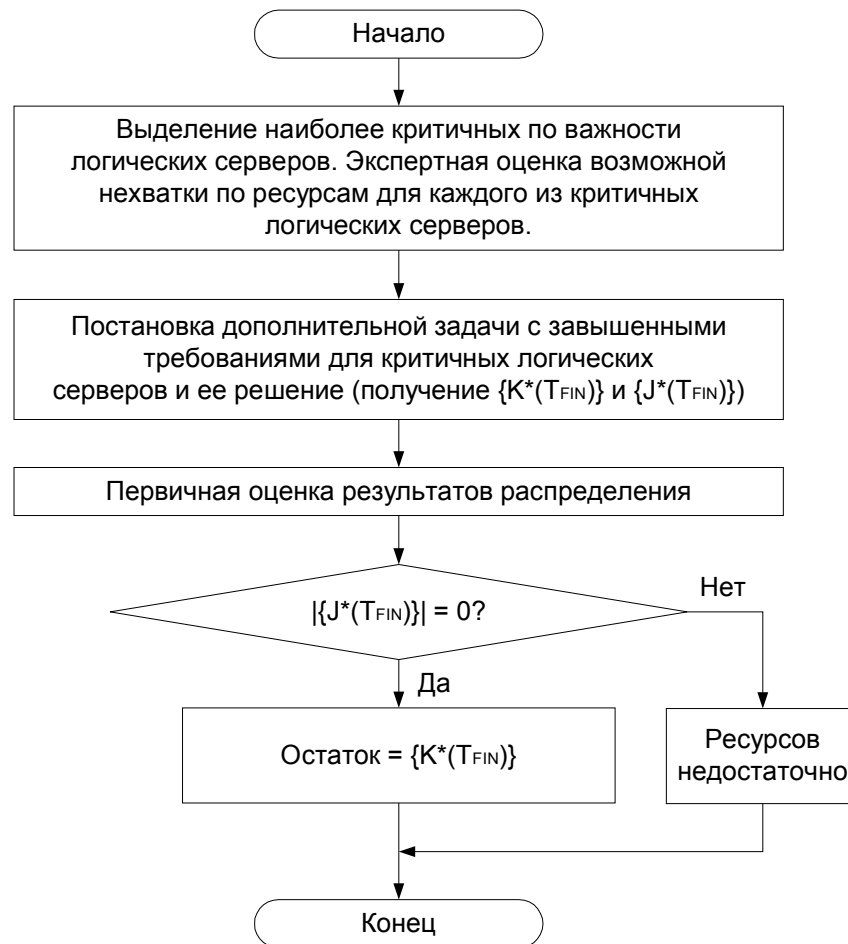


Рис 2.5. Схема алгоритма вторичной оценки распределения

- Блок оценки выгоды проекта реорганизации по остатку  $\{K^*(T_{FIN})\}$  после вторичной оценки полученного распределения для первого этапа и целесообразности проведения очередной корректировки по остатку  $\{K(T_{FIN})\}$  после первичной оценки полученного распределения для второго этапа. Этот блок фактически выполняется заказчиком, и для нас в общем случае данный блок является “черным ящиком”, конкретная реализация которого определяется самим заказчиком. На вход мы подаем объем незадействованного оборудования (или суммарную стоимость освободившихся компьютеров), на выходе – некоторая коммерческая оценка, на основе которой заказчик принимает окончательное решение. В рамках данной диссертации мы не будем детализировать этот блок.

## 2.5. Первичная реорганизация серверного парка

В случае положительного решения о целесообразности проведения реорганизации серверного парка на первом этапе реорганизации (рис. 2.1) вступает в действие алгоритм для второго этапа реорганизации (рис. 2.2). Второй этап начинается с проведения работ по реорганизации серверного парка в соответствии с первичным распределением, полученного на первом этапе. В связи с этим возникает необходимость проведения ряда мероприятий:

- Планирование времени проведения работ и остановка функционирования логических серверов, задействованных в распределении.
- Подготовка дополнительных логических серверов, требуемых для дублирования функций логических серверов.
- Выполнение резервного копирования данных и конфигурации логических серверов для возможности восстановления серверного парка в исходное состояние в случае необходимости.
- Проведение ряда операций для подготовки логических серверов для переноса на виртуальную платформу. Сохранение данных жестких дисков логических серверов в виде файлов образов.
- Удаление на физических компьютерах, задействованных в распределении, всех данных. Установка и конфигурирование базовой операционной системы и специального программного обеспечения, реализующего технологию виртуальных машин.
- Подготовка и настройка конфигурации виртуальных машин в соответствии с первичным распределением. Копирование содержимого жестких дисков логических серверов на виртуальные диски виртуальных машин.

Рассмотрим подробнее вышеперечисленные пункты работ:

### **Планирование и остановка логических серверов**

По согласованию с заказчиком выбирается время проведения работ и к началу работ производится остановка логических серверов. Работы должны планироваться таким образом, чтобы они по возможности не нарушали бизнес-процессы заказчика, и не приводили к дополнительным затратам.

Разумеется, зачастую возможны случаи высокой загрузки логических серверов, когда допускается только кратковременная остановка серверов и, желательно, не более одного сервера за раз – этот случай мы рассмотрим в отдельном порядке.

### **Подготовка дополнительных логических серверов**

Ранее было рассмотрено, что при первичном сборе информации для повышения надежности функционирования серверного парка выделяются критичные по важности логические серверы, для которых требуется серверы, дублирующие их функции. Соответственно, если в серверном парке для них уже были серверы-дублиеры, то в таком случае просто формировались ограничения-исключения для задачи распределения, чтобы предотвратить размещение логического сервера и серверов, дублирующих его функции, на одном физическом компьютере. Если же для какого-либо логического сервера серверы-дублиеры отсутствовали, то возникала необходимость в создании хотя бы одного дополнительного дублирующего сервера, и, соответственно, множество логических серверов расширялось, и в исходные условия задачи распределения добавлялись дополнительные ограничения-исключения.

Однако, очевидно, что к моменту первичной реорганизации дополнительные логические серверы еще не существуют, и их необходимо создать. Соответственно, поскольку изначально все физические компьютеры серверного парка находятся в работе, то от исполнителя работ потребуются один или более компьютеров, для развертывания на них временных базовых ОС, подготовки виртуальных машин и развертывания на них дополнительных логических серверов. Мы не будем подробно останавливаться на том, как именно конфигурируются дополнительные логические серверы и синхронизируются с теми, чьи функции они должны дублировать, поскольку в каждом конкретном случае она решается по-своему. Если необходимо создать дополнительный контроллер домена – это одна процедура, если дополнительный файловый сервер, хранящий зеркальную копию данных другого сервера – это другая. В частности для ОС MS Windows 2000 в

литературе [1, 18] достаточно подробно описано то, как именно разворачиваются ОС и конфигурируются для дублирования тех или функций. Приведем кратко два примера:

- Если необходимо подготовить дополнительный контроллер домена, то достаточно развернуть ОС MS Windows 2000, присвоить ей уникальное сетевое имя, сетевой адрес и с помощью стандартной описанной в литературе процедуры [1, 18] назначить серверу роль дополнительного контроллера существующего домена. В этом случае сервер самостоятельно синхронизирует данные с основным контроллером.
- Если необходимо создать файловый сервер, данные файловых ресурсов которого должны синхронизироваться с данными ресурсов другого сервера, то в этом случае используется технология распределенного хранения данных (Distributed File System) и служба файловой репликации (File Replication Services), которые также освещены в литературе [1, 18]. Соответственно, достаточно развернуть ОС MS Windows 2000, присвоить ей уникальное сетевое имя, сетевой адрес и создать необходимые файловые ресурсы и произвести настройки в конфигурации распределенного хранилища данных.

### **Резервное копирование данных логических серверов**

Любой логический сервер (кроме дополнительно созданного сервера) изначально функционирует на некотором физическом компьютере и хранит свои данные и настройки на одном или нескольких жестких дисках. Соответственно, перед проведением каких-либо работ содержимое жестких дисков компьютеров необходимо сохранить в каком-нибудь виде для возможности полного восстановления исходного состояния логического сервера в случае необходимости.

На сегодняшний день используется технология резервного хранения данных жестких дисков в виде файлов образов, содержащих в себе полную информацию о разделах, логических дисках, файловых системах и всех данных. Для “свертывания” данных дисков в файлы-образы и “развертывания” данных дисков из файлов-образов существует множество дисковых утилит от

различных производителей программного обеспечения, самые известные это – Symantec Norton Ghost и Power Quest Drive Image Pro. Утилиты достаточно просты в использовании и, кроме того, поставляются с достаточно подробным руководством для пользователя. Поэтому мы не будем останавливаться на описании и особенностях этих утилит.

Очевидно то, что для резервного хранения данных дисков логических серверов на время реорганизации серверного парка, требуются определенные дополнительные средства хранения данных (как правило, это жесткие диски). Соответственно, крайне желательно то, чтобы исполнитель работ располагал своими собственными средствами для резервного хранения данных, поскольку недопустимо рассчитывать на дополнительные средства заказчика, пусть даже и только на время работ по реорганизации.

### **Подготовка для переноса на виртуальную платформу**

Как известно, большинство операционных систем в той или иной степени привязаны к оборудованию, на котором они были установлены и сконфигурированы. Причем такая привязка имеет место быть, как правило, не по причине защит от нелегального копирования программного обеспечения, а необходимости тесного взаимодействия ОС и оборудования на этапе ее загрузки. Соответственно, так называемая машинно-зависимая часть ОС, как минимум содержит в себе некоторые ключевые системные драйверы, предназначенные для взаимодействия с конкретной аппаратной платформой, с контроллерами дисков и т.п. По этой причине при переносе системы на другую аппаратную платформу, при смене контроллеров дисков и самих дисков, как правило, для большинства ОС возникает ряд проблем:

- Невозможность загрузки ОС из-за различий в ядре аппаратной платформы – различия в особенности управления ресурсами материнской платы, организации мультипроцессорности и т.д. Большинство ОС при установке определяет тип материнской платы, тип процессоров и их количество и устанавливает для них специфичные драйверы, являющиеся компонентами машинно-зависимой части.

- Невозможность загрузки из-за различий контроллеров дисков. Большинство ОС при установке определяет тип контроллер дисков, конфигурацию дисков и устанавливает соответствующие специфичные драйверы устройств.
- Невозможность входа в рабочий сеанс пользователя системы. Данная проблема возникает, как правило, после смены системного диска. Большинство ОС жестким дискам и его разделам сопоставляет некоторые уникальные коды и хранит их в реестре, эти коды используются при назначении букв для разделов. При смене системного диска, система распознает этот диск как новый, генерирует для него и его разделов новые коды и заново выполняет процедуру назначения букв разделам. В таких случаях, нередко системному разделу присваивается буква, отличная от той, которая была ранее, а поскольку большая часть информации о местоположении программного обеспечения, персональных данных и настройках пользователей и т.п. хранится в виде путей, содержащих букву раздела, то после смены буквы раздела диска многие данные становятся недоступными. В частности, недоступность профилей пользователей делает невозможным вход в рабочий сеанс пользователя системы, что делает операционную систему непригодной для работы.

Вышеперечисленный список проблем отнюдь не является полным: он актуален в основном для семейства ОС MS Windows 2000. Соответственно, в других ОС проблем может быть меньше или наоборот больше. Так, например, ОС MS Windows 95/98 гораздо более “терпима” к замене материнской платы, нежели чем ОС MS Windows 2000: как правило, после переноса системного диска на другой компьютер ОС MS Windows 95/98 без проблем загружается и работает. В то же время, ОС MS Windows XP к вышеуказанным трем ключевым проблем добавляет еще, так называемую проблему активации – специальный механизм, предназначенный для защиты от незаконного копирования – после переноса системы на другой компьютер, ОС требует повторной активации, без которой она отказывается работать.



В рамках данной диссертации нет возможности рассмотреть большое множество ОС в сочетании с множеством аппаратных платформ и различных дисковых конфигураций при различных контроллерах дисков. Однако, в рамках исследований, проводившихся при эксплуатации серверных систем, автором было разработано руководство по переносу ОС MS Windows 2000 с физического компьютера на виртуальную машину (см. приложение 2), которое уже несколько лет используется автором и специалистами ИВЦ МЭИ (ТУ). Время для развертывания “с нуля” ОС, ее настройки и установки всего необходимого программного обеспечения в сложных случаях может занимать до нескольких дней или даже больше, перенос же системы занимает обычно не более 2-3 часов. В разработанном руководстве подробно рассмотрены ОС MS Windows 2000 и ее особенности с точки зрения привязок к оборудованию, подробно описаны вышеуказанные три ключевые проблемы, возникающие при переносе, и способы их обхода, приведен порядок действий “шаг за шагом” при переносе системы с физического компьютера на виртуальную машину. Однако, очевидно то, что в случае других ОС, возможно, потребуются дополнительные исследования – это остается за рамками данной диссертационной работы. Таким образом, используя те или иные технологии переноса, специфичные для каждой ОС, можно подготовить логические серверы к переносу. Для этого логические серверы запускаются, после этого в них вносятся необходимые подготовительные корректировки, после чего серверы останавливаются, и выполняется сохранение системного диска в виде файла образа для последующего развертывания на виртуальной платформе.

Следует четко различать то, что на предыдущем шаге первичной реорганизации серверного парка выполняется резервное копирование логического сервера в его исходном состоянии, а на данном шаге выполняется копирование логического сервера в модифицированном (для возможности успешного переноса) состоянии. Соответственно, на данном шаге также могут потребоваться некоторые дополнительные дисковые ресурсы.

### **Настройка базовой ОС на физических компьютерах**

Для того, чтобы размещать виртуальные машины, необходим базовый физический компьютер с установленной базовой ОС и специальным программным обеспечением. Для этого первоначально требуется то, чтобы на физическом компьютере были удалены все данные для того, чтобы проводить настройку на “чистой” машине. Поскольку на физических компьютерах, задействованных в распределении, изначально функционируют логические серверы, которые также задействованы в этом распределении, то еще на третьем шаге первичной реорганизации серверного парка выполняется резервное копирование данных всех логических серверов. Кроме того, на четвертом шаге выполняется резервное копирование данных логических серверов, подготовленных для переноса. Следовательно, на данном шаге можно удалить все данные всех физических компьютеров, задействованных в распределении. После этого на них устанавливается базовая ОС и специальное программное обеспечение. Мы не будем останавливаться на описании установки и настройки, потому что такие вопросы достаточно подробно освещаются в документации к программному обеспечению.

### **Развертывание виртуальных машин на физических компьютерах**

После того, как все физические компьютеры подготовлены для размещения на них виртуальных машин, для каждого задействованного в распределении физического компьютера при помощи утилит специального программного обеспечения, реализующего технологию виртуальных машин, в базовой ОС создаются виртуальные машины в соответствии с распределением:

- К базовому компьютеру подключается носитель с файлами-образами логических серверов, подготовленных к переносу и предназначенных для размещения на данном физическом компьютере.
- Создаются новые виртуальные машины, для которых, согласно требованиям соответствующего логического сервера, в конфигурации указывается необходимая емкость для оперативной памяти, необходимое количество виртуальных дисковых устройств и их емкости. Обязательно наличие в

конфигурации дисководов для гибких дисков либо привода CD-ROM для возможности первоначальной настройки виртуальной машины и развертывания логического сервера. Наконец, в конфигурацию обязательно включается носитель с файлами-образами логических серверов как дополнительный виртуальный жесткий диск на базе существующего физического диска, причем физический диск для безопасности данных желательно подключать в режиме “только для чтения”.

- В каждой виртуальной машине по очереди выполняется первичная загрузка с дискеты или компакт-диска, содержащие необходимые дисковые утилиты, производится разбиение на разделы виртуальных жестких дисков логического сервера. После этого с подключенного к виртуальной машине дополнительного виртуального диска, использующего физический диск с файлами-образами логических серверов, при помощи дисковых утилит производится развертывание разделов дисков логических серверов на соответствующие разделы виртуальных жестких дисков.
- После развертывания данных логических серверов на виртуальные диски, виртуальных машины необходимо выключить и обязательно отключить в конфигурации виртуальный жесткий диск, хранящий файлы-образы.
- Запустить виртуальные машины, загрузить системы и произвести окончательные настройки на логических серверах: настройка видеоадаптера, стека сетевых протоколов и т.п.

В качестве примера подробного описания вышеуказанной процедуры “шаг за шагом” с множеством графических иллюстраций можно воспользоваться разделом “Порядок действий при переносе системы на виртуальную платформу” руководства по переносу ОС MS Windows 2000 с физического компьютера на виртуальную машину (см. приложение 2). Поэтому здесь мы не будем подробно останавливаться на этом. Порядок подготовки и настройки виртуальных машин подробно описано в фирменной документации разработчиков специального программного обеспечения, реализующего технологию виртуальных машин.

Таким образом, выполнив шаг за шагом первичную реорганизацию серверного парка, мы внедряем первичный вариант распределения. Однако, как упоминалось ранее, часто возможна такая ситуация, что заказчиком не допускается одновременное отключение множества функционирующих серверов. В таком случае реорганизацию выполняют постепенно: заранее подготавливают дополнительные логические серверы, останавливают первый логический сервер, сохраняют его резервную копию, готовят к переносу, сохраняют их данные в виде файлов-образов, и таким образом получают первый базовый компьютер, доступный для развертывания базовой ОС. Далее логические серверы, которые должны быть размещены на этом компьютере, по очереди останавливают, сохраняют их резервные копии, готовят к переносу, сохраняют их данные в виде файлов-образов, после чего они уже не должны запускаться. Далее они переносятся на виртуальную платформу и запускаются на базовом компьютере. Соответственно, после этого один или несколько физических компьютеров становятся свободными, и они могут быть использованы для дальнейшего размещения логических серверов в соответствии с распределением. Таким образом, постепенно все логические серверы будут переведены на виртуальную платформу и размещены на соответствующих физических компьютерах.

На рисунке 2.6 представлен алгоритм первичной реорганизации серверного парка.

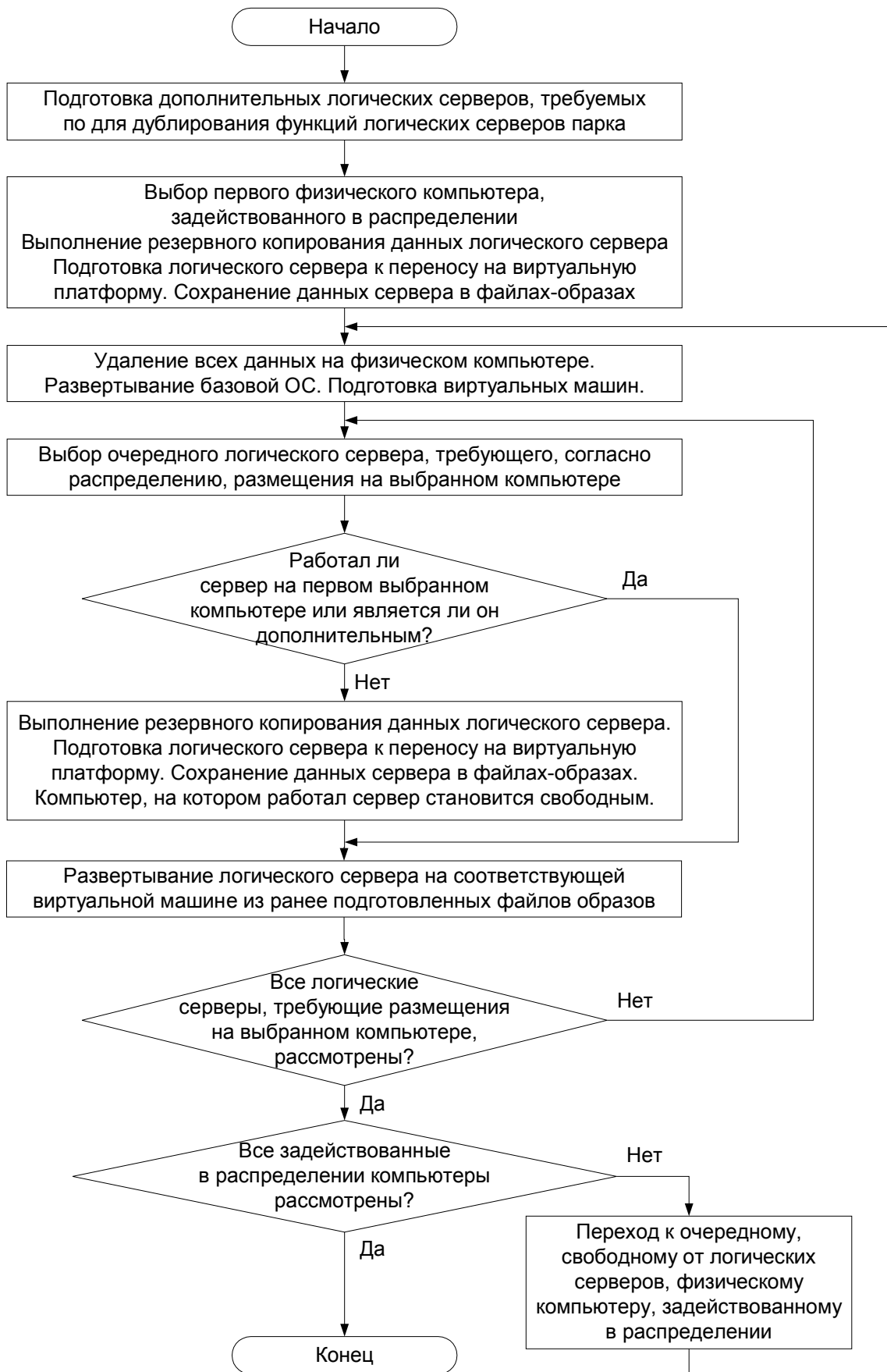


Рис. 2.6. Схема алгоритма первичной реорганизации

## 2.6. Оценка качества функционирования серверного парка

После завершения работ по первичной реорганизации либо после внесения очередных корректировок возникает необходимость в оценке качества функционирования серверного парка. На сегодняшний день существует большое множество программных продуктов с различными требованиями к аппаратным средствам и особенностями в реализации и, разумеется, невозможно заранее предусмотреть абсолютно все возможные проблемные ситуации, которые могут возникнуть после реорганизации. Реальные проблемы будут проявляться после завершения работ по реорганизации, причем, вполне вероятно то, что найдутся такие проблемы, которые проявятся только через некоторое время. Соответственно, полноценная оценка качества функционирования в общем случае это не только рутинный, но и достаточно длительный процесс. Так или иначе, в данной ситуации и исполнителю и заказчику приходится искать компромисс между качеством оценки и требуемым временем для ее проведения, и в каждом конкретном случае оговариваются свои конкретные условия и сроки.

Что касается основных параметров, по которым можно судить о качестве функционирования, то, как правило, у исполнителя нет возможности вдаваться глубоко в особенности каждого программного продукта. Как правило, достаточно хотя бы выполнить следующий минимальный набор действий:

- В первом приближении выявить те ключевые для организации сервисы, которые после реорганизации по тем или иным причинам перестали функционировать вообще. На практике, такая ситуация крайне маловероятна, и может появиться только в том случае, если на первом этапе реорганизации (рис. 2.1) были допущены грубые ошибки при первичном сборе информации по серверному парку.
- Выявить те сервисы, которые стали обрабатывать запросы с неудовлетворительным временем обслуживания (рост времени отклика до неприемлемого уровня). Такая ситуация вполне возможна, поскольку ключевой сервис может зависеть от множества конкретных сервисов

конкретных логических серверов. Поскольку на физическом компьютере после реорганизации размещается уже не один логический сервер, а множество, то если какой-либо сервис какого-либо сервера периодически либо при некоторых обстоятельствах создает большую нагрузку на процессор компьютера, то, очевидно, что время отклика других сервисов других логических серверов в такие моменты окажется большим нежели, чем до реорганизации. Это, в свою очередь, приведет к росту времени обслуживания по конечному ключевому сервису. Однако, следует упомянуть то, что во введении диссертационной работе было оговорено то, что методику нет смысла применять в случае, если на время отклика сервисов накладываются жесткие ограничения или какие-либо ресурсы постоянно используются с полной загрузкой (в данном случае процессор). Тем не менее, в алгоритме для первого этапа реорганизации (рис. 2.1) присутствует блок оценки “ухудшенной” выгоды путем завышения требований логических серверов на случай возможных корректировок.

- По возможности следует в течение некоторого времени эксплуатировать серверный парк в режиме наблюдения для того, чтобы иметь возможность получения дополнительной оценки качества функционирования на основе отзывов пользователей организации заказчика. В данном случае полезно при участии пользователей и заказчика смоделировать некоторую “тяжелую” ситуацию, (например, отчетный период или что-то иное) когда на серверный парк создается максимальная нагрузка (максимально возможная на практике для данной конкретной организации). Соответственно, если в этом случае пользователи будут удовлетворены временем обработки их запросов, то, очевидно, что этого будет достаточно для положительной оценки.

С учетом всего вышесказанного, как один из возможных вариантов детализации блока выявления негативных последствий после реорганизации серверного парка для второго этапа реорганизации (рис. 2.2) можно привести алгоритм, представленный на рисунке 2.7.

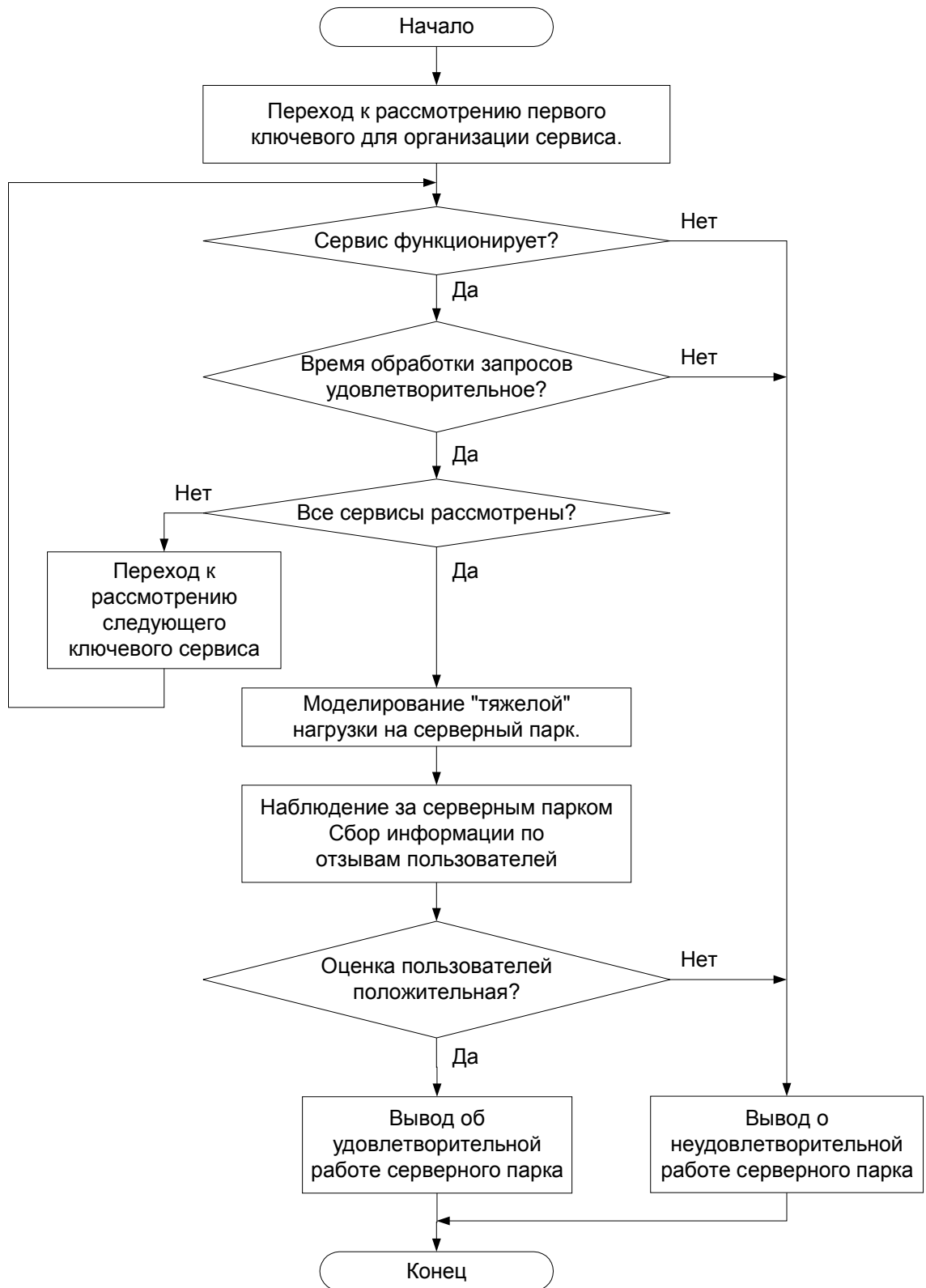


Рис. 2.7. Схема алгоритма по оценке качества функционирования серверного парка



## 2.7. Поиск причин неудовлетворительного функционирования

Если при оценке качества функционирования серверного парка работа сводится в основном к выявлению каких-либо внешних признаков “плохого качества”, то поиск причин “плохого качества” представляет куда более трудоемкую работу. Сложность поиска причин проблем, в первую очередь, заключается в том, что конечный сервис, имеющее ключевое значение для пользователя, в общем случае может являться лишь конечной вершиной в сложном ориентированном графе, состоящего из сервисов (вершин-служб) и зависимостей (направленных дуг). Приведем в качестве примера такого графа для системы безопасной корпоративной почты организации на базе ОС MS Windows 2000 [1, 18, 46, 47, 48, 49, 50] и программного обеспечения MS Exchange Server [41, 42, 44] (рис. 2.8):

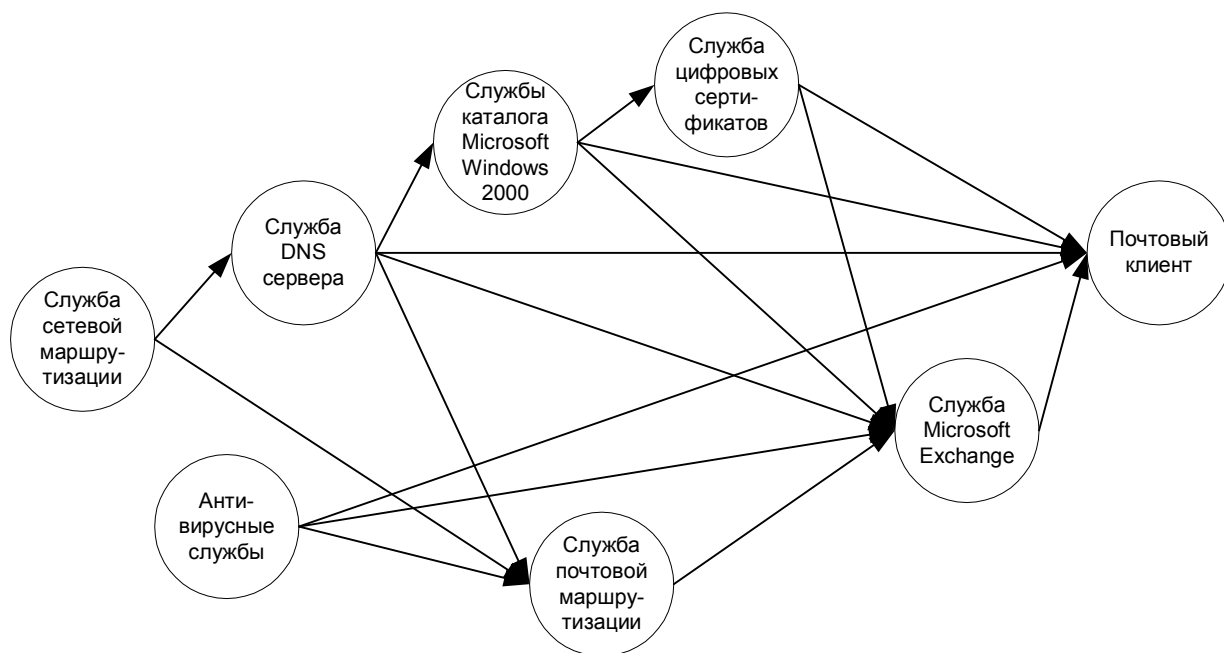


Рис. 2.8. Пример графа зависимостей сервисов для почтовой системы

В примере хорошо видно то, что качество предоставляемой пользователю услуги “корпоративной почты” зависит от качества работы множества зависимых друг друга от сервисов, которые в общем случае могут размещаться на различных логических серверах. Соответственно, если пользователя не удовлетворяет время доставки почтовых сообщений внутри организации между

пользователями, то, очевидно, что внешние почтовые службы Интернета здесь не причем, и приходится искать “узкое место” во внутренней системе корпоративной почты. Если же возникают проблемы с доставкой почты на внешние почтовые серверы, то задача поиска усложняется и выходит за рамки внутренней почтовой системы.

В качестве еще одного примера можно привести граф для такого ключевого сервиса как доступ в Интернет (в упрощенном виде) на базе ОС MS Windows 2000, программного обеспечения MS Proxy Server [44, 45] (рис. 2.9):

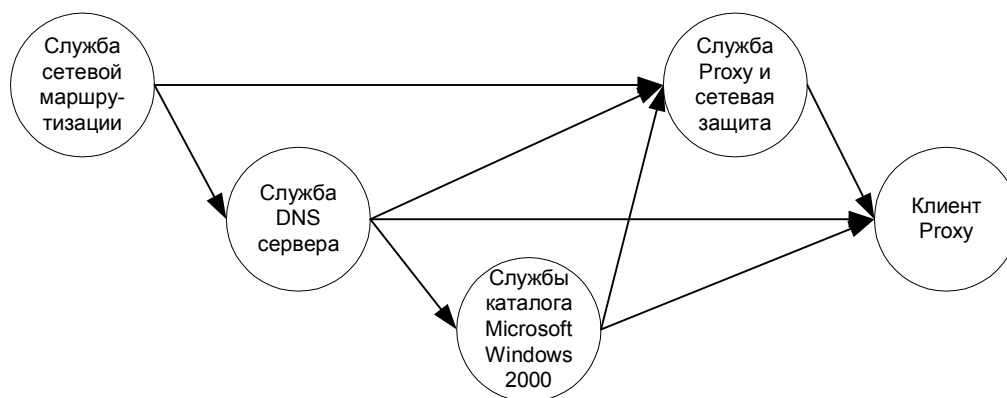


Рис. 2.9. Пример графа зависимостей сервисов, обеспечивающих доступ к ресурсам Интернет, находящихся вне корпоративной сети.

Причины проблем с доступом к ресурсам Интернет могут крыться как внутри, так и вне корпоративной сети. Для локализации причин проблемы обычно проверяют доступ к заведомо доступным ресурсам Интернет, проводят трассировку цепочки маршрутизации от рабочего места пользователя до конечного сервера, предоставляющего ресурс Интернета и т.д.

В качестве последнего примера, приведем граф (рис. 2.10) для ключевого сервиса, о котором пользователи, как правило, не знают и не используют непосредственно для выполнения своих задач, но который существенно упрощает администрирование сети, путем автоматического конфигурирования настроек сетевых протоколов на рабочих местах пользователей – это службы DHCP (Dynamic Host Configuration Protocol) [43, 44]. Причины проблем с этим сервисом кроются, как правило, внутри корпоративной сети.

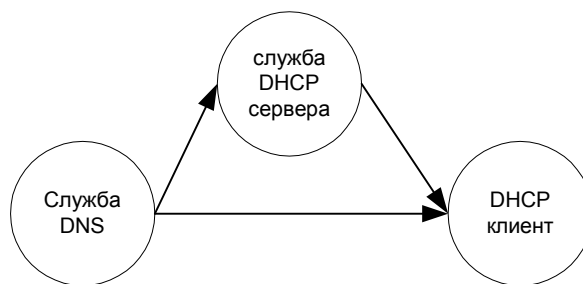


Рис. 2.10. Пример графа зависимостей сервисов, обеспечивающих автоматическое конфигурирование настроек сетевых протоколов.

Вышеприведенные примеры – это лишь самая незначительная доля огромного множества схем зависимостей сервисов, которые могут встретиться на сегодняшний день. Практически невозможно разработать какую-либо универсальную процедуру для поиска причин тех или иных проблем, связанных с каким-либо сервисом (необязательно, конечным в графе зависимостей и важным для конечного пользователя). В каждом конкретном случае причины могут крыться в особенностях программной реализации тех или иных сервисных функций, специфики зависимостей сервисов друг от друга, а также в возможностях физических компьютеров, коммутационного оборудования и линий связи. Часто встречаются случаи, когда простая замена какого-либо программного обеспечения с одной версии на другую может решить проблему, но зачастую такая замена может вызвать ряд других проблем. Однако, следует упомянуть то, что в нашем случае мы должны обращать внимание только на те проблемы, которые возникли именно после реорганизации, и результатом поиска причин проблем должно быть выявление нехватки по каким-либо из ресурсов или принципиальной несовместимости с виртуальной платформой для каких-либо логических серверов. Причины такого рода позволяют внести конкретные корректировки в исходные условия задачи: изменение требований к ресурсам для логических серверов, добавление новых ограничений типа исключений или вообще исключение из рассмотрения какого-либо логического сервера вместе с физическим компьютером, на котором он работает.

Рассмотрим следующий пример: допустим, что в результате реорганизации некоторого серверного парка время обслуживания по какому-либо ключевому сервису существенно возросло. В такой ситуации лучше всего действовать по следующему алгоритму (рис. 2.11): выбрать какую-либо заведомо работоспособную рабочую машину пользователя и выделить сервисы, от которых ключевой сервис непосредственно зависит. Далее установить и запустить программы-анализаторы сетевых пакетов данных на рабочей машине пользователя и всех компьютерах, на которых работают логические серверы, содержащие выделенные сервисы. Программы-анализаторы позволяют всегда точно выяснить время отправки или получения пакетов данных. После этого необходимо послать несколько тестовых запросов по заданному ключевому сервису с рабочей машины пользователя. Далее, тщательно анализируя данные, полученные анализаторами сетевых пакетов, выявить сервисы, задержки в работе которых “вносят существенный вклад” в конечную задержку. Среди выявленных сервисов, пометить как “проблемные”, те сервисы, которые либо не зависят от других сервисов либо те сервисы, от которых они зависят, не являются причиной задержки. Остальные выявленные сервисы, зависящие от других сервисов, являющихся причиной задержек в работе выявленных сервисов, нуждаются в дальнейшем анализе взаимодействия между ними и теми сервисами, от которых они зависят. Таким образом, рано или поздно образуется список “проблемных” сервисов, являющихся корневыми источниками проблемы задержки. Таким образом, локализируются “проблемные” сервисы “проблемных” логических серверов, работающих на одном или нескольких “проблемных” компьютерах. Далее, на “проблемных” компьютерах анализируется состав логических серверов и создаваемая ими нагрузка на компоненты (по типам ресурсов) компьютера. После этого для каждого “проблемных” компьютера выделяются логические серверы, для которых наблюдается нехватка ресурсов. Далее, экспертным путем устанавливаются скорректированные требования к ресурсам для выделенных логических серверов, и ставится новая задача поиска распределения.

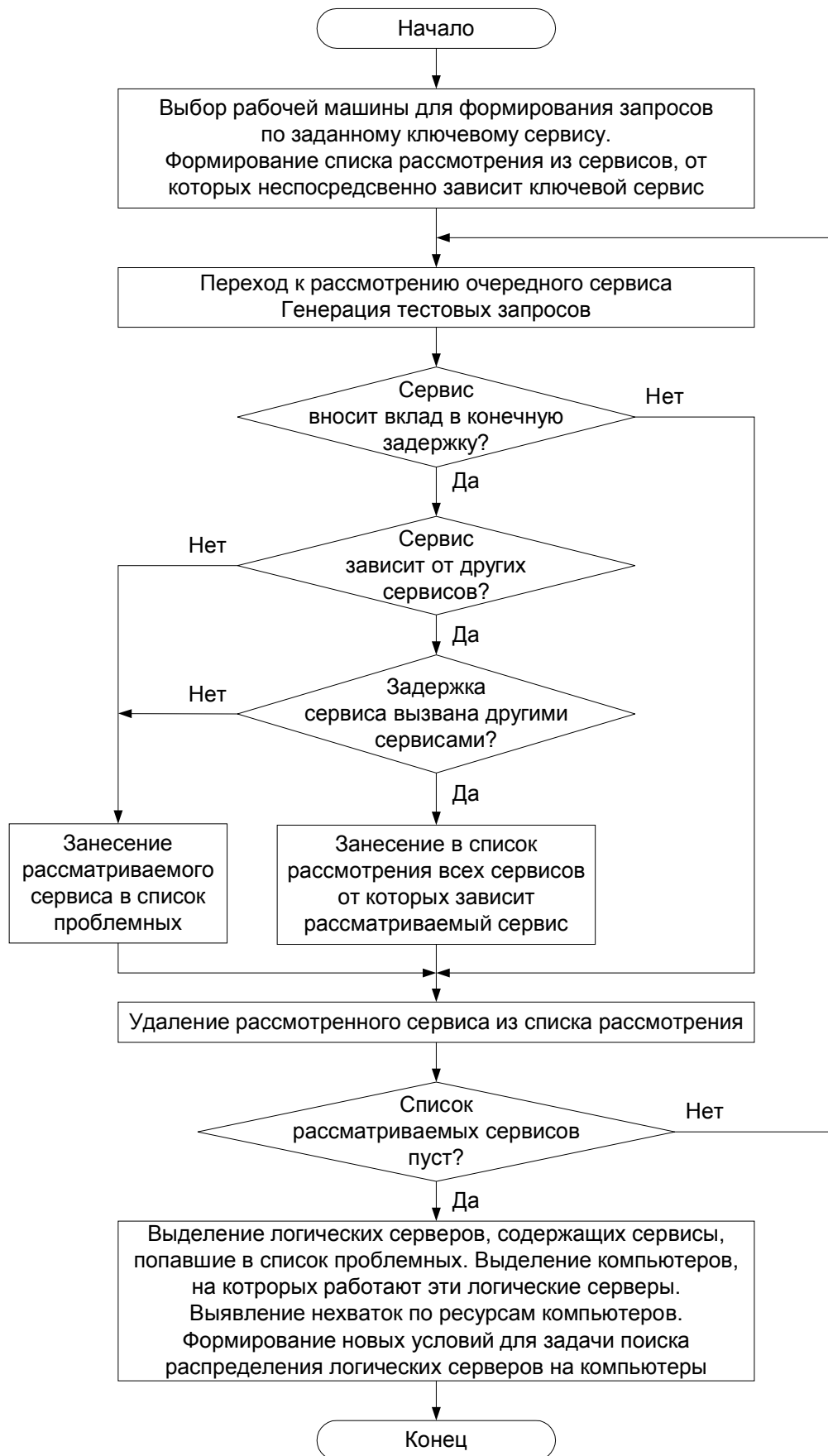


Рис. 2.11. Схема примера алгоритма поиска причин проблем с ключевым сервисом и формирования новых скорректированных условий.

## 2.8. Корректировочная реорганизация серверного парка

После выявления проблемных сервисов, являющихся причиной неудовлетворительной работы серверного парка, поиска причин проблем, выявления причин и корректировки исходных условий задачи распределения и повторного ее решения, и в случае если конечная выгода с учетом скорректированного решения остается приемлемой для заказчика, то возникает необходимость внедрения нового варианта распределения. На первый взгляд может показаться то, что объем работ по внедрению скорректированного распределения сопоставим с объемом работ первичной реорганизации. На практике же, как правило, требуется вовлечения одного или нескольких дополнительных из списка ранее незадействованных компьютеров для размещения на них логических серверов, которые в силу новых условий и ограничений не могут быть размещены на задействованных в первичном варианте распределения компьютерах. Возможен так же и тот случай, когда в новом варианте распределения задействованные компьютеры остаются те же самые, и требуются лишь некоторая “перетасовка” логических серверов работающих на них. Так или иначе, мы будем исходить из того, что после первичной или очередной корректировочной реорганизации оставалось достаточное количество (выгодное для заказчика по его оценке) незадействованных компьютеров и в новом скорректированном распределении также остается достаточное количество незадействованных компьютеров, привлекательное с коммерческой точки зрения для заказчика.

С технологической точки зрения корректировочная реорганизация серверного парка не отличается от первичной реорганизации. Тем не менее, объем работ в корректировочной реорганизации значительно меньше: не требуется резервное копирование данных логических серверов, подготовки к переносу на виртуальную платформу и сохранение подготовленных систем в виде файлов образов, поскольку все это было выполнено при первичной реорганизации. В случае если количество задействованных физических компьютеров становится больше (дополнительные компьютеры берутся из

списка ранее незадействованных компьютеров рассматриваемого серверного парка), то на дополнительных компьютерах можно сразу удалять все данные и разворачивать базовую операционную систему, поскольку все операции по резервному копированию данных были выполнены при первичной реорганизации. После того как все дополнительные компьютеры подготовлены, выполняется перераспределение логических серверов в соответствии с новым вариантом распределения – на практике это сводится к простому копированию файлов-образов логических серверов с одного компьютера на другой (одно из наиболее существенных достоинств технологии виртуальных машин), более того логические серверы не требуют никакой предварительной подготовки (это было сделано при первичной реорганизации).

Следует отметить то, что возможна ситуация когда в новом варианте распределения какие-либо ранее задействованные компьютеры могут оказаться не востребованными, в таком случае после того, как все логические серверы будут перемещены с них на другие компьютеры они станут свободными (незадействованными). Кроме того, при перераспределении логических серверов возможна следующая ситуация: на конечном компьютере, куда должен быть какой-либо логический сервер недостаточно дискового пространства, которое станет достаточным, только тогда, когда, в свою очередь, с этого компьютера согласно новому варианту распределения некоторые логические серверы будут перемещены на другие компьютеры. В таких случаях, очевидно, потребуются дополнительные дисковые носители, их предоставляет исполнитель работ.

Соответственно, мы можем детализировать блок корректировочной реорганизации серверного парка алгоритма для второго этапа реорганизации (рис 2.2) и представить этот в виде отдельного алгоритма (рис. 2.12).

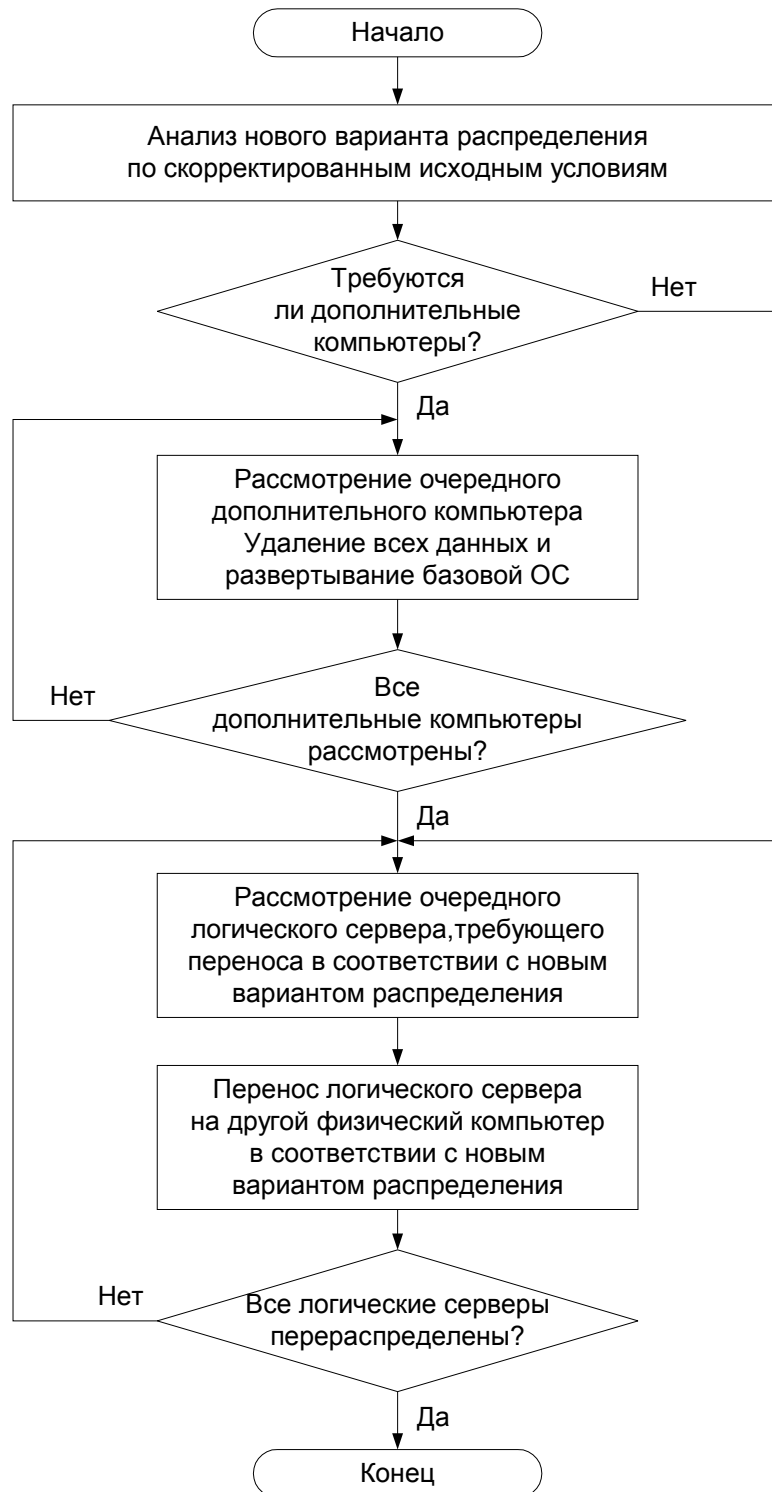


Рис. 2.12. Схема алгоритма  
корректировочной реорганизации серверного парка



## 2.9. Возврат к исходному состоянию или компромиссы

Как упоминалось ранее, в случае если качество функционирования серверного парка не удовлетворяет заказчика, и если при этом предложенный вариант внесения корректировок в реорганизацию серверного парка либо снижает, в конечном счете, ожидаемую выгоду до неприемлемого уровня, либо вообще требует дополнительных затрат, которые также сводят на нет всю выгоду, то возможна крайняя ситуация – отказ от проекта реорганизации с восстановлением исходного состояния серверного парка. Однако, здесь следует отметить то, что технология виртуальная машин помимо явной выгоды в виде более эффективного использования ресурсов компьютера при размещении нескольких виртуальных машин, дает и другие преимущества: легкость переноса логических серверов с одного компьютера на другой, упрощение процесса резервного копирования данных, программного обеспечения и их настроек для серверов и, соответственно, возможность быстрого восстановления в случае даже полного краха системы и данных. Исходя из этого, можно сказать то, что технология виртуальных машин, как минимум, всегда дает большую гибкость в управлении и обслуживании серверных систем. Поэтому даже в худших случаях возможны компромиссные варианты, но это уже зависит от заказчика, от его краткосрочных и долгосрочных целей и в каждом конкретном случае проблема решается по-своему.

Рассмотрим ситуацию, когда какие-либо компромиссные варианты отсутствуют, и возврат серверного парка к исходному состоянию неизбежен. В таком случае, самым надежным будет восстановление всех логических серверов на физические компьютеры из резервных копий, сделанных при первичной реорганизации. Очевидно то, что при этом будут потеряны все изменения данных и настроек логических серверов, произошедших в период реорганизации серверного парка, однако, в данном случае это приходится принимать так, как есть. Конечно, допускается попытка обратного переноса систем логических серверов с виртуальной платформы на физические компьютеры, но это крайне сложный и рискованный вариант.

Рассмотрим же теперь для примера один из типовых случаев принятия компромиссных решений. Допустим, что в некоторой организации заказчик желает реорганизовать следующий серверный парк: 20 одинаковых компьютеров с достаточно мощными процессорами, жесткими дисками большой емкости и с сетевой подсистемой (включая линии связи и коммуникационное оборудование) с высокой пропускной способностью и оперативной памятью емкостью 512 МБ. Логические серверы (допустим, что с одинаковыми требованиями), работающие на этих компьютерах, создают довольно слабую нагрузку на процессор и сетевую подсистему, занимают около 30% дискового пространства и в среднем используют около 256 МБ памяти. Исполнитель работ, учитывая то, что базовой ОС требуется не менее 128 МБ памяти, решает пойти на риск, и принимает требования логических серверов к оперативной памяти за 192 МБ – это позволит разместить по 2 логических сервера вместе с базовой ОС на компьютер  $(192*2 + 128 \leq 512)$ . Более того, он исходит из оптимистичных соображений и полагает, что корректировки не потребуются. Однако, после реализации первичного варианта распределения: переноса логических серверов на виртуальную платформу и размещения на 10 компьютерах, качество функционирования серверного парка стало неудовлетворительным. При выяснении причин обнаружилось то, что если виртуальной машине, на котором работает логический сервер, выделяется не 192 МБ, а 256 МБ памяти, то ситуация исправляется. Соответственно, придется размещать логические серверы на 20 компьютерах  $(256 + 128 \leq 512)$ . В такой ситуации, как самый простой вариант заказчик может потребовать возвратить серверный парк в исходное состояние. Однако, есть и компромиссный вариант: приобрести для 10 компьютеров модули памяти емкостью 128 МБ и тогда на этих компьютерах будет 640 МБ памяти, а этого достаточно для размещения на них 2-х логических серверов вместе с базовой ОС  $(256*2 + 128 \leq 640)$ . При этом качество функционирования станет приемлемым, и 10 других компьютеров останутся свободными, а их стоимость куда больше, нежели чем стоимость 10 модулей памяти.

## Выводы по главе 2

Данная глава была посвящена разработке методики реорганизации серверного парка с целью повышения эффективности использования его вычислительных ресурсов. Методика строго привязана к применению технологии виртуальных машин и опирается на поиск оптимального распределения логических серверов на физические компьютеры.

Ввиду сложности и трудоемкости задачи реорганизации серверного парка, а также возможных негативных последствий или возможного отсутствия ожидаемой прибыли после проведения реорганизации, в начале главы сделан вывод о том, что задача должна быть разбита на два этапа:

- Сбор первичной информации, разработка первичного решения на основе первичного распределения логических серверов на компьютеры, оценка качества решения с учетом возможных корректировок после реорганизации, анализ выгоды, которое дает это решение, и оценка целесообразности проведения реорганизации. Цель первого этапа – максимально обезопасить проектное решение от провала на втором этапе – этапе реализации первичного решения.
- Непосредственная реализация первичного решения, выявление негативных последствий реорганизации. В случае неудовлетворительного качества работы серверного парка – корректировка решения (получение распределения по скорректированным исходным данным) и реализация скорректированного решения, далее повторяется анализ качества. Если на каком-либо шаге корректировка невозможна в силу неприемлемого снижения коммерческой выгоды – то поиск компромиссных решений либо отказ от проекта с возвратом серверного парка в исходное состояние.

Поскольку каждый этап реорганизации содержит достаточно сложные и укрупненные блоки, то сделан вывод о том, что требуется дальнейшая детализация для следующих блоков, используемых на первом и втором этапах реорганизации серверного парка:

- Первичный сбор информации по серверному парку.
- Поиск оптимального распределения логических серверов по компьютерам.
- Оценка полученного распределения логических серверов по компьютерам.
- Поиск компромиссов в случае получения неудовлетворительных или невысоких результатов после реорганизации и невозможности исправления ситуации по тем или иным причинам.
- Первичная реорганизация серверного парка.
- Оценка качества функционирования серверного парка.
- Поиск причин неудовлетворительного качества.
- Коррекционная реорганизация серверного парка.

Для всех вышеуказанных блоков кроме блока поиска оптимального распределения логических серверов по компьютерам разработаны соответствующие рекомендации и алгоритмы.

Ввиду большой сложности блока поиска оптимального распределения сделан вывод о необходимости его рассмотрения в отдельной главе. Разработке математической модели и поиску метода решения задачи поиска распределения логических серверов на физические компьютеры посвящена глава 3.

Также следует отметить, что поскольку в постановке задачи четко различалось число физических компьютеров и логических серверов, то это позволяет использовать методику не только для реорганизации, но и при развертывании серверного парка “с нуля”. При этом методика с одной стороны становится проще в применении – нет необходимости в резервном копировании, подготовке логических серверов к переносу на виртуальную платформу, с другой стороны – сложнее оценивать требования будущих логических серверов, поскольку серверный парк развертывается впервые.

### 3. Разработка математической модели и метода решения задачи поиска распределения логических серверов по компьютерам

Разработанная в предыдущей в главе методика реорганизации серверного парка при применении технологии виртуальных машин опирается на поиск оптимального распределения логических серверов на компьютеры. Задача поиска оптимального распределения решается один раз на первом этапе и несколько раз в зависимости от результатов реорганизации серверного парка на втором этапе реорганизации.

Задача поиска оптимального распределения является задачей дискретной оптимизации. Входными данными задачи являются: множество типов ресурсов  $\{1, \dots, NC\}$ , множество компьютеров  $\{H_k\}$  и матрица базовых уровней их ресурсов  $\{R_{i,k}\}$ ,  $k = 1..NH$ ,  $i = 1..NC$ , множество логических серверов  $\{S_j\}$  и матрица их требований  $\{Q_{i,j}\}$ ,  $j = 1..NS$ ,  $i = 1..NC$ , базовая ОС и ее требования  $\{V_i\}$ ,  $i = 1..NC$ , дополнительные ограничения  $\{E_{d,j}\}$ ,  $d = 1..NX$ ,  $i = 1..NC$ , на одновременное размещение логических серверов на один и тот же компьютер, и, наконец, двоичный вектор  $\{O_i\}$ ,  $i = 1..NC$ , определяющий то, по каким именно типам ресурсов предпочтительно решение проблемы их неэффективного использования. Целью является повышение эффективности использования ресурсов. Выходными данными задачи (решением задачи) является матрица распределения  $\{X_{k,j}\}$ ,  $k = 1..NH$ ,  $j = 1..NS$ , логических серверов по физическим компьютерам.

Соответственно, для решения задачи оптимизации требуется разработка математической модели и метода решения задачи.

### 3.1. Разработка математической модели

Поскольку рассматриваемая задача оптимизации является достаточно сложной и не может быть приведена к какой-либо известной задаче оптимизации (типовой задаче, описанной в литературе с известным методом решения), то рассмотрим для начала родственную ей простейшую задачу дискретной оптимизации – задачу о рюкзаке, и далее попытаемся, постепенно усложняя математическую модель задачи о рюкзаке, подойти вплотную к нашей задаче оптимизации, построить для нее математическую модель и предложить некоторый метод решения.

#### Первое приближение

В задаче о рюкзаке [24, 36, 37, 40] задано множество вещей  $\{P_1, \dots, P_N\}$  различного веса  $\{a_j\}$  и ценности  $\{c_j\}$ , рюкзак с определенной вместимостью ( $b$ ) по весу и требуется найти такой набор вещей, при котором рюкзак их мог бы вместить, а суммарная ценность помещенных вещей была максимальной. Математическая модель задачи о рюкзаке выглядит следующим образом:

$$\left\{ \begin{array}{l} \sum_{j=1}^N a_j x_j \leq b \\ L = \sum_{j=1}^N c_j x_j \rightarrow \max \\ \forall j \in [1, N]: x_j \in \{0, 1\} \end{array} \right. \quad (3.1a)$$

#### Второе приближение

Во втором приближении – отсутствуют ценности вещей, но их роль выполняют уровни загрузки ресурсов, для задачи о рюкзаке это отношение веса вещи к вместимости рюкзака по весу. Соответственно, математическая модель немного преобразуется:

$$\left\{ \begin{array}{l} \sum_{j=1}^N a_j x_j \leq b \\ L = \sum_{j=1}^N \frac{a_j}{b} x_j \rightarrow \max \\ \forall j \in [1, N]: x_j \in \{0, 1\} \end{array} \right. \quad (3.16)$$

### Третье приближение

В третьем приближении, мы имеем дело с многомерным ресурсом. Соответственно, рюкзак и вещи становятся многомерным. Можно представить это следующим образом – в рюкзаке есть множество отсеков различной вместимости по весу  $\{b_i\}$ ,  $i = 1..M$ , а вместо вещей теперь однородные комплекты более мелких вещей весом  $\{a_{ij}\}$ , причем  $i$ -я вещь любого  $j$ -го комплекта может быть помещена только в  $i$ -й отсек рюкзака и не иначе. Кроме того, комплект может размещаться только целиком со всеми входящими в него вещами, разделение комплекта не допускается. Очевидно, в таком случае помимо того, что число ограничений увеличивается до  $M$ , число критериев оптимизации также может быть равно  $M$ , но в то же время, если число ограничений строго фиксировано, то оптимизацию проводить вовсе необязательно по всем отсекам рюкзака, можно, в частности, оптимизировать только один конкретный  $i^*$ -й отсек, где  $1 \leq i^* \leq M$ . Тогда математическую модель можно представить следующим образом:

$$\left\{ \begin{array}{l} \sum_{j=1}^N a_{ij} x_j \leq b_i, i = 1..M \\ L = \sum_{j=1}^N \frac{a_{i^*j}}{b_{i^*}} x_j \rightarrow \max, 1 \leq i^* \leq M \\ \forall j \in [1, N]: x_j \in \{0, 1\} \end{array} \right. \quad (3.1в)$$

### **Четвертое приближение**

В четвертом приближении, критериев может быть несколько – предпочтительно решение проблемы неэффективного использования по многим (или даже по всем  $M$ ) типам ресурсов (отсеков рюкзака). То, по каким именно отсекам проводить оптимизацию можно задать при помощи того же вектора маски  $\{O_i\}$ , о котором шла речь в предыдущем разделе главы, где  $O_i$  – булева переменная, равная “1” – если по  $i$ -му типу ресурса предпочтительно повышение эффективности его использования, “0” – в противном случае. Однако, как известно, большинство точных и приближенных методов решения задач оптимизации рассчитаны только на одну целевую функцию, и поэтому нам необходимо пойти на некоторый компромисс и выработать некоторую общую целевую функцию для множества заданных критериев. Общую целевую функцию можно представить как сумму целевых функций для каждого отдельного оптимизируемого по эффективности использования типа ресурса. Кроме того, для удобства можно нормировать эту сумму, поделив ее на число целей по отдельным типам ресурсам, для которых проводится оптимизация. Тогда математическая модель принимает следующий вид:

$$\left\{ \begin{array}{l} \sum_{j=1}^N a_{ij} x_j \leq b_i, i = 1..M \\ L = \frac{1}{\sum_{i=1}^M O_i} \left( \sum_{i=1}^M \frac{O_i}{b_i} \left( \sum_{j=1}^N a_{ij} x_j \right) \right) \rightarrow \max \\ \forall j \in [1, N]: x_j \in \{0,1\} \end{array} \right. \quad (3.1r)$$

### **Пятое приближение**

В пятом приближении, при решении задачи оптимизации необходимо еще учитывать так называемые требования базовой ОС – для задачи о рюкзаке это некоторый специальный комплект, который не участвует в размещении, но обязательно находится в рюкзаке, занимая некоторое место  $\{V_i\}$  в каждом его отсеке. Тогда математическая модель принимает следующий вид:



$$\left\{ \begin{array}{l} \sum_{j=1}^N a_{ij} x_j \leq b_i - V_i, i=1..M \\ L = \frac{1}{\sum_{i=1}^M O_i} \left( \sum_{i=1}^M \frac{O_i}{b_i - V_i} \left( \sum_{j=1}^N a_{ij} x_j \right) \right) \rightarrow \max \\ \forall j \in [1, N]: x_j \in \{0,1\} \end{array} \right. \quad (3.1д)$$

Следует заметить то, что если для какого-либо типа ресурса  $b_i \leq V_i$ , то сразу можно сказать то, что ни один комплект не разместится в рюкзаке, и задачу нет смысла решать. Второй важный момент – это то, что целевая функция ограничена снизу нулем: случай, когда ни один комплект мы не размещаем в рюкзаке –  $\forall j: x_j = 0$ , и ограничена сверху единицей: в силу заданных ограничений сумма требований по каждому типу ресурсов не может превышать базовый уровень имеющихся ресурсов.

### **Шестое приближение**

В шестом приближении, с целью учета вопросов надежности для логических серверов могли быть заданы специальные ограничения для исключения размещения определенных логических серверов (виртуальных машин содержащих в себе серверы), дублирующих функции друг друга, на один и тот же компьютер. Для задачи о рюкзаке – это специальные ограничения, оговаривающие то, что определенные сочетания комплектов вещей недопустимы для размещения в рюкзаке. Такие ограничения задать несложно: например, если мы не хотим, чтобы в рюкзак одновременно попали 1-й и 5-й комплект, то задаем ограничение следующего вида:  $x_1 + x_5 \leq 1$ . Однако, для задания дополнительных ограничений в матричной форме, где строки – ограничения, столбцы – номера (индексы) комплектов, удобнее использовать булеву матрицу  $\{E_{d,j}\}$ , где  $E_{d,j}$  – элемент матрицы, равный “1”, если  $j$ -й комплект присутствует в  $d$ -м ограничении, “0” – в противном случае,  $d = 1..D$ . Тогда математическая модель принимает следующий вид:

$$\left\{ \begin{array}{l} \sum_{j=1}^N a_{ij} x_j \leq b_i - V_i, i = 1..M \\ \sum_{j=1}^N E_{dj} x_j \leq 1, d = 1..D \\ L = \frac{1}{\sum_{i=1}^M O_i} \left( \sum_{i=1}^M \frac{O_i}{b_i - V_i} \left( \sum_{j=1}^N a_{ij} x_j \right) \right) \rightarrow \max \\ \forall j \in [1, N]: x_j \in \{0,1\} \end{array} \right. \quad (3.1e)$$

### Седьмое приближение

Наконец, в седьмом приближении мы имеем дело не с одним, а с множеством рюкзаков. Задача оптимизации в данном приближении резко усложняется, поскольку практически она из простой задачи о рюкзаке превращается в задачу поиска оптимального распределения множества комплектов вещей на множество рюкзаков с множеством отсеков. Основная проблема здесь в выборе правильного подхода к достижению цели. Целью реорганизации является то, что за счет использования технологии виртуальных машин, мы пытаемся освободить некоторое множество компьютеров, причем идем к этой цели, добиваясь максимально эффективного использования ресурсов компьютеров. Однако очень важно подчеркнуть то, что можно сделать ошибку и пытаться добиваться эффективного использования ресурсов физических компьютеров всех сразу и тем самым решить задачу балансировки нагрузки. В таком случае нагрузка будет распределена так, чтобы ни один физический компьютер не простаивал, и не было так, чтобы один компьютер сильно загружен, а другой очень слабо. Этим мы лишь добьемся приблизительно равномерной загрузки ресурсов среди множества физических компьютеров, и ни один физический компьютер не будет освобожден. Истинная же цель – решить задачу, обратную задаче балансировки нагрузки, то есть сосредоточить логические серверы на некотором множестве компьютеров таким образом, чтобы задействованные компьютеры были загружены максимально, а все незадействованные компьютеры были свободны.

**Основная цель – минимизация числа задействованных компьютеров путем максимизации загрузки ресурсов не всех физических компьютеров, а только тех, которые окажутся задействованными в результате распределения. На всех незадействованных компьютерах загрузка ресурсов должна быть нулевой.**

Тогда, используя теперь уже обозначения не задачи о рюкзаке, а обозначения нашей основной задачи поиска оптимального распределения, представим следующую математическую модель (в 3-х частях):

1) Для любого  $k$ -го физического компьютера ( $k=1..NH$ ), для которого  $\forall i: R_{i,k} - V_i > 0$ , справедлива следующая система ограничений:

$$\left\{ \begin{array}{l} \sum_{j=1}^{NS} Q_{ij} x_{kj} \leq R_{ik} - V_i, i = 1..NC \\ \sum_{j=1}^{NS} E_{dj} x_{kj} \leq 1, d = 1..NX \\ \forall j \in [1, NS]: x_{kj} \in \{0,1\} \end{array} \right. \quad (3.1\text{ж})$$

где,

$NC$  – число типов ресурсов.

$NH$  – число физических компьютеров.

$NS$  – число логических серверов (общее число с учетом дополнительных логических серверов, которые были подготовлены для дублирования функций критичных по важности логических серверов)

$NX$  – число дополнительных ограничений (подмножество дублирующих друг друга логических серверов).

$R_{i,k}$  – базовый уровень  $i$ -го типа ресурса  $k$ -го физического компьютера.

$Q_{ij}$  – требование  $j$ -го логического сервера к  $i$ -му ресурсу.

$E_{dj}$  – элемент булевой матрицы, равный “1”, если  $j$ -й логический сервер присутствует в  $d$ -м ограничении, “0” – в противном случае.

$X_{kj}$  – элемент булевой матрицы, равный “1”, если  $j$ -й логический сервер (виртуальная машина с логическим сервером) размещается на  $k$ -м физическом компьютере, “0” – в противном случае.

Первая система ограничений – это ограничения по ресурсам, вторая – ограничения, исключающие одновременное размещение определенных логических серверов на один и тот же физический компьютер.

2) Для любого  $j$ -го, где  $j = 1..NS$ , логического сервера справедливо то, что он может быть размещен на одном и только одном физическом компьютере либо не размещен вообще:

$$\sum_{k=1}^{NH} x_{kj} \leq 1 \quad (3.1ж)$$

3) Целевая функция:

$$\left\{ \begin{aligned} L_{\gamma} &= \frac{1}{\sum_{i=1}^{NC} O_i} \left( \sum_{i=1}^{NC} \frac{O_i}{R_{i\gamma} - V_i} \left( \sum_{j=1}^{NS} Q_{ij}^x x_{\gamma j} \right) \right) \rightarrow \max \\ \forall \gamma &\in \{k^*\} \end{aligned} \right. \quad (3.1ж)$$

где,

$V_i$  – требование базовой ОС к  $i$ -му ресурсу

$O_i$  – элемент вектора маски, принимающий значение “1”, если соответствующему  $i$ -му типу ресурса предпочтительно повышение эффективности его использования, “0” – в противном случае.

$\{k^*\}$  – множество индексов физических компьютеров, которые окажутся задействованными после решения задачи оптимального распределения.

В данном приближении в математической модели фигурирует множество целевых функций, причем то, сколько именно их и каким именно физическим компьютерам они соответствуют неизвестно, это можно выяснить, лишь решив всю задачу целиком, чтобы получить множество  $\{k^*\}$ . Выделить одну глобальную целевую функцию практически невозможно:

- Можно было бы, например, поставить целью минимизацию числа задействованных компьютеров, однако, в общем случае компьютеры все различаются по уровням своих ресурсов. Более того, для различных задач один и тот же компьютер может иметь различную ценность. Наконец,

неоправданно большое количество задействованных компьютеров – это всегда следствие неэффективного использования ресурсов компьютеров.

- Можно было бы пытаться суммировать по каждому компьютеру общий объем простаивающих ресурсов и пытаться минимизировать эту сумму. Однако, здесь нам снова неизвестно сколько и какие именно компьютеры будут задействованы, а сумма простаивающих ресурсов по всем изначально заданным физическим компьютерам не зависит от распределения (при условии, конечно, что все логические серверы размещены), поскольку сумма требований по всем логическим серверам остается постоянной.

В такой ситуации, на первой взгляд, модель кажется неразрешимой из-за неопределенности в целевых функциях, однако, как будет показано ниже, выход из этой ситуации есть.

## 3.2. Поиск метода решения

В математической модели (3.1ж) основная проблема – необходимо повысить уровень загрузки ресурсов задействованных физических компьютеров, но заранее неизвестно какие именно компьютеры окажутся задействованными. Количество и индексы (номера) задействованных физических компьютеров можно получить, лишь решив целиком задачу. Получается неразрешимая ситуация, однако, выход из нее есть – для этого необходимо уйти от попытки охватывать в каждый момент времени решения все физические компьютеры и все логические серверы одновременно и попытаться разбить задачу на множество более простых подзадач.

Рассмотрим основные подходы для решения поставленной задачи.

### 3.2.1 Существующие методы решения поставленной задачи

#### Полный перебор

Самый простой способ – полный перебор. Объем перебора составляет:  $(NH + 1)^{NS}$  – любой логический сервер может быть размещен на одном из физических компьютеров либо не размещен вовсе. Причем каждый вариант

необходимо проверить по всем ограничениям и на оптимальность. Очевидно, что такой подход неприемлем, хотя, он дал бы точный оптимум. Тем не менее, при небольшом количестве логических серверов и компьютер применение полного перебора не исключено.

**Разбиение на подзадачи локальной оптимизации по каждому компьютеру и последовательное их решение без учета глобальных целей оптимизации.**

Разбиение задачи на множество более простых задач с последовательным приближением к решению: для каждого отдельного физического компьютера ставится локальная задача оптимизации, модель локальной задачи в таком случае это модель шестого приближения (3.1e) – это типовая задача булевого линейного программирования и для нее существуют как точные, так и приближенные методы решения. Причем при каждом “большом шаге” на очередной рассматриваемый физический компьютер делается попытка размещения оптимального сочетания логических серверов из множества доступных на этом шаге, после этого рассмотренный физический компьютер и распределенные на него логические серверы удаляются из дальнейшего рассмотрения. В таком случае, даже в худшем случае, когда ни один логический сервер не удастся разместить ни на одном физическом компьютере, задача решится за  $NH$  “больших шагов”, причем даже если на каждом шаге локальная задача будет решаться полным перебором всех сочетаний логических серверов, то общий объем перебора составит:  $NH * 2^{NS}$  – это уже существенно меньше чем при полном переборе для всей задачи, однако, очевидно то, что полученное решение не будет являться глобально оптимальным, хотя бы из-за того, что на каждом шаге мы “теряем из виду” все физические компьютеры, кроме рассматриваемого на этом шаге компьютера: на каждом шаге, вполне возможно, что для какого-нибудь из оставшихся компьютера удавалось бы находить более лучшее распределение логических серверов, чем для всех остальных оставшихся на этом шаге компьютеров.

### **Метод динамического программирования Беллмана**

Самый известный и популярный метод решения сложных задач оптимизации с разбиением на множество более простых с обеспечением глобальной оптимальности решения исходной задачи – это метод динамического программирования Беллмана [13, 14]. Рассмотрим суть метода.

Пусть рассматривается задача, распадающаяся на  $m$  шагов или этапов, например планирование инвестиций, управление производственными мощностями в течение длительного срока. Показатель эффективности задачи в целом обозначим через  $W$ , а показатели эффективности на отдельных шагах – через  $\varphi_i$ ,  $i=1..m$ . Если  $W$  обладает свойством аддитивности, т.е.  $W = \varphi_1 + \dots + \varphi_m$ , то можно найти оптимальное решение задачи методом динамического программирования. Таким образом, динамическое программирование – это метод оптимизации многошаговых или многоэтапных процессов, критерий эффективности которых обладает вышеуказанным свойством. В задачах динамического программирования критерий эффективности называется выигрышем. Данные процессы управляемые, и от правильного выбора управления зависит величина выигрыша.

**Определение 1.** Переменная  $x_i$ , от которой зависят выигрыши на  $i$ -м шаге и, следовательно, выигрыш в целом, называется шаговым управлением,  $i=1..m$ .

**Определение 2.** Управлением процесса в целом ( $x$ ) называется последовательность шаговых управлений  $x=(x_1, x_2, \dots, x_i, \dots, x_m)$ .

**Определение 3.** Оптимальное управление  $x^*$  – это значение управления  $x$ , при котором значение  $W(x^*)$  является максимальным (или минимальным, если требуется уменьшить проигрыш):  $W^* = W(x^*) = \max\{W(x)\}$  по всем  $x$ , принадлежащим области допустимых управлений.

Оптимальное управление  $x^*$  определяется последовательностью оптимальных шаговых управлений  $x^*=(x_1^*, x_2^*, \dots, x_i^*, \dots, x_m^*)$ .

В основе метода динамического программирования лежит принцип оптимальности Беллмана, формулирующийся следующим образом: **управление на каждом шаге надо выбрать так, чтобы оптимальной была сумма**

*выигрышей на всех оставшихся до конца процесса шагах, включая выигрыш на данном шаге.* Объясняется это правило так: при решении задачи динамического программирования на каждом шаге выбирается управление, которое должно привести к оптимальному выигрышу. Если считать все шаги независимыми друг от друга, то оптимальным шаговым управлением будет то управление, которое приносит максимальный выигрыш именно на данном шаге. Но, например, при покупке новой техники взамен устаревшей на ее приобретение затрачиваются определенные средства, поэтому прибыль от ее эксплуатации вначале может быть небольшой. Однако в следующие годы новая техника будет приносить большую прибыль и наоборот, если руководитель примет решение оставить старую технику для получения прибыли в текущем году, то в дальнейшем это приведет к значительным убыткам. Данный пример демонстрирует следующий факт: в многошаговых процессах все шаги зависят друг от друга, и, следовательно, управление на каждом конкретном шаге надо выбирать с учетом его будущих воздействий на этот процесс.

Другой момент, который следует учитывать при выборе управления на данном шаге – это возможные варианты окончания предыдущего шага. Эти варианты определяют состояние процесса. Например, при определении количества средств, вкладываемых в предприятие в  $i$ -м году, необходимо знать, какая прибыль получена в предыдущем  $(i-1)$ -м году. Таким образом, при выборе шагового управления необходимо учитывать:

- Возможные исходы предыдущего шага.
- Влияние управления на все оставшиеся до конца процесса шаги.

В задачах динамического программирования первый пункт учитывают, делая на каждом шаге условные предположения о возможных вариантах окончания предыдущего шага и приводя для каждого из вариантов условную оптимизацию. Выполнение второго пункта обеспечивается тем, что в задачах динамического программирования условная оптимизация проводится от конца процесса к началу. Сначала оптимизируется последний  $m$ -й шаг, на котором нет необходимости учитывать возможные воздействия выбранного управления



$x_m$  на все последующие шаги, так как эти шаги просто отсутствуют. Делая предположения об условиях окончания  $(m-1)$ -го шага, делая предположения об исходах окончания  $(m-2)$ -го шага, определяют условное оптимальное управление на  $(m-1)$ -м шаге, приносящее оптимальный выигрыш на двух последних шагах –  $(m-1)$ -м и  $m$ -м. Так же действуют на всех остальных шагах до первого. На первом шаге, как правило, нет необходимости делать условных предположений, т.к. состояние системы перед первым шагом обычно известно. Для этого состояния выбирают оптимальное шаговое управление, обеспечивающее оптимальный выигрыш на первом и всех последующих шагах. Это управление является безусловным оптимальным управлением на первом шаге и, зная его, определяются оптимальное значение выигрыша и безусловные оптимальные управления на всех шагах.

Казалось бы, метод Беллмана наиболее подходит для решения нашей оптимизационной задачи и дает возможность получения глобально-оптимального распределения, но есть серьезная проблема: число шагов (подзадач) неизвестно, оно может колебаться от 1 до  $NH$ . Глобальная целевая функция не может быть сформулирована из-за неизвестного числа физических компьютеров, которые окажутся задействованными. Если в качестве глобальной целевой функции взять, например, сумму загрузки (интегральные загрузки по всем ресурсам) по всем физическим компьютерам и максимизировать ее, то придем к решению обратной задачи – задачи балансировки нагрузки между всеми исходными физическими компьютерами. Поэтому, при всех достоинствах метода Беллмана, он не может быть применен в чистом виде. Соответственно, остается выбрать некоторый компромиссный вариант с использованием аппарата динамического программирования.

### 3.2.2 Предлагаемый вариант решения с разбиением на подзадачи

Поскольку в результате обзора методов решения задачи остается только выбор между полным перебором и простым разбиением на подзадачи оптимизации по каждому физическому компьютеру в отдельности, то приемлем только второй подход. Однако, мы его можем модифицировать таким образом, чтобы на каждом шаге выбирать не просто следующий по порядку (или произвольный) физический компьютер, а рассматривать по очереди все оставшиеся, и выбирать наилучше подходящий компьютер.

Предлагается следующий подход: изначально задано исходное множество физических компьютеров и множество логических серверов, но в общем случае в процессе решения эти множества будут изменяться. При каждом очередном “большом шаге” мы будем пытаться распределить некоторое “наилучшее сочетание логических серверов” на “наилучше подходящем” физическом компьютере – компьютер, у которого достигается наибольшая средняя загрузка по всем ресурсам среди остальных компьютеров при оптимальном распределении на него логических серверов. В случае успеха после этого мы будем исключать из множества физических компьютеров “наилучше подошедший компьютер”, а из множества логических серверов – “распределенное наилучшее сочетание логических серверов”. Так будет продолжаться до тех пор, пока либо множество физических компьютеров или множество логических серверов не станет пустым, либо не наступит ситуация, когда не удастся распределить ни один из оставшихся логических серверов ни на один из оставшихся физических компьютеров.

Конечно, при использовании вышеуказанного подхода мы далеко не всегда будем находить глобальный оптимум. Тем не менее, мы разбиваем сложную задачу (с множеством физических компьютеров) на множество более простых задач (с одним физическим компьютером) и в то же время на каждом шаге “не теряем из виду” ни один из всех оставшихся физических компьютеров. Объем перебора при таком подходе в худшем случае (когда на каждом “большом шаге” распределяется только один логический сервер,

причем  $NS \geq NH$ ) составит:  $NH * 2^{NS} + (NH - 1) * 2^{NS-1} + \dots + 1 * 2^{NS - (NH - 1)} = (2^{NH} + NH - 1) * 2^{NS-1}$ , при достаточно больших  $NH$  это  $\sim NH * 2^{NS-1}$ , в случае если подзадачи решаются полным перебором. Такой объем перебора существенно меньше, чем при полном переборе при решении исходной задачи –  $(NH + 1)^{NS}$  и немного больше, чем при последовательном рассмотрении подзадач оптимизации по отдельным физическим компьютерам без поиска на каждом шаге наилучшего среди них –  $NH * 2^{NS}$ .

Рассмотрим теперь, как образуется математическая модель подзадачи и как решается исходная задача в целом по “большим шагам”.

Пусть,  $T$  – это порядковый номер “большого шага”,  $T \geq 0$ .

Пусть,  $\{K(T)\}$  – множество индексов физических компьютеров, оставшихся к моменту шага  $T$ , причем  $\{K(0)\} = \{1, \dots, NH\}$ .

Пусть,  $\{J(T)\}$  – множество индексов логических серверов, оставшихся к моменту шага  $T$ , причем  $\{J(0)\} = \{1, \dots, NS\}$ .

Тогда, математическая модель подзадачи на шаге  $T$  при рассмотрении  $k$ -го физического компьютера,  $k \in \{K(T)\}$ , для которого  $\forall i: R_{i,k} - V_i > 0$ , представляется следующим образом:

$$\left\{ \begin{array}{l} \sum_{j \in \{J(T)\}} Q_{ij} x_{kj} \leq R_{ik} - V_i, i = 1..NC \\ \sum_{j \in \{J(T)\}} E_{dj} x_{kj} \leq 1, d = 1..NX \\ L(T, k) = \frac{1}{\sum_{i=1}^{NC} O_i} \left( \sum_{i=1}^{NC} \frac{O_i}{R_{ik} - V_i} \left( \sum_{j \in \{J(T)\}} Q_{ij} x_{kj} \right) \right) \rightarrow \max \\ \forall j \in \{J(T)\} : x_{kj} \in \{0, 1\} \end{array} \right. \quad (3.2)$$

где,  $L(T, k)$  – целевая функция на шаге  $T$  при решении подзадачи для  $k$ -го физического компьютера.

На каждом шаге  $T$  последовательно рассматриваются все физические компьютеры с индексами  $k \in \{K(T)\}$ , оставшиеся к моменту шага  $T$ , и выбирается тот, для которого в результате решения соответствующей подзадачи, достигается наивысшее значение целевой функции среди значений,

полученных при решении подзадач для компьютеров с индексами  $k \in \{K(T)\}$ . Соответственно, наилучший физический компьютер с индексом  $k^*$  выбирается из следующего условия:

$$\begin{cases} L_{k^*}(T) = \max_{k \in \{K(T)\}} \{L_{\max}(T, k)\} \\ L_{k^*}(T) > 0 \end{cases} \quad (3.3)$$

где,  $L_{\max}(T, k)$  – оптимальное значение целевой функции на шаге  $T$  при решении подзадачи для  $k$ -го физического компьютера.

$L_{k^*}(T)$  – наивысшее оптимальное значение целевой функции на шаге  $T$  среди значений  $L_{\max}(T, k)$  для всех  $k \in \{K(T)\}$ .

В условии (3.3) особенно важно условие того, что  $L_{k^*}(T)$  не должна быть нулевой, это гарантирует, что хотя бы один логический сервер распределится и хотя бы один физический компьютер будет задействован. Если же  $L_{k^*}(T) = 0$ , то это означает, что дальнейшее распределение логических серверов невозможно и решение исходной задачи должно быть прекращено.

Если же  $k^*$  успешно найден из условия (3.3), то выполняются следующие преобразования: из множества оставшихся физических компьютеров исключается компьютер с индексом  $k^*$ , а из множества логических серверов – множество серверов, распределенных на этот компьютер:

$$\begin{aligned} \{K(T+1)\} &= \{K(T)\} \setminus k^* \\ \{J(T+1)\} &= \{J(T)\} \setminus \{j^*\} \end{aligned} \quad (3.4)$$

где,  $\{j^*\}$  – множество индексов логических серверов, которые были распределены на  $k^*$ -й физический компьютер.

Если в результате преобразования (3.4), множество  $K(T+1)$  или  $J(T+1)$  окажется пустым, то решения задачи завершается, в противном случае переход к шагу  $T+1$ .

Результатом решения задачи является распределение  $\{X_{kj}\}$ ,  $k=1 \dots N_H$ ,  $j=1 \dots N_S$ , логических серверов по физическим компьютерам. На рис. 3.1 представлен алгоритм решения задачи в целом (по “большим шагам”).

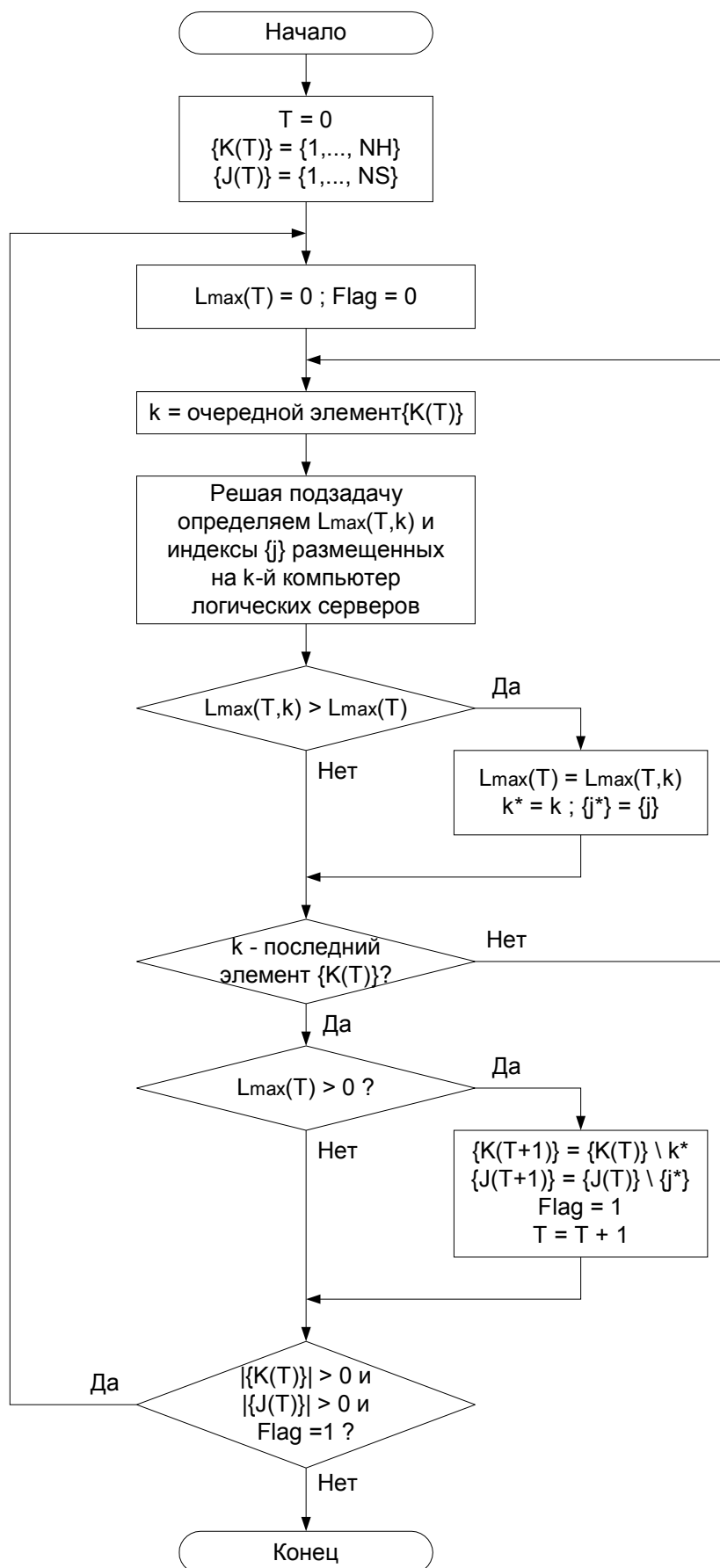


Рис. 3.1. Схема алгоритма решения задачи в целом

Таким образом, используя компромиссный подход, дающий схему решения всей задачи в целом (без детализации подзадач), нам удалось свести неразрешимую стационарную математическую модель (3.1ж) с неопределенными целевыми функциями к некоторому множеству (число подзадач заранее неизвестно) вполне разрешимых математических моделей (3.2). Число “больших шагов”, в худшем случае (когда множество физических компьютеров опустошается раньше или вместе с множеством логических серверов) составит:  $NH$ , а общее число решаемых подзадач оптимизации по отдельным физическим компьютерам:  $0.5 * NH * (NH + 1)$ . Так что, на глобальном уровне оптимизации сложность компромиссного метода полиномиальная ( $\sim n^2$ ). Однако, в алгоритме (рис. 3.1) каждая подзадача (мат. модель 3.2), в свою очередь, представляет собой класс задач условной псевдобулевой оптимизации [15], для которой также необходим эффективный способ решения. Подзадачу на шаге  $T$  при заданном  $k \in K(T)$  несложно представить в следующем виде:

$$\left\{ \begin{array}{l} \sum_{j=1}^N a_{ij} x_j \leq b_i, i=1..M \\ L = \sum_{j=1}^N c_j x_j \rightarrow \max \\ \forall j \in [1, N]: x_j \in \{0, 1\} \end{array} \right. \quad (3.5)$$

где,

$M = NC + NX$ ,  $N = |\{J(T)\}|$ .  $T, k$  – заданы,  $k \in K(T)$ .

Для всех  $j = 1 \dots N$  и  $q = j$ -й элемент из  $\{J(T)\}$ :

- Для всех  $i = 1 \dots NC$ :  $a_{i,j} = Q_{i,q}$ ;  $b_i = R_{i,k} - V_i$
- Для всех  $i = NC + 1 \dots NC + NX$ :  $a_{i,j} = E_{i-NC,q}$ ;  $b_i = 1$

$$c_j = \frac{1}{\sum_{i=1}^{NC} O_i} \left( \sum_{i=1}^{NC} \frac{O_i Q_{iq}}{R_{ik} - V_i} \right)$$

Следует отметить следующие особенности задачи: элементы матрицы ограничений  $a_{ij}$  неотрицательны, поскольку требования  $\{Q_{ij}\}$  логических серверов неотрицательные величины по определению. Правые части  $b_i$  – должны быть неотрицательными, поскольку если для какого-либо ограничения значение  $b_i = R_{i,k} - V_i$  окажется меньше либо равно нулю, то задача не будет иметь решения по определению – это случай явной нехватки ресурсов (базовые уровни ресурсов  $\{R_{i,k}\}$  физических компьютеров и требования  $\{V_i\}$  базовой ОС также неотрицательны по определению). Соответственно, в случае выявления нехватки ресурсов задача не должна решаться.

Коэффициенты  $c_j$ , очевидно, также являются неотрицательными величинами, поскольку  $O_i$  – булевы переменные, а  $R_{i,k} - V_i > 0$ , поскольку в противном случае задача не решается.

Математическая модель (3.5) представляет собой класс задач условной псевдобулевой оптимизации: в ней присутствуют линейные неравенства с булевыми переменными, причем сами коэффициенты при них и правые части неравенств в общем случае являются вещественными числами (по этой причине класс задач называют псевдобулевой, а не чисто булевой). Также задана целевая функция с булевыми переменными и с вещественными коэффициентами, которую требуется максимизировать: найти такое булево решение, при котором выполняются все ограничения и целевая функция достигает наивысшего значения при заданных ограничениях. Существует также и другое название вышеуказанного класса – класс задач булевого линейного программирования.

Для решения задач псевдобулевой оптимизации (булевого линейного программирования) на сегодняшний день используются приближенные методы решения [36, 37, 38, 39, 40]. Такие методы позволяют решить задачи больших размерностей с линейными ограничениями любой сложности, причем за приемлемое время, однако, как правило, дают приближенное (субоптимальное) решение. Кратко рассмотрим суть приближенных методов.

### 3.2.3. Существующие приближенные методы решения подзадач

Введем некоторые обозначения, определения и утверждения.

Пусть задано  $N$ -мерное булево пространство, состоящее из множества точек  $X=(x_1, \dots, x_N)$  с булевыми координатами:  $\forall j \in [1, N]: x_j \in \{0, 1\}$ .

Определение 1.  $D(X_A, X_B)$  будем называть расстоянием между точками  $X_A$  и  $X_B$ , используя метрику Хэмминга:  $D(X_A, X_B) = \sum_{j=1}^N |x_j^A - x_j^B|$ .

Утверждение 1. Расстояние между двумя точками  $X_A$  и  $X_B$  численно равно числу координат, по которым эти точки отличаются.

По определению 1 расстояние вычисляется как  $D(X_A, X_B) = \sum_{j=1}^N |x_j^A - x_j^B|$ .

Очевидно то, что поскольку каждая координата может принимать значение либо “0” либо “1”, то модуль разности для различающихся координат всегда будет равен 1, а для совпадающих координат – 0. Тогда сумма модулей разности будет равна количеству различающихся координат. Следовательно, расстояние между точками определяется числом различающихся координат.

Определение 2.  $\Theta_r(X_0)$  будем называть  $N$ -мерной сферой поиска с радиусом  $r$  и центром  $X_0$ , состоящей из множества точек, отличающихся от точки  $X_0$  ровно  $r$  координатами. Иными словами  $\forall X \in \Theta_r(X_0) \rightarrow D(X, X_0) = r$ .

Определение 3.  $Z_r(X_0)$  будем называть зоной поиска с радиусом  $r$  и центром  $X_0$ , состоящей из всех сфер с радиусами от 0 до  $r$ . Иными словами  $\forall X \in Z_r(X_0) \rightarrow \exists q \in [0, r]: X \in \Theta_q(X_0)$ , т.е.  $Z_r(X_0) = \Theta_0(X_0) \cup \dots \cup \Theta_r(X_0)$ .

Утверждение 2. Точка  $X^*$  принадлежит зоне поиска  $Z_r(X_0)$  тогда и только и тогда, когда расстояние  $D(X^*, X_0) \leq r$ .

Зона  $Z_r(X_0)$  представляет собой совокупность сфер поиска с радиусами от 0 до  $r$ . Соответственно, все точки всех сфер отличаются от центра  $X_0$  зоны поиска не более чем  $r$  координатами. Тогда в силу утверждения 1 расстояние от любой точки, находящейся в зоне  $Z_r(X_0)$ , до центра  $X_0$  не превышает  $r$ .

Следствие.  $\forall X^*: D(X^*, X_0) > r \rightarrow X^* \notin Z_r(X_0)$ .



Утверждение 3. Сфера  $\Theta_r(X_0)$  радиусом  $r$  содержит  $C_N^r$  различных точек.

Действительно, число точек, отличающихся ровно  $r$  координатами от точки  $X_0$ , определяется числом различных сочетаний  $r$  координат по всем  $N$  координатам. Так, например, для  $r=2$  и  $N=4$ , это сочетания:  $\{1,2\}$ ,  $\{1,3\}$ ,  $\{1,4\}$ ,  $\{2,3\}$ ,  $\{2,4\}$  и  $\{3,4\}$  – всего  $4!/(2!(4-2)!) = 6$  сочетаний.

Утверждение 4. Для рассмотрения всех точек  $N$ -мерного булевого пространства достаточно рассмотреть зону поиска радиусом  $N$ .

Зона поиска  $Z_N(X_0)$  радиусом  $N$  по определению 3 это совокупность всех сфер поиска  $\Theta_r(X_0)$  с радиусами  $r = 0 \dots N$ . В соответствии с утверждением 3 общее число точек в каждой сфере радиусом  $r$  составляет  $C_N^r$ . Тогда общее число точек во всех сферах зоны поиска составит  $C_N^0 + C_N^1 + \dots + C_N^N = 2^N$ . С другой стороны, общее число всевозможных точек в  $N$ -мерном булевом пространстве также  $2^N$  (для каждой из  $N$  координат возможно одно из двух значений: либо “0” либо “1”). Следовательно, для рассмотрения всех точек  $N$ -мерного булевого пространства достаточно рассмотреть зону  $Z_N(X_0)$ .

Утверждение 5. Две зоны поиска  $Z_{r1}(X_A)$  и  $Z_{r2}(X_B)$  пересекаются (имеют хотя бы одну общую точку) тогда и только тогда, когда  $D(X_A, X_B) \leq r1 + r2$ .

Если зоны  $Z_{r1}(X_A)$  и  $Z_{r2}(X_B)$  пересекаются, то мы всегда можем найти такую точку  $X^*$ , которая будет одновременно находиться и в 1-й и во 2-й зоне. Но тогда, в соответствии с утверждением 2, для такой точки должны выполняться следующие условия:  $D(X^*, X_A) \leq r1$  и  $D(X^*, X_B) \leq r2$ . Тогда, легко заметить то, что  $D(X^*, X_A) + D(X^*, X_B) \leq r1 + r2$ . Точки  $X^*$ ,  $X_A$  и  $X_B$  либо образуют треугольник, а в нем, как известно, любая из сторон имеет меньшую длину нежели, чем сумма двух других сторон, либо все три точки лежат на одной прямой. Тогда следует то, что длина  $D(X_A, X_B)$  не может быть больше, чем сумма длин  $D(X^*, X_A)$  и  $D(X^*, X_B)$ , следовательно,  $D(X_A, X_B) \leq D(X^*, X_A) + D(X^*, X_B) \leq r1 + r2$ .

**Гриди-методы.** Гриди-методы являются интуитивными эвристиками, в которых на каждом шаге принимается решение о том, являющееся наиболее выгодным в данный момент, без учета того, что происходит на последующих шагах поиска. Гриди-методы [23, 24] предназначены для решения задач псевдодобулевой максимизации с одним ограничением (задача о рюкзаке, мат. модель 3.1а). Причем коэффициенты целевой функции ( $c_j$ ), и ограничений ( $a_j$  и  $b$ ) – неотрицательны, соответственно, точка  $X^0=(0,0,\dots,0)$  – всегда допустима, а точка  $X^1=(1,1,\dots,1)$  – в силу ограничения является недопустимой.

Кратко опишем их суть:

**Метод 1.** Вокруг начальной точки  $X^0=(0,0,\dots,0)$  просматриваются точки сферы  $\Theta_1(X^0)$  и эти точки проверяются на условие выполнения ограничения, среди допустимых точек выбирается такая точка  $X^*$ , для которой отношение  $\frac{c_1 x_1^* + \dots + c_N x_N^*}{a_1 x_1^* + \dots + a_N x_N^*}$  – максимально. Далее просматриваются точки сферы  $\Theta_1(X^*)$  и т.д. Если на каком-то шаге для текущего оптимума  $X^*$  в сфере  $\Theta_1(X^*)$  нет ни одной допустимой точки, то  $X^*$  – решение.

**Метод 2.** Во втором методе начальной точкой является недопустимая точка  $X^1=(1,1,\dots,1)$ , просматриваются точки сферы  $\Theta_1(X^1)$  и эти точки проверяются на условие выполнения ограничений, если условие не выполняется, то среди них находится такая точка  $X^*$ , для которой отношение  $\frac{c_1 x_1^* + \dots + c_N x_N^*}{a_1 x_1^* + \dots + a_N x_N^*}$  – максимально. Далее просматриваются точки сферы  $\Theta_1(X^*)$  и т.д. Если на каком-то шаге для текущего оптимума  $X^*$  в сфере найдены допустимые точки, то среди них выбираем точку  $X^{**}$  для которой значение целевой функции максимально, и она принимается за решение.

Объем перебора в обоих вариантах гриди-метода  $\sim N^2$ .

Поскольку гриди-методы рассчитаны только на одно ограничение, то для нашего случая (мат. модель 3.5) они неприемлемы.

**Метод случайного поиска.** Суть метода [23, 24] заключается в том, что от некоторой начальной точки делаются шаги в случайном направлении на некоторое заданное расстояние. Если в новой точке значение целевой функции улучшается, то она принимается за текущий оптимум и от нее снова ведется случайный поиск. Если в новой точке значение целевой функции хуже, то возвращаемся к старой точке и продолжаем вести от нее случайный поиск. Критерием останова метода может служить то, что, допустим, от последней оптимальной точки некоторое конечное число попыток случайного поиска не привели к улучшению целевой функции. Также может быть задано ограничение на максимальный объем перебора как дополнительное исходное данное задачи либо аналогично может быть задано ограничение на время поиска.

Достоинство метода заключается в том, что в отличие от метода локального поиска, который будет рассмотрен ниже, случайный поиск “не притягивается безвозвратно” к локальному экстремуму и легко может уйти к другим экстремумам, что делает его очень полезным в многоэкстремальных случаях. Однако, следует помнить то, что это фактически метод проб и ошибок: он перебирает небольшое количество случайных точек, однако, никогда не просматривает окрестность с точками вокруг текущей. В случае условной оптимизации, метод может попадать на точки, не удовлетворяющие ограничениям и отбрасывать их, а рядом с ними, возможно, могли быть точки, удовлетворяющие ограничениям, причем с более лучшими значениями целевой функции в них. Поэтому метод случайного поиска чаще используют не как самостоятельный метод, а совместно с другими методами, поскольку он может дать неплохие стартовые точки.

**Метод локального поиска.** Метод локального поиска [23, 24, 25] является одним из наиболее распространенных и простых для программной реализации приближенных методов дискретной оптимизации. В первом приближении рассмотрим этот метод для задач безусловной оптимизации.

Суть метода локального поиска заключается в следующем: выбирается некоторая начальная точка  $X^0$ , вычисляется в ней целевая функция. Начальная

точка принимается за текущий оптимум  $X_{\text{опт}}$ . Далее ведется перебор всех точек  $X \in Z_1(X_{\text{опт}})$  и в них вычисляется значение целевой функции (в классическом варианте метода, радиус зон поиска не задается, он полагается равным 1, и на каждом шаге просматриваются точки, отличающиеся не более одной координатой от точки  $X_{\text{опт}}$ ). В процессе перебора возможны две стратегии:

- Просмотр всей зоны  $Z_1(X_{\text{опт}})$  целиком и нахождение наилучшей точки  $X^*$ , в которой значение целевой функции лучше, чем во всех остальных точках зоны (включая центр зоны – точку  $X_{\text{опт}}$ ). Эта точка принимается за новый текущий оптимум и далее ведется уже поиск вокруг нее.
- Перебор до первого улучшения, когда в ходе перебора, как только находится первая точка, в которой значение целевой функции улучшилось, то эта точка принимается за новый текущий оптимум и далее ведется поиск вокруг нее.

Для обеих стратегий работа алгоритма завершается, когда в зоне поиска нельзя найти ни одну точку, в которой значение целевой функции лучше.

Очевидно то, что при использовании второй стратегии задача решится быстрее, поскольку в этом случае перебирается меньшее количество точек, однако, по этой же причине результат в случае многоэкстремальной целевой функции может быть хуже, чем при использовании первой стратегии.

Метод локального поиска достаточно быстро находит решение: число шагов при классическом варианте (радиус поиска жестко задан и равен 1) не более  $N + 1$  шагов приближения к оптимуму – это тот случай, например, когда, начальная точка  $(0, 0 \dots 0)$ , а оптимальная:  $(1, 1 \dots 1)$ , после каждого шага будет меняться только одна координата. На каждом шаге просматривается не более  $N + 1$  точек (включая центр поиска). Соответственно, объем перебора  $\leq (N+1)^2$ .

В случае условной оптимизации применяют, так называемые, штрафные функции, которые выбираются таким образом, чтобы при нарушении любого из ограничений, значение целевой функции за счет штрафной функции корректировалось таким образом, чтобы оно оказывалась всегда наихудшим.

Например, для задачи условной псевдобулевой максимизации скорректированную целевую функцию можно сформировать так:

$$F = L - P * \sum_{i=1}^M \max\{0, \sum_{j=1}^N a_{ij} x_j - b_i\} \quad (3.6)$$

где,

$P$  – некоторое большое положительное число.

$F$  – скорректированная целевая функция.

В формуле (3.6) легко видеть то, что при нарушении любого из ограничений штраф, вычитаемый из целевой функции, принимает положительное значение, если же все условия по ограничениям выполняются то, значение штрафной функции равно нулю. Важно выбрать величину  $P$  достаточно большой для того, чтобы даже при достаточно больших значениях целевой функции, при любых нарушениях ограничений штраф мог “подавить” выигрыш, в противном случае в качестве оптимума может быть выбрана недопустимая точка.

Очевидным недостатком метода локального поиска является то, что в случае многоэкстремальной целевой функции, есть немалая вероятность получения в качестве решения не глобального оптимума, а одного из первого попавшегося субоптимального решения. При радиусе поиска равного 1, практически невозможны случаи, когда методу после “попадания в зону притяжения” одного из локальных оптимумов удастся выйти из этой зоны и найти другой оптимум. Однако, этот недостаток существенно ослабляется при использовании радиусов зон поиска  $\gg 1$ , разумеется, в этом случае также существенно увеличивается перебор, так что здесь приходится выбирать между количеством точек перебора и качеством решения. Кроме того, для получения эффективных решений при многоэкстремальных случаях, используют также, так называемый, локальный поиск с мультистартом, когда решение проводится для нескольких случайных стартовых точек и выбирается наилучшее. Использование даже нескольких случайно взятых стартовых точек значительно ослабляет влияние проблемы “попадания в зону притяжения” локальных экстремумов. Для решения задачи (3.5) мы будем ориентироваться именно на метод локального поиска с мультистартом с использованием функции штрафов.

### 3.2.4. Предлагаемый вариант локального поиска с использованием мультистарта, функции штрафов и управляемого радиуса зон поиска.

Для использования локального поиска в качестве выбранного метода решения, преобразуем задачу условной псевдобулевой оптимизации (мат. модель 3.5) в следующий вид:

$$\begin{cases} F = \sum_{j=1}^N c_j x_j - P * \sum_{i=1}^M \max\{0, \sum_{j=1}^N a_{ij} x_j - b_i\} \rightarrow \max \\ \forall j \in [1, N]: x_j \in \{0, 1\} \end{cases} \quad (3.7)$$

В преобразованной математической модели (3.7) ограничения явным образом отсутствуют, но они учитываются при помощи штрафной функции. При рассмотрении задачи (3.5) было отмечено, что для этой задачи элементы  $a_{ij}$ ,  $b_i$  и  $c_j$  – неотрицательные величины. Соответственно, скорректированная целевая функция ( $F$ ) в этом случае для всех допустимых точек всегда неотрицательна (штраф в таких случаях равен нулю). В случае же недопустимых точек  $F$  может быть как неотрицательной, так и отрицательной величиной – это зависит от выбора коэффициента  $P$ . Во избежание ошибок при решении задач, коэффициент  $P$  лучше всего выбрать настолько большим, чтобы для любой недопустимой точки скорректированная целевая функция ( $F$ ) была отрицательной. После этого задача (3.7) может быть решена при помощи метода локального поиска.

Для расширения возможностей базового метода локального поиска с целью получения качественных решений мы применим следующие подходы:

- Задача решается не один, а множество раз для различных стартовых точек, причем вовсе необязательно допустимых: чем больше различных стартовых точек и чем более качественный генератор случайных чисел, тем больше шансов обнаружить глобальный оптимум. Число стартовых точек задается как дополнительное исходное данное задачи, обозначим его **MS**. Соответственно, в результате решения задачи MS число раз получаем различные оптимумы, сравниваем значения целевой функции и выбираем наилучший оптимум.

- Радиус зон поиска не равен жестко единице, а может быть задан как дополнительное исходное данное задачи. Обозначим его **MR**. Соответственно, для заданной случайной стартовой точки, задача решается методом локального поиска, причем параметр MR задает радиус зон  $Z_r(X)$ , в которых ведется поиск более лучших точек. Как было упомянуто ранее, чем больше радиус зон, тем больше шансов выйти на глобальный оптимум. При  $MR = N$  глобальный оптимум находится однозначно, однако, в таком случае локальный поиск фактически превращается в полный перебор.

В итоге, локальный поиск становится **обобщенным** и многоуровневым: на самом верхнем уровне решается задача при MS различных стартовых точек, на следующем уровне внутри задачи при конкретной стартовой точке идет переход от одной зоны поиска к другой до тех пор, пока внутри какой-то зоны невозможно будет найти более лучшую точку, на следующем уровне внутри конкретной зоны ведется перебор сфер поиска с радиусами от 0 до MR, наконец, на самом нижнем уровне ведется перебор точек внутри конкретной сферы поиска.

Следует особо отметить то, что при решении задачи при переходе от одной зоны поиска к другой зоны всегда частично перекрываются. Это очевидно, поскольку, согласно методу поиска, новая локальная оптимальная точка найденная в текущей зоне становится центром для следующей, а расстояние между центром текущей зоны и центром будущей зоны в лучшем случае равна MR (когда новый центр находится на краю текущей зоны), в худшем случае равна 1. В таком случае, при просмотре следующей зоны, в любом случае, часть точек из предыдущей зоны будут совершенно бесполезным образом заново “просмотрены”. На качество результата это никак не повлияет (нет никакого толка от повторного просмотра проверенных точек), а объем перебора существенно может возрасти, особенно при значениях параметра MR близких к N (но не строго равных N). Для такой ситуации существует следующий способ обхода лишнего перебора: при просмотре точек

в новой зоне достаточно проверять просматриваемые точки на принадлежность предыдущей зоны. Для этого перед переходом в новую зону поиска сохраняются координаты центра старой зоны. Далее для каждой просматриваемой точки новой зоны вычисляется расстояние до центра старой зоны и если оно меньше  $MR$ , то она сразу же отбрасывается. Следует заметить то, что вычисление расстояния – это всего лишь подсчет простой числа различающихся координат, в то время как повторный просмотр точки – это вычисление целевой функции с вещественными коэффициентами, вычисление штрафа, сравнение значений целевой функции и т.п. Очевидно, что первая операция куда более простая и быстрая, нежели чем вторая. Поэтому применение способа обхода лишнего перебора крайне желательно, причем даже при  $MR = 1$ . Математически этот способ можно выразить так: допустим  $X_T$  – текущий оптимум, он же и центр текущей зоны  $Z_{MR}(X_T)$ , допустим в этой зоне найден более лучшая точка  $X^*$ , тогда текущий центр становится старым  $X_{CT} = X_T$ , а  $X^*$  принимается за новый текущий центр  $X_T = X^*$  новой зоны поиска  $Z_{MR}(X_T)$ . Тогда, при дальнейшем поиске в новой зоне мы просматриваем не  $\forall X \in Z_{MR}(X_T)$ , а  $\forall X \in Z_{MR}(X_T) \setminus Z_{MR}(X_{CT})$  – множество всех точек новой зоны за исключением точек старой зоны. Таким образом, мы избавимся от лишнего перебора.

На рисунке 3.2 представлен алгоритм обобщенного локального поиска с использованием заданного количества случайных стартовых точек и заданного радиуса зон поиска, а также с учетом вышерассмотренного подхода к исключению повторного просмотра точек.

Данный алгоритм реализует блок поиска оптимального распределения логических серверов на  $k$ -й физический компьютер алгоритма решения задачи в целом (рис. 3.1) на текущем шаге  $T$  – т.е. фактически получаем индексы  $\{j\}$  распределенных логических серверов и значение целевой функции  $L_{\max}(T, k)$ .



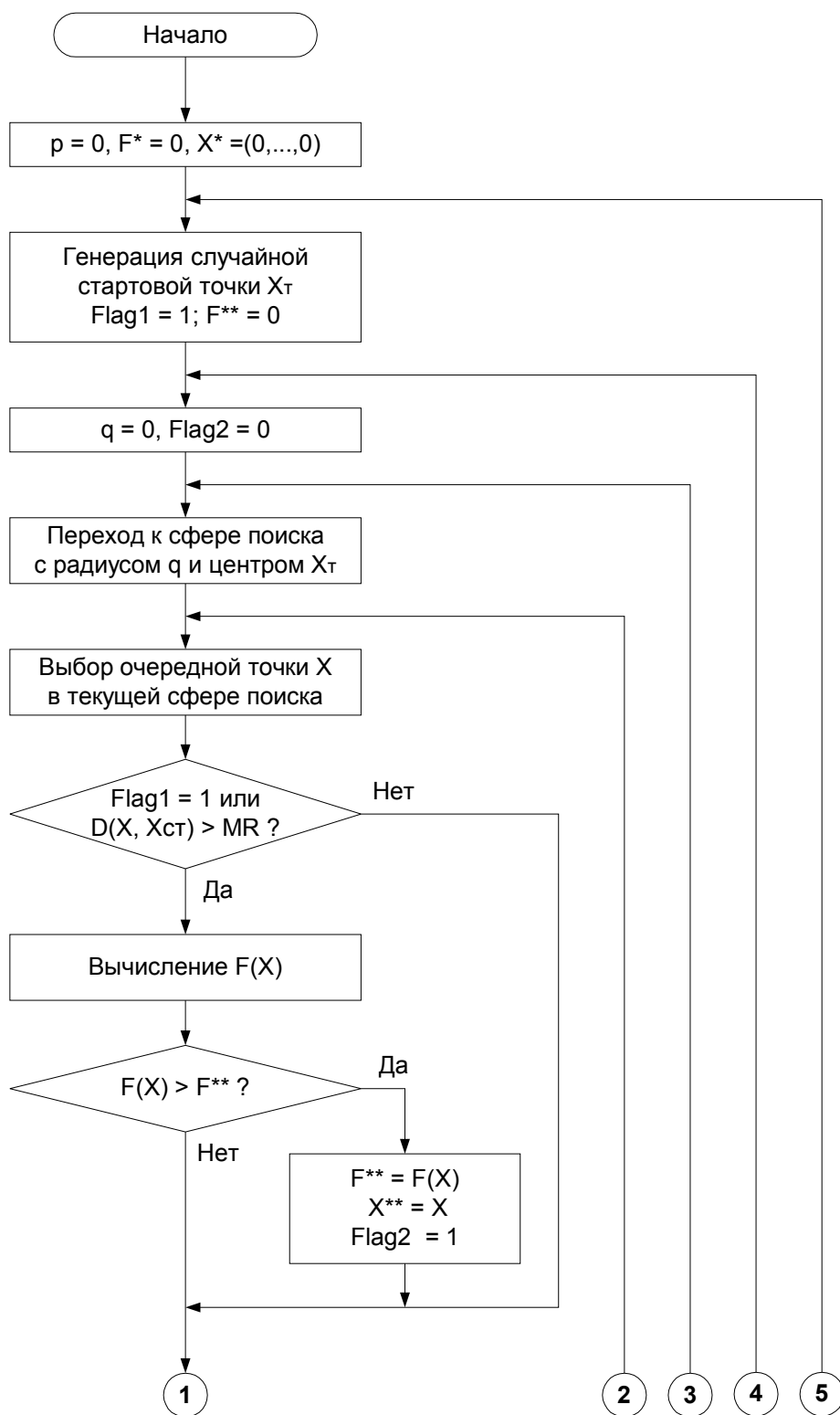


Рис. 3.2. Схема алгоритма обобщенного локального поиска

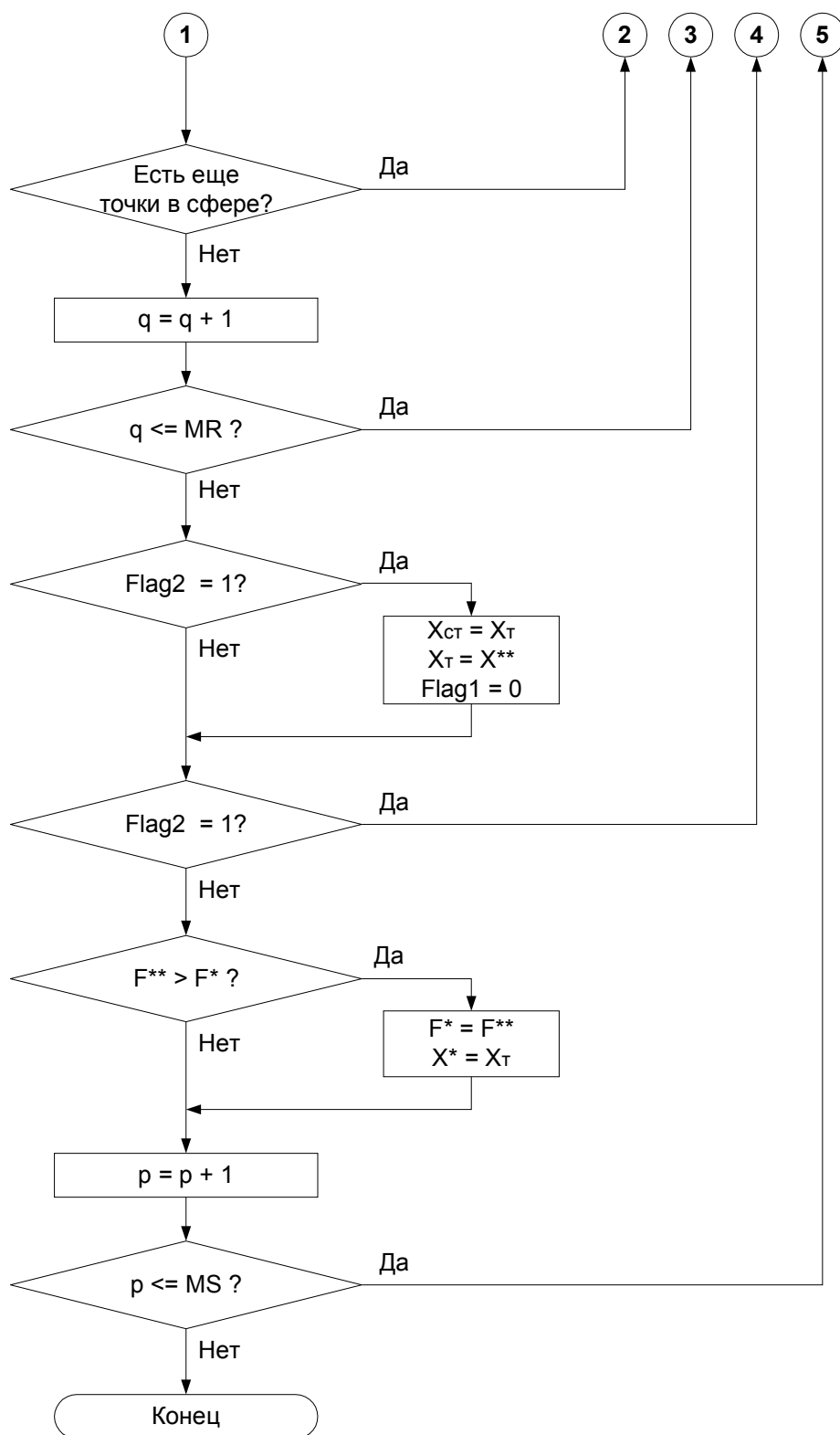


Рис. 3.2 (продолжение). Схема алгоритма обобщенного локального поиска

В алгоритме (рис. 3.2) используются следующие основные и вспомогательные обозначения:

$p$  – порядковый номер стартовой точки.

$q$  – радиус текущей просматриваемой сферы поиска в текущей зоне.

$X_T$  – центр текущей зоны поиска (текущий оптимум).

$X_{CT}$  – центр предыдущей зоны поиска (предыдущий оптимум).

$X^{**}$  – текущая наилучшая точка в текущей зоне поиска. При переходе в новую зону поиска  $X_T$  совпадает с последней наилучшей точкой  $X^{**}$  в предыдущей зоне. Поэтому, когда поиск заканчивается (в последней рассматриваемой зоне не нашлось более лучших точек), последней наилучшей точкой является  $X_T$ .

$F^{**}$  – значение целевой функции в точке  $X^{**}$ .

$X^*$  – текущая наилучшая точка среди тех, которые были получены при решении задачи при различных стартовых точках – фактически это решение всей задачи.

$F^*$  – значение целевой функции в точке  $X^*$ .

Flag1 – признак того, что просматривается первая зона поиска, соответственно,  $X_{CT}$  еще не существует, поэтому в зоне поиска просматриваются все без исключения точки.

Flag2 – признак того, что в зоне поиска была найдена лучшая точка.

Таким образом, используя алгоритмы 3.1 и 3.2, мы можем решить задачу поиска и получить матрицу распределения  $\{X_{k,j}\}$ ,  $k=1\dots NH$ ,  $j=1\dots NS$ , логических серверов по компьютерам. Тогда, используя матрицу распределения  $\{X_{k,j}\}$  несложно также вычислить матрицу загрузки ресурсов компьютеров  $\{\eta_{k,i}\}$ ,  $k=1\dots NH$ ,  $i=1\dots NC$ , для оценки загрузки ресурсов:

$$\eta_{ki} = \begin{cases} \sum_{j=1}^{NS} \frac{X_{kj} Q_{ij}}{R_{ik} - V_i}, & \text{если } R_{ik} - V_i > 0 \\ 0, & \text{в противном случае} \end{cases} \quad (3.8)$$

$\forall i \in [1, NC], \forall k \in [1..NH]:$

### 3.3. Анализ характеристик предложенных методов.

Предложенные алгоритмы решения задачи поиска распределения в целом и решения подзадач условной псевдобулевой оптимизации достаточно непросты, а результат решения, довольно сложным образом зависит от исходных данных. Тем не менее, попытаемся провести хотя бы базовый анализ предложенных алгоритмов с точки зрения оценки таких характеристик и показателей как: объем перебора, качество решений и сходимость алгоритма.

#### **Сходимость**

Сходимость метода решения задачи поиска распределения достаточно легко показать. На уровне решения задачи в целом (алгоритм 3.1) возможны только три конечные ситуации: множество логических серверов опустошается, множество компьютеров опустошается, ни один из оставшихся логических серверов не размещается ни на один из оставшихся компьютеров. На уровне решения подзадачи псевдобулевой оптимизации (алгоритм 3.2) сходимость определяется тем, что сам метод локального поиска обладает сходимостью. Введенные усложнения метода локального поиска – множество стартовых точек (MS) и управляемый радиус зон поиска (MR) – влияют на объем перебора, но не делают его бесконечным. Это будет показано ниже.

#### **Оценка объема перебора**

На самом верхнем уровне алгоритма 3.2 обобщенного локального поиска подзадача псевдобулевой оптимизации решается MS раз для различных стартовых точек, генерируемых случайным образом. На следующем уровне алгоритм 3.2 находит решение не более чем за  $N + 1$  шагов в худшем случае. Рассмотрим следующий пример: пусть ограничения заданы таким образом, что даже решение  $X^* = (1, 1, \dots, 1)$  является допустимым, и оно является глобальным оптимумом, при этом пусть радиус зоны поиска ограничен  $MR = 1$ , а стартовая точка –  $(0, 0, \dots, 0)$ . Тогда, за каждый шаг улучшение будет происходить не более чем по одной по координате, поскольку в зоне радиусом 1 присутствуют только

точки, отличающиеся не более одной координатой от точки центра. Соответственно, чтобы все “0” улучшились до всех “1” потребуется  $N$  шагов, а на шаге  $N + 1$  будет просматриваться последняя зона с центром  $(1, 1, \dots, 1)$ , после чего этот центр будет принят как решение подзадачи. На последних двух нижних уровнях алгоритма 3.2 ведется перебор точек внутри зон радиусом  $MR$ , причем в первой зоне поиска всегда просматриваются все точки без исключения, в последующих же только те, которые не принадлежат предыдущей зоне. Рассмотрим для начала частный случай  $MR = 1$ . В этом случае каждая зона поиска состоит из  $C_N^0 + C_N^1 = N + 1$  точек. Первая зона просматривается целиком, каждая следующая содержит 2 точки, принадлежащие предыдущей зоне – центр текущей зоны и центр предыдущей зоны. Действительно, пусть, например, центр предыдущей зоны был  $(0, 0, \dots, 0)$ , а центр текущей зоны  $(0, 0, \dots, 1)$ . Предыдущая зона состоит из точки  $(0, 0, \dots, 0)$  и  $N$  точек:  $(0, 0, \dots, 1) - (1, 0, \dots, 0)$ . Текущая зона состоит из точки  $(0, 0, \dots, 1)$ , из  $N - 1$  точек:  $(0, \dots, 1, 1) - (1, 0, \dots, 1)$  и точки  $(0, 0, \dots, 0)$ . Легко видеть, что только  $N - 1$  точек не принадлежат предыдущей зоне (имеют по две единицы в координатах, в предыдущей зоне таких точек нет). Соответственно, в первой зоне будет просмотрено  $N + 1$  точек, во всех последующих по  $N - 1$  точек.

Тогда общий объем перебора для алгоритма 3.2 в худшем случае (завышенная оценка перебора) при  $MR = 1$  составит:  $MS * (1 * (N + 1) + N * (N - 1)) = MS * (N^2 + 1)$ .

В случае если радиус для зон поиска  $MR > 1$ , то оценить объем перебора довольно трудно, хотя при некоторых допущениях и приближениях это можно сделать. Основная трудность состоит в том, что центр каждой следующей зоны поиска может находиться на расстоянии  $D \in [1, MR]$  от центра предыдущей зоны, соответственно, сложно оценить число перекрываемых точек. Однако, мы хотя бы оценим число точек, рассматриваемых в первой зоне при решении подзадачи. В первой зоне просматривается  $C_N^0 + C_N^1 + \dots + C_N^{MR}$  точек. На каждом следующем шаге будет просматриваться меньшее число точек за счет

перекрытия зон поиска. Число шагов (даже для худшего случая, когда стартовая точка  $(0,0,\dots,0)$ , а допустимый оптимум  $-(1,1,\dots,1)$ ) при этом может колебаться от  $\lceil N / MR \rceil + 1$  до  $N + 1$ , поскольку на каждом шаге центр новой зоны поиска может отстоять от предыдущего центра на расстоянии вплоть до  $MR$ , соответственно, приближение к оптимуму будет происходить быстрее. В предельном же случае при радиусе поиска  $MR \rightarrow N$ , сумма  $C_N^0 + C_N^1 + \dots + C_N^{MR} \rightarrow 2^N$  – это полный перебор и гарантирует получение глобального оптимума для подзадачи, при этом подзадача решается фактически за один шаг, поскольку на втором шаге все точки второй зоны поиска будут принадлежать первой зоне, и не будут повторно просматриваться. Тогда при  $MR = N$  объем перебора для алгоритма 3.2 составит  $MS * 2^N$ .

Попробуем теперь оценить объем перебора при произвольном радиусе зон поиска  $MR$ . В первой зоне  $Z_{MR}(X_0)$ , необходимо просматривать все точки без исключения, их число (обозначим как  $W_0$ ), легко вычисляется по формуле:

$$W_0(MR, N) = \sum_{p=0}^{MR} C_N^p \quad (3.9)$$

При  $MR \ll N$ ,  $W_0(MR, N) \sim N^{MR} / MR!$ , при  $MR = N$ ,  $W_0(MR, N) = 2^N$ .

В каждой следующей зоне за счет перекрытия зон, очевидно, объем перебора будет меньшим. Пусть  $Z_{MR}(X_{CT})$  – предыдущая зона поиска,  $Z_{MR}(X_T)$  – текущая зона поиска, причем  $D(X_{CT}, X_T) \leq MR$ . Тогда, очевидно, новыми (непроверенными) точками будут являться точки:  $\forall X \in Z_{MR}(X_T) \setminus Z_{MR}(X_{CT})$ . Аналитическим путем была выведена точная формула (в рамках диссертации не будем рассматривать описание вывода формулы и экспериментов по проверке ее достоверности) для подсчета числа новых точек, или иными словами, мощности множества  $Z_{MR}(X_T) \setminus Z_{MR}(X_{CT})$ , при условии  $D(X_{CT}, X_T) \leq MR$  и  $MR \leq N$  (расстояние  $D(X_{CT}, X_T)$  в формуле обозначено буквой  $D$ ):

$$\sum_{p=MR-D+1}^{MR} \left( \sum_{q=0}^{\lceil (D+p-MR)/2 \rceil - 1} \left( C_D^q * C_{N-D}^{p-q} \right) \right) \quad (3.10)$$

$MR \leq N, D \leq MR$

Мы будем ориентироваться на худший случай, когда в каждой следующей зоне новый локальный оптимум находится на самом краю зоны, поскольку в этом случае перекрытие новой зоны поиска и старой будет наименьшим, а число точек, которое необходимо просматривать в новой зоне – наибольшим, иными словами будем считать, что  $D(X_{CT}, X_T) = MR$ . Тогда, число новых точек (обозначим как  $W_1$ ) можно будет вычислить по формуле:

$$W_1(MR, N) = \sum_{p=1}^{MR} \left( \sum_{q=0}^{\lceil p/2 \rceil - 1} \left( C_{MR}^q * C_{N-MR}^{p-q} \right) \right) \quad (3.11)$$

$MR \leq N$

При  $MR = N$ ,  $W_1(MR, N) = 0$ .

Тогда, поскольку мы исходим из того, что центр новой зоны находится на расстоянии  $MR$  от старой, то, как было показано ранее, алгоритм 3.2 должен прийти к решению не более чем за  $1 + \lceil N / MR \rceil$  шагов. Например, в худшем случае если стартовая точка – это точка  $(0, 0 \dots, 0)$ , а решение – это точка  $(1, 1 \dots, 1)$ , то на каждом шаге будут “улучшаться”  $MR$  координат, а значит за  $1 + \lceil N / MR \rceil$  шагов “улучшатся” все координаты. Причем на первом шаге рассматривается целиком вся первая зона поиска, а на следующих  $\lceil N / MR \rceil$  шагах – точки следующих зон, не принадлежащих предыдущим зонам. Тогда общий объем перебора алгоритма 3.2 в худшем случае составит:

$$MS * \left( \left\lceil \frac{N}{MR} \right\rceil * W_1(MR, N) + W_0(MR, N) \right) \quad (3.12)$$

Теперь попытаемся получить общую оценку объема перебора алгоритма 3.1 поиска распределения. Для простоты рассмотрим худший случай, когда на каждом большом шаге на физический компьютер распределяется только один логический сервер – это необходимый минимум для того, чтобы алгоритм поиска не завершился раньше времени и при этом для каждого следующего “большого шага” будет оставаться максимально возможное количество логических серверов. В таком случае число “больших шагов” будет равно  $\min(NH, NS)$ : алгоритм завершается, когда  $|\{K(T)\}| = 0$  или  $|\{J(T)\}| = 0$ . Тогда окончательная оценка общего объема перебора для худшего случая:

$$\sum_{T=0}^{\lambda-1} \left( (NH - T) * MS * \left( \left\lceil \frac{(NS - T)}{\psi} \right\rceil * W_1(\psi, NS - T) + W_0(\psi, NS - T) \right) \right) \quad (3.13)$$

$$\psi = \min(NS - T, MR), \lambda = \min(NH, NS)$$

Получить сумму ряда в общем виде не представляется возможным. Тем не менее, можно приблизительно оценить зависимость объема перебора от числа компьютеров: при  $NS \gg NH$  (соответственно,  $\lambda = NH$  и  $NS - T \sim NS$ ) объем перебора в худшем случае будет  $\sim 0.5 * NH * (NH + 1)$ . Далее, достаточно очевидно, что объем перебора  $\sim MS$ . Далее, если положить  $NH = 1$ , то  $\lambda = 1$ ,  $T = 0$ ,  $NS - T = NS$  и  $\psi = MR$  (т.к.  $MR \leq NS$ ). Тогда зависимость объема перебора от числа серверов и радиуса зон поиска: при  $MR = NS$ , объем перебора  $\sim 2^{NS}$  (т.к.  $W_1(NS, NS) = 0$ ,  $W_0(NS, NS) = 2^{NS}$ ,  $\lceil NS / MR \rceil = 1$ ). При  $MR \ll NS$ , для упрощения считая  $W_1(MR, NS) \approx W_0(MR, NS)$ , причем  $W_0(MR, NS) \sim NS^{MR} / MR!$ , объем перебора  $\sim ((1 + \lceil NS / MR \rceil) * (NS^{MR} / MR!)) \sim (NS^{MR+1} / (MR * MR!))$ .

Для наглядности приведем несколько графиков зависимостей общего объема перебора для алгоритма распределения в зависимости (рис. 3.3, 3.4, 3.5, 3.6) от числа логических серверов и числа компьютеров при заданном числе стартовых точек и радиусе зоны поиска для подзадач локального поиска.

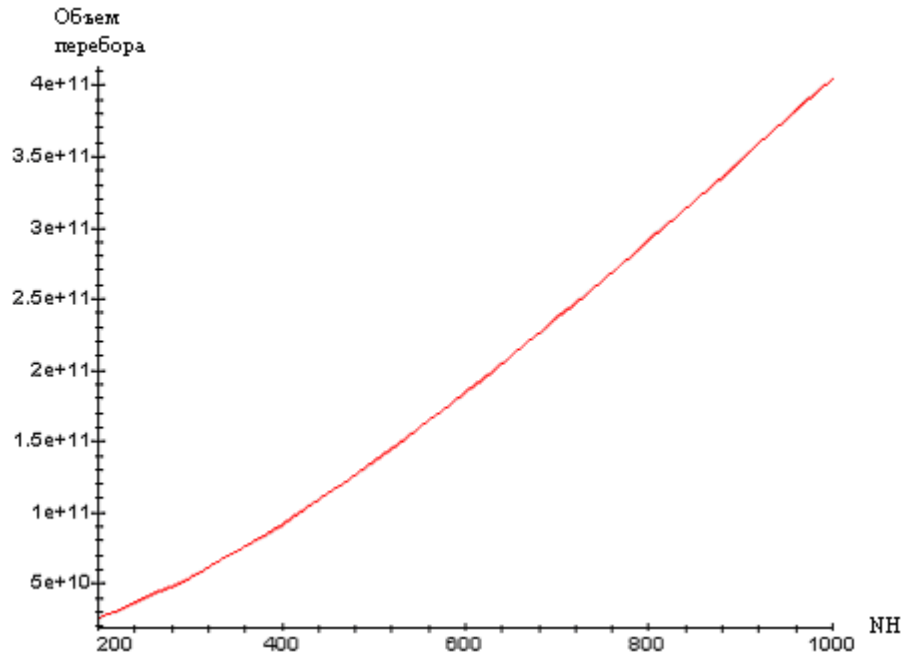


Рис 3.3. График зависимости общего объема перебора от числа компьютеров при  $MR = 1$ ,  $MS = 1$  и при 1200 логических серверах.



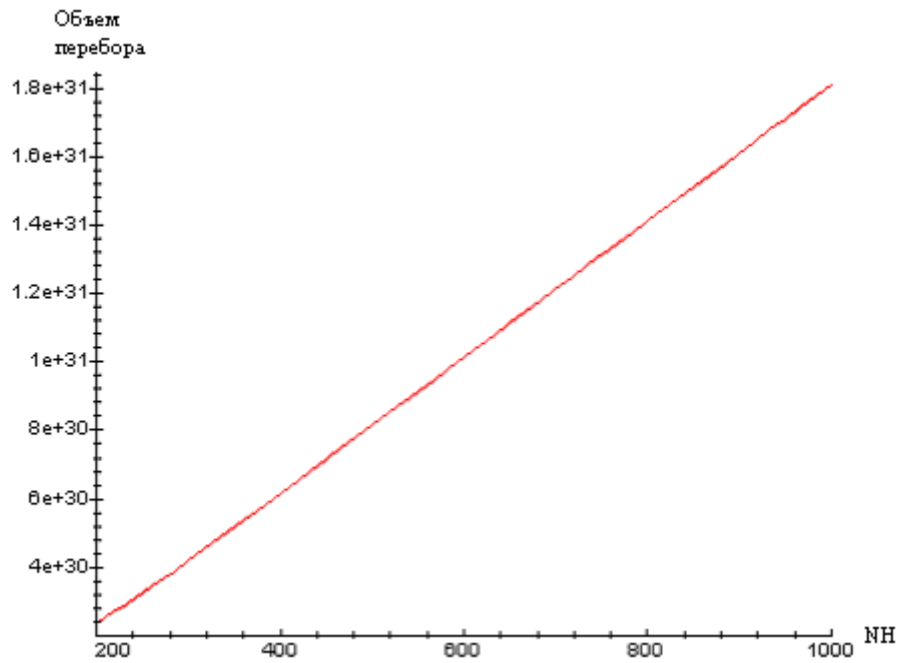


Рис 3.4. График зависимости общего объема перебора от числа компьютеров при  $MR = 10$ ,  $MS = 1$  и при 1200 логических серверов.

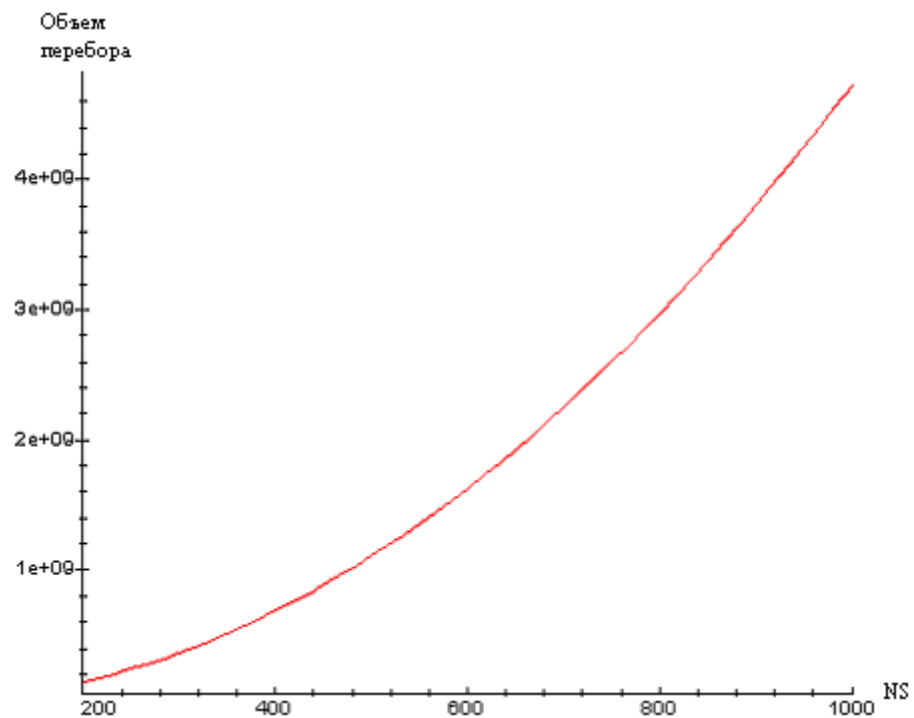


Рис 3.5. График зависимости общего объема перебора от числа логических серверов при  $MR = 1$ ,  $MS = 1$  и при 100 компьютерах.

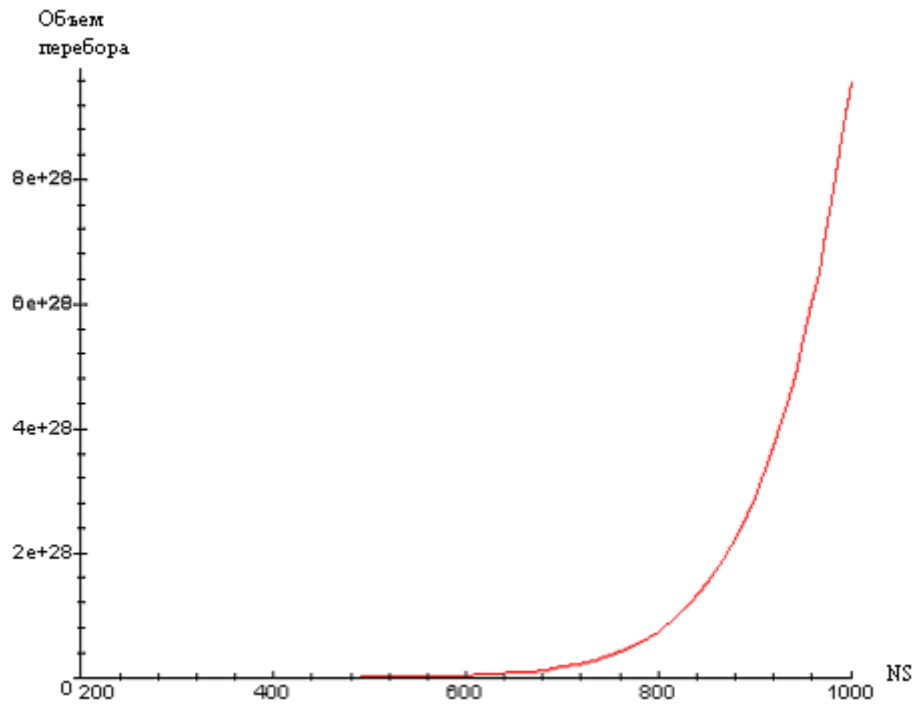


Рис 3.6. График зависимости общего объема перебора от числа логических серверов при  $MR = 10$ ,  $MS = 1$  и при 100 компьютерах.

Из графиков 3.3 и 3.4 видно то, что объем перебора  $\sim$  квадратично зависит от числа физических компьютеров независимо от параметра  $MR$  – радиуса зоны поиска. Что же касается графиков 3.5 и 3.6, то из них хорошо видно то, что зависимость объема перебора от числа логических серверов сильно зависит от параметра  $MR$ : скорость нарастания графика определяется этим параметром. Кроме этого следует особо отметить то, что даже при очень большом количестве логических серверов и физических компьютеров порядок объема перебора в основном определяется радиусом зон поиска ( $MR$ ).

Как частный случай мы можем рассмотреть графики зависимостей объема перебора от числа логических серверов при одном единственном физическом компьютере (рис. 3.7 и рис. 3.8). В этом случае решение задачи распределения сводится к решению одной единственной подзадачи.

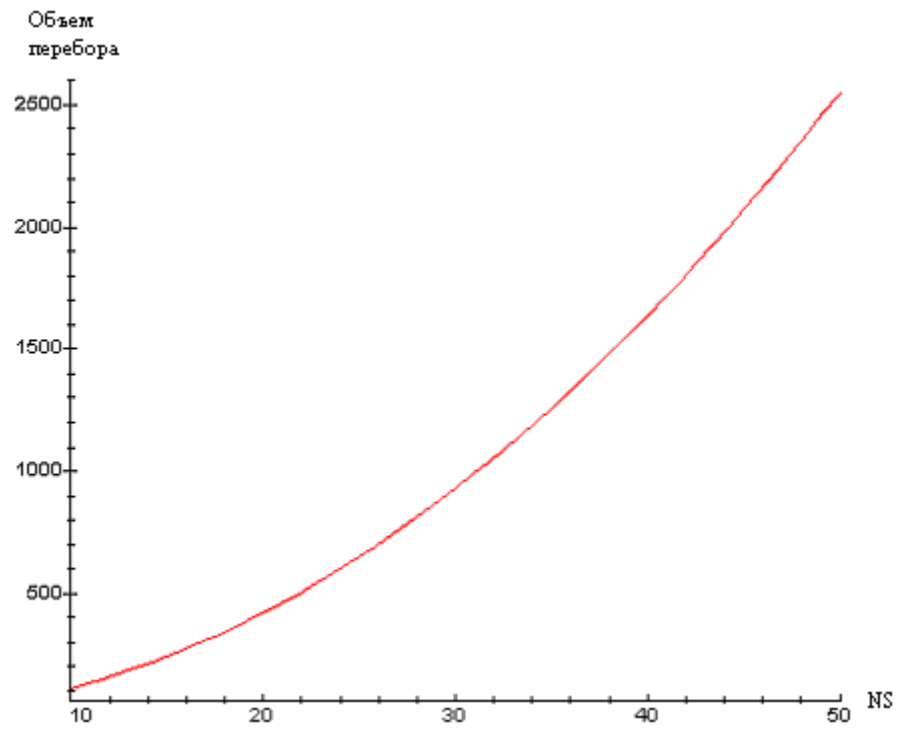


Рис 3.7. График зависимости общего объема перебора от числа логических серверов при  $MR = 1$ ,  $MS = 1$  и при 1 компьютере.

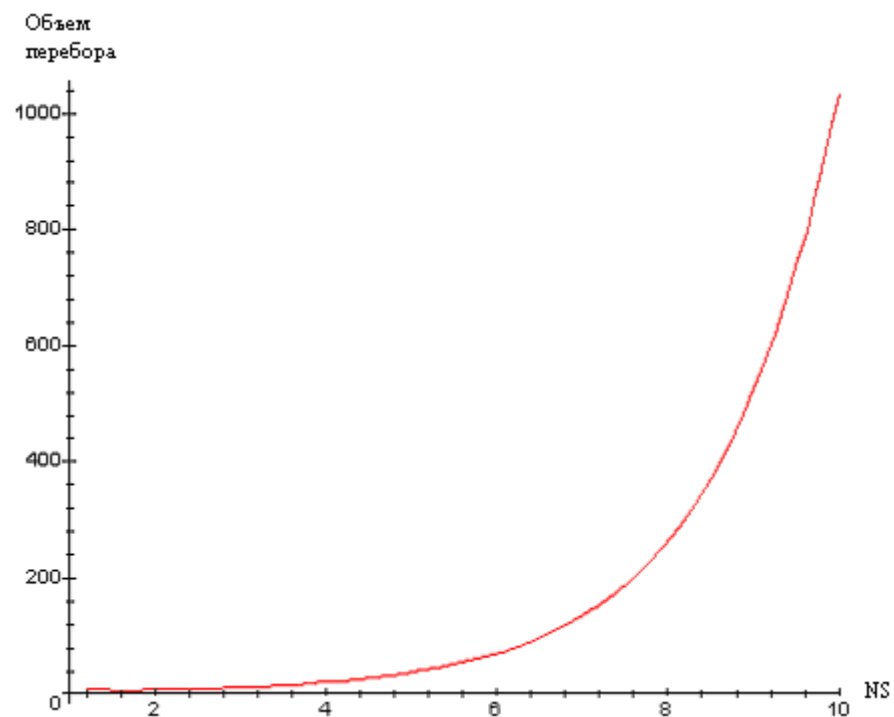


Рис 3.8. График зависимости общего объема перебора от числа логических серверов при  $MS = 1$ , при 1 компьютере, а  $MR$  равно числу логических серверов (радиус зон поиска всегда максимальный).

Из графика 3.7 хорошо видно то, что зависимость объема перебора от числа логических серверов  $\sim NS^2$  и это не противоречит оценке объема перебора ( $MS * (N^2 + 1)$ ) алгоритма локального поиска при  $MR = 1$  и при  $MS = 1$ . Из графика 3.8 хорошо видно то, что зависимость объема перебора от числа логических серверов  $\sim 2^{NS}$  и это не противоречит оценке объема перебора ( $MS * 2^{NS}$ ) алгоритма локального поиска при  $MR = NS$ .

Наконец, приведем график (рис. 3.9) зависимости объема перебора от радиуса зоны поиска при небольшом числе логических серверов и одном физическом компьютере, чтобы выявить характер влияния параметра  $MR$  на объем перебора.

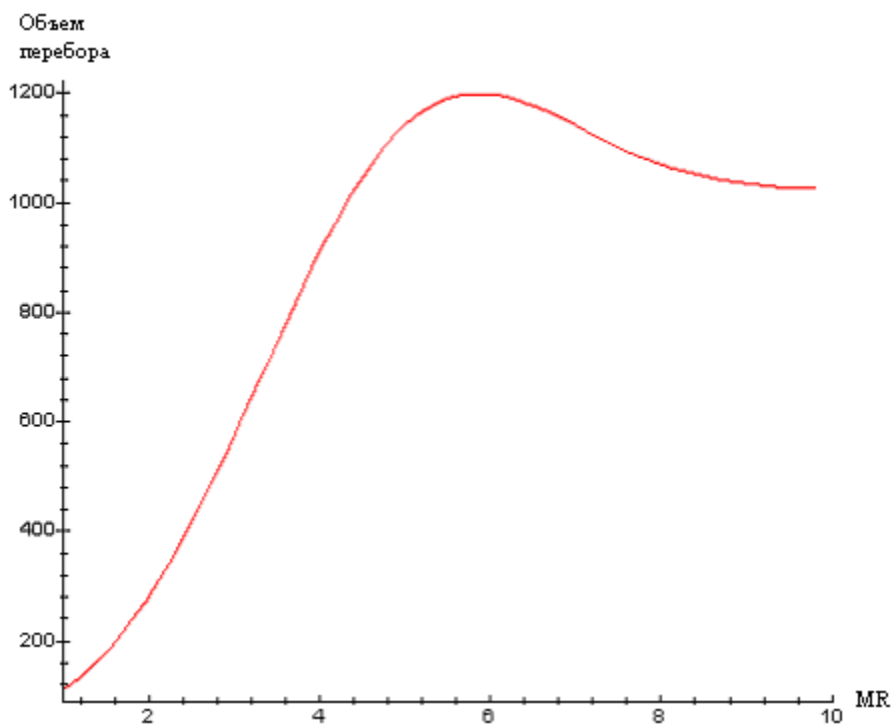


Рис 3.9. График зависимости общего объема перебора от параметра  $MR$  при 1 компьютере 10 логических серверах и  $MS = 1$ .

Как видно из графика 3.9 объем перебора, как следовало ожидать, сначала возрастает, но затем снижается — это легко объясняется: при приближении  $MR$  к числу логических серверов зоны поиска сильно перекрываются и, кроме того, алгоритм локального поиска сходится быстрее.

### Оценка качества решений

Метод решения задачи поиска распределения в целом, а также и используемый в нем метод решения подзадач условной псевдобулевой оптимизации, достаточно сложны и результаты решения крайне непростым образом зависят от всех исходных данных и в первом приближении получить какую-либо оценку качества решений достаточно трудно.

Тем не менее, также как при поиске оценки объема перебора, мы попытаемся начать с простейших случаев исходных данных и попытаемся с помощью некоторых простых представительных примеров выявить хотя бы зависимость качества решений подзадач условной псевдобулевой оптимизации от ключевого параметра  $MR$  – радиуса зон поиска. Представительные примеры выбираются таким образом, чтобы выделить и подчеркнуть недостатки предложенного обобщенного варианта метода локального поиска в плане субоптимальности получаемых решений.

Мы будем рассматривать примеры с одним компьютером и единственным типом ресурса, по которому и будет проводиться оптимизация. Тогда можно ввести понятие *точности решения* как отношение загрузки оптимизируемого ресурса компьютера в субоптимальном распределении, полученном при некотором заданном радиусе зон поиска, к загрузке ресурса компьютера в глобальном оптимуме. Точность будем обозначать буквой  $\xi$ .

Пример 1.1. Рассмотрим простейший пример: пусть задан всего один тип ресурсов ( $NC = 1$ ), один компьютер  $H_1$  ( $NH = 1$ ) и множество логических серверов  $\{S_1, \dots, S_{NS}\}$ . Базовый уровень ресурса компьютера пусть будет равен  $R$  (причем  $R \gg 1$ ). Требование 1-го логического сервера  $S_1$  равно  $\lfloor R / 2 \rfloor + 2$  (чуть больше половины базового уровня ресурса компьютера), а требования остальных  $NS - 1$  серверов  $\{S_2, \dots, S_{NS}\}$  равны  $\lfloor R / 2 \rfloor$ . Для простоты требования базовой ОС будут нулевыми, дополнительные ограничения - исключения отсутствуют ( $NX = 0$ ), число стартовых точек равно одному ( $MS = 1$ ). Оптимизация будет проводиться по единственному типу ресурсов.

Пусть сначала радиус зон поиска  $MR = 1$ .

При решении этой задачи алгоритм локального поиска начнет с точки  $(0,0,\dots,0)$  и просмотрит зону поиска радиусом 1 – это точки:  $(1,0,\dots,0),\dots, (0,0,\dots,0,1)$  и выберет точку  $(1,0,\dots,0)$ , так как в ней загрузка ресурса достигает наибольшего значения равного  $(\lfloor R / 2 \rfloor + 2) / R$ . Соответственно, точка  $(1,0,\dots,0)$  будет центром новой зоны поиска. На следующем шаге будут просмотрены новые точки:  $(1,1,0,\dots,0),\dots, (1,0,\dots,0,1)$  и все они окажутся недопустимыми в силу ограничений по ресурсам. Таким образом, точка  $(1,0,\dots,0)$  будет принята за решение задачи. Загрузка ресурса компьютера при этом составит  $(\lfloor R / 2 \rfloor + 2) / R$  (очевидно, что для любого  $R \gg 1$ , загрузка будет чуть более 50%).

Нетрудно заметить, что решение  $(0,1,1,0,\dots,0)$  – является глобальным оптимумом. При этом загрузка ресурса компьютера составляет  $2 * (\lfloor R / 2 \rfloor) / R$  (очевидно, что для любого  $R \gg 1$ , загрузка будет около 100,00%). Тогда точность субоптимального решения составит  $(\lfloor R / 2 \rfloor + 2) / (2 * \lfloor R / 2 \rfloor)$ . Если допустить, что требования логических серверов могут принимать нецелые значения, то для любого  $R \gg 1$  точность будет равна  $1 / 2$ .

Если же радиус зон поиска  $MR > 1$ , то алгоритм локального поиска гарантированно выходит на глобальный оптимум. Например, при радиусе зон поиска  $MR = 2$ , на первом шаге алгоритм рассмотрит точки  $(1,0,\dots,0,0),\dots,(0,\dots,0,1,1)$  и выберет точку  $(0,1,1,0,\dots,0)$  как наилучшую. На следующем шаге алгоритм в новой зоне с центром  $(0,1,1,0,\dots,0)$  просмотрит точки  $(1,1,1,0,\dots,0),\dots, (0,1,1,0,\dots,0,1,1)$ , и все они будут недопустимыми в силу ограничений по ресурсам. Тогда за решение будет принята точка  $(0,1,1,0,\dots,0)$ .

Пример 1.2. Рассмотрим теперь пример, аналогичный предыдущему, за исключением того, что требование 1-го логического сервера  $S_1$  равно  $2 * (\lfloor R / 3 \rfloor) + 3$  (чуть больше двух третей базового уровня ресурса компьютера), а требования остальных  $NS - 1$  серверов  $\{S_2,\dots,S_{NS}\}$  равны  $\lfloor R / 3 \rfloor$ .

Пусть сначала радиус зон поиска  $MR = 2$ .

При решении этой задачи алгоритм локального поиска начнет с точки  $(0,0,\dots,0)$  и просмотрит зону поиска радиусом 1 – это точки  $(1,0,\dots,0,0),\dots, (0,\dots,0,1,1)$ , и выберет точку  $(1,0,\dots,0)$ , так как в ней загрузка ресурса достигает

наибольшего значения равного  $2 * (\lfloor R / 3 \rfloor + 3) / R$ . Далее приняв точку  $(1, 0, \dots, 0)$  за центр новой зоны алгоритм рассмотрит точки  $(1, 1, 0, \dots, 0), \dots, (1, 0, \dots, 0, 1, 1)$ , но все они окажутся недопустимыми в силу ограничений по ресурсам и точка  $(1, 0, \dots, 0)$  будет принята за решение задачи. Загрузка ресурса компьютера при этом составит  $2 * (\lfloor R / 3 \rfloor + 3) / R$  (очевидно, что для любого  $R \gg 1$ , загрузка будет чуть более 66,66%).

Нетрудно заметить, что глобальным оптимумом является решение  $(0, 1, 1, 1, 0, \dots, 0)$ . При этом загрузка ресурса компьютера составляет  $3 * (\lfloor R / 3 \rfloor) / R$  (очевидно, что для любого  $R \gg 1$ , загрузка будет около 100,00%). Тогда, точность субоптимального решения составит  $(2 * (\lfloor R / 3 \rfloor + 3)) / (3 * (\lfloor R / 3 \rfloor))$ . Если допустить, что требования логических серверов могут принимать нецелые значения, то для любого  $R \gg 1$  точность будет равна  $2 / 3$ .

Если же радиус зон поиска  $MR > 2$ , то алгоритм локального поиска гарантированно выходит на глобальный оптимум. Например, при радиусе зон поиска  $MR = 3$ , на первом шаге алгоритм рассмотрит точки  $(1, 0, \dots, 0, 0), \dots, (0, \dots, 0, 1, 1, 1)$  и выберет точку  $(0, 1, 1, 1, 0, \dots, 0)$  как наилучшую. На следующем шаге алгоритм в новой зоне с центром  $(0, 1, 1, 1, 0, \dots, 0)$  просмотрит точки  $(1, 1, 1, 1, 0, \dots, 0), \dots, (0, 1, 1, 1, 0, \dots, 0, 1, 1, 1)$ , но все они будут недопустимыми в силу ограничений по ресурсам. Тогда за решение будет принята точка  $(0, 1, 1, 1, 0, \dots, 0)$ .

Подбирая далее аналогичные примеры можно показать, что при одном компьютере  $NH = 1$ , при числе логических серверов  $NS \geq 3$  и произвольном количестве типов ресурсов  $(NC)$ , но при отсутствии дополнительных ограничений-исключений  $(NX = 0)$  и только при одном оптимизируемом ресурсе, точность в наихудшем случае не ниже, чем  $MR / (MR + 1)$ . Однако, чтобы был возможен наихудший случай радиус зон поиска должен быть  $MR \leq NS - 2$ . Это условие можно пояснить так: чтобы получить наихудшую точность, должен быть 1 “неудачный” логический сервер и сочетание из  $\Omega$  “удачных” логических серверов  $(\Omega \leq NS - 1)$ , дающих наибольшую загрузку ресурса

компьютера. При радиусе  $MR < \Omega$  алгоритм локального поиска на первом шаге размещает “неудачный” логический сервер на компьютере, поскольку его требование обеспечивает более высокую загрузку ресурса, чем суммарные требования любого сочетания из  $MR$  “удачных” логических серверов. На втором шаге в силу ограничений по ресурсам ни один из “удачных” серверов не сможет быть размещен вместе с “неудачным”.

Таким образом, хотя и с некоторыми допущениями (см. выше), но все же мы можем выбирать параметр  $MR$ , оценивая нижнюю границу точности по следующей формуле:

$$\left\{ \begin{array}{l} \xi \geq \left( \frac{MR}{MR+1} \right) \\ \xi \in [0,1], NS \geq 3 \\ 1 \leq MR \leq NS - 2 \end{array} \right. \quad (3.14)$$

В вышерассмотренных примерах отсутствовали дополнительные ограничения-исключения, считалось, что на одном компьютере могут размещаться любые сочетания логических серверов, если соблюдаются ограничения по ресурсам. При использовании же дополнительных ограничений можно выявить более неблагоприятные ситуации для точности субоптимальных решений. Однако, необходимо отметить, что поскольку дополнительные ограничения предназначены для исключения возможности размещения логических серверов на один и тот же компьютер, то наихудшие ситуации могут проявляться только, если в задаче либо один компьютер, либо их множество, но только один компьютер имеет достаточные ресурсы, а на все остальные компьютеры невозможно разместить ни один логический сервер.

Пример 2.1. Рассмотрим следующий пример: пусть имеется один тип ресурса ( $NC = 1$ ), один компьютер  $H_1$  с базовым уровнем ресурса равным  $R$  (причем  $R \gg 1$ ), и множество логических серверов  $\{S_1, \dots, S_{NS}\}$ . Требование 1-го логического сервера  $S_1$  равно  $\lfloor R / (NS - 1) \rfloor + 1$ , требования остальных  $NS - 1$  серверов  $\{S_2, \dots, S_{NS}\}$  равны  $\lfloor R / (NS - 1) \rfloor$ . Кроме того, заданы дополнительные



ограничения, которые исключают возможность размещения логического сервера  $S_1$  с любым другим сервером из множества  $\{S_2, \dots, S_{NS}\}$ , число таких ограничений равно  $NX = NS - 1$ . Требования базовой операционной системы нулевые. Оптимизация будет проводиться по заданному типу ресурсов. Число стартовых точек равно 1 ( $MS = 1$ ).

Пусть сначала  $MR = 1$ ,  $NS \gg MR$ .

При решении этой задачи на первом шаге алгоритм локального поиска начинает с точки  $(0, \dots, 0)$  и рассматривает точки  $(1, 0, \dots, 0, 0), \dots, (0, 0, \dots, 0, 1)$  и точка  $(1, 0, \dots, 0, 0)$  будет выбрана как наилучшая. На втором шаге алгоритм рассмотрит новые точки  $(1, 1, 0, \dots, 0, 0), \dots, (1, 0, 0, \dots, 0, 1)$ , но ни одна из них в силу дополнительных ограничений не будет допустимой. Тогда за решение будет принята точка:  $(1, 0, \dots, 0, 0)$  – то есть на компьютере будет размещен только логический сервер  $S_1$ . При этом загрузка ресурса компьютера составит  $(\lfloor R / (NS - 1) \rfloor + 1) / R$  (при  $NS = 11$ , и  $R = 10000$  загрузка равна 10,01%).

Нетрудно заметить, что глобальным оптимумом является решение  $(0, 1, 1, \dots, 1, 1)$ . При этом загрузка ресурса компьютера составляет  $((NS - 1) * \lfloor R / (NS - 1) \rfloor + 1) / R$  (при  $NS = 11$ , и  $R = 10000$  загрузка равна 100%). Тогда точность субоптимального решения составит  $(\lfloor R / (NS - 1) \rfloor + 1) / ((NS - 1) * \lfloor R / (NS - 1) \rfloor + 1)$  (при  $NS = 11$ , и  $R = 10000$  точность составит 0,1001 – это хуже, чем оценка по формуле 3.11, которая при  $MR = 1$ , дает оценку 0,5). Если для простоты считать, что допустимы нецелые значения для требований логических серверов, а также, если базовый уровень ресурса  $R \gg 1$ , то точность можно будет оценить по формуле:  $1 / (NS - 1)$ . Например, при  $NS = 11$ , нижняя граница точности равна 0,1.

Если решать вышеописанную задачу при радиусе зон поиска  $MR > 1$ , то алгоритм гарантированно выходит на глобальный оптимум. Например, при радиусе зон поиска  $MR = 2$ , на первом шаге алгоритм рассмотрит точки  $(1, 0, \dots, 0, 0, 0), \dots, (0, 0, \dots, 0, 1, 1)$  и выберет  $(0, 1, 1, 0, \dots, 0)$  как наилучшую точку. После этого через некоторое число шагов алгоритм поиска придет к решению  $(0, 1, 1, \dots, 1, 1)$ .

Пример 2.2. Рассмотрим теперь еще один пример: пусть имеется один тип ресурса ( $NC = 1$ ), один компьютер  $H_1$  с базовым уровнем ресурса равным  $R$  (причем  $R \gg 1$ ), и множество логических серверов  $\{S_1, \dots, S_{NS}\}$ . Требования 1-го логического сервера  $S_1$  равно  $2 * (\lfloor R / (NS - 1) \rfloor + 1)$ , требования остальных  $NS - 1$  серверов  $\{S_2, \dots, S_{NS}\}$  равны  $\lfloor R / (NS - 1) \rfloor$ . Кроме того, заданы дополнительные ограничения, которые исключают возможность размещения логического сервера  $S_1$  с любым другим сервером из множества  $\{S_2, \dots, S_{NS}\}$ , число таких ограничений равно  $NX = NS - 1$ . Требования базовой операционной системы нулевые. Оптимизация будет проводиться по заданному типу ресурсов. Число стартовых точек равно ( $MS = 1$ ).

Пусть сначала  $MR = 2$ ,  $NS \gg MR$ .

При решении этой задачи на первом шаге алгоритм локального поиска начинает с точки  $(0, \dots, 0)$  и рассматривает точки  $(1, 0, \dots, 0, 0), \dots, (0, 0, \dots, 0, 1, 1)$  и точка  $(1, 0, 0, \dots, 0)$  будет выбрана как наилучшая. На втором шаге алгоритм рассмотрит множество новых точек  $(1, 1, 0, \dots, 0), \dots, (1, 0, 0, \dots, 1, 1)$ , но ни одна из них в силу дополнительных ограничений не будет допустимой. Тогда за решение будет принята точка  $(1, 0, \dots, 0, 0)$  – на компьютере может быть размещен только логический сервер  $S_1$ . При этом загрузка ресурса компьютера составит  $(2 * (\lfloor R / (NS - 1) \rfloor + 1)) / R$  (при  $NS = 11$ , и  $R = 10000$  загрузка равна 20,01%).

Нетрудно заметить, что глобальным оптимумом является решение  $(0, 1, 1, \dots, 1)$ . При этом загрузка ресурса компьютера составляет  $((NS - 1) * \lfloor R / (NS - 1) \rfloor) / R$  (при  $NS = 11$ , и  $R = 10000$  загрузка равна 100%). Тогда, точность субоптимального решения составит  $(2 * (\lfloor R / (NS - 1) \rfloor + 1)) / ((NS - 2) * \lfloor R / (NS - 1) \rfloor)$  (при  $NS = 11$ , и  $R = 10000$  точность составит 0,2002 – это хуже, чем оценка по формуле 3.11, которая при  $MR = 2$ , дает оценку 0,6667). Если для простоты считать, что допустимы нецелые значения для требований логических серверов, а также, если базовый уровень ресурса  $R \gg 1$ , то точность можно будет оценить по формуле:  $2 / (NS - 1)$ . Например, при  $NS = 11$ , нижняя граница точности равна 0,2.

Если решать вышеописанную задачу при радиусе зон поиска  $MR > 2$ , то алгоритм гарантированно выходит на глобальный оптимум. Например, при радиусе зон поиска  $MR = 3$ , на первом шаге алгоритм рассмотрит точки  $(1,0,\dots,0,0,0),\dots, (0,0,\dots,0,1,1,1)$  и выберет  $(0,1,1,1,0,\dots,0)$  как наилучшую точку. После этого через некоторое число шагов алгоритм поиска придет к решению  $(0,1,1,\dots,1,1)$ .

Подбирая далее аналогичные примеры можно показать, что при одном компьютере  $NH = 1$ , при числе логических серверов  $NS \geq 3$  и произвольном количестве типов ресурсов  $(NC)$  и при произвольном количестве дополнительных ограничений-исключений  $(NX)$ , но только при одном оптимизируемом ресурсе, точность в наихудшем случае не ниже, чем  $MR / (NS - 1)$ . Однако, чтобы был возможен наихудший случай радиус зон поиска должен быть  $MR \leq NS - 2$ . Это условие можно пояснить так: чтобы получить наихудшую точность, должен быть 1 “неудачный” логический сервер и сочетание из  $\Omega$  “удачных” логических серверов ( $\Omega \leq NS - 1$ ), дающих наибольшую загрузку ресурса компьютера. При радиусе  $MR < \Omega$  алгоритм локального поиска на первом шаге размещает “неудачный” логический сервер на компьютере, поскольку его требование обеспечивает более высокую загрузку ресурса, чем суммарные требования любого сочетания из  $MR$  “удачных” логических серверов. На втором шаге в силу дополнительных ограничений ни один из “удачных” серверов не сможет быть размещен вместе с “неудачным”.

Таким образом, хотя и с некоторыми допущениями (см. выше), но все же мы можем выбирать параметр  $MR$ , оценивая нижнюю границу точности по следующей формуле:

$$\left\{ \begin{array}{l} \xi \geq \left( \frac{MR}{NS-1} \right) \\ \xi \in [0,1], NS \geq 3 \\ 1 \leq MR \leq NS - 2 \end{array} \right. \quad (3.15)$$

Подводя итог, можно сказать, что при отсутствии дополнительных ограничений рекомендуется выбирать радиус зон поиска исходя из оценки нижней границы точности по формуле 3.14, при наличии дополнительных ограничений – по формуле 3.15.

На первый взгляд полученные оценки точности субоптимальных решений могут показаться довольно удручающими. Особенно обращает на себя внимание оценка по формуле 3.15, поскольку согласно ей, например, при числе логических серверов равном 101 и радиусе зон поиска равном 1, нижняя граница точности составляет всего 0,01. Но с другой стороны, такая оценка была получена при условии, что в рассмотрении присутствует один “неудачный” логический сервер, который не может быть размещен на каком-либо компьютере ни с одним другим сервером. Вполне очевидно, что лучше было бы оставить этот логический сервер на том компьютере, на котором он работал, удалить его вместе с этим компьютером из рассмотрения и тем самым избежать худших решений задачи поиска распределения.

Кроме того, следует отметить, что оценки точности были получены при условии, что всегда используется только одна стартовая точка ( $MS = 1$ ), причем нулевая –  $(0, \dots, 0)$ . Нетрудно заметить, что если использовать несколько случайных стартовых точек, то появляется некоторая вероятность “ухода из зоны худших решений”, а также вероятность “попадания в зону глобального оптимума”. Так, например, в примерах 1.1 и 2.1, если стартовой точкой оказывается одна из точек  $\{(0, 1, 0, \dots, 0, 0), \dots, (0, 0, 0, \dots, 0, 1)\}$ , то алгоритм локального поиска гарантированно выходит на глобальный оптимум. В рамках данной диссертации не проводится статистический анализ качества решений в зависимости от числа стартовых точек (параметра  $MS$ ). Тем не менее, достаточно, очевидно, что чем больше стартовых точек, тем выше вероятность выхода на хорошие субоптимальные решения или на глобальный оптимум. При этом в худшем случае с ростом числа стартовых точек объем перебора возрастает линейно.

## Выводы по главе 3

Данная глава была посвящена разработке математической модели и поиску метода решения задачи поиска оптимального распределения логических серверов на физические компьютеры.

В первой части главы разработана математическая модель, описывающая класс задач поиска оптимального распределения множества логических серверов на множество компьютеров, предоставляющих множество ресурсов различных типов. Модель получена путем нескольких приближений от базовой модели задачи о рюкзаке к конечной модели. Описание конечной модели показывает, что она учитывает ключевые моменты поставленной задачи поиска оптимального распределения: множество типов ресурсов, дополнительные ограничения на размещение логических серверов на один и тот же компьютер, требования базовой ОС, возможность оптимизации по нескольким ресурсам одновременно и множество компьютеров, предоставляющих ресурсы.

Во второй части в результате обзора математических методов для решения задачи поиска распределения в целом, ввиду ее особой сложности сделан вывод о необходимости разбиения ее на множество более простых подзадач. Предложена схема разбиения на подзадачи с использованием аппарата динамического программирования. В свою очередь, в результате обзора математических методов для решения подзадач, являющихся задачами условной псевдобулевой оптимизации сделан вывод о предпочтительности использования некоторого приближенного метода дискретной оптимизации. Предложен метод решения задачи, опирающийся на метод локального поиска оптимума с использованием функции штрафов для учета ограничений (условий), множества случайных стартовых точек, что позволяет повысить вероятность нахождения более хороших решений, и управляемого радиуса зоны поиска, что расширяет зону поиска (при этом одновременно увеличивается и объем перебора).

В третьей части главы в результате анализа сходимости, оценки объема перебора и качества получаемых решений в предложенном методе решения задачи поиска оптимального распределения установлено следующее:

- Метод решения задачи распределения обладает сходимостью. На уровне решения задачи в целом возможны только три конечные ситуации: множество логических серверов опустошается, множество компьютеров опустошается, ни один из оставшихся логических серверов не размещается ни на один из оставшихся компьютеров. На уровне решения подзадачи псевдобулевой оптимизации сходимость определяется тем, что сам метод локального поиска обладает сходимостью.
- Объем перебора полиномиально  $\sim O(NH^2)$  зависит от числа компьютеров. Объем перебора полиномиально  $\sim O(MS)$  зависит от числа стартовых точек (MS). Объем перебора полиномиально  $\sim O(NS^{MR+1})$  зависит от числа логических серверов при малых значениях параметра радиуса зоны поиска ( $MR \ll NS$ ), и экспоненциально  $\sim O(2^{NS})$  – при значении параметра равного числу логических серверов ( $MR = NS$ ). Объем перебора не зависит от числа типов ресурсов (NC) и числа дополнительных ограничений (NX).
- Точность субоптимальных решений для алгоритма локального поиска достаточно сложным способом зависит от исходных условий задач. Тем не менее, при помощи нескольких простых представительных задач установлено, что в задачах без дополнительных ограничений ( $NX = 0$ ) при числе логических серверов  $NS \geq 3$  и радиусе зон поиска  $MR \leq NS - 2$  точность не ниже  $MR / (MR + 1)$ . При наличии же дополнительных ограничений ( $NX > 0$ ) при числе логических серверов  $NS \geq 3$  и радиусе зон поиска  $MR \leq NS - 2$ , точность не ниже  $MR / (NS - 1)$ . Оценки нижней границы точности были получены для случая одной стартовой точки ( $MS = 1$ ). В случае же, если используется множество стартовых точек, то чем больше стартовых точек, тем больше вероятность того, что фактическая точность будет выше ее нижней границы или вообще будет равна 1.

## 4. Программная реализация алгоритма поиска оптимального распределения

### 4.1. Требования к программной реализации

Разработка программного обеспечения, решающего какие-либо математические задачи заданного класса – это сложная работа, состоящая из множества этапов. Непосредственное кодирование на каком-либо языке программирования является лишь одним из этапов. Разработка программного обеспечения требует тщательного описания входных и выходных параметров, анализа допустимых областей для этих параметров, проработки вопросов реакции программы на некорректные значения входных параметров, обзора существующих подходов к решению задач заданного класса, разработки метода решения и представления его в виде какого-либо алгоритма, выбора наиболее подходящего языка программирования и реализации алгоритма на этом языке. После реализации также требуется разрабатывать некоторые контрольные наборы тестов для выявления и устранения ошибок в разработанном программном обеспечении.

Помимо этого, следует учитывать особенности современных ОС, средств разработки и подходов к разработке программного обеспечения, которые широко используются в течение многих лет и уже фактически стали обязательными для программистов-профессионалов:

- Графический интерфейс для ввода/вывода информации.
- Возможности многозадачности в операционных системах.
- Высокоуровневый программный интерфейс ОС для разработки.
- Принципы объектно-ориентированного программирования.
- Встроенные в ОС и средства разработки системные компоненты, компоненты графики, СУБД, сетевых служб и т.д.
- Многоуровневая обработка некорректных входных данных, системных сбоев и исключительных ситуаций, возникающих во время выполнения.

С учетом вышесказанного мы можем сформулировать базовые требования к программной реализации алгоритма поиска распределения, как с точки зрения конечных пользователей, так и с точки зрения современных подходов программирования:

- Программа должна разрабатываться на языке высокого уровня для работы под управлением наиболее распространенных операционных систем. Программа должна максимально использовать встроенные возможности операционной системы [56, 57, 60] и средства для разработки приложений.
- Программа должна быть разработана в соответствии с современными подходами модульного и объектно-ориентированного программирования (ООП) [51, 52, 55]. Желательно, чтобы математическая обработка была вынесена в отдельный модуль, причем модуль должен быть независимым от других модулей, а также гибким в плане расширения функциональности для возможности его применения при разработке других приложений.
- Программа должна использовать возможности многозадачности в современных ОС [20, 58, 59]. Поскольку в программе имеет место сложная математическая обработка, то, очевидно, требуется использование как минимум двух потоков: главного потока процесса и потока математической обработки. Интерфейс программы должен своевременно обрабатывать системные события и реагировать на действия пользователя.
- Программа должна предоставлять удобный графический интерфейс для пользователя, обеспечивать возможность быстрого ввода и корректировки исходных данных, загрузки и сохранения исходных данных, сохранения результатов в файлах, гибкого управления потоком решения задачи.
- Программа должна корректно обрабатывать различные проблемные ситуации: некорректные входные данные, нехватка памяти, сбой устройства хранения данных при операциях чтения или записи данных. Программа должна разрабатываться с использованием современных подходов тестирования и отладки программного обеспечения [53, 54].



## 4.2. Описание разработанного программного обеспечения

С учетом вышеприведенных требований автором было разработано многопоточное приложение DTSOLVEX, с графическим интерфейсом пользователя, с разбиением исходного кода на модули и с использованием принципов объектно-ориентированного программирования. В качестве класса операционных систем, для которого было разработано программное обеспечение, был выбран класс ОС MS Windows, поскольку на сегодняшний день он имеет наибольшее распространение. В качестве средства разработки был выбран Borland Delphi 7.0 – данный программный продукт содержит большие возможности для разработки и отладки объектно-ориентированных многопоточных приложений для ОС MS Windows, а также имеет богатый набор визуальных компонент, для построения красивого и удобного графического интерфейса пользователя. Среда разработки Borland Delphi [65, 66, 67, 68, 69, 70] является развитием сред Turbo [61, 62, 64] и Borland Pascal [63], включает в себя возможности языка программирования классического и объектного Pascal-а и значительно расширяет его.

В приложении 4 приводится краткое руководство для пользователя, а в приложении 5 приводится исходный код программы. В данной же главе мы кратко рассмотрим ключевые моменты и особенности программы.

### **Основная программа и подключаемые модули**

Исходный код расположен в тексте основной программы и трех модулей:

- Исходный код основной программы при разработке в Borland Delphi не представляет собою ничего особого – это не более 10-15 строк кода: подключение модулей, инициализация и запуск приложения. Основной же код сосредоточен в подключаемых модулях.
- Главный модуль программы: в нем содержатся компоненты графического интерфейса: меню, поля данных, кнопки управления, а также обработчики событий, поступающих от них, исходный код для главного потока процесса, потока решения задачи и потока мониторинга.

- Вспомогательный промежуточный модуль между математическим модулем и модулем графического интерфейса. Этот модуль необходим для того, чтобы расширить функциональность математического модуля: обеспечить взаимодействие с графическим интерфейсом, обеспечить файловые операции с исходными данными задач и результатами.
- Модуль математической обработки: в нем содержится иерархия классов в терминах ООП, соответствующих классам математических задач.

Кроме того, в Delphi форма главного окна вместе со всеми содержащимися графическими элементами и элементами управления описывается в отдельном файле программы. Содержимое этого файла не представляет особого интереса.

### **Главный модуль. Графический интерфейс программы**

На рисунке 4.1. и 4.2 приведены вид окна программы и его меню.

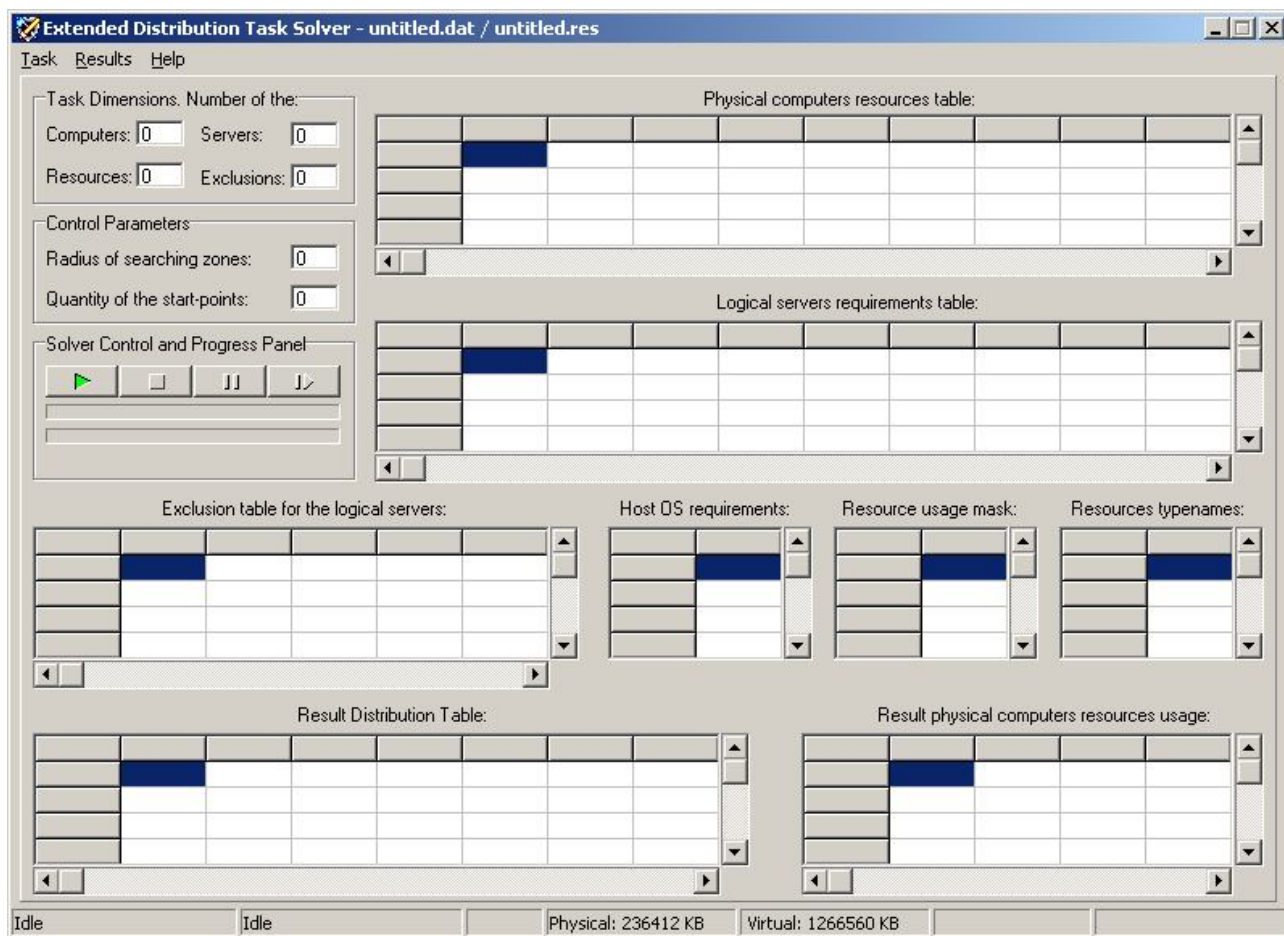


Рис. 4.1. Вид главного окна разработанной программы

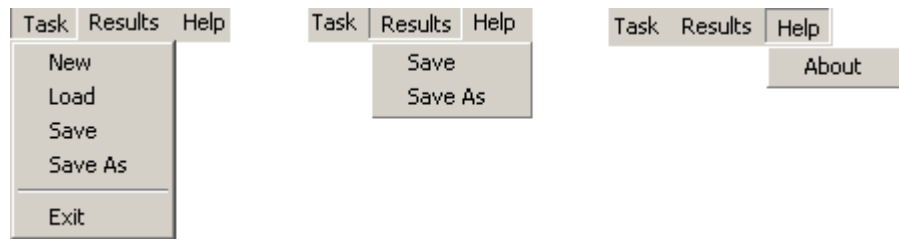






Рис. 4.2. Вид элементов меню программы

Мы не будем подробно останавливаться на описании главного окна, полей ввода/вывода данных и элементов управления – все они достаточно подробно рассмотрены в приложении 4. Через меню доступны возможности создания новой, открытия, сохранения задачи и сохранения результатов, а также закрытия программы и вывода информации об авторе.

### **Главный модуль. Логика работы программы.**

В отличие от DOS-приложений и консольных приложений, Windows-приложения с графическим интерфейсом не находятся постоянно в каких-либо циклах обработки или циклах ожидания действий пользователя. Windows-приложения при отсутствии внешних воздействий (событий) находятся в спящем режиме, если они только не обрабатывают ранее поступившее событие, и они тратят минимум процессорного времени. При такой событийной модели работы ОС и приложений, невозможно представить работу программы в виде какого-то алгоритма, который в каждый момент времени находится на каком-то этапе выполнения и процессор непрерывно занят его обработкой до самого завершения работы программы. Кроме того, множество событий, возникающих в ОС и в приложениях, очень велико (до тысячи и выше), и многие из них (десятки, сотни) обрабатываются конкретной программой, а остальные события обрабатываются системным обработчиком ОС, реализация которого скрыта внутри ОС. Представить в виде единого алгоритма работу программы вместе со всеми моментами ее взаимодействия с ОС практически невозможно. В такой ситуации гораздо легче наглядно представить работу программы (его потоков) в виде диаграммы состояний. При запуске Windows-приложения всегда запускается, как минимум один поток – главный поток приложения.

В разработанной программе присутствуют три потока: главный поток, поток решения задач и поток мониторинга (сбор сведений о состоянии потока решения задачи, главного потока и ресурсов системы). При запуске программы автоматически создается главный поток программы, который запускает поток мониторинга. При закрытии программы главный поток перед завершением останавливает работу потока мониторинга. Главный поток программы в зависимости от действий пользователя может запускать (кнопка  Run), завершать (кнопка  Stop), приостанавливать (кнопка  Pause) и возобновлять (кнопка  Resume) поток решения задачи поиска распределения. Главный поток также ответственен за обработку всех событий.

На рисунках 4.3, 4.4, и 4.5 приведены упрощенные диаграммы состояний главного потока, потока решения и потока мониторинга программы.

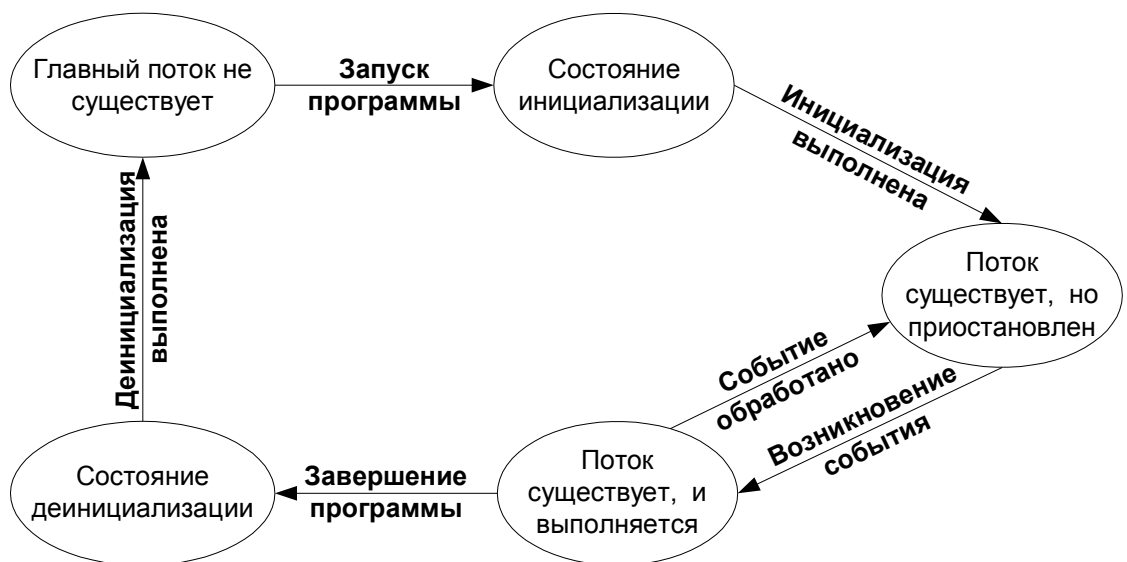


Рис. 4.3. Диаграмма состояний главного потока

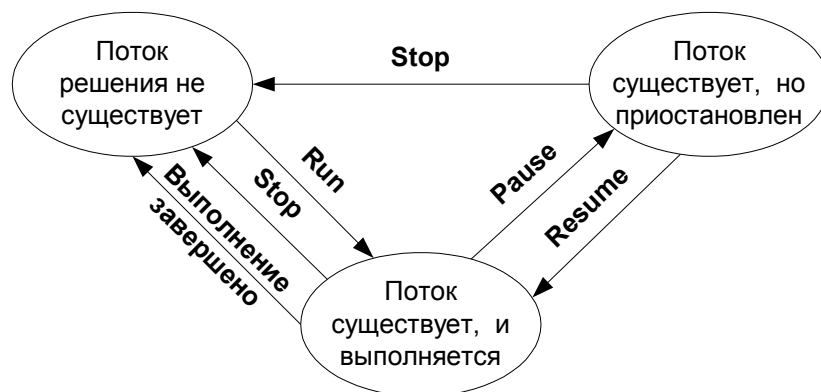


Рис. 4.4. Диаграмма состояний для потока решения

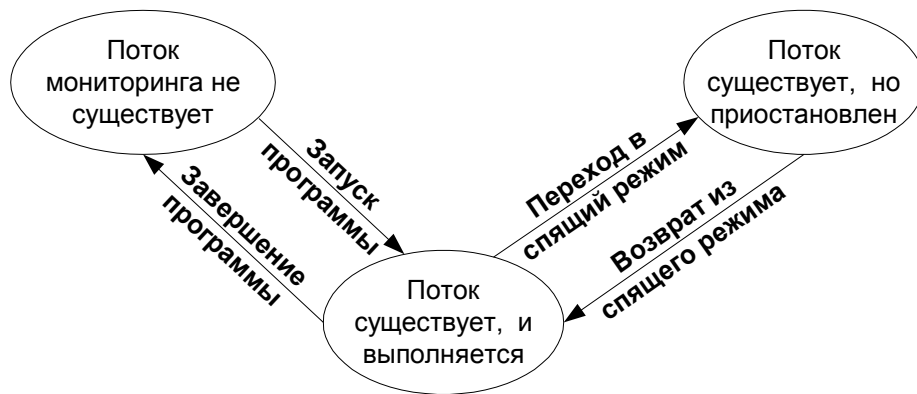


Рис. 4.5. Диаграмма состояний для потока мониторинга

**Синхронизация потоков.** Главный поток, поток решения и поток мониторинга не используют одновременно каких-либо общих ресурсов.

В потоке решения ничего кроме вычислений не происходит. Загрузка исходных данных из интерфейса выполняется главным потоком и только после этого запускается поток решения. Вывод результатов в интерфейс выполняется в обработчике события завершения работы потока решения, который также выполняется главным потоком. Поскольку событие завершения работы потока решения автоматически возникает сразу же после завершения работы исполняемого кода потока решения, то есть в случае готовности решения задачи или какой-либо ошибки, то нет необходимости в главном потоке отслеживать этот момент. Результаты решения или ошибка будут автоматически выведены в интерфейс обработчиком события.

Поток мониторинга отслеживает информацию о системе, состояние потока решения и флаги модификации исходных данных и обновления результатов, и ему требуется выводить информацию в интерфейс. Однако, прямой вывод информации в поля интерфейса крайне нежелательно было использовать, поскольку, обновление данных в полях, как правило, влечет вызов функций SendMessage – посылка системного сообщения типа Window Message окну приложения, с обязательным ожиданием того, когда сообщение будет обработано. Поскольку сообщение обрабатывает, как правило, главный поток, то если сложится такая ситуация, что главный поток ожидает какого-либо события или освобождения семафора, которое должно произойти в другом

потоке, а этот поток, в свою очередь, ожидает завершения обработки `SendMessage`, то, очевидно, произойдет взаимоблокировка (DEADLOCK). Поэтому поток мониторинга не обновляет ничего напрямую, а лишь изменяет специальные переменные, и посылает сообщение типа `Window Message` окну приложения функцией `PostMessage` (без ожидания обработки этого сообщения). Главный поток обработает это сообщение тогда, когда до него дойдет очередь и обновит поля интерфейса значениями из специальных переменных. Таким образом, обходится опасность взаимоблокировки.

### Модуль математической обработки

Модуль математической обработки является ядром программы, основным назначением которого является обработка входных данных и непосредственное решение задачи. Для использования преимуществ и принципов объектно-ориентированного программирования в модуле была разработана иерархия классов (рис. 4.6) математических задач со свойствами, предназначенных для хранения исходных данных и результатов, и методами инициализации, решения и завершения задачи. Объем исходных данных и сложность решаемых задач возрастает от родительских классов к дочерним классам: чем ниже по иерархии, тем больше дополнительных свойств (исходных условий задач) и, соответственно, тем сложнее решать задачу.



Рис 4.6. Структура иерархии классов математических задач.

При разработке иерархии классов в первом приближении был выделен абстрактный класс, который не решает какие-либо задачи, но является корневым классом для всех классов поисковых задач. Поскольку задача поиска распределения опирается на решение подзадач булевой оптимизации, то были выделены 2 ветви дочерних классов: левая ветвь – классы задач булевой оптимизации, правая ветвь – классы задач поиска распределения.

Отметим ключевые свойства классов и их методы:

Класс `TLOCALSEARCHTASK` имеет два свойства `MAX_RADIUS` (радиус зон поиска) и `MAX_STARTS` (число стартовых точек) и пять методов:

- Конструктор класса, выполняет проверку входных данных для свойств класса и инициализирует экземпляр класса.
- Деструктор класса, уничтожает экземпляр класса.
- Процедура решения, в данном классе ничего не делает – заглушка.
- Процедура запуска решения, специальная процедура, реализующая полиморфизм: вызывается процедура решения без указания класса.
- Специальная процедура, проверяющая глобальный флаг остановки решения.

Класс `TBOOLOPTIMUMTASK` наследует все свойства класса `TLOCALSEARCHTASK` и добавляет новые: `N` (число переменных), `S` (указатель на вектор коэффициентов в целевой функции), `X` (указатель на результирующий оптимальный вектор). Кроме того, класс переопределяет конструктор (инициализация новых свойств, вызов унаследованного конструктора, выделение памяти для временных переменных), деструктор (освобождение памяти, вызов унаследованного деструктора) и процедуру решения (ничего не делает, поскольку в диссертации не требуется решение задач безусловной оптимизации).

Класс `TDISTRIBUTIONTASK` наследует все свойства класса `TLOCALSEARCHTASK` и добавляет новые: `NH` (число компьютеров), `NS` (число логических серверов), `NC` (число типов ресурсов), `HST` (указатель на матрицу базовых уровней ресурсов компьютеров), `SST` (указатель на матрицу требований логических серверов), `BST` (указатель на вектор требований

базовой ОС), CM (указатель на вектор маски), DT (указатель на матрицу распределения) и RL (указатель на матрицу загрузки ресурсов). Кроме того, класс переопределяет конструктор (инициализация новых свойств, вызов унаследованного конструктора, выделение памяти для временных переменных), деструктор (освобождение памяти, вызов унаследованного деструктора) и процедуру решения (ничего не делает, поскольку в диссертации не требуется решение задач поиска распределения без дополнительных ограничений).

Класс TBOOLOPTIMUMTASKEXT наследует все свойства класса TBOOLOPTIMUMTASK и добавляет новые: M (число ограничений), A (указатель на матрицу, левую часть ограничений), B (указатель на вектор, правую часть ограничений). Кроме того, класс переопределяет конструктор (инициализация новых свойств, вызов унаследованного конструктора, выделение памяти для временных переменных), деструктор (освобождение памяти, вызов унаследованного деструктора) и процедуру решения – в ней полностью реализован алгоритм обобщенного локального поиска (рис 3.2).

Класс TDISTRIBUTIONTASKEXT наследует все свойства класса TDISTRIBUTIONTASK и добавляет новые: NX (число дополнительных ограничений-исключений), EXT (указатель на матрицу дополнительных ограничений-исключений). Кроме того, класс переопределяет конструктор (инициализация новых свойств, вызов унаследованного конструктора, выделение памяти для временных переменных), деструктор (освобождение памяти, вызов унаследованного деструктора) и процедуру решения – в ней полностью реализован алгоритм решения задачи поиска распределения (рис 3.1). Поскольку решение опирается на решение задач условной псевдобулевой оптимизации, то процедура решения использует подзадачи – экземпляры класса TBOOLOPTIMUMTASKEXT.

Следует отметить, что для эффективного использования памяти во всех классах поля исходных данных и результатов, которые относятся к матрицам/векторам (размеры которых зависят от параметров размерности задач), хранят не сами матрицы/векторы, а только лишь указатели на них.



Таким образом, математический модуль полностью внутри себя содержит необходимую математическую обработку в виде иерархии классов с их входными/выходными свойствами и методами. Соответственно, для того, чтобы решить какую-либо задачу конкретного класса необходимо:

- Выделить память для матриц/векторов исходных данных, матриц/векторов результатов и создать экземпляр соответствующего класса с помощью конструктора, передать ему необходимые параметры размерности задачи, параметры управления поиском и указатели на матрицы/векторы исходных данных, матрицы/векторы результатов.
- Вызвать процедуру решения, которая решает задачу по исходным данным, находящимся в полях класса для исходных данных и заносит результаты решения в поля класса для результатов.
- Уничтожить экземпляр при помощи деструктора.

### **Вспомогательный модуль**

Математический модуль предоставляет удобные возможности создания, инициализации исходных данных, решения и уничтожения конкретных задач, являющихся экземплярами класса задач поиска распределения с дополнительными ограничениями-исключениями или класса задач условной булевой оптимизации. Тем не менее, этот модуль не обеспечивает следующий набор функциональности: загрузка задач из файла, сохранение задач в файл, сохранение результатов решения в файл, а также обмен данными между интерфейсом пользователя и экземплярами класса задач поиска распределения. Этот набор функциональности можно было бы реализовать в главном модуле, но в таком случае этот модуль стал бы слишком громоздким, а исходный код чрезвычайно сложным для понимания особенностей функционирования программного обеспечения. По этой причине вышеприведенный набор функциональности был вынесен в отдельный модуль – вспомогательный модуль, который фактически является промежуточным слоем взаимодействия между главным модулем и математическим модулем. Поскольку

вышеперечисленный набор функциональности связан с операциями, которые используют те же поля данных, что и класс задач поиска распределения с дополнительными ограничениями, то не было смысла в дублировании этих полей в каком-либо отдельном независимом классе. Поэтому во вспомогательном модуле создан вспомогательный класс TCUSTOMTASK, являющийся дочерним по отношению к классу TDISTRIBUTIONTASKEXT. Вспомогательный класс наследует свойства родительского и добавляет новое свойство RTN (указатель на вектор имен типов ресурсов). Для обеспечения вышеперечисленного набора функциональности в классе TCUSTOMTASK переопределены унаследованные методы и добавлены новые:

- Конструктор класса – инициализация полей класса пустыми данными.
- Процедура 1 – выделение памяти для матриц/векторов исходных данных и загрузка полей класса для исходных данных из полей графического интерфейса с проверкой корректности данных.
- Процедура 2 – выделение памяти для матриц/векторов исходных данных и загрузка полей класса для исходных данных из файла с проверкой корректности данных.
- Процедура 3 – сохранение полей класса для исходных данных в поля графического интерфейса.
- Процедура 4 – сохранение полей класса для исходных данных в файл.
- Процедура 5 – выделение памяти для матриц результатов и решение задачи. Исходные данные должны находиться в полях класса, результаты решения задачи процедура помещает в поля класса для результатов.
- Процедура 6 – выделение памяти для матриц результатов и загрузка полей класса для результатов из полей графического интерфейса для результатов.
- Процедура 7 – сохранение полей класса для результатов в поля графического интерфейса для результатов.
- Процедура 8 – сохранение полей класса для результатов в файл.
- Деструктор класса – освобождение памяти для всех матриц/векторов исходных данных и результатов.

Главный модуль программы использует экземпляры класса TCUSTOMTASK для обеспечения следующей функциональности:

- Загрузка исходных данных из файла (создается экземпляр класса TCUSTOMTASK конструктором, вызывается процедура 2, вызывается процедура 3, затем экземпляр уничтожается деструктором).
- Сохранение исходных данных в файл (создается экземпляр класса TCUSTOMTASK конструктором, вызывается процедура 1, вызывается процедура 4, затем экземпляр уничтожается деструктором).
- Решение задач (создается экземпляр класса TCUSTOMTASK конструктором, вызывается процедура 1, вызывается процедура 5, вызывается процедура 7, затем экземпляр уничтожается деструктором).
- Сохранение результатов в файл (создается экземпляр класса TCUSTOMTASK конструктором, вызывается процедура 6, вызывается процедура 8, затем экземпляр уничтожается деструктором).

### **Объемы требуемых машинных ресурсов для программы**

Программа работает на процессорах, начиная с Intel 386. К быстродействию процессоров никаких специальных требований нет, от этого зависит только время решения задач. Что же касается оперативной памяти, то требуется минимум 2 МБ для запуска программы. Для решения же задач максимальной размерности ( $NH=NS=NX=NC=1000$ ) и возможности во время решения задачи максимальной размерности одновременно сохранять ее исходные данные или результаты (ранее полученные), требуется не менее 48 МБ свободной оперативной памяти. На диске программа занимает 615 КБ, однако, в случае недостаточной емкости оперативной памяти компьютера, часть памяти будет отображаться операционной системой на дисковое пространство, и может понадобиться до 48 Мб свободного места на диске.

### 4.3. Экспериментальное исследование и внедрение

В рамках данной главы нет возможности рассмотреть все эксперименты, которые проводились при тестировании разработанной программы, оценке времени решения задач, оценке качества получаемых решений, оценке целесообразности использования декомпозиции исходной задачи на множество более простых при решении задач больших размерностей, а также все производственные задачи, решенные в той или иной организации. Тем не менее, рассмотрим несколько ключевых серий экспериментов и одну производственную задачу.

Эксперименты проводились с использованием разработанной программы DTSOLVEX на компьютере с одним процессором Intel Pentium 1000 МГц, 512 МБ ОЗУ, частотой системной шины 133 МГц.

#### Эксперименты по оценке времени решения задач

В данной серии экспериментов исследовалась зависимость времени решения задач от числа компьютеров ( $NH$ ), числа логических серверов ( $NS$ ) и параметра радиуса зоны поиска ( $MR$ ). Следующие исходные данные не изменялись и были одними и теми же во всех экспериментах:

- Тип ресурсов был один ( $NC = 1$ ).
- Для всех компьютеров базовый уровень ресурса был равен 1000 ( $R_{1,k} = 1000$ ,  $k = 1..NH$ ). Для всех логических серверов требование было равно 10 ( $Q_{1,j} = 10$ ,  $j = 1..NS$ ). Требование базовой ОС было нулевым ( $V_1 = 0$ ).
- Дополнительные ограничения-исключения не использовались ( $NX = 0$ ), соответственно, матрица  $\{E_{d,j}\}$ ,  $d = 1..NX$ ,  $j = 1..NS$ , отсутствовала.
- Оптимизация проводилась по единственному типу ресурса ( $O_1 = 1$ ).
- Использовалась одна стартовая точка ( $MS = 1$ ).

Эксперименты 1.1.1 – 1.1.10. Оценка среднего времени решения задач в зависимости от числа логических серверов  $NS$ . Число компьютеров  $NH = 1$ , радиус зон поиска  $MR = 1$ . Результаты сведены в таблицу 4.1 и представлены также в виде графика (рис. 4.7):

Таблица 4.1

NS	10	20	30	40	50	60	70	80	90	100
t, сек	0,00012	0,00055	0,0014	0,0035	0,0062	0,011	0,017	0,024	0,033	0,045

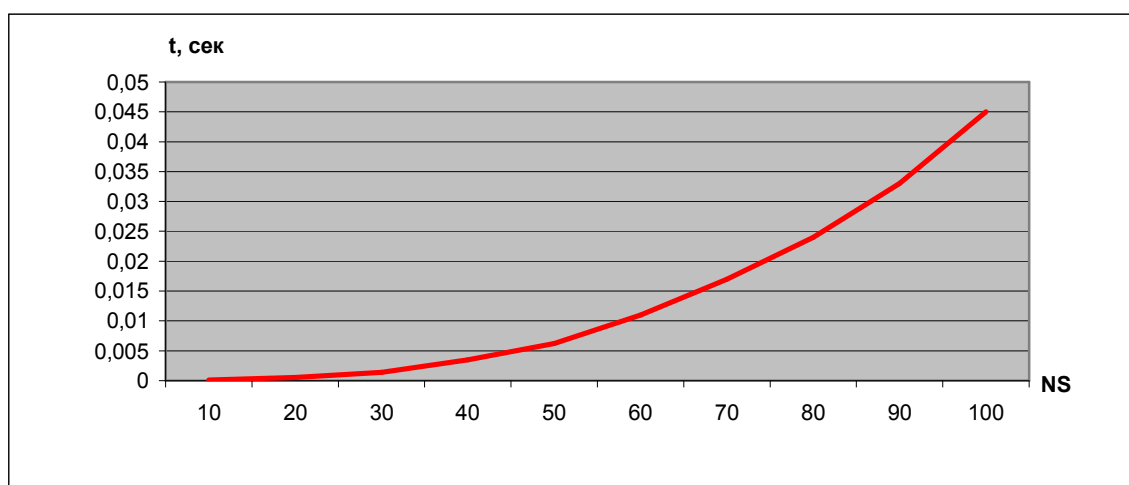


Рис. 4.7. График зависимости времени решения от числа логических серверов

Эксперименты 1.2.1 – 1.2.10. Оценка среднего времени решения задач в зависимости от числа логических серверов NS. Число компьютеров NH = 1, радиус зон поиска MR = 5. Результаты сведены в таблицу 4.2 и представлены также в виде графика (рис. 4.8):

Таблица 4.2

NS	10	20	30	40	50	60	70	80	90	100
t, сек	0,0011	0,117	1,72	14,6	58,2	238	692	1728	3722	7966

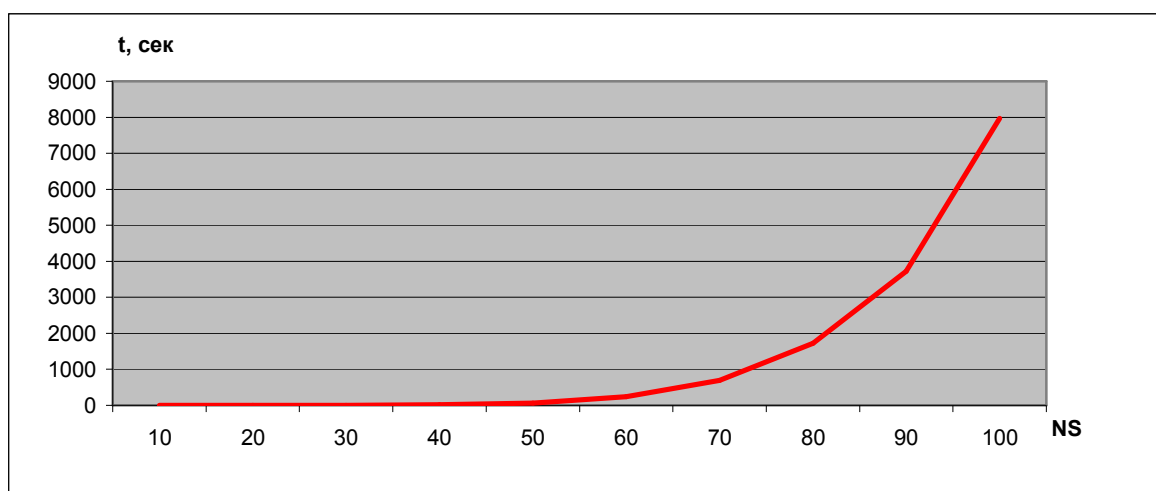


Рис. 4.8. График зависимости времени решения от числа логических серверов

Из графиков 4.7 и 4.8 видно то, как резко увеличивается скорость роста времени решения задач при увеличении радиуса зон поиска. Графики 4.7 и 4.8 подтверждают графики 3.5 и 3.6, построенные по формуле 3.13, выведенной в главе 3 для оценки объема перебора при решении задач.

Эксперименты 1.3.1 – 1.3.30. Оценка среднего времени решения задач в зависимости от радиуса зон поиска MR. Число компьютеров NH = 1, число логических серверов NS = 30. Результаты сведены в таблицу 4.3 и представлены также в виде графика (рис. 4.9):

Таблица 4.3

<b>MR</b>	1	2	3	4	5	6	7	8	9	10
<b>t, сек</b>	0,0011	0,009	0,066	0,377	1,68	6,46	23,06	67,2	245,2	482,4
<b>MR</b>	11	12	13	14	15	16	17	18	19	20
<b>t, сек</b>	783,4	1081	1459	1798	2111	2310	2511	2597	2543	2460
<b>MR</b>	21	22	23	24	25	26	27	28	29	30
<b>t, сек</b>	2375	2276	2211	2122	2066	2012	1962	1921	1867	1828

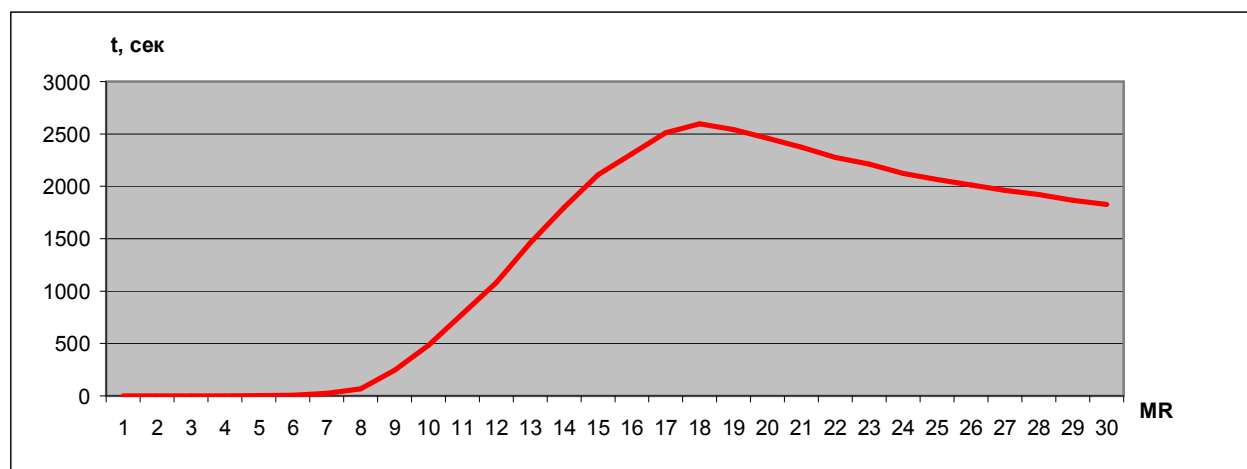


Рис. 4.9. График зависимости времени решения от радиуса зон поиска

Из графика 4.9 видно, что время решения задач не всегда возрастает с ростом радиуса зон поиска. График 4.9 подтверждает график 3.9, построенный по формуле 3.13, выведенной в главе 3 для оценки объема перебора.

Эксперименты 1.4.1 – 1.4.5. Оценка среднего времени решения задач в зависимости от числа компьютеров NH. Число логических серверов NS = 100,

радиус зон поиска  $MR = 1$ . Результаты сведены в таблицу 4.4 и представлены также в виде графика (рис. 4.10):

Таблица 4.4

<b>NH</b>	5	10	15	20	25
<b>t, сек</b>	0,024	0,073	0,142	0,219	0,304

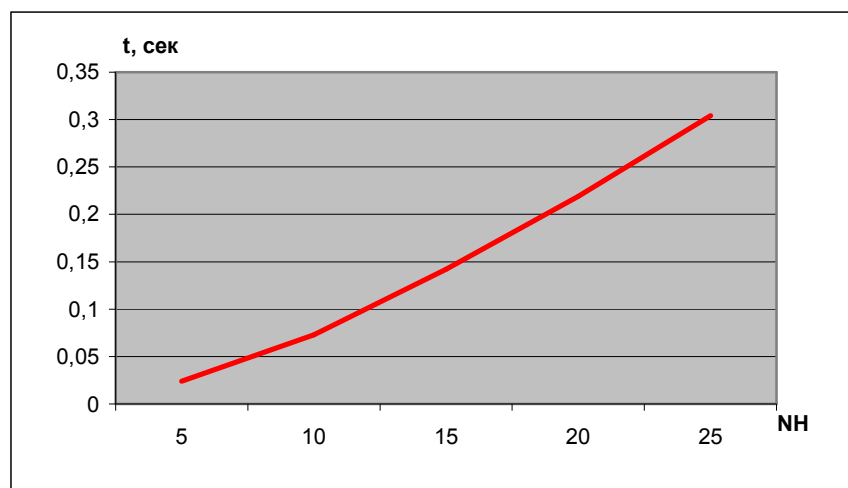


Рис. 4.10. График зависимости времени решения от числа компьютеров

Эксперименты 1.5.1 – 1.5.5. Оценка среднего времени решения задач в зависимости от числа компьютеров NH. Число логических серверов  $NS = 100$ , радиус зон поиска  $MR = 3$ . Результаты сведены в таблицу 4.5 и представлены также в виде графика (рис. 4.11):

Таблица 4.5

<b>NH</b>	5	10	15	20	25
<b>t, сек</b>	22	65	114	167	221

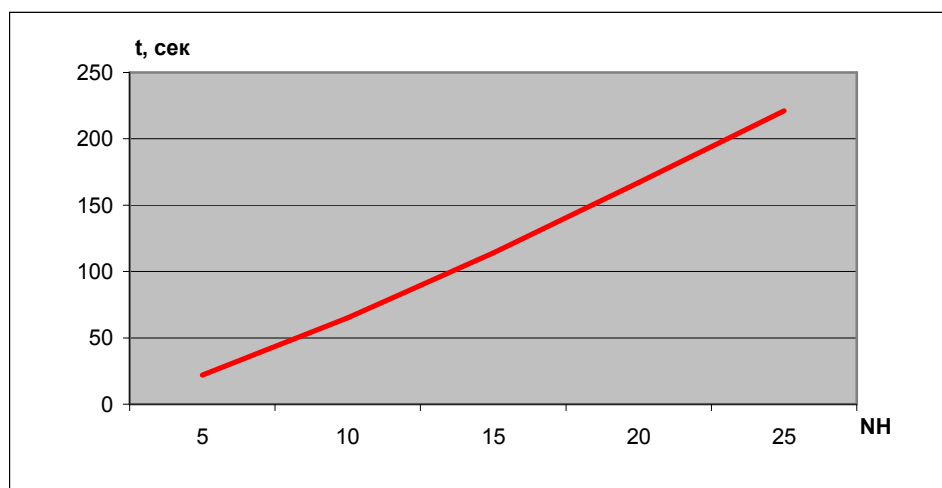


Рис. 4.11. График зависимости времени решения от числа компьютеров

Из графиков 4.10 и 4.11 видно, что скорость роста времени решения задач практически не меняется при увеличении радиуса зон поиска. Графики 4.10 и 4.11 подтверждают графики 3.3 и 3.4, построенные по формуле 3.13, выведенной в главе 3 для оценки объема перебора при решении задач.

### **Эксперименты по базовой оценке качества решений**

Данная серия экспериментов проводилась с целью проверки оценок (формулы 3.14 и 3.15) нижней границы точности субоптимальных решений на двух простых задачах поиска распределения при наличии дополнительных ограничений-исключений и без них для различных значений параметра радиуса зон поиска  $MR$ . Следующие исходные данные не изменялись и были одними и теми же во всех экспериментах:

- Число типов ресурсов  $NC = 1$ . Требование базовой ОС нулевое ( $V_1 = 0$ ).
- Число компьютеров  $NH = 1$ . Число логических серверов  $NS = 21$ .
- Оптимизация проводится по единственному типу ресурса ( $O_1 = 1$ ).
- Используется одна стартовая точка ( $MS = 1$ ).

Эксперименты 2.1.1 – 2.1.38. Оценка загрузки ресурса компьютера в различных задачах и при двух значениях параметрах радиуса зон поиска  $MR$ . Базовый уровень ресурса равен  $R_{1,1}$  – является варьируемым параметром. Требование 1-го логического сервера равно  $Q_{1,1}$  – является варьируемым параметром. Для всех остальных логических серверов требование равно 5000 ( $Q_{1,j} = 5000, j = 2..21$ ). Дополнительных ограничений-исключений нет ( $NX = 0$ ), соответственно, матрица  $\{E_{d,j}\}$ ,  $d = 1..NX, j = 1..NS$ , отсутствует.

**Таблица 4.6**

$\lambda$	1	2	3	...	10	...	17	18	19
$R_{1,1}$	10000	15000	20000	...	55000	...	90000	95000	100000
$Q_{1,1}$	5001	10001	15001	...	50001	...	85001	90001	95001
Загрузка ресурса при $MR = \lambda$	50%	66,67%	75%	...	90,91%	...	94,45%	94,74%	95%
Загрузка ресурса при $MR = NS$	100%	100%	100%	...	100%	...	100%	100%	100%
Точность субоптимального решения	0,5	0,6667	0,75	...	0,9091	...	0,9445	0,9474	0,95



Из таблицы видим, что при значении параметра радиуса зон поиска  $MR = \lambda$  ( $1 \leq \lambda \leq 19$ ), точность субоптимального решения составляет  $\lambda / (\lambda + 1)$ , что подтверждает оценку точности по формуле 3.14.

Эксперименты 2.2.1 – 2.2.38. Оценка загрузки ресурса компьютера в различных задачах и при двух значениях параметрах радиуса зон поиска  $MR$ . Базовый уровень ресурса равен 100000 ( $R_{1,1} = 100000$ ). Требование 1-го логического сервера равно  $Q_{1,1}$  – является варьируемым параметром. Для всех остальных логических серверов требование равно 5000 ( $Q_{1,j} = 5000$ ,  $j = 2..21$ ). Задано 20 дополнительных ограничений-исключений ( $NX = 20$ ), соответственно, задана матрица исключений  $\{E_{d,j}\}$ ,  $d = 1..NX$ ,  $j = 1..NS$ , исключающая размещение логического сервера  $S_1$  с любым другим:

Таблица 4.7

S1	S2	S3	S4	...	S19	S20	S21
1	1	0	0	...	0	0	0
1	0	1	0	...	0	0	0
1	0	0	1	...	0	0	0
...	...	...	...	...	...	...	...
1	0	0	0	...	1	0	0
1	0	0	0	...	0	1	0
1	0	0	0	...	0	0	1

Результаты экспериментов сведены в таблицу:

Таблица 4.8

$\lambda$	1	2	3	...	10	...	17	18	19
$Q_{1,1}$	5001	10001	15001	...	50001	...	85001	90001	95001
Загрузка ресурса при $MR = \lambda$	5%	10%	15%	...	50%	...	85%	90%	95%
Загрузка ресурса при $MR = NS$	100%	100%	100%	...	100%	...	100%	100%	100%
Точность субоптимального решения	0,05	0,1	0,15	...	0,5	...	0,85	0,9	0,95

Из таблицы видим, что при значении параметра радиуса зон поиска  $MR = \lambda$  ( $1 \leq \lambda \leq 19$ ), точность субоптимального решения составляет  $\lambda / (NS - 1)$ , что подтверждает оценку точности по формуле 3.15.

### **Эксперименты с декомпозицией исходной задачи**

При решении задач реорганизации с большим количеством логических серверов и компьютеров, очевидно, возможно использование декомпозиции: разбиение исходного серверного парка на множество более мелких парков и постановка и решение задач реорганизации в каждом из них. Использование декомпозиции, безусловно, дает существенное сокращение времени решения исходной задачи, поскольку множество задач меньшей размерности вместе решаются намного быстрее, чем одна задача большой размерности, но с другой стороны декомпозиция может дать существенное ухудшение конечного результата – суммарного объема освободившегося оборудования.

В рамках данной диссертации нет возможности описать исходные данные и решения множества различных задач с использованием декомпозиции и без нее, тем не менее, рассмотрим две простейшие (но представительные) задачи и решим их с использованием декомпозиции и без нее и сравним результаты. Для сокращения объема описания исходных данных задач выпишем те исходные данные, которые были одинаковыми во всех экспериментах:

- Тип ресурсов один ( $NC = 1$ ).
- Число логических серверов  $NS = 100$ , число компьютеров  $NH = 100$ .
- Дополнительные ограничения-исключения не используются ( $NX = 0$ ), соответственно, матрица  $\{E_{d,j}\}$ ,  $d = 1..NX$ ,  $j = 1..NS$ , отсутствует.
- Оптимизация проводится по единственному типу ресурса ( $O_1 = 1$ ).
- Используется одна стартовая точка ( $MS = 1$ ).
- Требование базовой ОС нулевое ( $V_1 = 0$ ).

Эксперименты 3.1.1 – 3.1.6. Для всех компьютеров базовый уровень ресурса равен 100 ( $R_{1,k} = 100$ ,  $k = 1..100$ ). Для всех логических серверов требование равно 10 ( $Q_{1,j} = 10$ ,  $j = 1..100$ ). Решаем задачу с использованием декомпозиции и без нее при радиусах зон поиска  $MR = 1, 2$  и  $3$ .

Сначала решаем задачу целиком (без использования декомпозиции). Получаем следующее распределение логических серверов по компьютерам, причем результаты одинаковые для всех значений радиуса зон поиска:

Таблица 4.9

<b>H1</b>	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
...	...	...	...	...	...	...	...	...	...	...
<b>H10</b>	S91	S92	S93	S94	S95	S96	S97	S98	S99	S100
<b>H11</b>	-									
...	...									
<b>H100</b>	-									

Компьютеры  $H_1 - H_{10}$  задействованы, остальные компьютеры  $H_{11} - H_{100}$  не задействованы. Таким образом, 90 освободившихся компьютеров.

Теперь же разобьем исходный серверный парк на 10 более мелких серверных парков  $P_1: \{H_1...H_{10}\} \dots P_{10}: \{H_{91}...H_{100}\}$  и найдем распределение логических серверов по компьютерам внутри каждого парка.

В результате, распределение внутри  $\mu$ -го серверного парка  $P_\mu$ ,  $\mu = 1..10$ , причем результаты одинаковые для всех значений радиуса зон поиска:

Таблица 4.10

<b>H(1+<math>\lambda</math>)</b>	S(1+ $\lambda$ )	S(2+ $\lambda$ )	S(3+ $\lambda$ )	S(4+ $\lambda$ )	S(5+ $\lambda$ )	S(6+ $\lambda$ )	S(7+ $\lambda$ )	S(8+ $\lambda$ )	S(9+ $\lambda$ )	S(10+ $\lambda$ )
<b>H(2+<math>\lambda</math>)</b>	-									
...	...									
<b>H(10+<math>\lambda</math>)</b>	-									

Где  $\lambda = 10 * (\mu - 1)$  – смещение при вычислении индексов компьютеров и логических серверов внутри исходного серверного парка. Внутри парка  $P_\mu$  компьютер  $H_{1+\lambda}$  задействован, остальные компьютеры  $H_{2+\lambda} - H_{10+\lambda}$  не задействованы. Всего 9 освободившихся компьютеров в парке  $P_\mu$ .

Тогда, возвращаясь к исходной задаче и подводя итоги ее решения, видим, что в результате задействованы 10 компьютеров  $H_1, H_{11}, H_{21}, H_{31}, H_{41}, H_{51}, H_{61}, H_{71}, H_{81}, H_{91}$  остальные 90 компьютеров остаются свободными – то есть решение не хуже, чем при решении исходной задачи целиком.

Теперь же сведем в таблицу время решения задачи при использовании декомпозиции и без нее, при радиусах зон поиска  $MR = 1, 2$  и  $3$ :

Таблица 4.11

	MR = 1	MR = 2	MR = 3
Время решения всей задачи без ее декомпозиции	1,11 сек	22,28 сек	536 сек
Суммарное время решения подзадач при декомпозиции исходной задачи	0,0078 сек	0,0183 сек	0,0482 сек

Очевидно, что декомпозиция дает существенный выигрыш по времени решения исходной задачи, причем, чем больше радиус зон поиска, тем больше выигрыш. Таким образом, в данной задаче применение декомпозиции оказалось целесообразным.

Эксперименты 3.2.1 – 3.2.6. Для компьютеров  $H_1 - H_{50}$  базовый уровень ресурса равен 50, а для компьютеров  $H_{51} - H_{100}$  базовый уровень ресурс равен 100: ( $R_{1,k} = 50, k = 1..50; R_{1,k} = 100, k = 51..100$ ). Для логических серверов  $S_1 - S_{50}$  требование равно 30, а для логических серверов  $S_{51} - S_{100}$  требования равно 60: ( $Q_{1,j} = 30, k = 1..50; Q_{1,j} = 60, k = 51..100$ ). Решаем задачу с использованием декомпозиции и без нее при радиусах зон поиска  $MR = 1, 2$  и  $3$ .

Сначала решаем задачу целиком (без использования декомпозиции). Получаем следующее распределение логических серверов по компьютерам, причем результаты одинаковые для всех значений радиуса зон поиска:

Таблица 4.12

<b>H1</b>	-	-
...	...	...
<b>H50</b>	-	-
<b>H51</b>	S1	S51
...	...	...
<b>H100</b>	S50	S100

Компьютеры  $H_{51} - H_{100}$  задействованы, остальные компьютеры  $H_1 - H_{50}$  не задействованы. Таким образом, 50 освободившихся компьютеров.

Теперь же разобьем исходный серверный парк на 10 более мелких серверных парков  $P_1: \{H_1...H_{10}\} \dots P_{10}: \{H_{91}...H_{100}\}$  и найдем распределение логических серверов по компьютерам внутри каждого парка.

В результате распределение внутри  $\mu$ -го серверного парка  $P_\mu, \mu = 1..10$ , причем результаты одинаковые для всех значений радиуса зон поиска:

Таблица 4.13

<b>H(1+<math>\lambda</math>)</b>	<b>S(1+<math>\lambda</math>)</b>
...	...
<b>H(10+<math>\lambda</math>)</b>	<b>S(10+<math>\lambda</math>)</b>

Где  $\lambda = 10 * (\mu - 1)$  – смещение при вычислении индексов компьютеров и логических серверов внутри исходного серверного парка. Внутри каждого серверного парка  $P_\mu$ ,  $\mu = 1..10$ , в результате распределения все 10 компьютеров  $H_{1+\lambda} - H_{10+\lambda}$  задействованы, ни одного освободившегося компьютеры.

Тогда, возвращаясь к исходной задаче и подводя итоги ее решения, видим, что в результате задействованы все 100 компьютеров. Освободившихся компьютеров нет, декомпозиция дает бесконечное ухудшение результата, если судить по числу освободившихся компьютеров (50 / 0).

Теперь же сведем в таблицу время решения задачи с использованием декомпозиции и без нее, при радиусах зон поиска  $MR = 1, 2$  и  $3$ :

Таблица 4.14

	MR = 1	MR = 2	MR = 3
Время решения всей задачи без ее декомпозиции	0,849 сек	25,4 сек	694 сек
Суммарное время решения подзадач при декомпозиции исходной задачи	0,0083 сек	0,0191 сек	0,0431 сек

Очевидно, что декомпозиция дает существенный выигрыш по времени решения исходной задачи, причем, чем больше радиус зон поиска, тем больше выигрыш, но результат при этом ухудшается и становится нулевым.

Таким образом, в данной задаче применение декомпозиции оказалось совершенно нецелесообразным.

Подводя итог можно сказать, что эффект от использования декомпозиции целиком зависит от исходных данных задачи, и если в простейших задачах проанализировав исходные данные можно предварительно оценить целесообразность применения декомпозиции, то при более сложных исходных данных сделать это будет уже крайне затруднительно. Можно, конечно, использовать различные варианты декомпозиции и сравнивать получаемые результаты, но то насколько ухудшается решение при использовании какого-либо варианта декомпозиции, можно будет выяснить, только решив задачу целиком без использования декомпозиции.

### **Решение производственных задач с использованием методики**

Разработанная методика была использована при решении ряда производственных задач реорганизации серверного парка, однако, в рамках данной диссертации нет возможности перечислить их всех. Тем не менее, в рамках данной главы мы рассмотрим одну из решенных задач, кроме того, еще пять задач приведены в приложении 1.

Постановка задачи. Рассмотрим некоторую компанию, которая занимается разработкой проектов по строительству сооружений различных видов: начиная от гостиниц, офисных зданий, торговых центров, заканчивая скоростными магистралями, мостами и сооружениями для защиты от наводнений. Изначально компания располагалась в пределах одного небольшого арендуемого здания, но по мере увеличения числа проектов, руководство арендовало дополнительные помещения в других зданиях, расположенных в пределах одного города: отдельный офис, штат сотрудников, техническое оборудование, в том числе компьютерное оборудование и локальная вычислительная сеть, для каждого нового проекта. Локальные сети множества офисов были связаны друг с другом с помощью защищенных виртуальных каналов в общедоступной сети Интернет. В каждом офисе было некоторое множество компьютеров сотрудников и 3-4 компьютера, выполняющих функции сервера DNS, DHCP, контроллера домена, файлового сервера, сервера печати и т.п. Впоследствии в связи с завершением строительства собственного многоэтажного здания для компании, персонал и оборудование всех офисов были размещены в новом здании. Для серверного оборудования и ключевого коммутационного оборудования было выделено отдельное помещение. Поскольку серверы множества офисов оказались в пределах одного помещения, то руководство из экономических соображений, а также с учетом вопросов вирусной и сетевой безопасности, совместимости программ, нежелательности перенастройки компьютеров сотрудников, приняло решение о реорганизации серверного парка с применением технологии виртуальных машин.

Решение задачи. Поставленная задача решалась с использованием разработанной методики. Рассмотрим кратко решение задачи:

### Первичный сбор информации

Число компьютеров и логических серверов было равно 15 ( $NH = NS = 15$ ). Критичными ресурсами являлись процессор, память и диск. Сетевая подсистема и каналы передачи данных не являлись критичными, поскольку сетевой обмен между серверами и между серверами и пользовательскими компьютерами был невелик: не более 50-100 Кб/час. Кроме того, все компьютеры располагались внутри одного здания и были связаны высокоскоростной и надежной локальной вычислительной сетью. Соответственно, число типов ресурсов было равно 3 ( $NC = 3$ ).

Базовые уровни ресурсов компьютеров  $\{R_{i,k}\}$ ,  $i = 1..3$ ,  $k = 1..15$ :

Таблица 4.15а

	<b>H1</b>	<b>H2</b>	<b>H3</b>	<b>H4</b>	<b>H5</b>	<b>H6</b>	<b>H7</b>	<b>H8</b>
<b>Память, MB</b>	512	256	256	512	1024	1024	384	640
<b>Диск, MB</b>	20000	40000	30000	120000	120000	120000	60000	40000
<b>Процессор, МТОРС</b>	5200	5200	4500	4800	4300	3700	2200	2333
	<b>H9</b>	<b>H10</b>	<b>H11</b>	<b>H12</b>	<b>H13</b>	<b>H14</b>	<b>H15</b>	
<b>Память, MB</b>	512	512	384	384	768	640	512	
<b>Диск, MB</b>	80000	40000	25000	40000	50000	80000	120000	
<b>Процессор, МТОРС</b>	900	1200	2300	2300	3800	3800	4200	

Требования логических серверов  $\{Q_{i,j}\}$ ,  $i = 1..3$ ,  $j = 1..15$ :

Таблица 4.15б

	<b>S1</b>	<b>S2</b>	<b>S3</b>	<b>S4</b>	<b>S5</b>	<b>S6</b>	<b>S7</b>	<b>S8</b>
<b>Память, MB</b>	100	120	150	200	350	600	220	200
<b>Диск, MB</b>	9000	4500	2500	45000	32000	66000	20000	12000
<b>Процессор, МТОРС</b>	300	500	600	200	300	400	120	330
	<b>S9</b>	<b>S10</b>	<b>S11</b>	<b>S12</b>	<b>S13</b>	<b>S14</b>	<b>S15</b>	
<b>Память, MB</b>	125	220	120	200	270	340	120	
<b>Диск, MB</b>	35000	20000	10000	12000	22000	45000	90000	
<b>Процессор, МТОРС</b>	220	440	233	666	122	744	222	

Требования базовой ОС  $\{V_i\}$ ,  $i = 1..3$ :

Таблица 4.15в

	Базовая ОС
Память, МВ	96
Диск, МВ	2000
Процессор, МТОРS	10

В целях получения наибольшего сокращения объема оборудования было принято решение не вводить дополнительные логические серверы, дублирующие функции каких-либо других серверов. Однако, после анализа функций серверов было рекомендовано исключить одновременное размещение на одном компьютере следующих логических серверов:

- Серверы  $S_7$ ,  $S_8$ , и  $S_9$  не должны размещаться на одном компьютере.
- Серверы  $S_{10}$ ,  $S_{11}$  и  $S_{12}$  не должны размещаться на одном компьютере.
- Серверы  $S_{13}$ ,  $S_{14}$  и  $S_{15}$  не должны размещаться на одном компьютере.

Соответственно, число дополнительных ограничений было равно 3 ( $NX = 3$ ), а матрица дополнительных ограничений-исключений  $\{E_{d,j}\}$ ,  $d = 1..3$ ,  $j = 1..15$ , выглядела следующим образом:

Таблица 4.15г

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15
1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Наконец, в качестве типов ресурсов, для которых предпочтительно решение проблемы их неэффективного использования были выбраны память и дисковая подсистема. Соответственно, вектор маски  $\{O_i\}$ ,  $i = 1..3$ , выглядел следующим образом:

Таблица 4.15д

	Маска
Память, МВ	1
Диск, МВ	1
Процессор, МТОРS	0



### Получение первичного распределения

При помощи разработанной программы DTSOLVEX, реализующей алгоритм поиска распределения логических серверов по компьютерам, после ввода всех исходных данных и запуска решения, были получены следующие матрицы распределения и загрузки ресурсов компьютеров:

Таблица 4.15e

Матрица распределения					Матрица загрузки ресурсов		
					Память	Диск	Процессор
H1					0	0	0
H2					0	0	0
H3					0	0	0
H4	S10	S15			81,73%	93,22%	13,82%
H5	S2	S4	S6		99,14%	97,88%	25,64%
H6	S5	S7	S14		98,06%	82,20%	31,54%
H7					0	0	0
H8					0	0	0
H9					0	0	0
H10					0	0	0
H11	S1	S11			76,39%	82,61%	23,28%
H12	S3	S9			95,49%	98,68%	35,81%
H13	S8	S12	S13		99,70%	95,83%	29,50%
H14					0	0	0
H15					0	0	0

### Первичная и вторичная оценка результатов

Из матрицы распределения видно, что задействовано 6 компьютеров: H<sub>4</sub>, H<sub>5</sub>, H<sub>6</sub>, H<sub>11</sub>, H<sub>12</sub>, H<sub>13</sub>, остальные 9 компьютеров освободились: H<sub>1</sub>, H<sub>2</sub>, H<sub>3</sub>, H<sub>7</sub>, H<sub>8</sub>, H<sub>9</sub>, H<sub>10</sub>, H<sub>14</sub>, H<sub>15</sub>. Таким образом, в первом приближении было возможно сокращение числа компьютеров в 2.5 раза (15 / 6).

Все логические серверы были распределены, причем не было ситуаций, когда какой-либо логический сервер распределен на тот же компьютер, на котором он работал, и при этом на этот компьютер больше ничего не размещается. Также отсутствовали ситуации, когда какой-либо логический сервер никуда не распределен, а соответствующий компьютер не задействован.

Во время сбора первичной информации требования логических серверов были заранее завышены на 20% (чтобы был некоторый запас по ресурсам после реорганизации), и поэтому поиск вторичного плана распределения при завышенных требованиях логических серверов не проводился.

### Реорганизация серверного парка и качество его функционирования

После получения распределения логических серверов по компьютерам заказчик принял решение о целесообразности проведения реорганизации серверного парка. После этого была проведена реорганизация серверного парка в соответствии с полученным распределением на физические компьютеры. Тестирование реорганизованного серверного парка не выявило каких-либо существенных проблем, качество функционирования вполне удовлетворяло требованиям заказчика. Таким образом, поставленная задача была решена.

Теперь же было бы интересным сравнить распределение, полученное при использовании методики, с распределением, которое можно получить вручную – последовательное “заполнение” компьютеров логическими серверами с соблюдением ограничений по ресурсам и дополнительных ограничений на размещение логических серверов на одном и том же компьютере. Распределение, полученное вручную, имеет следующий вид:

Таблица 4.15ж

Матрица распределения					Матрица загрузки ресурсов		
					Память	Диск	Процессор
H1	S1	S2	S3		88,94%	88,89%	26,97%
H2	S9				78,13%	92,11%	4,24%
H3	S11				75,00%	35,71%	5,19%
H4	S4	S8			96,15%	48,31%	11,06%
H5	S5	S7	S10		85,13%	61,02%	20,05%
H6	S6	S12			86,12%	66,10%	28,89%
H7	S13				93,75%	37,93%	5,57%
H8					0	0	0
H9	S14				81,73%	57,69%	83,60%
H10					0	0	0
H11					0	0	0
H12					0	0	0
H13					0	0	0
H14					0	0	0
H15	S15				28,85%	76,27%	5,30%

Очевидно, что распределение, полученное вручную существенно хуже: число задействованных компьютеров сократилось только в 1.66 раза (15 / 9). В распределении, полученном программой DTSOLVEX при решении задачи, сокращение числа задействованных компьютеров было в 2.5 раза (15 / 6).

**Внедрение.** Разработанная методика и программа использовались в ИВЦ МЭИ (ТУ) при решении задач реорганизации серверного парка с целью повышения эффективности использования вычислительных ресурсов компьютеров. Часть решенных задач приведены в приложении 1. Акт о внедрении находится в приложении 6.

ИВЦ МЭИ (ТУ) является подразделением МЭИ (ТУ), состоящее из множества специалистов, занимающихся исследованиями по различным направлениям современных информационных технологий, поддержкой и развитием информационно – вычислительной сети МЭИ (ТУ), сопровождением рабочих мест пользователей МЭИ, сопровождением компьютерных учебных классов и разработкой прикладного программного обеспечения. Серверный парк ИВЦ МЭИ (ТУ) содержит более 100 компьютеров, выполняющих те или иные серверные функции, обеспечивающие безопасный доступ в Интернет, быстрый и эффективный документооборот, возможности использования защищенных корпоративных почтовых систем, публикацию информационных ресурсов в Интернет и т.д. В связи с постоянным ростом количества серверов ввиду развития информационных технологий, возрастания потребностей пользователей, расширения круга решаемых задач, разработки нового прикладного программного обеспечения, периодически возникают задачи расширения и реорганизации той или иной части серверного парка, а также задачи развертывания дополнительных парков с применением технологии виртуальных машин. В такой ситуации отсутствие какой-либо методики значительно усложняет и увеличивает общее время решения задач реорганизации или развертывания серверного парка, кроме того, получаемые распределения логических серверов по компьютерам далеки от оптимальных, и, наконец, возникает множество непредсказуемых негативных последствий, существенно снижающих качество функционирования серверного парка. Внедрение в ИВЦ МЭИ (ТУ) методики, разработанной в данной диссертации, позволило решить эти проблемы и значительно снизить совокупную стоимость владения серверным парком.

## Выводы по главе 4

Данная глава была посвящена разработке программного инструмента, реализующего наиболее сложную и трудоемкую часть разработанной в главе 2 методики – поиск оптимального распределения логических серверов на физические компьютеры, алгоритм которого был разработан в главе 3.

В первой части главы приведен краткий обзор особенностей интерфейса пользователя современного программного обеспечения, принципов программирования, особенностей современных многозадачных операционных систем и средств разработки, а также особенностей решаемой прикладной задачи. В результате обзора сформулирован ряд требований к разрабатываемой программе. Во второй части главы кратко рассмотрены структура, особенности реализации многопоточности и логика работы разработанной программы, которые достаточно наглядно показывают, что все требования соблюдены.

В третьей части главы рассмотрено несколько серий экспериментов, а также рассмотрена одна из решенных производственных задач:

- Результаты экспериментов по оценке времени решения задачи подтвердили оценку объема перебора, полученную аналитическим путем в главе 3.
- Результаты экспериментов с использованием и без использования декомпозиции исходной задачи большой размерности на множество задач меньших размерностей показали, что декомпозиция существенно снижает время решения задач. Однако, при этом качество решения исходной задачи может резко ухудшиться по сравнению с решением задачи без использования декомпозиции, и ввиду сложной зависимости результатов решения от исходных данных невозможно дать каких-либо рекомендаций по применению декомпозиции.
- Сравнение результатов решения производственной задачи при помощи программного инструмента и вручную наглядно показало существенное преимущество качества получаемых решений при использовании программного инструмента, реализующего алгоритм решения задачи поиска распределения, предложенного в главе 3.

## Заключение

В диссертационной работе разработана методика по реорганизации серверных парков с целью повышения эффективности использования вычислительных ресурсов компьютеров. Методика также включает в себя предварительную оценку целесообразности проведения реорганизации.

В диссертационной работе проведен обзор существующих способов повышения эффективности использования вычислительных ресурсов, рассмотрены несколько существующих подходов, выявлены их основные недостатки и сделан вывод о том, что использование технологии виртуальных машин является наиболее приемлемым вариантом. Технология виртуальных машин предоставляет новые возможности по построению или реорганизации сетевой инфраструктуры. Виртуальные машины, позволяющие запускать несколько достаточно хорошо изолированных ОС на одном компьютере, решают проблемы информационной безопасности, совместимости приложений, неизменности логической структуры, а также избавляют от необходимости подбора адекватных аппаратных решений, поскольку размещение на одном компьютере нескольких логических серверов позволяет существенно повысить эффективность использования вычислительных ресурсов.

Тем не менее, при более подробном обзоре технологии виртуальных машин выявлено, что технология виртуальных машин имеет и некоторые недостатки: снижение надежности функционирования серверных систем, небольшие накладные расходы ресурсов для размещения базовой ОС, возрастание времени отклика и времени обслуживания запросов для серверных систем. Помимо недостатков возникают дополнительные сложности при применении технологии: сложность поиска оптимального распределения логических серверов по физическим компьютерам и сложность переноса серверных систем с физической аппаратной платформы на виртуальную платформу в силу ряда особенностей операционных систем. Все эти проблемные вопросы были подробно рассмотрены в рамках диссертации и в разработанной методике учтены все вышеперечисленные проблемы.

В рамках диссертации получены следующие результаты:

- Для поставленной проблемы повышения эффективности использования вычислительных ресурсов разработана методика реорганизации серверного парка, позволяющая решить эту проблему, на базе модели задачи поиска оптимального распределения логических серверов по компьютерам при применении технологии виртуальных машин.
- Разработана технология переноса операционных систем с физической аппаратной платформы на виртуальную платформу, которая делает методику реорганизации серверного парка при применении технологии виртуальных машин технически реализуемой.
- Разработана математическая модель задачи поиска оптимального распределения множества логических серверов на множество компьютеров при применении технологии виртуальных машин.
- Предложена схема разбиения, с использованием аппарата динамического программирования, исходной задачи на множество подзадач поиска распределения логических серверов на один компьютер, являющихся задачами условной псевдобулевой оптимизации. Для решения подзадач предложен модифицированный метод локального поиска с использованием функции штрафов, управляемого радиуса зон поиска и множества случайных стартовых точек. Выполнены анализ сходимости, оценки объема перебора и качества решений для предложенного метода решения.
- Разработана программная реализация алгоритма решения задачи поиска оптимального распределения логических серверов по компьютерам.
- Проведено экспериментальное исследование на ряде модельных примеров и производственных задач при внедрении результатов диссертации с использованием разработанного программного обеспечения и предложенной методики.

Разработанная методика имеет практическую ценность, поскольку она предлагает самодостаточный и полноценный подход для решения задач по реорганизации серверного парка с целью повышения эффективности использования вычислительных ресурсов. Методика охватывает практически все этапы решения этой задачи:

- Первичный сбор информации, поиск первичного варианта распределения с учетом вопросов надежности функционирования серверного парка, оценка первичного распределения логических серверов по физическим компьютерам, получение и оценка распределения при прогнозировании ситуаций некачественного функционирования реорганизованного серверного парка и принятие совместно с заказчиком окончательного решения о целесообразности проведения реорганизации.
- Внедрение первичного варианта распределения, оценка качества функционирования серверного парка после реорганизации, в случае неудовлетворительности поиск причин и внесение дополнительных ограничений и условий для задачи распределения, получение скорректированных вариантов распределения, их оценка, внедрение и повторная оценка качества функционирования серверного парка.

Конечный специалист, поставленный перед проблемой неэффективного использования ресурсов компьютеров серверного парка заказчика, может использовать данную методику для эффективного решения поставленной задачи. Более того, методика разработана таким образом, что она также может быть легко применена при первичном развертывании (проектировании с нуля нового) серверного парка за счет того, что в математическом аппарате число логических серверов и физических компьютеров различается (при реорганизации же изначально  $NS$  логических серверов функционируют на  $NH$  физических компьютерах, то есть  $NS = NH$ ).

## Литература

1. Зубанов Ф.В. Microsoft Windows 2000. Планирование, развертывание, установка. – М.: ИТД “Русская Редакция”, 2000.
2. Медведовский И.Д., Семьянов П.В., Платонов В.В. Атака через Интернет. – М.: НПО “Мир и семья-95”, 1997.
3. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы. – 2-е изд. – СПб.: Издательский дом “Питер”, 2004.
4. Олифер В.Г., Олифер Н.А. Сетевые операционные системы – СПб.: Издательский дом “Питер”, 2001.
5. Тормасов А. Виртуализация операционных систем // Открытые системы. – 2002. – № 1. – С. 15-22.
6. Гук М. Процессоры Intel: от 8086 до Pentium II. – СПб.: “Питер Ком”, 1997.
7. Гук М., Юров В.. Процессоры Intel Pentium IV, Pentium III, AMD Athlon и Duron. – СПб.: Издательский дом “Питер”, 2001.
8. Intel Corporation. Official site with technical documentation. – Santa Clara: Intel Corporation, CA, USA, 2005. <http://www.intel.com>
9. VMware Inc. Official site with technical documentation. – Palo Alto: VMware Inc., CA, USA, 2005. <http://www.vmware.com>
10. Костромин В. Виртуальный компьютер. Обмен данными с реальными миром // Открытые системы. – 2001. – № 11. – С. 17-25.
11. Microsoft Corporation. Official site with technical documentation. – Redmond: Microsoft Corporation, WA, USA, 2005. <http://www.microsoft.com/technet>
12. Гук М. Аппаратные средства IBM PC. Энциклопедия. – 2-е изд. – СПб.: Издательский дом “Питер”, 2002.
13. Беллман Р., Дрейфус С. Прикладные задачи динамического программирования. – М.: Наука, 1965.
14. Беллман Р., Калаб Р. Динамическое программирование и современная теория управления. – М.: Наука, 1969.



15. Бронштейн И.Н. Семендяев К.А. Справочник по математике для инженеров и учащихся втузов. – 13-е изд. – М.: Наука, Гл. изд. физ-мат. лит., 1986.
16. Дж.П.Мюллер, И.Чоудри. Microsoft Windows 2000. Настройка и оптимизация производительности. / Пер. с англ. – М.: Изд. "Эком", 2001.
17. Стивен Норткатт, Джуди Новак. Обнаружение вторжений в сеть. / Пер. с англ. – М.: Изд. "Лори", 2001.
18. Microsoft Press. Учебный курс MCSE Windows 2000 Server. / Пер. с англ. – М.: Изд. ИТД "Русская Редакция", 2000.
19. Сойер Б. Проектирование экспертных систем. – М.: Изд. "Наука", 1993.
20. Джефффри Рихтер. Windows для профессионалов / Пер. с англ. – 4-е изд. – СПб.: Питер, 2004.
21. Еремин И.И., Астафьев Н.Н. Введение в теорию линейного и выпуклого программирования. – М.: Наука, 1979.
22. Карманов В.Г. Математическое программирование. – М.: Наука, 1986.
23. Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность – М.: Мир, 1985.
24. Антамошкин А.Н., Масич И.С. Эффективные алгоритмы условной оптимизации монотонных псевдобулевых функций // Вестник СибГАУ. – 2003. – № 4. – С. 15-24.
25. Семенкина О.Э., Жидков В.В. Оптимизация управления сложными системами методом обобщенного локального поиска – М.: Изд. "МАКС Пресс", 2002.
26. Барский А.Б. Параллельные процессы в вычислительных системах. Планирование и организация. – М.: Радио и связь, 1990.
27. Головкин Б.А. Расчет характеристик и планирование параллельных вычислительных процессов. – М.: Радио и связь, 1983.
28. Гэри М.Р., Джонсон Д.С. Вычислительные машины и труднорешаемые задачи. – М.: Мир, 1982.
29. Липаев В.В. Распределение ресурсов в вычислительных системах. – М.: Статистика, 1979.

30. Михалевич В.С., Кукса А.И. Методы последовательной оптимизации в дискретных сетевых задачах распределения ресурсов. – М.: Наука, 1983.
31. Танаев В.С., Гордон В.С., Шафранский Я.М. Теория расписаний. Одностадийные системы. – М.: Наука, 1984.
32. Теория расписаний и вычислительные машины. / Под ред. Э.Г. Коффмана. – М.: Наука, 1984.
33. Топорков В.В. Модели распределенных вычислений. – М.: ФИЗМАТЛИТ, 2004.
34. Топорков В.В. Оптимизация распределения ресурсов в системах жесткого реального времени // Известия РАН. Теория и системы управления. – 2004. – № 3 – С. 61-71.
35. Топорков В.В. Разрешение коллизий параллельных процессов в масштабируемых вычислительных системах // Автоматика и телемеханика. – 2003. – № 5. – С. 180-189.
36. Barth P. A. Davis-Putnam. Based Enumeration Algorithm for Linear Pseudo-Boolean Optimization. Max-Planck-Institut für Informatik, January 1995.
37. Boros E., Hammer P.L. Pseudo-Boolean Optimization. Rutcor Research Report, RRR 48-2001, September 2001.
38. Boros E., Lari I., Simeone B. Block Linear Majorants in Quadratic 0-1 Optimization. Rutcor Research Report, RRR 18-2000, March 2000.
39. Ekin O., Foldes S., Hammer P.L., Hellerstein L. Equational Theories of Boolean Function. DIMACS Technical Report 97-79, December 1997.
40. Hammer P.L., Peled U.N. On the Maximization of a Pseudo-Boolean Function. Journal of the Association for Computing Machinery, Vol. 19, No. 2, April 1972.
41. Джим Макби. Microsoft Exchange 5.5 Server для профессионалов. Проблемы и решения. / Пер. с англ. – СПб.: “Питер”, 2000.
42. Сью Мошер. Руководство по Microsoft Exchange. / Пер. с англ. – М.: “Русская Редакция”, 1998.
43. Пол Робишо. Администрирование инфраструктуры сети Windows 2000. / Пер. с англ. – М.: “Лори”, 2001.

44. Энил Десай. Администрирование служб каталога Windows 2000. / Пер. с англ. – М.: “Лори”, 2001.
45. Microsoft Press. Учебный курс Internet Security And Acceleration Server 2000. / Пер. с англ. – М.: “Русская Редакция”, 2002.
46. Microsoft Press. Безопасность сети на основе Windows 2000. Учебный курс MCSE. / Пер. с англ. – М.: “Русская Редакция”, 2002.
47. Microsoft Press. Сети TCP/IP. Ресурсы MS Windows 2000 Server. / Пер. с англ. – М.: “Русская Редакция”, 2001.
48. Microsoft Press. Распределенные системы. Ресурсы MS Windows 2000 Server. / Пер. с англ. – М.: “Русская Редакция”, 2001.
49. Microsoft Press. Сопровождение сервера. Ресурсы MS Windows 2000 Server. / Пер. с англ. – М.: “Русская Редакция”, 2001.
50. Microsoft Press. Межсетевое взаимодействие. Ресурсы MS Windows 2000 Server. / Пер. с англ. – М.: “Русская Редакция”, 2002.
51. Керниган Б. Пайк Р. Практика программирования. / Пер. с англ. – СПб.: “Невский Диалект”, 2001.
52. Антлг Элианс. Принципы объектно-ориентированной разработки программ. / Пер. с англ. – М.: “Вильямс”, 2002.
53. Джон Макгрегор, Дэвид Сайке. Тестирование объектно-ориентированного программного обеспечения. / Пер. с англ. – М.: ТИД “ДС”, 2002.
54. Сэм Канер. Тестирование программного обеспечения. / Пер. с англ. – Киев: “Диасофт”, 2000.
55. К. Ауэр. Р. Миллер. Практическое руководство по экстремальному программированию. / Пер. с англ. – СПб.: “Питер”, 2003.
56. Microsoft Press. Архитектура Microsoft Windows для разработчиков / Пер. с англ. – М.: “Русская Редакция”, 1998.
57. Ари Каплан, Мортер Нильсен. Windows 2000 Изнутри. / Пер. с англ. – М.: “ДМК”, 2000.
58. Ал. Вильямс. Системное программирование в Windows 2000. / Пер. с англ. – СПб.: “Питер”, 2001.

59. М. Руссинович. Соломон Д. Внутреннее устройство Microsoft Windows 2000. / Пер. с англ. – М.: “Русская Редакция”, 2004.
60. Румянцев П.В. Основы программирования в Win32 API. – М.: “Радио и Связь”, 1998.
61. Д.Б. Поляков, И.Ю. Круглов. Программирование в среде Турбо Паскаль версии 5.5. – М.: Изд-во МАИ, 1992.
62. В.В. Фаронов. Турбо Паскаль. Книга 1. “Основы Турбо Паскаля”. – М.: “МВТУ - ФЕСТО ДИДАКТИК”, 1992.
63. Федоров А., Рогаткин Д. Borland Pascal в среде Windows. – Киев: “Диалектика”, 1993.
64. А.М. Епанешников, В.А. Епанешников. Программирование в среде Turbo Pascal 7.0. – 3-е изд. стер. – М.: “ДИАЛОГ - МИФИ”, 1996.
65. Федоров А.Г. Создание Windows-приложений в среде Delphi. – М.: “Компьютер Пресс”, 1995.
66. Том Сван. Основы программирования на Delphi для Windows 95. / Пер. с англ. – Киев: “Диалектика”, 1996.
67. Федоров А.Г. Delphi 2.0 для всех. – М.: “Компьютер Пресс”, 1997.
68. Тексейра С. Delphi 5.0. Основные методы и технологии программирования. / Пер. с англ. – М.: “Вильямс”, 2000.
69. Культин Н.Б. Самоучитель Delphi 6.0. Программирование на Object Pascal. – СПб.: “ВНУ-Санкт-Петербург”, 2001.
70. Гофман В. Delphi 6.0 в подлиннике. Руководство для разработчика. – СПб.: “ВНУ-Санкт-Петербург”, 2001.

## Приложение 1. Примеры использования разработанной методики.

Пример 1. Задан простейший парк, состоящий из 8 серверов, физически находящихся в одной подсети (отсутствуют какие-либо промежуточные маршрутизаторы) и связанных высокоскоростной (100 Мбит/с) и надежной ЛВС (подключены к одному коммутатору и находятся в одном серверном помещении). Заказчик, планируя замену устаревших 8 компьютеров серверного парка, приобрел 2 мощных компьютера с расчетом применения технологии виртуальных машин. Оценить то, возможен ли полный вывод из эксплуатации всех старых компьютеров. В положительном случае требуется произвести развертывание базовых ОС на компьютеры, произвести перенос логических серверов на виртуальные машины, разместить эти машины на соответствующих компьютерах и ввести их в тестовую эксплуатацию. Оценить качество функционирования логических серверов и в случае неудовлетворительности определить причины, внести коррекции в исходные условия, получить и внедрить скорректированный вариант распределения.

### Первичный сбор информации

Критичными ресурсами являются процессор, память и диск. Сетевая подсистема и каналы передачи данных не являются критичными.

Ресурсы новых физических компьютеров:

	H1	H2
Процессор, МТОPS	6200	5600
Память, МВ	2048	2048
Диск, МВ	100000	75000

Требования логических серверов (предоставлены специалистами заказчика со специально завышенными на 25% требованиями):

	S1	S2	S3	S4	S5	S6	S7	S8
Процессор, МТОPS	1200	1200	600	600	1500	1500	3000	1500
Память, МВ	256	256	128	128	256	256	512	256
Диск, МВ	5120	2048	4096	8192	8192	4096	32768	10240

Требования базовой операционной системы:

	Базовая ОС
Процессор, МТОPS	10
Память, МВ	128
Диск, МВ	2000

Специальные требования к надежности не предъявляются.

Оптимизация проводится по всем типам ресурсов.

### Получение первичного распределения

При помощи разработанной программы DTSOLVEX получаем следующее распределение серверов по новым компьютерам:

H1:	S2	S5	S6	S8
H2:	S1	S3	S4	S7

Все логические серверы распределены на два новых физических компьютера, соответственно, все старые компьютеры могут быть освобождены.

Загрузка ресурсов новых компьютеров:

	Процессор	Память	Диск
H1:	92,08%	53,33%	25,08%
H2:	96,60%	53,33%	68,73%

### Первичная оценка результатов

Поскольку все серверы распределены, причем распределены на новые компьютеры (старые вообще не рассматривались), то здесь нет ситуаций, когда какой-либо логический сервер распределен на тот же компьютер, на котором он работал, и при этом на этот компьютер больше ничего не размещается. Также отсутствуют ситуации, когда какой-либо логический сервер нераспределен, а соответствующий физический компьютер не задействован.

### Вторичная оценка результатов

На этапе сбора информации требования логических серверов были завышены на 25%, соответственно, получение варианта распределения для завышенных требований (на случай возможных корректировок после внедрения первичного варианта распределения) не требуется.

Таким образом, возможен полный вывод из эксплуатации всех старых компьютеров, и размещение логических серверов на два новых компьютера после их переноса на виртуальную платформу, и развертывания базовой ОС на новых компьютерах.

### Первичная реорганизация серверного парка

Производится подготовка базовой ОС на 2-х новых компьютерах в соответствии с первичным вариантом распределения. Кроме того, подготавливаются 8 виртуальных машин, выполняется резервное копирование данных логических серверов S1–S8 и подготовка их к переносу на виртуальную платформу, после этого выполняется перенос на виртуальные машины и окончательная настройка логических серверов. Наконец, виртуальные машины размещаются на компьютерах в соответствии с первичным вариантом распределения.

### Выявление негативных последствий реорганизации

В результате тестирования серверного парка (8 логических серверов) в течение нескольких дней никаких проблем не возникло, а скорость обработки запросов пользователей вполне удовлетворяло заказчика.

### Заключение

Таким образом, используя технологию виртуальных машин и разработанную методику, удалось полностью освободить старое оборудование (8 компьютеров), разместив логические серверы на 2-х новых компьютерах.

Доказательство оптимальности полученного решения

В данном примере окончательным было следующее решение:

H1:	S2	S5	S6	S8
H2:	S1	S3	S4	S7

Покажем, что не существует более лучшего допустимого решения. Более лучшими могли бы быть решения, при котором можно было бы обойтись меньшим количеством компьютеров (одним компьютером H1 или H2). Однако, очевидно, что они не допустимы, поскольку сумма требований логических серверов к быстродействию процессора равна 11100, и это больше чем возможности процессоров компьютеров H1 и H2 по отдельности: 6200 и 5600. Очевидно, что одним компьютером в любом случае не удастся обойтись.

Для сравнения методов решения можно попытаться решить задачу реорганизации более простым способом – последовательное “заполнение” компьютеров логическими серверами с проверкой ограничений по ресурсам. Для этого сначала на компьютер H1 будем пытаться размещать логические серверы, рассматривая их по порядку. В результате на компьютер H1 можно разместить логические серверы следующим образом:

H1:	S1	S2	S3	S4	S5
-----	----	----	----	----	----

Оставшиеся логические серверы попытаемся разместить на компьютер H2. В результате на компьютер H2 можно разместить логические серверы следующим образом:

H2:	S6	S7
-----	----	----

Оставшийся логический сервер S8 не удастся разместить ни на H1, ни на H2, поскольку ресурс процессора на компьютере H1 заполнен на  $1200 + 1200 + 600 + 600 + 1500 + 10$  (Базовая ОС) = 5110 из 6200 МТОPS, а на компьютере H2 ресурс процессора заполнен на  $1500 + 3000 + 10$  (Базовая ОС) = 4510 из 5600 МТОPS, а серверу S8 требуется 1500 МТОPS. Таким образом, при использовании более простого метода потребуются дополнительный компьютер H3, что, очевидно, является более худшим вариантом.



Пример 2. Задан простейший парк (часть парка), состоящий из 10 серверов, физически находящихся в одной подсети (отсутствуют какие-либо промежуточные маршрутизаторы) и связанных высокоскоростной (100 Мбит/с) и надежной ЛВС (подключены к одному коммутатору и находятся в одном серверном помещении). Серверы выполняют функции контроллеров доменов (по одному контроллеру на каждый домен). Ввиду низких требований серверов к компьютерам, на которых они работают, ресурсы используются неэффективно. Кроме того, заказчик желает повысить надежность функционирования доменов, как минимум, удвоив число контроллеров доменов (по два контроллера на каждый домен). Оценить то, возможно ли размещение удвоенного числа серверов на имеющихся компьютерах при применении технологии виртуальных машин. В положительном случае требуется произвести развертывание базовых ОС на компьютеры, произвести перенос логических серверов на виртуальные машины, разместить эти машины на соответствующих компьютерах и ввести их в тестовую эксплуатацию. Оценить качество функционирования логических серверов и в случае неудовлетворительности определить причины, внести коррективы в исходные условия, получить и внедрить скорректированный вариант распределения.

### Первичный сбор информации

Критичными ресурсами для контроллеров доменов являются память и диск. Процессор и сетевая подсистема и каналы передачи данных не являются критичными. Тем не менее, приведем требования для процессоров также.

Ресурсы физических компьютеров:

Конфигурация компьютеров идентична.

	H1-H10
Процессор, МТОPS	1000
Память, МВ	512
Диск, МВ	10240

Требования логических серверов:

Требования серверов одинаковы.

	S1-S10
Процессор, МТОPS	100
Память, МВ	96
Диск, МВ	2400

Требования базовой операционной системы:

	Базовая ОС
Процессор, МТОPS	10
Память, МВ	128
Диск, МВ	2000

По соображениям надежности функционирования, число логических серверов удваиваются, соответственно, добавляются серверы S11-S20, причем их требования такие же, как у серверов S1-S10. Кроме того, два контроллера одного домена не должны размещаться на одном и том же компьютере. Соответственно, необходимо использовать таблицу исключений (10 исключений для каждой пары контроллеров доменов):

S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20
1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1

Оптимизация проводится по всем типам ресурсов.

#### Получение первичного распределения

При помощи разработанной программы DTSOLVEX получаем следующее распределение серверов по новым компьютерам:

H1:	S1	S2	S3
H2:	S4	S5	S6
H3:	S7	S8	S9
H4:	S10	S11	S12
H5:	S13	S14	S15
H6:	S16	S17	S18
H7:	S19	S20	
H8:			
H9:			
H10:			

Все логические серверы распределены.

Компьютеры H8, H9 и H10 свободны.

Загрузка ресурсов компьютеров:

	Процессор	Память	Диск
H1:	30,30%	75,00%	87,38%
H2:	30,30%	75,00%	87,38%
H3:	30,30%	75,00%	87,38%
H4:	30,30%	75,00%	87,38%
H5:	30,30%	75,00%	87,38%
H6:	30,30%	75,00%	87,38%
H7:	20,20%	50,00%	58,25%
H8:	0,00%	0,00%	0,00%
H9:	0,00%	0,00%	0,00%
H10:	0,00%	0,00%	0,00%

### Первичная оценка результатов

Здесь нет ситуаций, когда какой-либо логический сервер распределен на тот же компьютер, на котором он работал, и при этом на этот компьютер больше ничего не размещается. Также отсутствуют ситуации, когда какой-либо логический сервер нераспределен, а соответствующий физический компьютер не задействован.

### Вторичная оценка результатов

Для учета возможных корректировок решим ту же задачу, но при завышенных на 25% требованиях логических серверов. В таком случае получаем следующее распределение:

H1:	S1	S2
H2:	S3	S4
H3:	S5	S6
H4:	S7	S8
H5:	S9	S10
H6:	S11	S12

H7:	S13	S14
H8:	S15	S16
H9:	S17	S18
H10:	S19	S20

Все логические серверы распределены. Свободных компьютеров нет. Таким образом, размещение удвоенного числа серверов на имеющихся компьютерах при применении технологии виртуальных машин возможно.

### Первичная реорганизация серверного парка

Производится подготовка базовой ОС на 7-и компьютерах в соответствии с первичным вариантом распределения. Подготавливаются 20 виртуальных машин, выполняется резервное копирование данных 10 логических серверов и подготовка их к переносу на виртуальную платформу. После этого выполняется перенос логических серверов S1–S10 на соответствующие 10 виртуальные машины и окончательная настройка логических серверов. После этого на других 10 виртуальных машинах производится развертывание и настройка логических серверов S11–S20. Наконец, виртуальные машины размещаются на компьютерах в соответствии с первичным вариантом распределения. После этого логические серверы S11–S20 настраиваются в режиме контроллеров соответствующих доменов, и синхронизируются с серверами S1–S10.

### Выявление негативных последствий реорганизации

В результате тестирования серверного парка в течение нескольких дней никаких проблем не возникло, а скорость обработки запросов пользователей вполне удовлетворяло заказчика.

### Заключение

Таким образом, используя технологию виртуальных машин и разработанную методику, удалось полностью освободить 3 из 10 компьютера, при всем при этом удвоив число контроллеров доменов с целью повышения надежности функционирования доменов.

### Доказательство оптимальности полученного решения

В данном примере окончательным было следующее решение:

H1:	S1	S2	S3
H2:	S4	S5	S6
H3:	S7	S8	S9
H4:	S10	S11	S12
H5:	S13	S14	S15
H6:	S16	S17	S18
H7:	S19	S20	
H8:			
H9:			
H10:			

Покажем, что не существует более лучшего допустимого решения. Более лучшими в данном примере могли бы быть решения, при которых можно было бы освободить более чем 3-х компьютеров. Однако, поскольку базовые уровни ресурсов компьютеров одинаковы и требования логических серверов также одинаковы, то можно сделать простой расчет по требованиям логических серверов к емкости диска ( $2400 \cdot x + 2000$  (Базовая ОС)  $\leq 10240$ ,  $x \in \mathbb{N}$ ) увидеть, что на один компьютер невозможно размещение более трех логических серверов ( $x \leq 3$ ). Тогда поскольку требования логических серверов одинаковы, а также одинаковы требования компьютеров, то, очевидно, что для размещения 20 серверов требуется не менее 7 компьютеров.

Для сравнения методов решения можно попытаться решить задачу реорганизации более простым способом – последовательное “заполнение” компьютеров логическими серверами с проверкой ограничений по ресурсам. Последовательное “заполнение” компьютеров даст следующее решение:

H1:	S1	S2	S3
H2:	S4	S5	S6
H3:	S7	S8	S9
H4:	S10	S11	S12
H5:	S13	S14	S15
H6:	S16	S17	S18
H7:	S19	S20	
H8:			
H9:			
H10:			

Это решение идентично полученному выше решению (но в любом случае оно не лучше, чем полученное выше решение). В данном случае более простой способ дал “такое же наилучшее решение”, но это лишь “удача” при конкретном частном случае.

Пример 3. В некоторой фирме по разработке программного обеспечения в связи с развитием направления WEB-разработки возникла необходимость в подготовке 10 тестовых компьютеров с различными версиями ОС MS Windows и обозревателей Internet для тестирования и отладки WEB-приложений. Основные требования предъявляются к процессору, памяти и диску, к сетевой подсистеме особых требований нет. Как известно, в пределах одной ОС несколько версий обозревателя Internet Explorer сосуществовать не могут, также на одном компьютере невозможна одновременная работа нескольких ОС без использования технологии виртуальных машин. Использование для тестирования приложений компьютеров разработчиков крайне нежелательно, кроме того, на компьютерах разработчиков установлены одинаковые ОС, обозреватели Internet Explorer и прочие приложения. Соответственно, руководство фирмы стремится снизить затраты на решение проблемы и, рассчитывая использовать технологию виртуальных машин, планирует приобрести не 10, а гораздо меньшее число компьютеров относительно недорогих конфигураций следующего вида:

	Н
Процессор, МТОПС	1000
Память, МВ	512
Диск, МВ	20480

Требования тестовых ОС с обозревателями Internet Explorer:

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Процессор, МТОПС	200	200	200	200	200	200	200	200	200	200
Память, МВ	64	64	64	64	128	128	256	256	256	256
Диск, МВ	2500	2500	2500	2500	2500	2500	3000	3000	3000	3000

Требования базовой ОС:

	Базовая ОС
Процессор, МТОПС	10
Память, МВ	96
Диск, МВ	2000

Требуется оценить то, каким минимальным количеством компьютеров вышеуказанной конфигурации можно обойтись для решения проблемы. После определения минимального числа требуемых компьютеров и приобретения заказчиком компьютеров, требуется произвести развертывание базовых ОС на компьютеры, произвести развертывание и настройку тестовых ОС на виртуальные машины и разместить их на соответствующих компьютерах и ввести серверный парк в тестовую эксплуатацию. Оценить качество функционирования тестовых систем и в случае неудовлетворительности определить причины, внести коррекции в исходные условия, получить и внедрить скорректированный вариант распределения.

### Первичный сбор информации

Все исходные данные заданы в условиях задачи. Для решения поставленной задачи поступим следующим образом: представим, что мы располагаем 10-ю компьютерами вышеуказанной конфигурации, и получим вариант распределения для этих условий. Таким образом, мы выясним то, сколько реально необходимо компьютеров для размещения тестовых ОС.

### Получение первичного распределения

При помощи разработанной программы DTSOLVEX получаем следующее распределение серверов по новым компьютерам:

H1:	S5	S7	
H2:	S6	S8	
H3:	S1	S2	S9
H4:	S3	S4	S10
H5:			
H6:			
H7:			
H8:			
H9:			
H10:			

Загрузка ресурсов компьютеров:

	Процессор	Память	Диск
H1:	40,40%	92,31%	66,75%
H2:	40,40%	92,31%	66,75%

H3:	60,61%	92,31%	97,09%
H4:	60,61%	92,31%	97,09%
H5:	0,00%	0,00%	0,00%
H6:	0,00%	0,00%	0,00%
H7:	0,00%	0,00%	0,00%
H8:	0,00%	0,00%	0,00%
H9:	0,00%	0,00%	0,00%
H10:	0,00%	0,00%	0,00%

Очевидно то, что в первом приближении достаточно 4 компьютера заданной конфигурации.

### Первичная оценка результатов

Поскольку все серверы распределены, причем распределены на новые компьютеры (старые вообще не рассматривались), то здесь нет ситуаций, когда какой-либо логический сервер распределен на тот же компьютер, на котором он работал, и при этом на этот компьютер больше ничего не размещается. Также отсутствуют ситуации, когда какой-либо логический сервер нераспределен, а соответствующий физический компьютер не задействован.

### Вторичная оценка результатов

Для учета возможных корректировок решим ту же задачу, но при завышенных на 25% требованиях логических серверов. В таком случае получаем следующее распределение:

H1:	S1	S7
H2:	S2	S8
H3:	S3	S9
H4:	S4	S10
H5:	S5	S6
H6:		
H7:		
H8:		
H9:		
H10:		

Во втором приближении с учетом случая возможных корректировок полученного распределения требуется не менее 5 компьютеров.



### Первичная организация серверного парка

После приобретения заказчиком 5-ти (с учетом вторичной оценки результатов) компьютеров производится подготовка базовой ОС на 4-х из них (Н1-Н4) в соответствии с первичным вариантом распределения. Кроме того, подготавливаются 10 виртуальных машин, на которые устанавливается требуемое программное обеспечение. После этого виртуальные машины размещаются на компьютерах в соответствии с первичным вариантом распределения. Наконец, полученный серверный парк вводится в эксплуатацию в тестовом режиме.

### Выявление негативных последствий реорганизации

В результате эксплуатации серверного парка в течение определенного времени выявляется то, что тестирование WEB-приложений одновременно на серверах S1, S2, S9 или на серверах S3, S4, S10 происходит существенно медленнее, чем на серверах S5, S7 или S6, S8. Соответственно, принимается решение о вводе дополнительных условий – исключений на размещение серверов S1, S2, S9 или серверов S3, S4, S10 на один и тот же компьютер.

S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
1	1	0	0	0	0	0	0	1	0
0	0	1	1	0	0	0	0	0	1

Остальные исходные условия остаются неизменными.

### Получение скорректированного распределения

После этого при помощи разработанной программы DTSOLVEX определяется новый (скорректированный) вариант распределения:

H1:	S5	S7
H2:	S6	S8
H3:	S1	S10
H4:	S3	S9
H5:	S2	S4
H6:		
H7:		
H8:		
H9:		
H10:		

Очевидно, требуется задействовать запасной 5-й компьютер (H5).

### Первичная оценка результатов

Поскольку все серверы распределены, причем распределены на новые компьютеры (старые вообще не рассматривались), то здесь нет ситуаций, когда какой-либо логический сервер распределен на тот же компьютер, на котором он работал, и при этом на этот компьютер больше ничего не размещается. Также отсутствуют ситуации, когда какой-либо логический сервер нераспределен, а соответствующий физический компьютер не задействован.

### Корректировочная реорганизация серверного парка

На компьютере Н5 производится развертывание базовой ОС, после чего выполняется перераспределение виртуальных машин в соответствии со скорректированным вариантом распределения.

### Выявление негативных последствий реорганизации

В результате эксплуатации серверного парка в течение определенного времени никаких проблем выявлено не было. Также функционирование серверного парка удовлетворило требованиям заказчика и WEB-разработчиков.

### Заключение

Используя разработанную методику, было определено минимальное количество требуемых компьютеров: 4 – в первом приближении, 5 – во втором. После внедрения первичного распределения (требующего 4 компьютера) и оценки качества функционирования парка были выявлены проблемы при тестировании WEB-приложений одновременно на 3-х серверах, размещенных на одном компьютере. Соответственно, были введены дополнительные ограничения и получено скорректированное распределение (требующего 5 компьютеров), что оправдало оценку количества требуемых компьютеров во втором приближении. После внедрения скорректированного распределения, качество функционирования серверного парка стало удовлетворительным. Таким образом, при решении поставленной проблемы удалось обойтись 5-ю вместо 10-и компьютеров.

### Доказательство оптимальности полученного решения

В данном примере окончательным было следующее решение (после ввода дополнительных ограничений на размещение серверов при коррекции):

H1:	S5	S7
H2:	S6	S8
H3:	S1	S10
H4:	S3	S9
H5:	S2	S4
H6:		
H7:		
H8:		
H9:		
H10:		

В данном примере ввиду сложности ограничений оптимальность полученного решения можно показать, лишь полностью перебрав всевозможные варианты. Полный перебор (описание проверок всех вариантов опустим), действительно, показал, что при заданных условиях нельзя обойтись меньше, чем 5 компьютерами (конфигурация компьютеров одинакова).

Для сравнения методов решения можно попытаться решить задачу реорганизации (с учетом дополнительных ограничений, полученных при коррекции) более простым способом – последовательное “заполнение” компьютеров логическими серверами с проверкой ограничений по ресурсам. Последовательное “заполнение” компьютеров даст следующее решение:

H1:	S1	S3	S5
H2:	S2	S4	S6
H3:	S7		
H4:	S8		
H5:	S9		
H6:	S10		
H7:			
H8:			
H9:			
H10:			

Таким образом, при попытке простого метода “заполнения” компьютеров логическими серверами с учетом ограничений ресурсов, дополнительных ограничений на размещение логических серверов и требований базовой ОС потребуется 6 компьютеров, что, очевидно, является более худшим решением.

Пример 4. Задан простейший парк (часть парка), состоящий из 6 серверов, физически находящихся в одной подсети (отсутствуют какие-либо промежуточные маршрутизаторы) и связанных высокоскоростной (100 Мбит/с) и надежной ЛВС (подключены к одному коммутатору и находятся в одном серверном помещении). Заказчика интересует первичный вариант распределения для определения целесообразности проведения реорганизации при применении технологии виртуальных машин.

#### Первичный сбор информации

Критичными ресурсами являются процессор, память и диск. Сетевая подсистема и каналы передачи данных не являются критичными, поскольку сетевой обмен между серверами и между серверами и пользовательскими рабочими местами невелик: не более 50-100 Кб/час.

Ресурсы физических компьютеров:

	H1	H2	H3	H4	H5	H6
Процессор, МТОПС	5867	408	5400	2333	525	6200
Память, МВ	256	256	512	1024	384	1024
Диск, МВ	37000	8500	37000	37000	64000	37000

Требования логических серверов:

	S1	S2	S3	S4	S5	S6
Процессор, МТОПС	1467	21	2700	2333	420	3720
Память, МВ	128	128	256	1024	384	512
Диск, МВ	4500	3000	5000	16000	45000	20000

Требования базовой операционной системы:

	Базовая ОС
Процессор, МТОПС	10
Память, МВ	128
Диск, МВ	2000

Специальные требования к надежности не предъявляются.

Оптимизация проводится по всем типам ресурсов.

### Получение первичного распределения

При помощи разработанной программы DTSOLVEX получаем следующее распределение серверов по компьютерам:

H1:			
H2:			
H3:	S3		
H4:			
H5:			
H6:	S1	S2	S6

Оставшиеся нераспределенные серверы – S4 и S5.

Незадействованные компьютеры – H1, H2, H4 и H5.

Загрузка ресурсов компьютеров:

	Процессор	Память	Диск
H1:	0,00%	0,00%	0,00%
H2:	0,00%	0,00%	0,00%
H3:	50,09%	66,67%	14,29%
H4:	0,00%	0,00%	0,00%
H5:	0,00%	0,00%	0,00%
H6:	84,14%	85,71%	78,57%

### Первичная оценка результатов

Легко видеть то, что серверы S4 и S5 никуда не распределены и в то же время компьютеры H4 и H5 остались незадействованными. Соответственно, серверы S4 и S5 нет смысла переносить на виртуальную платформу и лучше всего оставит их работать на компьютерах H4 и H5. Кроме того, сервер S3 размещается на компьютер H3 (на котором он и работал), и на H3 ничего более не размещается, поэтому сервер S3 также следует оставить работающим на компьютере H3. Таким образом, серверы S3, S4 и S5 и компьютеры H3, H4 и H5 исключаются из рассмотрения.

Тогда с учетом вышесказанного окончательный вариант распределения:

H1:			
H2:			
H6:	S1	S2	S6

В первом приближении возможно освобождение компьютеров H1 и H2.

### Вторичная оценка результатов

Для учета возможных корректировок решим ту же задачу, но при завышенных на 25% требованиях логических серверов. В таком случае получаем следующее распределение:

H1:		
H2:		
H3:	S3	
H4:	S1	
H5:		
H6:	S2	S6

Оставшиеся нераспределенные серверы – S4 и S5.

Незадействованные компьютеры – H1, H2 и H5.

Сервер S3 размещается на компьютер H3 (на котором он и работал), и на H3 ничего более не размещается, поэтому сервер S3 следует оставить работающим на компьютере H3. На компьютер H4 размещается сервер S1, в то же время компьютер H1 не задействован, более того, S4 никуда не распределен, поэтому S4 следует оставить работать на H4, а S1 на H1 и исключить их рассмотрения. Наконец, сервер S5 никуда не распределен, а H5 не задействован, поэтому S5 следует оставить работать на H5 и исключить их из рассмотрения. Таким образом, в рассмотрении остаются только серверы S2 и S6 и компьютеры H2 и H6. Тогда распределение имеет следующий вид:

H2:		
H6:	S2	S6

### Заключение

Во втором приближении возможно освобождение только компьютера H2. Учитывая невысокие характеристики компьютера H2, а также освобождение всего лишь одного компьютера, проведение работ по реорганизации вряд ли будет целесообразным.

### Доказательство оптимальности полученного решения

В данном примере при первоначальных требованиях логических серверов было получено следующее решение:

H1:			
H2:			
H3:	S3		
H4:			
H5:			
H6:	S1	S2	S6

Оставшиеся нераспределенные серверы – S4 и S5.

Незадействованные компьютеры – H1, H2, H4 и H5.

Поскольку для размещения S4 и S5, было принято решение оставить эти серверы на компьютерах H4 и H5, и исключить их из рассмотрения, то, свободными становились только компьютеры H1 и H2.

Доказать то, что не существует более лучших вариантов можно лишь перебрав всевозможные варианты. Полный перебор (описание проверок всех вариантов опустим), действительно, показал, что при заданных условиях не существует более лучшего решения.

Для сравнения методов решения можно попытаться решить задачу реорганизации более простым способом – последовательное “заполнение” компьютеров логическими серверами с проверкой ограничений по ресурсам.

Последовательное “заполнение” компьютеров даст следующее решение:

H1:	S1	S2
H2:		
H3:	S3	
H4:		
H5:		
H6:	S6	

Оставшиеся нераспределенные серверы – S4 и S5.

Незадействованные компьютеры – H2, H4 и H5.

Поскольку для размещения S4 и S5, необходимо оставить эти серверы на компьютерах H4 и H5, и исключить их из рассмотрения, то, свободным становится только компьютер H2, и это является более худшим вариантом при первоначальных исходных условиях.

Пример 5. В некоторой фирме в течение нескольких лет эксплуатируется небольшой серверный парк, состоящий из 10 компьютеров. Серверный парк представляет собой ЛВС 100Base-T IEEE 802.3 (100 мегабитный Ethernet). Компьютеры парка находятся в одном помещении. В худшем случае скорость обмена данными между компьютерами и рабочими компьютерами пользователей составляет не менее 2000 Кбайт/сек. Компьютеры по своим техническим характеристикам изначально соответствовали (с небольшим запасом) требованиям серверного программного обеспечения, функционирующего на этих компьютерах. Однако, ввиду устаревания компьютерного оборудования, руководством было решение о приобретении новых компьютеров и выводе из эксплуатации старых компьютеров. Новые компьютеры обладали значительно более высокими техническими характеристиками, существенно превышающими требования программного обеспечения, однако, поначалу этот момент остался без внимания. Спустя некоторое время, в связи с открытием второго офиса фирмы, возникла необходимость в развертывании дополнительного серверного парка в новом офисе. Соответственно, перед руководством встал вопрос о приобретении дополнительного компьютерного оборудования. К тому моменту специалисты фирмы представили руководству результаты анализа загрузки компьютеров серверного парка, после чего руководство решило попробовать использовать технологию виртуальных машин.

Требуется проанализировать ситуацию и оценить целесообразность реорганизации серверного парка при применении технологии виртуальных машин. В положительном случае требуется произвести развертывание базовых ОС на компьютеры, произвести перенос логических серверов на виртуальные машины, разместить эти машины на соответствующих компьютерах и ввести их в тестовую эксплуатацию. Оценить качество функционирования логических серверов и в случае неудовлетворительности определить причины, внести коррекции в исходные условия, получить и внедрить скорректированный вариант распределения.



### Первичный сбор информации

Критичными ресурсами являются процессор, память, диск и сетевая подсистема. Компьютеры парка имеют идентичную конфигурацию, характеристика для сетевой подсистемы берется для худшего случая – 2000 Кбайт/сек.

Ресурсы физических компьютеров:

	H1-H10
Процессор, МТОPS	4000
Память, МВ	1024
Диск, МВ	40000
Сеть, КБ/с	2000

Требования логических серверов:

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Процессор, МТОPS	700	500	800	900	800	900	1000	900	600	900
Память, МВ	192	256	128	128	256	384	256	512	128	96
Диск, МВ	2000	3000	5000	2000	2000	8000	1000	10000	4000	8000
Сеть, КБ/с	150	200	500	300	500	400	200	120	180	200

Требования базовой операционной системы:

	Базовая ОС
Процессор, МТОPS	10
Память, МВ	100
Диск, МВ	2200
Сеть, КБ/с	10

Специальные требования к надежности не предъявляются.

Оптимизация проводится по использованию процессора и диска.

### Получение первичного распределения

При помощи разработанной программы DTSOLVEX получаем следующее распределение серверов по новым компьютерам:

H1:	S3	S4	S8	S10
H2:	S1	S5	S7	S9
H3:	S2	S6		
H4:				
H5:				

H6:				
H7:				
H8:				
H9:				
H10:				

В первом приближении при применении технологии виртуальных машин можно обойтись всего 3-мя компьютерами, освободив 7 из 10 компьютеров.

Загрузка ресурсов:

	Процессор	Память	Диск	Сеть
H1:	87,72%	93,51%	66,14%	56,28%
H2:	77,69%	90,04%	23,81%	51,76%
H3:	35,09%	69,26%	29,10%	30,15%
H4:	0,00%	0,00%	0,00%	0,00%
H5:	0,00%	0,00%	0,00%	0,00%
H6:	0,00%	0,00%	0,00%	0,00%
H7:	0,00%	0,00%	0,00%	0,00%
H8:	0,00%	0,00%	0,00%	0,00%
H9:	0,00%	0,00%	0,00%	0,00%
H10:	0,00%	0,00%	0,00%	0,00%

### Первичная оценка результатов

Здесь нет ситуаций, когда какой-либо логический сервер распределен на тот же компьютер, на котором он работал, и при этом на этот компьютер больше ничего не размещается. Также отсутствуют ситуации, когда какой-либо логический сервер нераспределен, а соответствующий физический компьютер не задействован.

### Вторичная оценка результатов

Для учета возможных корректировок решим ту же задачу, но при завышенных на 25% требованиях логических серверов. В таком случае получаем следующее распределение:

H1:	S2	S3	S4	S10
H2:	S1	S6	S9	
H3:	S5	S7		
H4:	S8			
H5:				
H6:				
H7:				
H8:				
H9:				
H10:				

Во втором приближении при применении технологии виртуальных машин можно обойтись всего 4-мя компьютерами, освободив 6 из 10 компьютеров. Таким образом, учитывая то, что возможно освобождение 6-7 компьютеров, очевидно, что реорганизация будет целесообразной.

### Первичная реорганизация серверного парка

Производится подготовка базовой ОС на 3-х компьютерах в соответствии с первичным вариантом распределения. Подготавливаются 10 виртуальных машин, выполняется резервное копирование данных 10 логических серверов и подготовка их к переносу на виртуальную платформу. После этого выполняется перенос логических серверов S1–S10 на соответствующие 10 виртуальные машины и окончательная настройка логических серверов.

### Выявление негативных последствий реорганизации

В течение нескольких дней реорганизованный серверный парк эксплуатировался в тестовом режиме, в результате чего было выявлено то, что поскольку в силу специфики решаемых в фирме задач на серверы S3, S4 (размещенных на H1) либо на серверы S1, S5 (размещенных на H2) создается одновременно большая нагрузка и это сказывается на времени обработки запросов пользователей, то лучше их разместить на различных компьютерах. Соответственно, к исходным условиям добавляем исключение на размещение:

S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
1	0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0

Остальные исходные условия остаются неизменными.

### Получение скорректированного распределения

После этого при помощи разработанной программы DTSOLVEX определяется новый (скорректированный) вариант распределения:

H1:	S3	S8	S9	S10
H2:	S2	S4	S5	S7
H3:	S1	S6		
H4:				
H5:				

H6:				
H7:				
H8:				
H9:				
H10:				

Удалось перераспределить серверы, не привлекая дополнительных компьютеров из числа доступных в серверном парке.

### Первичная оценка результатов

Здесь нет ситуаций, когда какой-либо логический сервер распределен на тот же компьютер, на котором он работал, и при этом на этот компьютер больше ничего не размещается. Также отсутствуют ситуации, когда какой-либо логический сервер нераспределен, а соответствующий физический компьютер не задействован.

### Корректировочная реорганизация серверного парка

Выполняется перераспределение виртуальных машин в соответствии со скорректированным вариантом.

### Выявление негативных последствий реорганизации

В результате эксплуатации серверного парка в течение определенного времени никаких проблем выявлено не было. Также функционирование серверного парка удовлетворило требованиям заказчика.

### Заключение

Таким образом, применив технологию виртуальных машин и разработанную методику, удалось выявить и устранить крайнюю неэффективность использования вычислительных ресурсов, освободив 7 из 10 компьютеров. Очевидно, что затраты фирмы на компьютерное оборудование для нового офиса будут существенно ниже за счет использования освободившихся компьютеров.

### Доказательство оптимальности полученного решения

В данном примере окончательным было следующее решение (после ввода дополнительных ограничений на размещение серверов при коррекции):

H1:	S3	S8	S9	S10
H2:	S2	S4	S5	S7
H3:	S1	S6		
H4:				
H5:				
H6:				
H7:				
H8:				
H9:				
H10:				

Поскольку конфигурация компьютеров одинакова, то более лучшим решением может считаться решение, при котором можно было бы обойтись не 3-мя, а меньшим количеством компьютеров. Однако, легко показать то, что это невозможно: сумма требований к емкости памяти всех логических серверов равна 2336 МБ, а на каждом компьютере емкость памяти равна 924 МБ (учитываем то, что на любом компьютере базовая ОС для себя требует 100 МБ). Очевидно, что требуется не менее 3-х компьютеров.

Для сравнения методов решения можно попытаться решить задачу реорганизации более простым способом (с учетом дополнительных ограничений, полученных при коррекции) – последовательное “заполнение” компьютеров логическими серверами с проверкой ограничений по ресурсам. Последовательное “заполнение” компьютеров даст следующее решение:

H1:	S1	S2	S3	S7
H2:	S4	S5	S6	S9
H3:	S8	S10		
H4:				
H5:				
H6:				
H7:				
H8:				
H9:				
H10:				

Решение не отличается от полученного выше решения (такое же хорошее, как и полученное ранее решение: требуется лишь 3 из 10 компьютеров). Однако, следует помнить, что это лишь частный случай, когда удастся получить “такое же хорошее решение” более простым способом.

## **Приложение 2. Руководство по переносу системы MS Windows 2000 с физической аппаратной платформы на виртуальную**

### **Аннотация**

В настоящее время проекты корпоративных сетей, как правило, представляют собой достаточно сложные решения, требующие для реализации большого количества программных и аппаратных средств. В такой ситуации идея размещения части серверных систем на виртуальных платформах является весьма разумной и выгодной с экономической точки зрения. Однако, помимо проектирования сетей “с нуля”, специалистам зачастую приходится сталкиваться с ситуацией, когда корпоративная сеть уже существует и функционирует, но ее необходимо реорганизовать, перенести часть настроенных операционных систем с физических компьютеров на виртуальные машины, и здесь возникает ряд проблем, обзору и способам обхода которых посвящено данное руководство.

### **Условия применения**

В данном документе описывается порядок действий при переносе системы на базе MS Windows 2000 с физического компьютера на виртуальную машину, работающей под управлением VMware Workstation с использованием только IDE устройств. В качестве утилиты для создания разделов используется встроенная в ОС MS Windows 2000 утилита Disk Management. В качестве утилиты для копирования разделов используется Symantec Norton Ghost.

Автор должен предупредить, что он не несет ответственности в случае, если использование руководства повлекло нарушение функционирования каких-либо компьютерных систем, либо потерю каких-либо данных. Пользователи и специалисты могут использовать данное руководство только на свой собственный страх и риск. Также автор не несет ответственности в случае злоумышленного использования какими-либо физическими либо юридическими лицами данного руководства в качестве пособия для быстрого

создания массовых копий в целях незаконного использования либо распространения программных продуктов корпорации Microsoft, равно как и любых других производителей программного обеспечения.

Microsoft, Windows и Windows NT являются зарегистрированными торговыми марками фирмы Microsoft Corporation.

Intel, Pentium, Pentium Pro являются зарегистрированными торговыми марками фирмы Intel Corporation.

VMware, Workstation, GSX Server, ESX Server являются зарегистрированными торговыми марками компании VMware, Inc.

Symantec, Norton Ghost являются зарегистрированными торговыми марками фирмы Symantec Corporation.

PowerQuest, Partition Magic, Drive Image Pro являются зарегистрированными торговыми марками фирмы PowerQuest Corporation.

## Ключевые проблемы переноса систем

При переносе системы на базе ОС MS Windows 2000 с физического компьютера на другой физический компьютер либо виртуальную машину возникают три основные проблемы:

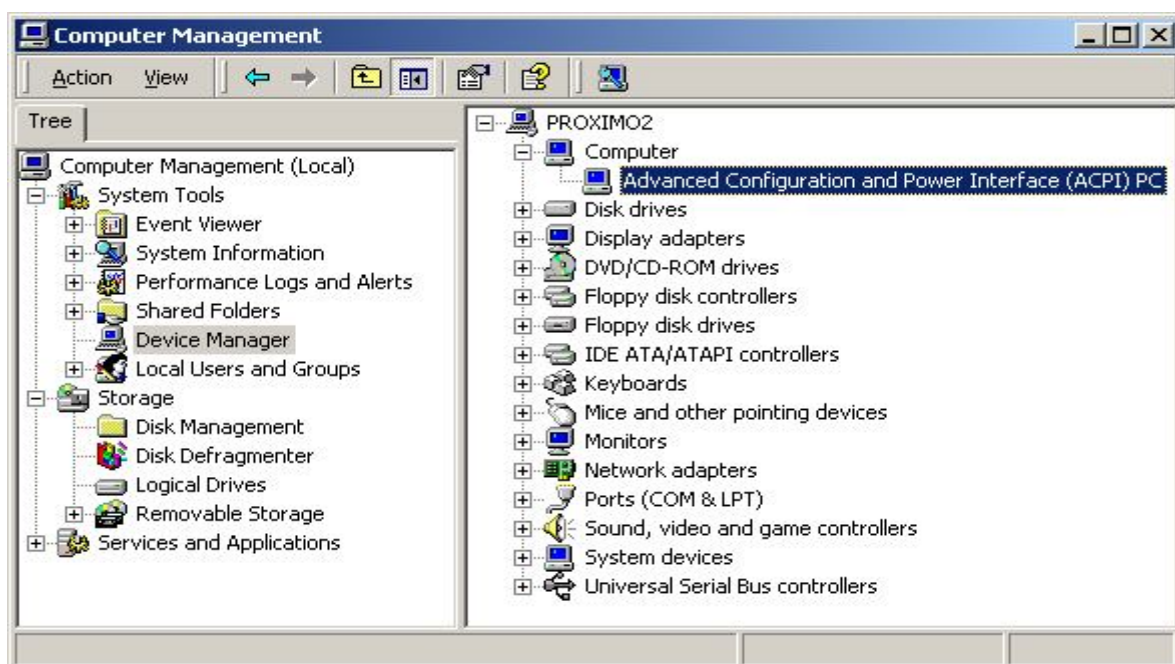
### 1) Несовместимость аппаратных платформ

На сегодняшний день ОС MS Windows 2000 распознает 6 ключевых аппаратных платформ: из них 3 с поддержкой технологии ACPI (Advanced Configuration and Power Interface), 3 без ее поддержки:

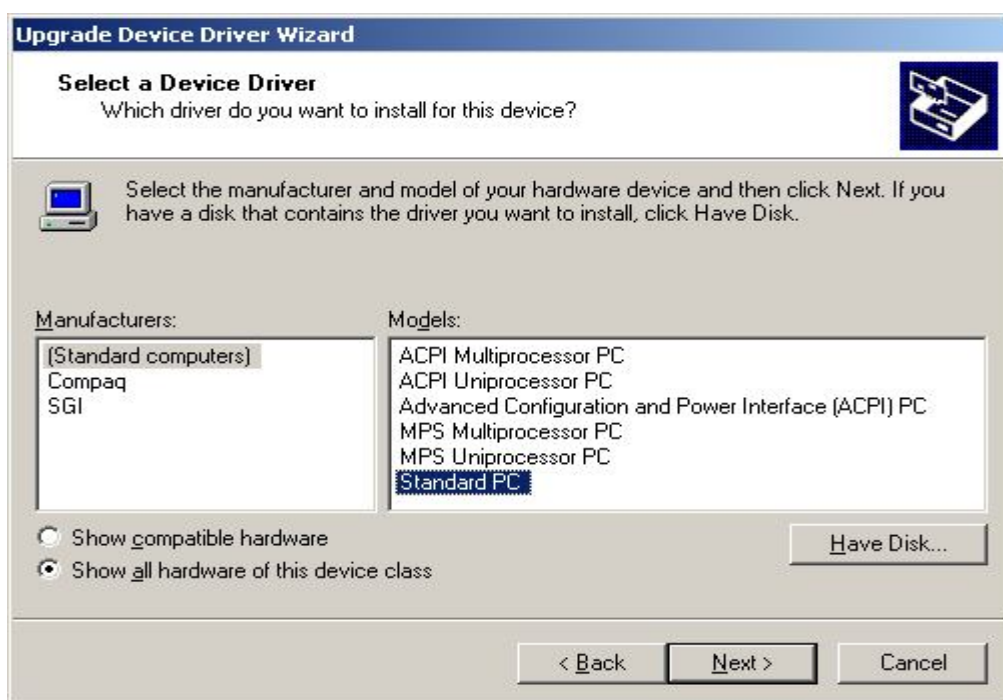
Платформа	Описание	Системный драйвер
Standard PC	Стандартный однопроцессорный ПК без поддержки ACPI	HAL.DLL
ACPI PC	Стандартный однопроцессорный ПК с поддержкой ACPI	HALACPI.DLL
MPS Uniprocessor PC	Многопроцессорный ПК, с одним установленным процессором без поддержки ACPI	HALAPIC.DLL
ACPI Uniprocessor PC	Многопроцессорный ПК, с одним установленным процессором с поддержкой ACPI	HALAACPI.DLL
MPS Multiprocessor PC	Многопроцессорный ПК, без поддержки ACPI	HALMPS.DLL
ACPI Multiprocessor PC	Многопроцессорный ПК, с поддержкой ACPI	HALMACPI.DLL

Большинство ПК на базе процессоров Intel принадлежат к одной из этих платформ. ОС MS Windows 2000 для каждой из вышеперечисленных аппаратных платформ использует соответствующий системный драйвер. В диспетчере устройств тип платформы определяется как устройство:





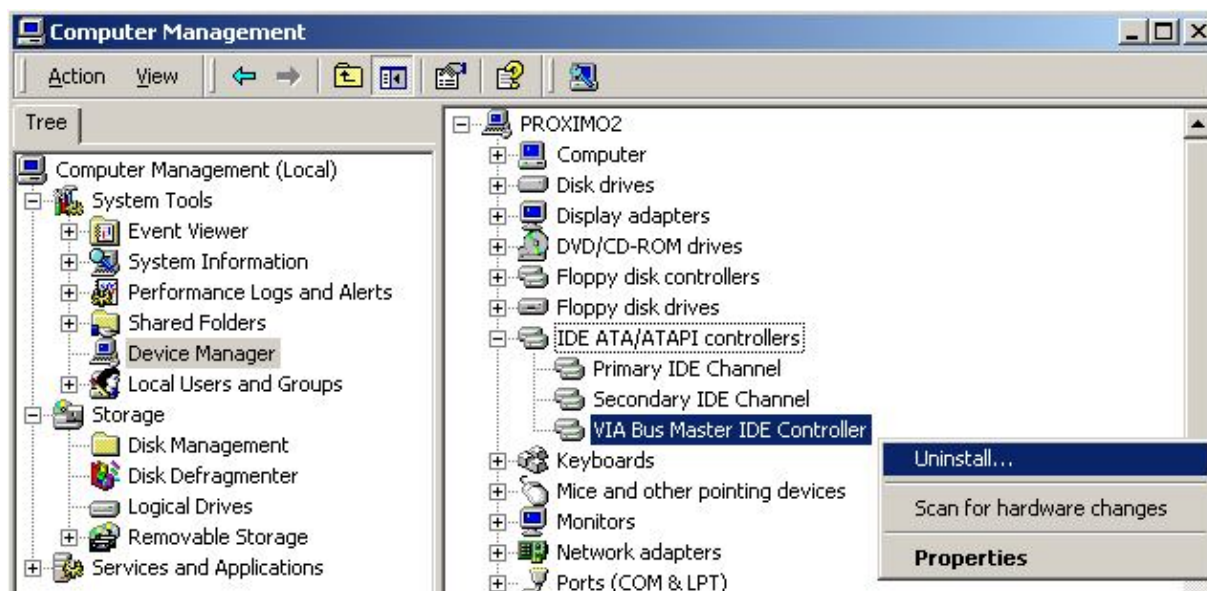
Соответственно, если мы переносим систему с одного типа платформы на другой, то система не может успешно загрузиться (ПК автоматически перегружается на фазе инициализации ядра системы) в силу принципиальной несовместимости платформ. Поэтому перед переносом на другую платформу необходимо заменить системный драйвер в диспетчере устройств, выбрав соответствующий. Например, при переносе системы на виртуальную платформу всегда выбирается тип платформы **Standard PC**:



**Важное примечание.** После операции по замене драйвера аппаратной платформы загрузка системы на исходном ПК недопустима, поскольку из-за несовместимости драйвера и платформы это может привести к необратимому нарушению работоспособности операционной системы.

## 2) Несовместимость контроллеров жестких дисков

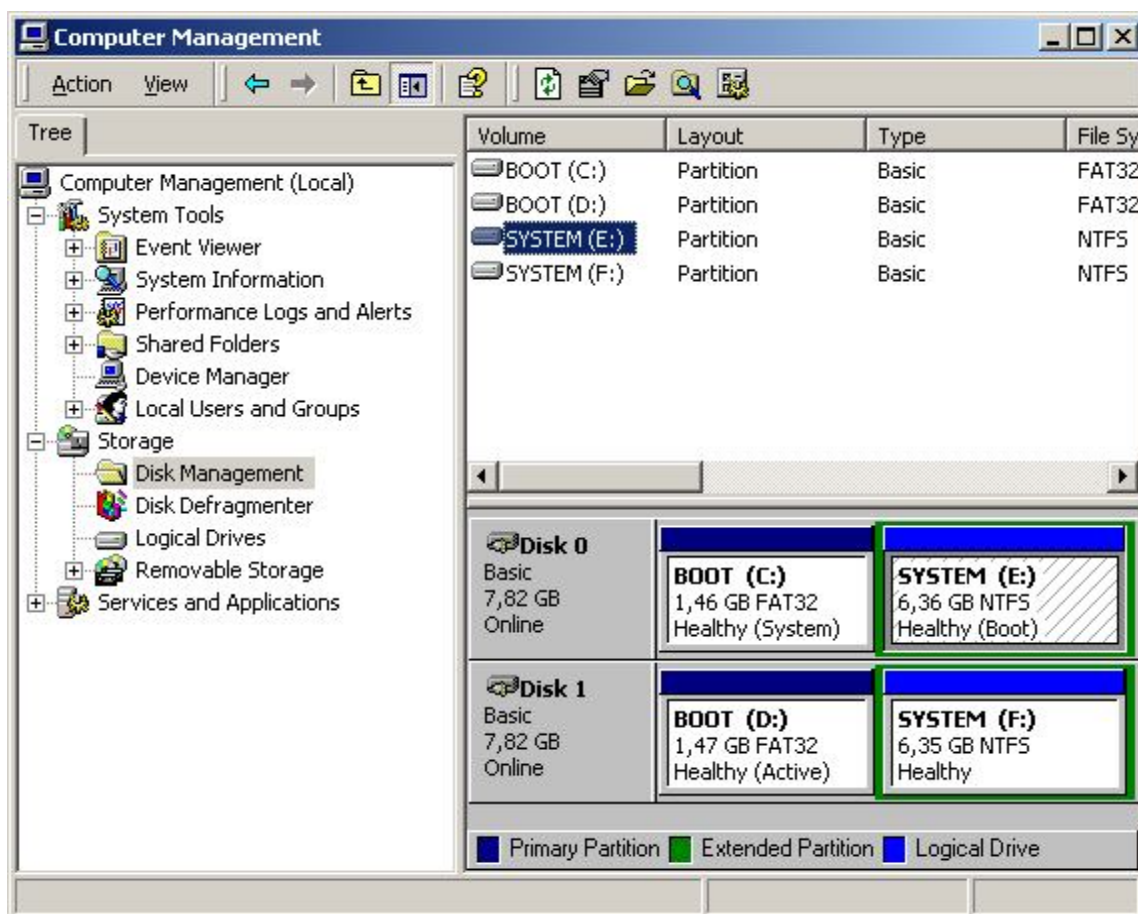
Вторая проблема при переносе системы связана с особенностями контроллера жестких дисков. Для успешной загрузки системы необходим доступ к системному разделу, а для этого необходимо, чтобы система корректно распознавала контроллер жестких дисков и сами диски. Если система переносится на аппаратную платформу с контроллером, отличным от того, который был в исходном компьютере, то система при загрузке останавливается с ошибкой **"STOP 0x0000007B – Inaccessible boot device"** и дальнейшая загрузка невозможна. Чтобы избежать этого, перед переносом системы необходимо удалять все драйвера RAID-массивов и прочие драйверы и утилиты, так или иначе связанные с работой жестких дисков. Также необходимо удалять все IDE-контроллеры в диспетчере устройств:



**Важное примечание.** После данной операции загрузка системы на исходном компьютере недопустима, т.к. в этом случае система вновь распознает дисковой контроллер и устанавливает специфичный для него драйвер, что приведет к невозможности загрузки на конечной аппаратной платформы после переноса.

### 3) Несоответствие буквы, назначаемой системному разделу в целевой системе, букве, назначенной в исходной системе

В отличие от первых двух проблем, данная проблема проявляется только при использовании, так называемых “нетипичных” букв для системных разделов. Например, если мы располагаем жестким диском с двумя разделами: загрузочным и системным – то по логике вещей загрузочный раздел должен иметь букву C:, а системный – D:. Однако, это соблюдается, как правило, если система изначально устанавливалась на компьютер, с одним IDE-устройством – единственным жестким диском. В общем же случае, системы могут устанавливаться на системы с несколькими жесткими дисками, приводами компакт дисков, ZIP приводами и прочими IDE-устройствами. Буквы разделам жестких дисков назначаются не в порядке их следования внутри одного жесткого диска, а сначала назначаются буквы первым разделам жестких дисков, всем IDE-устройств в порядке их следования в контроллере IDE-устройств. Например, если мы имеем два жестких диска по два раздела на каждом, то результат назначения букв получается следующим:



В данном примере, системный раздел имеет букву E:, в общем же случае он может иметь любую букву латинского алфавита, начиная с C:. В такой ситуации после подготовки к переносу системы на другой ПК и переноса данных разделов на другой диск, при запуске системы на исходном либо на другом компьютере с этого диска в ОС MS Windows 2000 возникает проблема с невозможностью входа в систему. Происходит это из-за того, что система при обнаружении нового диска начинает повторную процедуру назначения букв разделам. Например, если мы переносим жесткий диск с двумя разделами, с назначенными ранее буквами C: и E: на исходный либо другой ПК, на котором нет других жестких дисков, то система при первой загрузке присвоит системному разделу букву D:. Поскольку в реестре ОС MS Windows 2000 присутствует множество ссылок на диск E:, в том числе пути к профилям пользователей, то, несмотря на успешную загрузку системы, вход в рабочий сеанс пользователя будет невозможен. Данная ситуация разрешается путем внесения определенных корректировок в реестр “сбойной” системы.

К системному реестру “сбойной” системы можно получить доступ несколькими способами:

- ✗ Непосредственно в сеансе самой системы. Однако, этот вариант отпадает сразу, так как невозможно войти в рабочий сеанс пользователя, соответственно, невозможен запуск утилит редактирования реестра.
- ✗ Доступ по сети с помощью системы на другом компьютере. Это не всегда возможно, поскольку, как правило, после переноса система по сети недоступна в силу того, что зачастую система не может установить необходимый драйвер сетевого адаптера и, кроме того, сетевые протоколы требуют ручной настройки.
- ✓ С помощью подключения жесткого диска “сбойной” системы к другому компьютеру и доступа к системному разделу “сбойной” системы с помощью системы на другом компьютере. Этот вариант работает всегда, но для разных случаев переноса требует различные дополнительные действия:

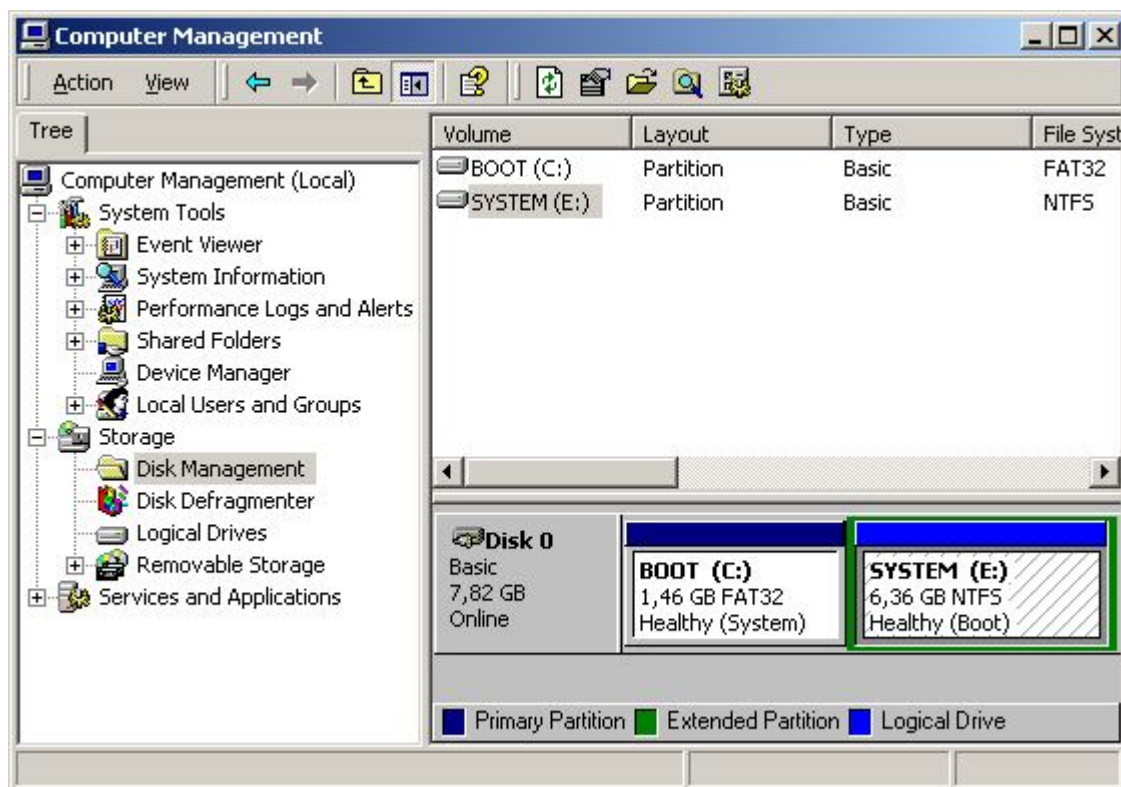
- В случае переноса системы на виртуальную машину, необходимо воспользоваться вспомогательной работоспособной виртуальной



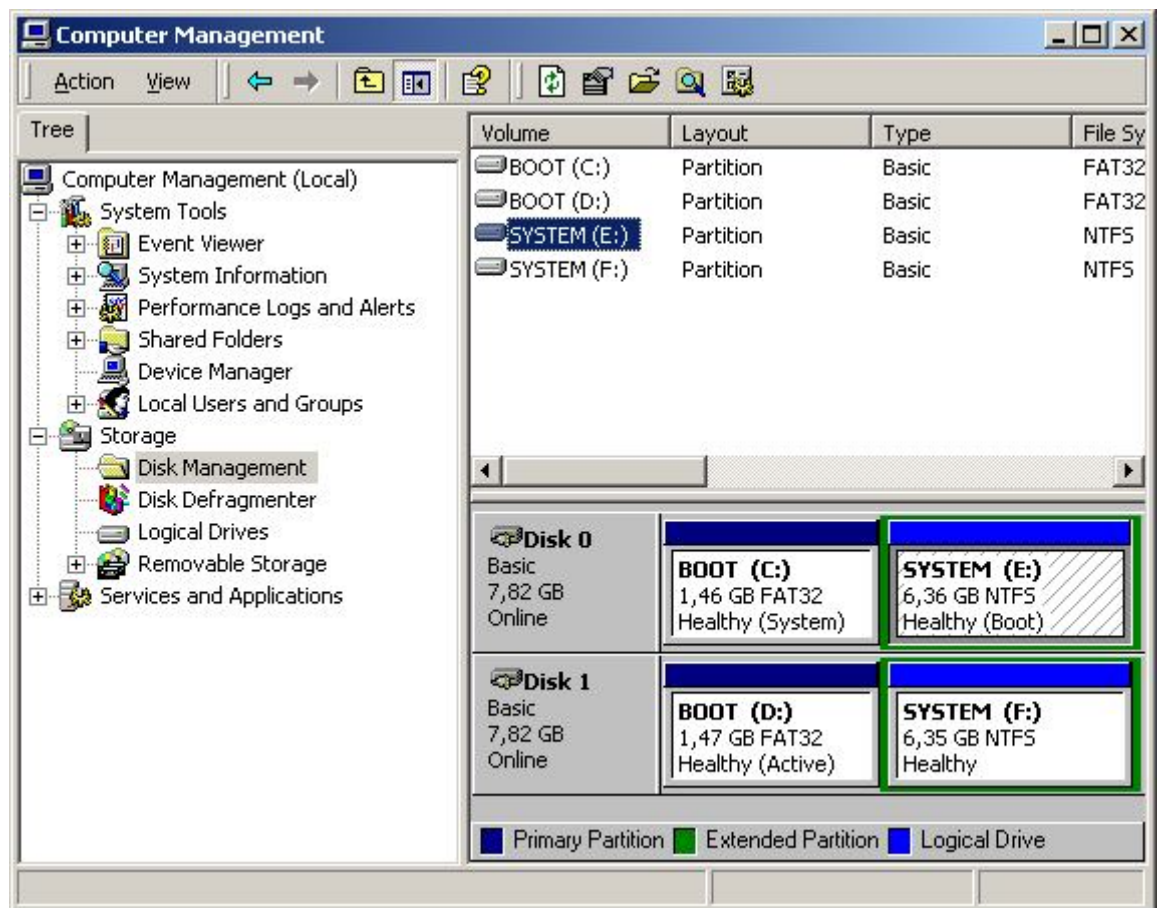
системой на базе MS Windows 2000, к которой необходимо подключить жесткий диск со “сбойной” системой как дополнительный жесткий диск.

- В случае переноса системы с одного реального ПК на другой достаточно подключить жесткий диск со “сбойной” системой к исходному ПК.

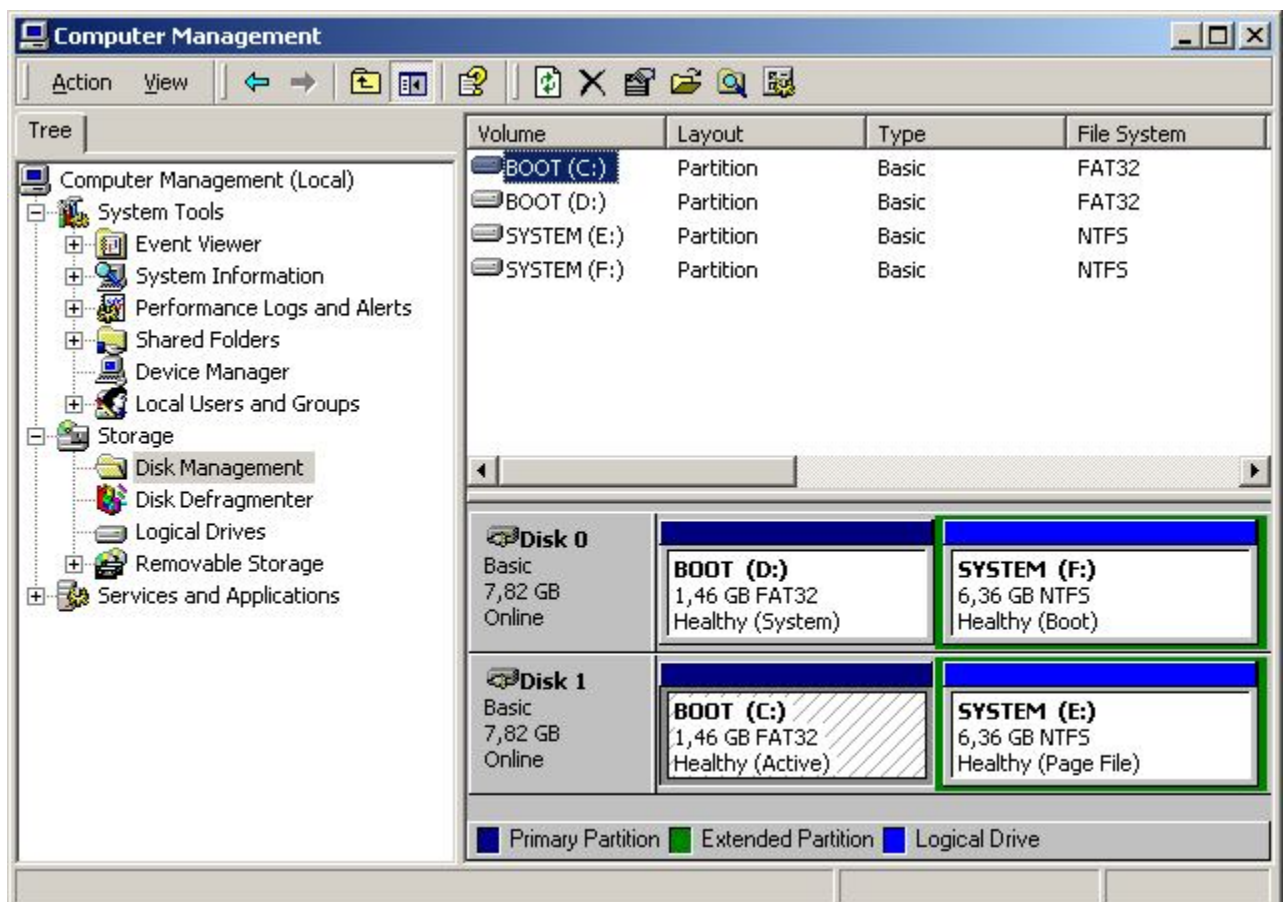
В такой ситуации жесткий диск со “сбойной” системой ни в коем случае не должен быть результатом операции копирования “диск в диск”, копирование должно проводиться по каждому разделу в отдельности с предварительным разбиением на разделы жесткого диска конечной системы. В противном случае, в дальнейшем станет невозможным вход в рабочий сеанс пользователя в системе на исходном диске и потребуются третья работоспособная система, с помощью которой необходимо будет вносить исправления в реестры исходной и конечной систем. Для наглядности приведем пример: допустим, что на исходном диске были два раздела, с назначенными буквами C: и E:.



После копирования каждого раздела по отдельности с исходного диска на диск конечной системы и при загрузке с исходного диска, с подключенным диском конечной системы получаем следующую картину (в этом случае все в порядке): на исходном диске не нарушены привязки букв к его разделам:



В случае же копирования “диск в диск”, получается следующее:



Назначение букв в случае копирования “диск в диск” нарушается: к разделам конечного диска привязываются буквы, принадлежавшие ранее исходному диску, а разделам исходного диска привязываются буквы, которые по логике вещей должны были бы принадлежать разделам конечного диска. Происходит это потому, что начальный загрузчик компьютера обнаруживает два активных загрузочных диска и выбирает последний по списку (а это диск конечной системы), далее загрузчик ищет на его первом разделе загрузочную запись и находит ее, далее этому разделу назначается буква С:. На этом разделе запускается начальный загрузчик операционной системы, называемый NTLOADER, который обращается к конфигурационному файлу BOOT.INI. В конфигурационном файле хранится информация о расположении системных файлов, при этом, самое важное то, что в качестве ссылок местоположения указываются не буквы дисков, а порядковые номера физических дисков и их разделов. Соответственно, начальный загрузчик операционной системы обнаруживает в этом файле запись типа Disk0\Partition2, что означает то, что система находится на 0-м физическом диске (нумерация начинается с 0), на 2-м разделе (нумерация начинается с 1). В результате ОС загружается именно со второго раздела первого жесткого диска, несмотря на то, что загрузчик был запущен с первого раздела второго жесткого диска. Далее, ОС считает второй жесткий диск первым и назначение букв происходит так, как будто второй и первый жесткий диск в порядке их следования поменяли местами. В результате второй раздел первого жесткого диска получает букву F:, а второй раздел второго диска букву E:. Далее, поскольку в реестре системы местоположение программного обеспечения, файла подкачки и профилей пользователей строго привязано именно к букве E:, то, несмотря на то, что система загружается со второго раздела первого диска, получившую букву F:, вышеуказанные данные извлекаются со второго раздела второго жесткого диска, получившего букву E:. При всем при этом используется реестр системы на исходном диске, расположенной на втором разделе первого диска (F:) и вся информация о привязке букв к конкретным разделам конкретных жестких дисков хранится

именно в реестре системы на исходном диске. В такой ситуации, если в дальнейшем отключить второй жесткий диск и попытаться загрузить ОС на исходном диске, то система без проблем обнаружит системные файлы на втором разделе, однако, она не сможет обнаружить диск E:, привязанный ко второму разделу второго жесткого диска, и будет считать потерянными программное обеспечение, файл подкачки и профили пользователей, в результате чего работа в исходной системе станет невозможной.

Таким образом, копирование данных должно производиться не “диск в диск”, а по разделам. После этого необходимо внести некоторые изменения в реестр “сбойной” системы на конечном диске.

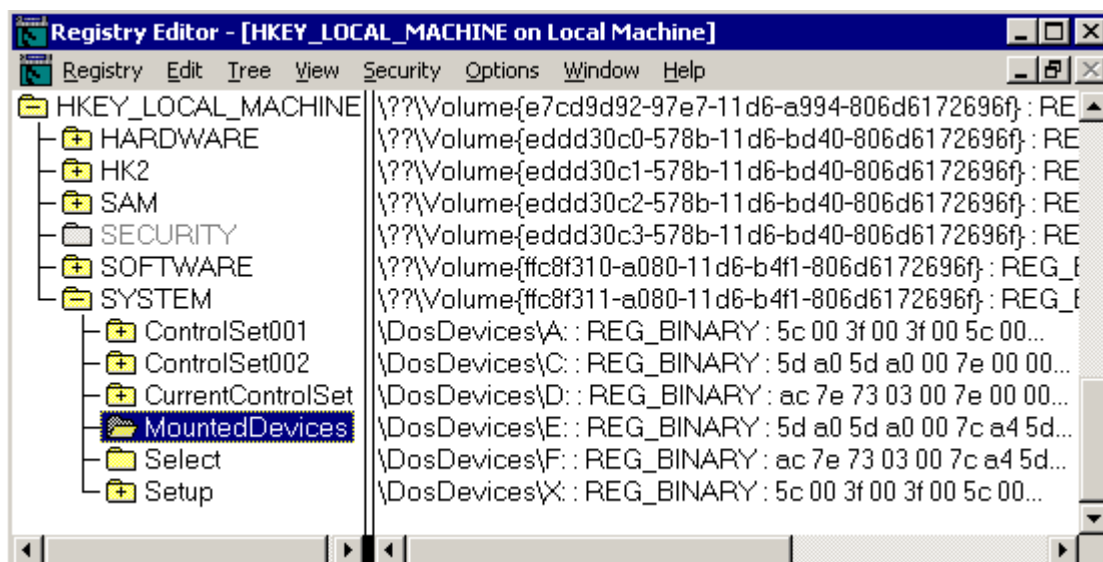
### **Корректировка системного реестра “сбойной” системы**

После того, как диск со “сбойной” системой подключен как дополнительный к компьютеру с работоспособной системой, в ней необходимо запустить утилиту редактирования реестра REGEDT32. Выделить мышью раздел HKEY\_LOCAL\_MACHINE, после чего в меню Registry выбрать Load Hive, и открыть файл “SYSTEM” находящийся в папке Winnt\system32\config системного раздела дополнительного диска, при этом необходимо точно знать какая именно буква соответствует этому разделу. Буква системного раздела дополнительного диска может быть различной при различных дисковых конфигурациях компьютера, поэтому эту букву необходимо уточнять через встроенную в ОС утилиту Disk Management. Например, как в примере выше, поскольку исходный диск использует буквы C: и E:, а дополнительный – D: и F:, причем F: – это и есть системный раздел “сбойной” системы, то в редакторе реестра необходимо будет загрузить файл F:\Winnt\system32\config\system. В качестве требуемого имени для загружаемой ветви реестра можно выбрать произвольное название, например HK2. После этого вносятся определенные корректировки в этой ветви реестра (системной ветви реестра “сбойной системы”) – об этом будет сказано ниже.



## Привязка букв к разделам жесткого диска в ОС Windows 2000

В системном реестре ОС Windows 2000 в ключе HKEY\_LOCAL\_MACHINE\SYSTEM\MOUNTEDDEVICES хранятся данные о привязках букв к устройствам и отдельным разделам жестких дисков. Обратимся к нашему старому примеру. Рассмотрим данные, которые хранятся в реестре исходной системы.



Как видно из вышеприведенного снимка экрана, каждой букве соответствует некоторое двоичное значение, которое как минимум зависит от конкретного устройства (уникально для каждого устройства) и физического местоположения раздела на нем. Кроме того, для букв C: и E: первые 5 байтов соответствующих им кодов совпадают, то же самое можно отметить и для букв D: и F:, что означает, что первыми 5-ю байтами однозначно идентифицируется жесткий диск. Следующими байтами однозначно определяется раздел. Именно таким образом ОС MS Windows 2000 привязывает буквы к разделам жестких дисков. Соответственно, подставляя двоичные коды, соответствующие одним буквам в коды для других букв, причем, внося изменения в реестр не исходной, а “сбойной” системы, можно добиваться искусственного назначения требуемой буквы для системного раздела в “сбойной” системе.

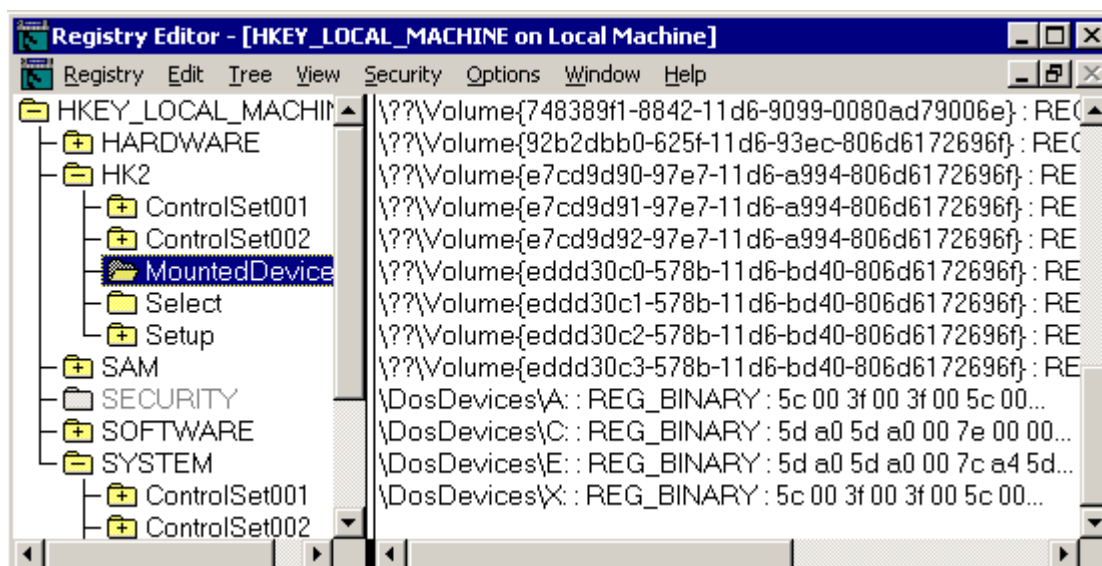
**Важное примечание.** В нашем примере до подключения дополнительного диска и операции копирования разделов исходную операционную систему ни разу не загружали с подключенным дополнительным диском. После же

операции копировании исходную систему загружали с дополнительным диском хотя бы один раз, после чего исходная система “распознала” дополнительный диск и назначила буквы его разделам.

Составим карту разделов, букв и двоичных кодов для нашего примера:

Disk0 / Partition1	C:	5DA05DA0007E000000000000
Disk0 / Partition2	E:	5DA05DA0007C54AD000000000
Disk1 / Partition1	D:	AC7E7303007E000000000000
Disk1 / Partition2	F:	AC7E7303007CA45D00000000

Disk0 – исходный диск, Disk1 – дополнительный диск со сбойной системой. Обратимся теперь к ключу MOUNTEDDEVICES системной ветви реестра “сбойной” системы (HK2):



Здесь мы видим то, что буквам C: и E: соответствуют коды разделов не дополнительного, а исходного диска. Это произошло по той причине, что в данном примере загрузка с дополнительного диска еще ни разу не проводилась, а также не выполнялась загрузка исходной системы с подключенным дополнительным диском до операции копирования разделов. Соответственно, реестр “один к одному” скопировался в том состоянии, каким он был до подключения дополнительного диска.

При попытке загрузиться с дополнительного диска, ОС обнаружит диск, отличающийся от исходного, и, соответственно, заново будет назначать буквы

для разделов “нового диска”. Если при загрузке кроме дополнительного диска других подключенных размеченных дисков не будет, то разделы получают буквы по порядку C: и D:, после чего система загрузится, но войти будет невозможно из-за того, что системному разделу назначена буква, отличная от E:.

Для такой ситуации существуют два способа решения проблемы:

- 1) Внести корректировки в реестр “сбойной” системы еще до того, как с дополнительного диска будут загружать систему. Назовем это *априорной корректировкой*.
- 2) Внести корректировки в реестр “сбойной” системы уже после того, как с дополнительного диска хотя бы один раз загружали систему и не могли войти в систему, после чего дополнительный диск подключили к работоспособной системе для внесения изменений в реестр. Назовем это *апостериорной корректировкой*.

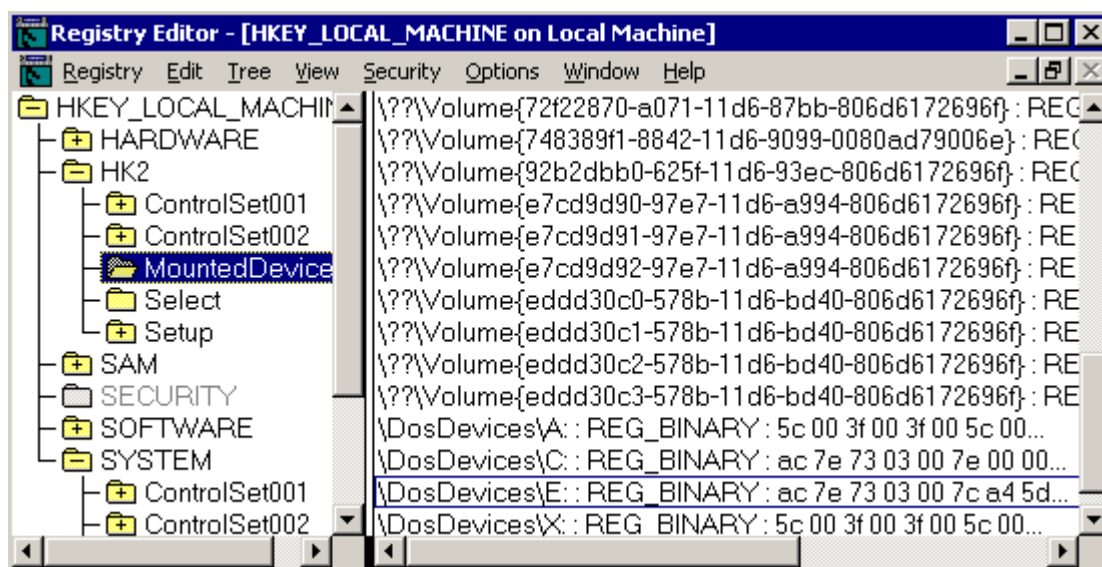
#### **Априорная корректировка**

В данном случае корректировки в реестр “сбойной” системы вносят, загрузив исходную систему сразу же после операций копирования на дополнительный диск. Вернемся к нашему примеру. Как видим из вышеприведенных снимков экрана, в реестре “сбойной” системы для букв C: и E: хранятся коды, соответствующие разделам дисками исходного диска. Следовательно, эти коды необходимо поменять на коды, соответствующие разделам дополнительного диска со “сбойной” системой. Для этого обращаемся к ключу SYSTEM\MOUNTEDDEVICES реестра исходной системы, по карте дисков видим то, что в исходной системе разделам дополнительного диска, соответствуют буквы D: и F:. Код, закрепленный за буквой F: – это и есть требуемый код для буквы E: в “сбойной” системы, а код для буквы D: в исходной системе – это искомый код для буквы C: в “сбойной”.

*Для корректировки необходимо скопировать двоичный код для буквы, соответствующей системному разделу дополнительного диска в реестре исходной системы, в код для буквы, соответствующей системному разделу исходного диска, в ключе MOUNTEDDEVICES реестра “сбойной” системы.*

Соответственно, также можно скопировать значение для буквы, соответствующей загрузочному разделу дополнительного носителя в исходном реестре в значение для буквы, соответствующей загрузочному разделу исходного диска в ключе MOUNTEDDDEVICES ветви реестра “сбойной” системы, хотя это и необязательно, поскольку первому разделу жесткого диска по умолчанию будет назначена буква C:. Проще говоря, мы с помощью исходной системы выясняем коды для разделов дополнительного диска, а потом эти данные заносим в реестр “сбойной” системы. В нашем примере, мы должны скопировать код для буквы F: в исходном реестре, в код для буквы E: в реестре “сбойной” системы. Также можно скопировать код для буквы D: в исходном реестре, в код для буквы C: в реестре “сбойной” системы.

В результате реестр будет выглядеть следующим образом:



Карта разделов жесткого диска для “сбойной” системы будет следующей:

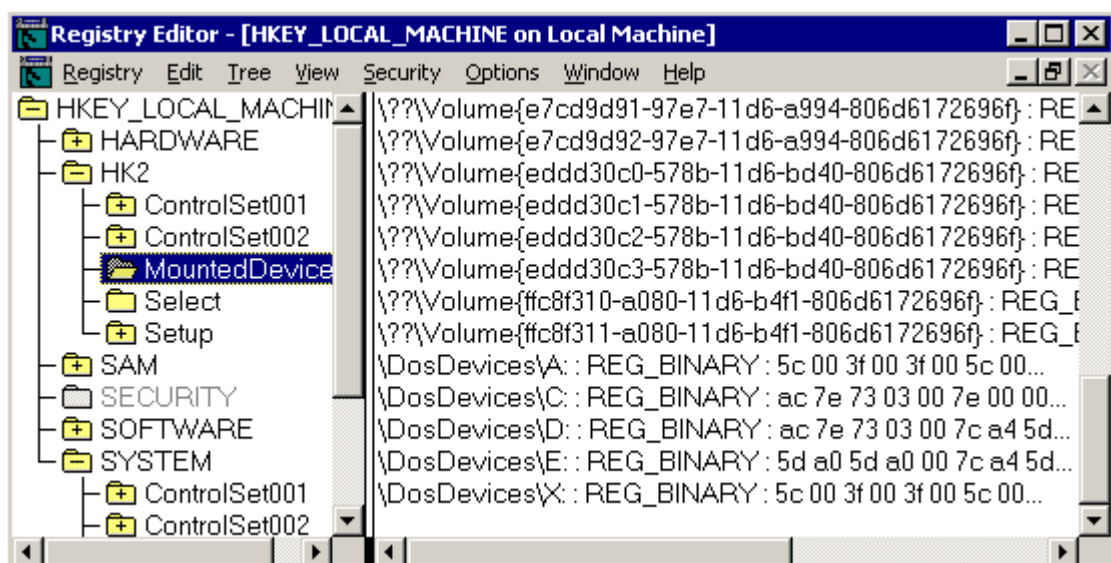
Disk0 / Partition1	C:	AC7E7303007E000000000000
Disk0 / Partition2	E:	AC7E7303007CA45D00000000

После внесения изменений **необходимо** выделить ветвь HK2 реестра “сбойной” системы и в меню **Registry** выполнить **Unload Hive** для того, чтобы сохранить изменения в реестре. В результате вышеописанных операций “сбойная” система, являющаяся копией исходной и находящаяся на дополнительном диске, становится работоспособной.

### Апостериорная корректировка

В данном случае мы имеем дело с дополнительным диском, с которого пытались загрузить систему хотя бы раз, в результате чего система назначила на свое усмотрение буквы разделам дисков и сохранила эту информацию в реестре в ключе HKEY\_LOCAL\_MACHINE\SYSTEM\MOUNTEDDEVICES.

Обратимся снова к нашему примеру. Допустим, что мы уже выполнили копирование загрузочного и системного разделов с исходного диска на дополнительный диск, и после этого подключили этот диск как основной на исходный либо конечный компьютер, на котором нет иных IDE устройств, и попытались выполнить загрузку. После безуспешных попыток входа в рабочий сеанс пользователя системы мы подключаем дополнительный диск вместе с исходным диском, используемым как основной, на исходный компьютер и выполняем загрузку исходной системы (в действительности, мы можем подключить дополнительный диск к любому компьютеру с работоспособной системой MS Windows 2000). Тогда, запустив редактор реестра REGEDT32 и загрузив ветвь HKEY\_LOCAL\_MACHINE\SYSTEM реестра “сбойной” системы, находящейся во втором разделе дополнительного диска (файл \Winnt\system32\config\system), и назвав эту ветвь HK2, обратимся к ключу HK2\MOUNTEDDEVICES и увидим следующее:



На снимке экрана мы видим то, что первому и второму разделу дополнительного диска соответствуют буквы C: и D:. Когда система пыталась загружаться с дополнительного диска, карта дисков была следующей:

Disk0 / Partition1	C:	AC7E7303007E000000000000
Disk0 / Partition2	D:	AC7E7303007CA45D00000000

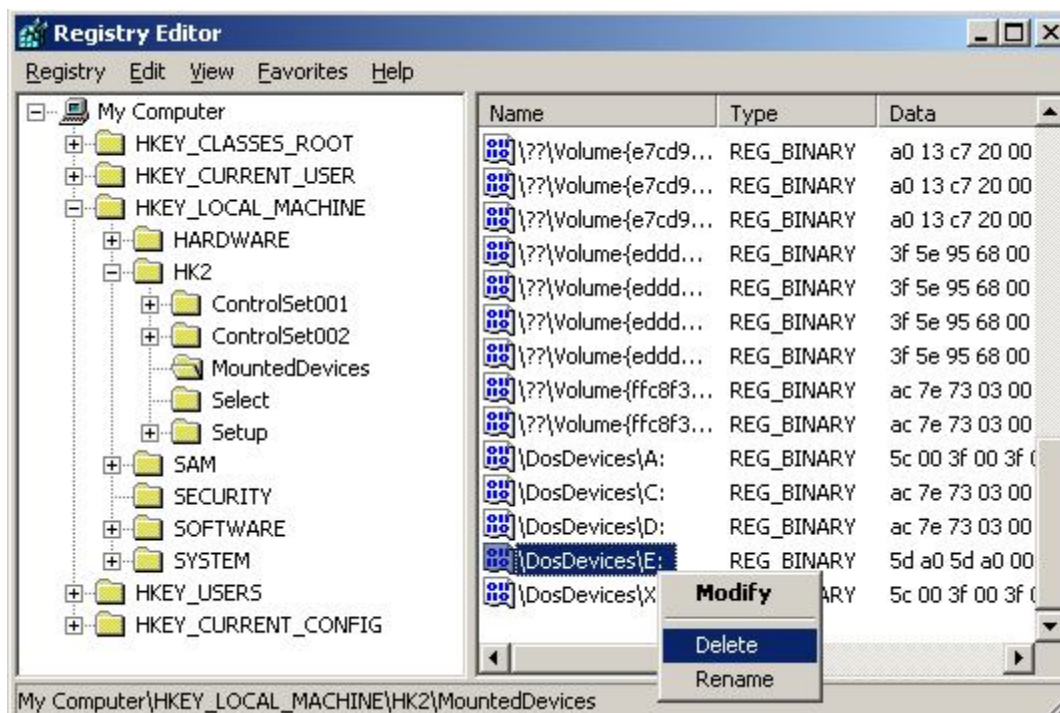
При первой загрузке с дополнительного диска, очевидно, система обнаружила неизвестный жесткий диск и выполнила повторное назначение букв разделам. В нашем примере загрузочный раздел получил букву C:, а системный – D:. Код для буквы E: остался без изменений и соответствует системному разделу исходного диска, который при загрузке системы физически недоступен. В общем случае, системный раздел может быть не вторым, а любым другим разделом дополнительного диска, и на компьютере, где выполнялась первая загрузка системы, может быть установлено множество IDE устройств, и системный раздел может получить любую букву от C: до Z:.

*Для корректировки необходимо привязать в реестре “сбойной” системы требуемую букву к двоичному коду системного раздела диска со “сбойной” системой. Для этого в реестре “сбойной” системы достаточно переименовать параметр \DosDevices\<текущая буква>, соответствующий системному разделу диска со “сбойной” системой, в параметр \DosDevices\<требуемая буква>. Требуемая буква в данном случае – буква, соответствующая системному разделу в исходной системе на исходном жестком диске.*

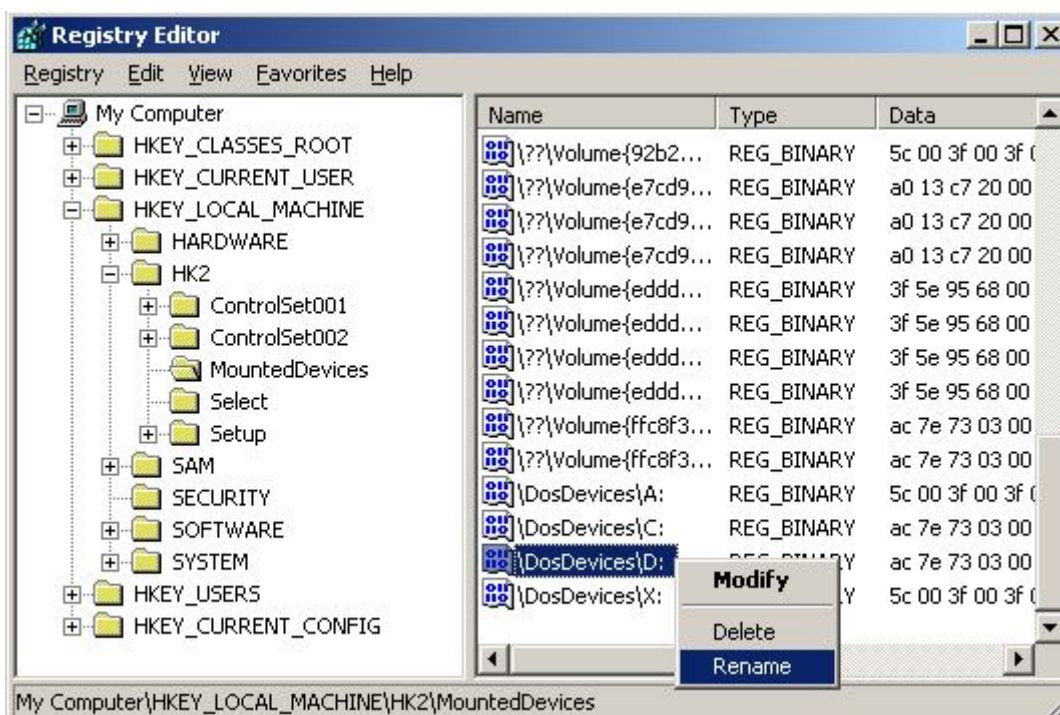
В нашем примере мы должны переименовать переменную \DosDevices\<D:> в переменную \DosDevices\<E:> в ветви HK2\MOUNTEDDEVICES. Поскольку редактор реестра REGEDT32 не позволяет переименовывать ключи и переменные, то можно воспользоваться альтернативным редактором REGEDIT, и в ветви HK2\MOUNTEDDEVICES выполнить следующие действия:



- Удалить существующий параметр \DosDevices\<E:>:



- Переименовать параметр \DosDevices\<D:> в \DosDevices\<E:>:



После этого необходимо закрыть редактор REGEDIT, вернуться в редактор REGEDT32, выделить ветвь HK2 и в меню **Registry** обязательно выполнить **Unload Hive** для того, чтобы сохранить изменения в реестре.

В результате этих операций “сбойная” система, являющаяся копией исходной и находящаяся на дополнительном диске станет работоспособной.

## Порядок действий при переносе системы с реального ПК на виртуальную машину

- Для подготовки к переносу необходимо воспользоваться дополнительным диском для того, чтобы не вносить изменения в оригинал исходной системы. На дополнительный жесткий диск по умолчанию необходимо копировать только загрузочный раздел (обычно это первый раздел первого жесткого диска системы), и раздел, на котором установлена система. Соответственно, на дополнительном жестком диске необходимо предварительно создать, как минимум, два (один, если система расположена на загрузочном разделе) раздела, первый раздел обязательно сделать активным (это можно сделать с помощью встроенной в ОС MS Windows 2000 утилиты Disk Management), выполнить копирование разделов по отдельности с исходного жесткого диска на дополнительный диск и в дальнейшем проводить работы только с ним.

**Примечание 1.** Перенос разделов и жестких дисков с другими (несистемными) данными при необходимости выполняется отдельно и это не представляет особой сложности.

**Примечание 2.** Системный раздел исходного диска может иметь в исходной операционной системе “нетипичную” букву. В таком случае необходимо выполнить априорную корректировку.

- Установить дополнительный диск на исходный ПК и загрузить с него операционную систему.

**Примечание.** Если в системе невозможно войти в рабочий сеанс пользователя (очень маловероятный случай, разве что после некачественного выполнения априорной корректировки в случае “нетипичной” буквы системного раздела жесткого диска с исходной системой), то в таком случае необходимо выполнить апостериорную корректировку, используя исходную систему, затем снова попытаться загрузиться с дополнительного диска.

- Подготовить систему к переносу. Для этого выполнить следующее:



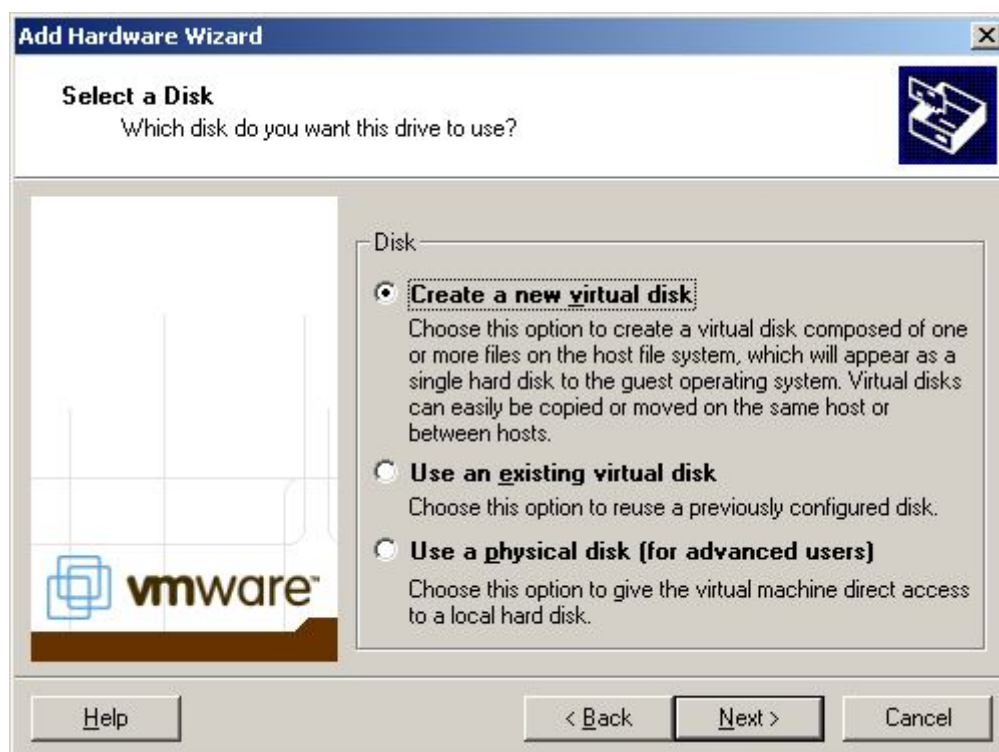
- Для устранения аппаратной несовместимости необходимо заменить тип платформы в диспетчере устройств на **Standard PC**.

**Примечание.** После смены драйвера аппаратной платформы система будет предлагать перезагрузить систему. От этого необходимо отказаться и проводить дальнейшие подготовительные операции.

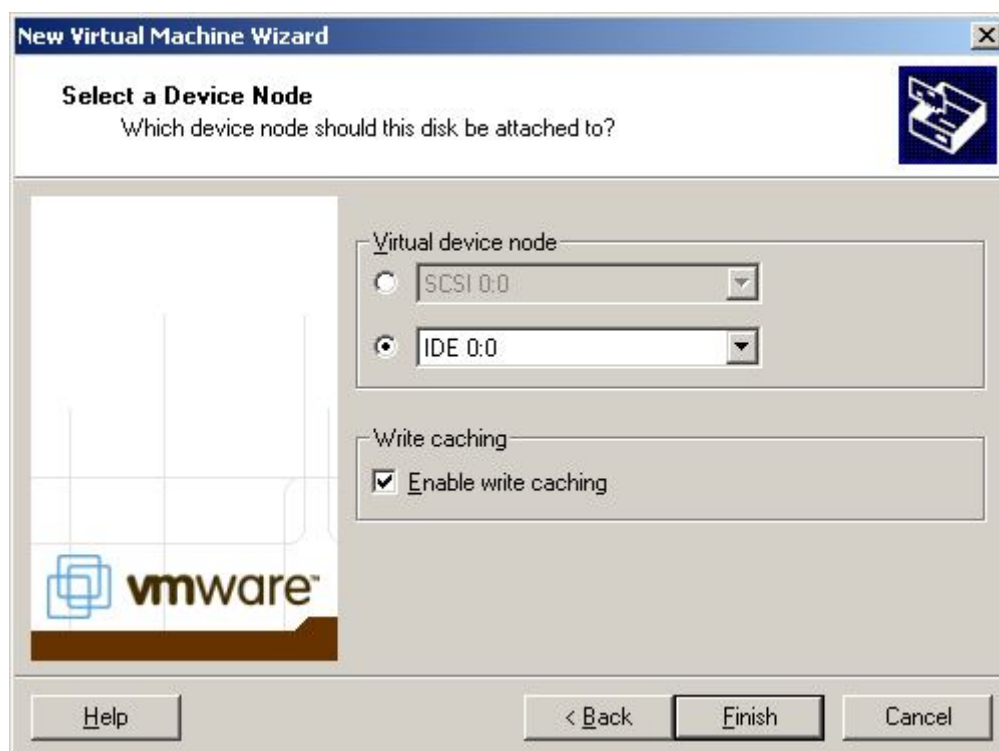
- Для избежания ошибки "**0x0000007B - Inaccessible boot device**", возникающей из-за несовместимости контроллеров жестких дисков необходимо удалить все драйверы RAID-массивов и прочие, так или иначе связанные с работой жестких дисков, а также удалить все IDE-контроллеры в диспетчере устройств.

**Примечание.** При удалении устройств система будет предлагать перезагрузку, от этого необходимо отказаться.

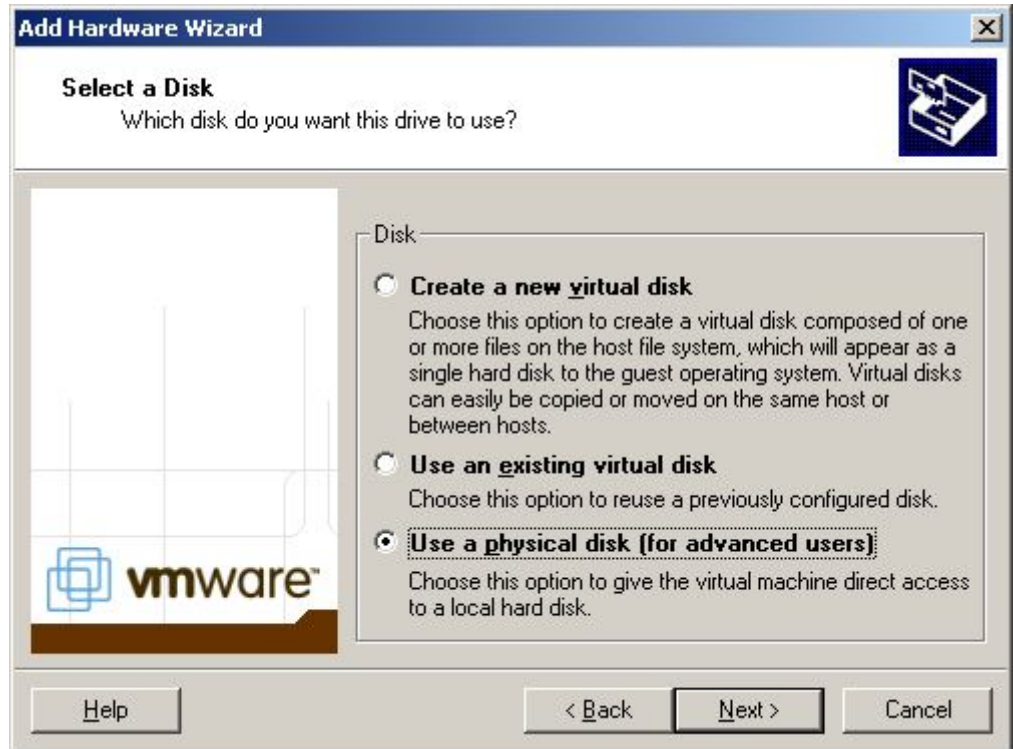
- Завершить работу системы и ни в коем случае не загружать ее на исходном компьютере.
- Подключить дополнительный диск с подготовленной к переносу системой к базовому компьютеру, где предполагается размещать виртуальные машины.
- Создать на базовом компьютере новую виртуальную машину:
  - При создании машины указать режим конфигурирования Custom и в качестве типа гостевой системы указать ОС MS Windows 2000.
  - Указать требуемую для системы емкость для оперативной памяти
  - Тип сети обязательно указать Bridged Networking, для возможности доступа в сеть, за пределами базового компьютера.
  - При начальном конфигурировании создать новый виртуальный жесткий диск с заданной необходимой емкостью (в зависимости от конкретной задачи переноса – суммарный размер загрузочного и системного раздела диска оригинала, либо размер, равный емкости диска оригинала, либо иное, отдельно оговоренное).



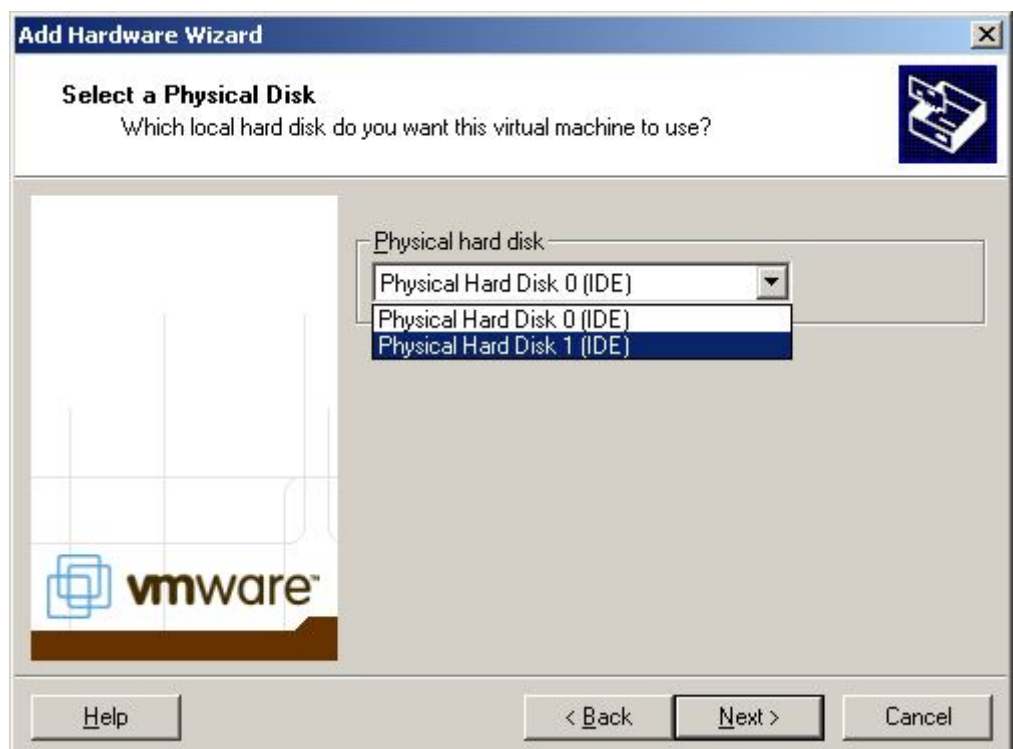
Очень важно то, чтобы диск был IDE устройством, так как по умолчанию для Windows 2000 систем VMWARE создает SCSI-диск, который нельзя потом обратить в IDE. Для этого в окне, в котором указывается имя файла образа диска, нажать кнопку Advanced и в появившемся окне выбрать устройство IDE 0:0 (primary master):



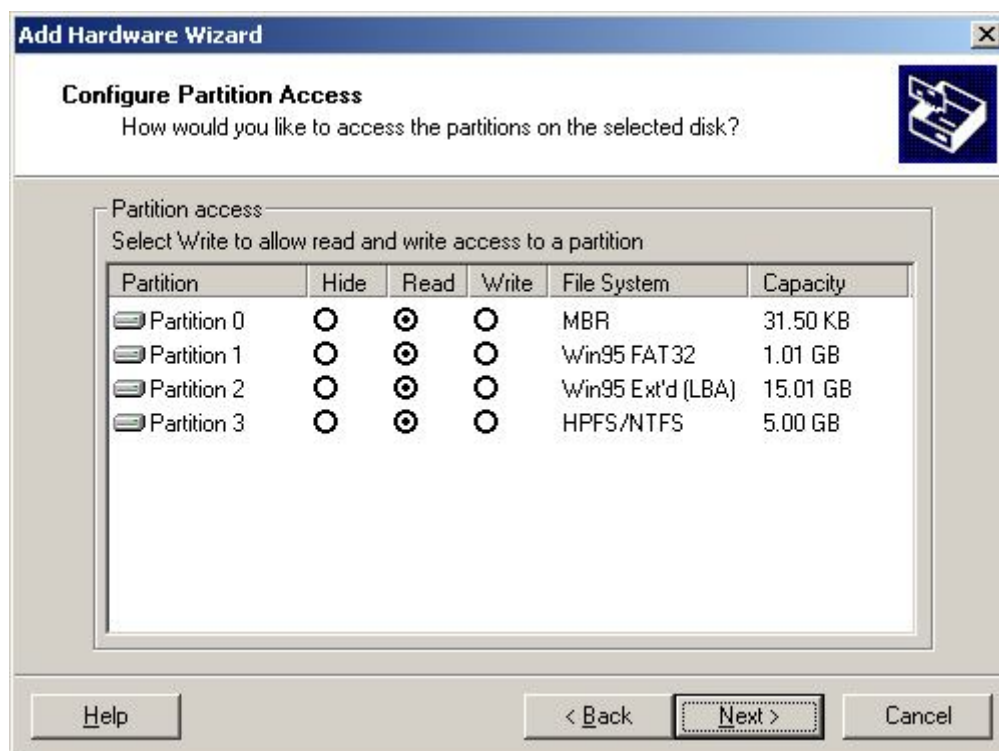
- Далее необходимо к виртуальной машине подключить дополнительный диск со скопированной исходной системой. Для этого в конфигурации виртуальной машины необходимо добавить второе дисковое устройство, которое будет использовать данные физического дополнительного диска:



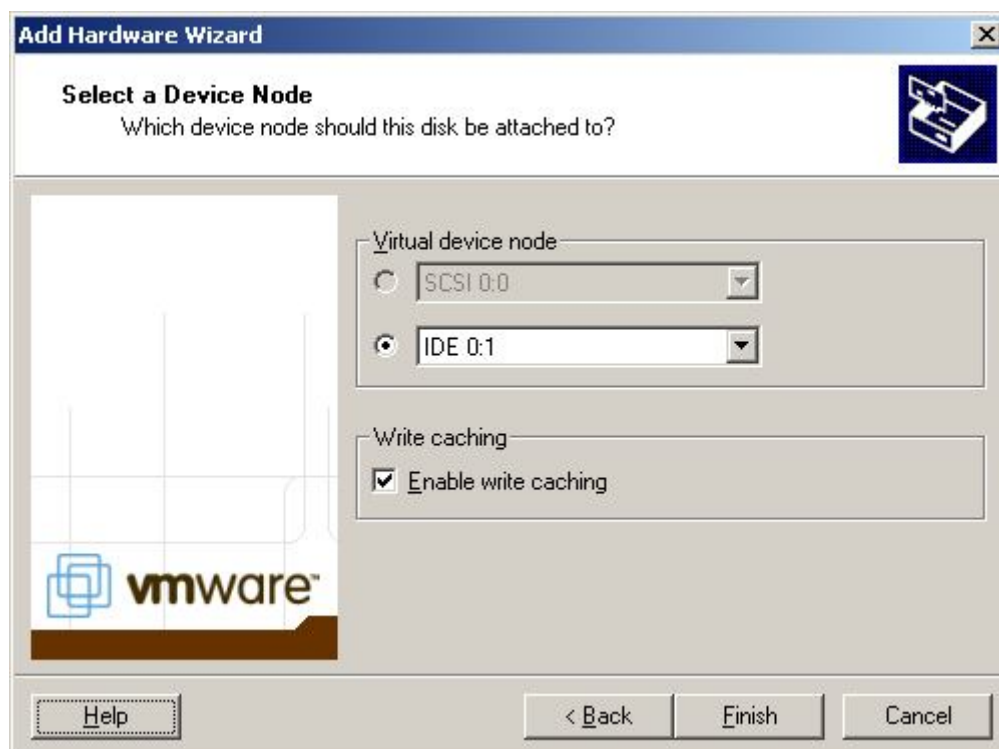
В следующем окне необходимо указать номер физического устройства базового компьютера, соответствующего дополнительному диску:



В окне конфигурирования доступа к разделам физического диска оставить по умолчанию разрешение только на чтение ко всем разделам:



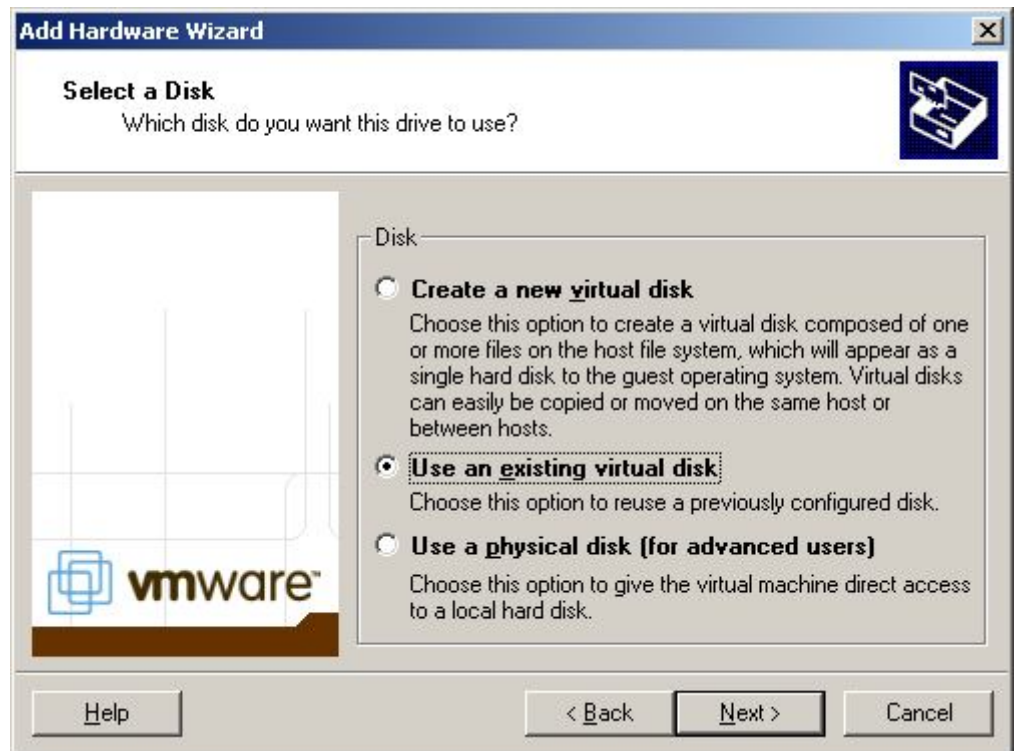
В окне, в котором указывается имя файла конфигурации диска, нажать кнопку **Advanced** и в появившемся окне выбрать устройство IDE 0:1:



- Запустить виртуальную машину и выполнить загрузку с заранее подготовленной дискеты. Дискета обязательно должна содержать дисковые утилиты для конфигурирования разделов и копирования данных разделов с одного диска на другой. При помощи дисковых утилит создать на первом жестком диске необходимые разделы, первый раздел сделать обязательно активным. Выполнить копирование загрузочного и системного разделов по отдельности со второго жесткого диска на первый.
- Запустить виртуальную машину, обязательно заранее отключив в конфигурации второй диск (дополнительный диск). Если этого не сделать, то при загрузке система будет использовать его в процессе назначения букв при обнаружении неизвестных ей носителей и системный раздел может получить букву, отличную от той, которая была в исходной системе, что приведет к невозможности входа в систему.

**Примечание.** Если в исходной системе была привязана “нетипичная” буква к системному разделу, либо при первой загрузке виртуальной машины дополнительный диск по неаккуратности был оставлен подключенным, то вход пользователя виртуальной системы в рабочий сеанс может стать невозможным. В этой ситуации необходимо выполнить **апостериорную корректировку**. Нужно воспользоваться вспомогательной работоспособной виртуальной системой на базе MS Windows 2000, к которой подключить основной диск “сбойной” машины как дополнительный диск. Для этого требуется скопировать на базовый компьютер файлы вспомогательной виртуальной ОС, открыть ее и перейти в окно конфигурирования, где добавить диск “сбойной” системы как дополнительный жесткий диск:

- В окне выбора типа диска указать использование существующего виртуального диска:



- В следующем окне указать путь к файлу образа загрузочного жесткого диска “сбойной” системы.
- В окне выбора типа подключения указать Persistent. Здесь же можно нажать кнопку Advanced и указать устройство IDE или SCSI, которому будет соответствовать дополнительный диск. Однако в данном случае это не так существенно. Важно лишь, чтобы этот диск был доступен во вспомогательной виртуальной ОС.
- Загрузить вспомогательную систему и выполнить апостериорную корректировку.
- Завершить работу вспомогательной системы.
- Загрузить виртуальную машину и убедиться в корректности ее работы.
- После обнаружения системой устройств и установки драйверов установить VMWARE Tools (описание установки есть в документации к VMWARE).

### Приложение 3. Обзор технологии виртуальных машин

#### 1) Причины появления виртуальных машин

*Виртуальная машина (Virtual Machine)* – это абстрактный компьютер, который эмулируется и поддерживается на реальном компьютере.

Само развитие вычислительной техники привело к появлению виртуализации для того, чтобы избавить пользователя – программиста от трудных, однообразных, отнимающих множество времени действий по непосредственному программированию аппаратной части, стали разрабатываться первые операционные системы, берущие на себя задачи по взаимодействию с аппаратными средствами.

*Операционная система (Operating system)* – упрощенно это комплекс управляющих и обрабатывающих программ, организующих вычислительный процесс на компьютере. С появлением операционных систем подавляющее большинство пользователей стало взаимодействовать уже не с реальной (физической) вычислительной машиной, а с машиной, которую ему предоставила операционная система, т. е. с виртуальной машиной.

Операционная система в наибольшей степени определяет облик всей вычислительной системы в целом. Основными функциями ОС являются повышение эффективности использования компьютера путем рационального управления его ресурсами и обеспечение пользователю удобств по взаимодействию с аппаратурой, посредством предоставления для него виртуальной машины.

#### **Операционная система как система управления ресурсами**

Современные вычислительные системы могут состоять из множества компонентов: процессоры, память, таймеры, диски, сетевые коммуникационные устройства, принтеры и т.д. Соответственно, основной функцией операционной системы является распределение ресурсов процессоров, памяти, устройств и данных между процессами, конкурирующими за эти ресурсы. Операционная система должна управлять

всеми ресурсами вычислительной машины таким образом, чтобы обеспечить максимальную эффективность ее функционирования.

Управление ресурсами включает решение двух общих, не зависящих от типа ресурса задач:

- планирование ресурса – определение того, какому потребителю, в какой момент времени, а для разделяемых ресурсов и в каком объеме, необходимо выделить данный ресурс.
- отслеживание состояния ресурса – поддержка оперативной информации о том, занят или не занят ресурс, а для разделяемых ресурсов – какой объем ресурса уже распределен, а какой еще свободен.

Для решения этих общих задач управления ресурсами разные ОС используют различные алгоритмы, что, в конечном счете, и определяет их облик в целом, включая характеристики производительности, область применения и даже пользовательский интерфейс. Так, например, алгоритм управления процессором в значительной степени определяет, является ли ОС системой разделения времени, системой пакетной обработки или системой реального времени.

### **Операционная система как виртуальная машина**

Идея о том, что операционная система это, прежде всего, некоторый механизм, управляющий всеми частями сложной системы, соответствует рассмотрению “снизу вверх” – от аппаратных средств к конечным приложениям. В идеале ОС должна не только исключать необходимость взаимодействия приложений с оборудованием напрямую, но максимально предотвращать возможность прямого взаимодействия. Тем не менее, до сих пор встречаются ОС, которые допускают это (MS DOS, Windows 95/98 позволяют приложениям напрямую работать с аппаратурой через порты ввода-вывода и в семействе ОС фирмы Microsoft только лишь начиная с Windows NT такая возможность была заблокирована), что резко негативно отражается на безопасности, надежности и устойчивости функционирования системы. Другой



взгляд, “сверху вниз”, дает представление об операционной системе как о механизме, обеспечивающим удобный интерфейс пользователям и его приложениям, нуждающихся в некоторых вычислительных ресурсах, системах хранения данных и устройствах ввода-вывода. Использование большинства компьютеров на уровне машинного языка затруднительно, особенно это касается ввода-вывода. Например, для организации чтения блока данных с гибкого диска программист может использовать 16 различных команд, каждая из которых требует 13 параметров, таких как: номер считывающей головки на диске, номер сектора на дорожке и т.п. Когда выполнение операции с диском завершается, контроллер возвращает 23 значения, отражающих наличие и типы ошибок, которые, очевидно, необходимо анализировать. Даже если не входить в курс реальных проблем программирования ввода-вывода, ясно, что среди программистов нашлось бы немного желающих непосредственно заниматься программированием этих операций. При работе с диском пользователю достаточно представлять его в виде некоторого набора файлов, каждый из которых имеет имя. Работа с файлом заключается в его открытии, выполнении чтения или записи, а затем в закрытии файла. Вопросы, подобные таким, как следует ли при записи использовать усовершенствованную частотную модуляцию или в каком состоянии сейчас находится двигатель механизма перемещения считывающих головок, не должны волновать пользователя. Точно так же, как операционная система ограждает пользователей от аппаратуры дискового накопителя и предоставляет ему простой файловый интерфейс, операционная система берет на себя все низкоуровневые процедуры, связанные с обработкой прерываний, управлением таймерами и распределением оперативной памяти и т.д.

С этой точки зрения функцией ОС является предоставление пользователю некоторой расширенной или виртуальной машины, которую легче программировать и с которой легче работать, чем непосредственно с аппаратурой, составляющей реальную машину.

### Эволюционное развитие ОС

Первые операционные системы имели монолитную структуру. На рисунке 1 приведена упрощенная структура подобных ОС.

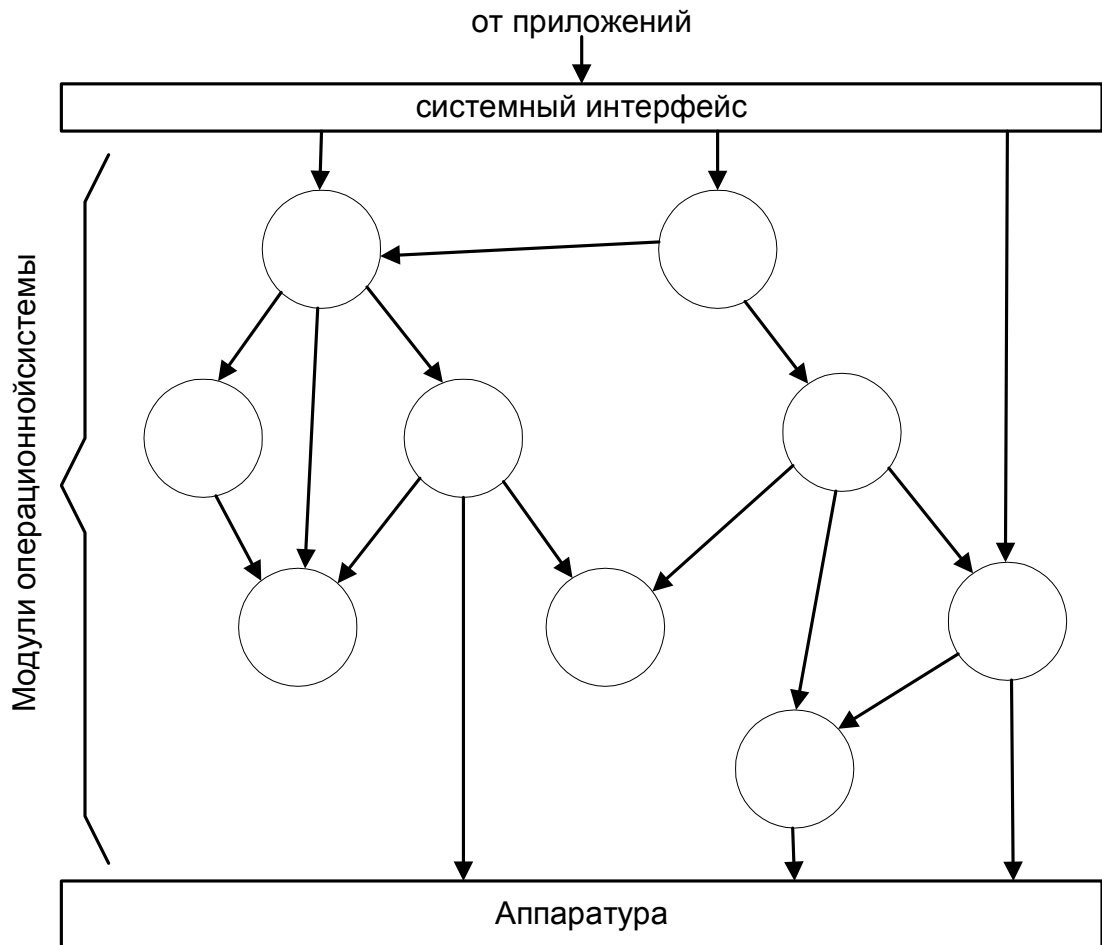


Рис. 1. Монолитная структура ОС

ОС представлялась как набор процедур, каждая из которых может вызывать другие, когда ей это нужно. При использовании такого подхода каждая процедура системы имеет хорошо определенный интерфейс в терминах параметров и результатов, и каждая способна вызвать любую другую для выполнения некоторой нужной для нее полезной работы. Для построения монолитной системы необходимо скомпилировать все отдельные процедуры, а затем связать их вместе в единый объектный файл с помощью компоновщика (примерами могут служить ранние версии ядра UNIX или Novell NetWare). Добавление новых функций и изменение существующих для монолитных ОС крайне сложно и требует очень хорошего знания всей структуры ОС.

Следующим этапом развития ОС стали многоуровневые ОС. ОС представлялась как иерархия уровней, которые образуются группами функций операционной системы – файловая система, управление процессами и устройствами и т.п. Каждый уровень может взаимодействовать только с выше или нижележащим уровнем. Прикладные программы или модули самой операционной системы передают запросы вверх и вниз по этим уровням. Кроме того, вовсе необязательно то, что какая-либо из компонентов ОС располагалась строго в пределах определенного уровня, она могла охватывать несколько уровней, обеспечивая внутри себя функциональность этих уровней (рис. 2).

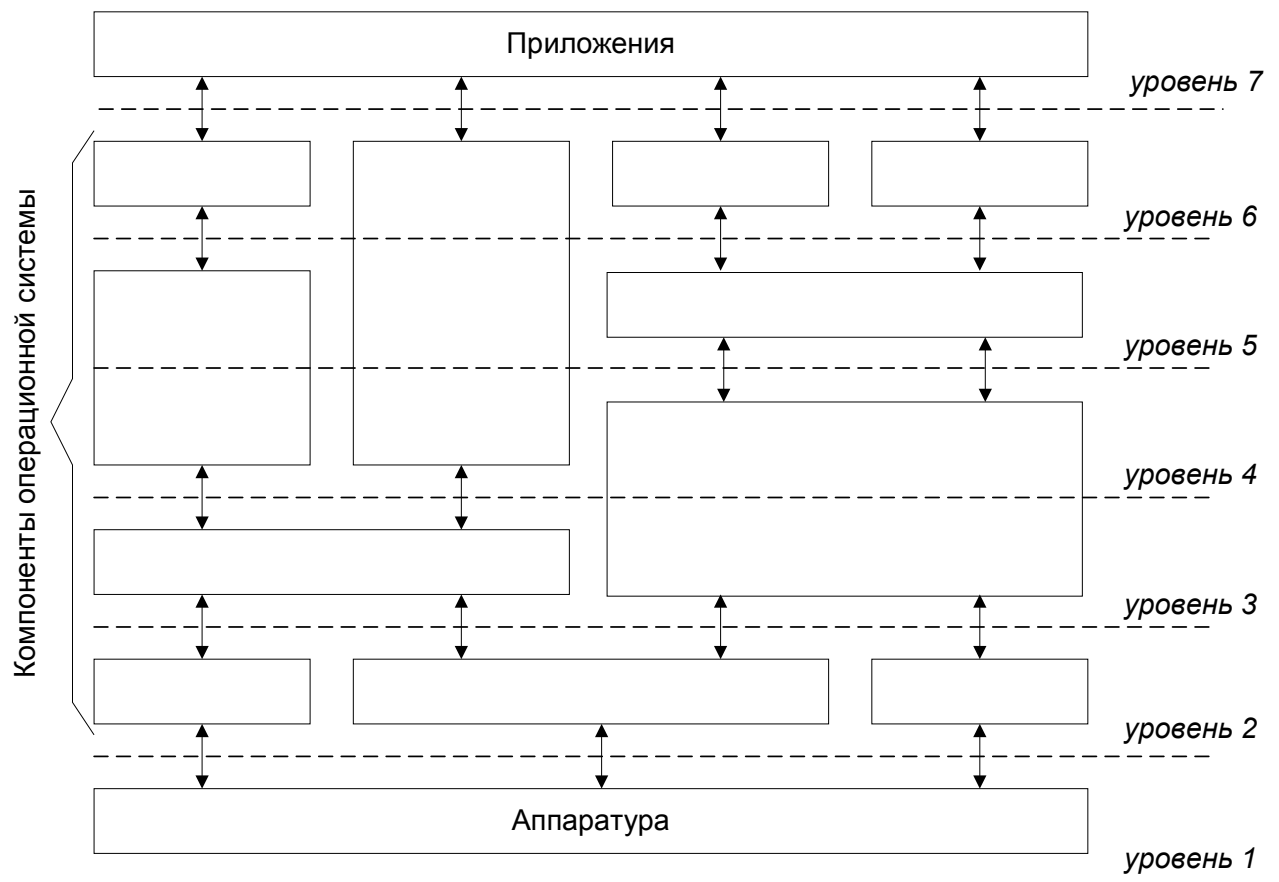


Рис. 2. Многоуровневая структура ОС

Первой системой, построенной таким образом, была простая пакетная система TNE, которую в 1968 году построил Эдсгер Дейкстра, выдающийся программист и один из основоположников информатики.

Тем не менее, в системах, имеющих многоуровневую структуру, нелегко удалить один слой и заменить его другим в силу множественности и размытости интерфейсов между слоями. Добавление новых функций и

изменение существующих также требует хорошего знания операционной системы и трудоемких работ, поэтому на смену многоуровневой модели пришла модель клиент-сервер и тесно связанная с ней концепция микроядра.

Модель клиент-сервер – это следующий этап в эволюции ОС. В широком смысле модель клиент-сервер предполагает наличие программного компонента – потребителя какого-либо сервиса – клиента, и программного компонента – поставщика этого сервиса – сервера. Взаимодействие между клиентом и сервером стандартизуется, так что сервер может обслуживать клиентов, реализованных различными способами и, может быть, разными производителями. Применительно к ОС, идея состоит в разбиении ее на несколько процессов – серверов, каждый из которых выполняет отдельный набор сервисных функций – например, управление памятью, планирование процессов и т.д. Каждый сервер выполняется в пользовательском режиме. Клиент, которым может быть либо другой компонент ОС, либо прикладная программа, запрашивает сервис, посылая сообщение на сервер (рис. 3).

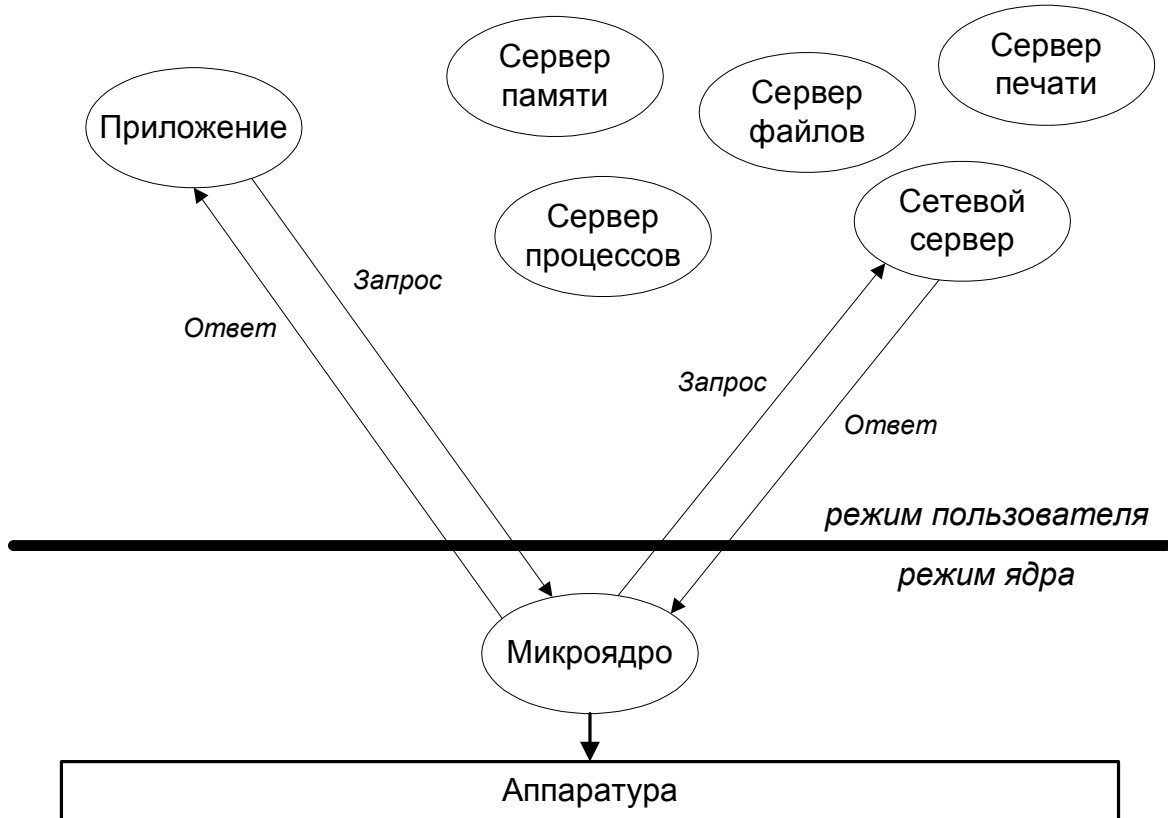


Рис. 3. Структура клиент-серверной ОС

Ядро ОС (называемое здесь микроядром), работая в привилегированном режиме, доставляет сообщение нужному серверу, сервер выполняет операцию, после чего ядро возвращает результаты клиенту с помощью другого сообщения. Подход с использованием микроядра заменил вертикальное распределение функций операционной системы на горизонтальное. Компоненты, лежащие выше микроядра, хотя и используют сообщения, пересылаемые через микроядро, взаимодействуют друг с другом непосредственно. Микроядро играет роль регулировщика: оно проверяет сообщения, пересылает их между серверами и клиентами, и предоставляет доступ к аппаратуре. Микроядро реализует жизненно важные функции, лежащие в основе операционной системы. Это базис для менее существенных системных служб и приложений. Главный принцип разделения работы между микроядром и окружающими его модулями – включать в микроядро только те функции, которым абсолютно необходимо исполняться в режиме супервизора и в привилегированном пространстве. Под этим обычно подразумеваются машинно-зависимые программы (включая поддержку нескольких процессоров), некоторые функции управления процессами, обработка прерываний, поддержка пересылки сообщений, некоторые функции управления устройствами ввода-вывода, связанные с загрузкой команд в регистры устройств. Эти функции операционной системы трудно, если не невозможно, выполнить программам, работающим в пространстве пользователя. Проблема заключается в том, что и как включать в состав ядра. На одном краю этой проблемы находится разрабатываемая фирмой IBM на основе микроядра Mach операционная система Workplace OS, придерживающаяся чистой микроядерной доктрины, состоящей в том, что все несущественные функции ОС должны выполняться не в режиме ядра, а в пользовательском режиме. На другом – Windows NT, в составе которой имеется исполняющая система (NT executive), работающая в режиме ядра и включающая ряд компонентов, которые управляют виртуальной памятью, объектами, вводом-выводом и файловой системой (включая сетевые драйверы), взаимодействием процессов, и частично системой безопасности.

**Принцип виртуализации в современных ОС**

Принцип виртуализации позволяет представить структуру системы в виде определенного набора планировщиков процессов и распределителей ресурсов и использовать единую централизованную схему распределения ресурсов.

По сути, любая операционная система, являясь средством распределения ресурсов и организуя по определенным правилам управление процессами, скрывает от пользователя и его приложений реальные аппаратные и иные ресурсы, заменяя их некоторой абстракцией. В результате пользователи видят и используют виртуальную машину как некое устройство, способное воспринимать их программы, написанные на определенном языке программирования, выполнять их и выдавать результаты. При таком языковом представлении пользователя совершенно не интересует реальная конфигурация вычислительной системы, способы эффективного использования ее компонентов и подсистем. Он мыслит и работает с машиной в терминах используемого им языка и тех ресурсов, которые ему предоставляются в рамках виртуальной машины. Виртуальная машина, предоставляемая пользователю, воспроизводит структуру реальной машины, но структурные элементы в таком представлении выступают с новыми или улучшенными характеристиками, часто упрощающими работу с системой. Характеристики могут быть произвольными, но чаще всего пользователи желают иметь собственную «идеальную» по техническим характеристикам машину в следующем составе:

- Единообразная по логике работы память (виртуальная) практически неограниченной емкости. Среднее время доступа соизмеримо со значением этого параметра оперативной памяти. Организация работы с информацией в такой памяти производится в терминах обработки данных – в терминах работы с сегментами данных на уровне выбранного пользователем языка программирования.
- Произвольное количество процессоров (виртуальных), способных работать параллельно и взаимодействовать во время работы. Способы управления процессорами, в том числе синхронизация и информационные

взаимодействия, реализованы и доступны пользователям на уровне используемого языка в терминах управления процессами.

- Произвольное количество внешних устройств (виртуальных), способных работать с памятью виртуальной машины параллельно или последовательно, асинхронно или синхронно по отношению к работе того или иного виртуального процессора, которые инициируют работу этих устройств. Информация, передаваемая или хранимая на виртуальных устройствах, не ограничена допустимыми размерами. Доступ к такой информации осуществляется на основе либо последовательного, либо прямого способа доступа в терминах соответствующей системы управления файлами. Предусмотрено расширение информационных структур данных, хранимых на виртуальных устройствах.

Степень приближения к «идеальной» виртуальной машине может быть большей или меньшей в каждом конкретном случае. Чем больше виртуальная машина, реализуемая средствами ОС на базе конкретной аппаратуры, приближена к «идеальной» по характеристикам машине и, следовательно, чем больше ее характеристики отличны от реально существующих, тем больше степень виртуальности у полученной пользователем машины.

## 2) Развитие идей виртуализации

### **Способы виртуализации**

Виртуализация устройств. Одной из основных функций операционной системы всегда была унификация способа взаимодействия пользовательских программ с аппаратурой, в частности, создание такого способа взаимодействия с устройством, при котором все специфические операции скрываются от пользователя. Таким образом, осуществляется виртуализация устройств — программа пользователя, работающая с подобным устройством, видит его как некое обобщенное, или виртуальное устройство.

Упомянутый способ виртуализации одного устройства обычно работает только внутри одной операционной системы, для ее клиентов (в роли которых,

обычно, выступают процессы). Только процессы ОС могут использовать подобные устройства. Если вспомнить, как процессор и программа пользователя могут взаимодействовать с устройствами, то оказывается, что на самом деле все взаимодействие с внешним миром у них ограничено некоторым специальным набором команд. Например, командами in/out в процессорах Intel или предопределенными адресами памяти, ассоциированными аппаратурой с управлением устройствами. В этом случае можно осуществить эмуляцию «внешнего мира» и устройств для подобных команд так, как если бы процессор при их выполнении работал с реальным периферийным или другим оборудованием.

Подобные идеи возникли достаточно давно и трансформировались в специфический подход создания виртуального компьютера “над” физическим. В общих чертах, создание виртуального компьютера как программы позволило бы использовать внутри него типовую операционную систему, с поддержкой типовых “внешних” (хотя и виртуальных) устройств и обеспечить дополнительный уровень изоляции копий операционных систем друг от друга (естественно, если на одном физическом компьютере есть возможность запустить несколько виртуальных).

Разделение ресурсов по разделам. Другой способ — это разделение ресурсов по разделам. По этому пути пошли компании Compaq, Sun Microsystems, IBM, HP. Покупая компьютер, на самом деле пользователь получает большой набор ресурсов, из которых он может «собрать» несколько виртуальных компьютеров. Так, например, получив 16-процессорную версию системы, можно разбить ее на 16 автономных однопроцессорных компьютеров или на 10 однопроцессорных и одну 6-процессорную вычислительную систему. Имеющиеся ресурсы отдаются в исключительное пользование какой-либо конкретной системе. Плюсами подобного решения являются простота конфигурации и эксплуатации в сочетании с высоким уровнем гарантии ресурсов. При таком подходе если необходимо быть уверенным, что при запросе на обслуживание для обработки требуется получить целиком один



процессор, то он будет предоставлен. Такой режим, одновременно является и минусом — отсутствие разделения ресурса между разными потребителями понижает степень его загруженности и, соответственно, повышает цену решения. Ярким примером такой системы служит мэйнфрейм от IBM - OS/390 с ОС VM и системой LPAR — Logical partitioning. С этой системой пользователь получал в свое распоряжение полноразмерный и полнофункциональный виртуальный компьютер, на который он мог поставить собственную версию ОС и установить собственное прикладное ПО. Этот компьютер включал оперативную память, возможность использования процессора, собственные виртуальные периферийные устройства типа диска или сетевой карты — практически все то, чем обладает обычный компьютер, только в виртуальном виде. Количество обслуживаемых виртуальных компьютеров на одном аппаратном компьютере определялось многими факторами, в частности, лицензией, доступными ресурсами типа памяти, диска, возможностями центрального процессора и т.д. На мощной физической машине можно было запустить множество виртуальных, например, IBM утверждает, что на S/390 возможно запустить более десятка тысяч копий виртуальных компьютеров с ОС Linux.

### **Виртуализация в архитектуре x86**

Стоит отметить, что возможность виртуализации была заложена в архитектуре x86 много лет тому назад, когда инженеры корпорации Intel, проявив прозорливость, изначально заложили в свои процессоры, начиная с Intel 80386 и выше, потенциальную возможность виртуализации, заключающуюся в следующих особенностях:

- Наличие защищенного режима, в котором параллельные вычисления могут быть защищены аппаратно – программными механизмами.
- 32-битная архитектура обеспечивает программные ресурсы, необходимые для поддержки "больших" систем, характеризующихся операциями с большими числами, большими структурами данных, большими программами (или

большим числом программ) и т.п. Физическое адресное пространство 80386 возможно до 4 Гбайт, а его логическое адресное пространство – до 64 терабайт (Тбайт).

- Уровни привилегий 0 – 3. Уровни привилегий используются для установки различных стратегий защиты. Уровень 0 является наиболее привилегированным, а 3 – наименее. Система без защиты может быть реализована путем простого помещения всех процедур в сегмент (или сегменты), чей уровень привилегии равен 0. Традиционное разделение на супервизора и пользователя может быть реализовано путем помещения прикладной задачи в сегмент с уровнем привилегии 3, а процедур супервизора – в сегмент с уровнем привилегий 0.
- Настраиваемый пул ввода-вывода. Системы, базирующиеся на 80386, могут распределять адресное пространство устройств ввода-вывода в пространство основной памяти процессора или в отдельное пространство ввода - вывода.
- Обеспечение работы с виртуальной памятью. Виртуальная память, позволяет ставить максимальный объем программы или группы программ в зависимость от имеющегося адресного пространства на диске, а не от емкости физической памяти.

Архитектура x86 предполагает, что ядру работающей на этой платформе операционной системе будут доступны абсолютно все ресурсы процессора, поэтому отчуждение отдельно взятой ОС от процессора теоретически невозможно. Попытка совместить две ОС на паритетных условиях приведет к конфликту между ними. Следовательно, если необходимо, чтобы несколько ОС параллельно сосуществовали на одном или нескольких процессорах, приходится эти ОС ранжировать, выделить из них одну, ту, которая будет обращаться непосредственно к процессорам на нулевом уровне; ее называют **базовой ОС**, а остальные ОС, работающие на виртуальных машинах, являются **гостевыми ОС**. Существующая на сегодняшний день модель виртуальной машины на основе архитектуры x86 представлена на рисунке 4.



Рис. 4. Модель виртуальной машины на основе архитектуры x86

Соответственно, можно говорить о так называемом *базовом ПК* – реальной машине работающей под управлением базовой ОС, и *гостевом ПК* – виртуальной машине работающей под управлением гостевой ОС.

В данной модели "уровень виртуализации" – это часть архитектуры являющаяся приложением по отношению к базовой ОС, и является инструментом для создания и управления виртуальными машинами.

Уровень "эмуляция устройств" отвечает за обращение напрямую к ресурсам процессора, работая в привилегированном режиме, чтобы не перегружать лишними обращениями базовую ОС.

### 3) Существующие системы виртуализации для архитектуры x86

На рынке присутствует несколько программных продуктов, работающих в современных многозадачных операционных системах, которые позволяют эмулировать несколько независимых виртуальных ПК на одном физическом, наиболее популярные из которых это: **Virtual PC** от компании Connectix Corp. и продукты компании VMware Inc: **VMware Workstation**, **GSX** и **ESX Server**.

**Virtual PC** – эмулятор, изначально предназначавшийся для выполнения программ Windows на машинах Apple Macintosh. Последняя редакция программы, “Version”, работает с Windows XP (Home и Professional), Windows 2000 Professional, Windows Me, Windows 98 SE и Windows NT 4.0. На основе этих базовых операционных систем можно построить разнообразные виртуальные машины, работающие не только с последними платформами Windows и OS/2 Warp (2.1, 3.0, 4.0, WSeB и eCS), но и DOS, всеми версиями Windows начиная с 3.1, NetWare (5.0, 5.1 и 6.0) и Solaris. Программное обеспечение Virtual PC также успешно прошло тестирование со многими версиями ОС Linux.

При использовании Virtual PC каждый гостевой ПК располагает собственными аудио, видео и сетевыми платами. Пользователю не составляет труда изменить емкость памяти и жесткого диска по своему усмотрению. При надлежащей настройке гостевого ПК может запрашивать пользователя, следует ли сохранить изменения, сделанные на жестком диске в ходе сеанса работы, или отказаться от них. В Virtual PC также заложена возможность дистанционного управления гостевыми ПК с других машин сети.

Основными достоинствами Virtual PC являются:

- Низкая цена пакета (цена при прямых поставках 199 долларов).
- Поддержка всех распространенных ОС в качестве гостевых.
- Высокая функциональность.

Однако, Virtual PC обладает серьезными недостатками, а именно:

- Отсутствие поддержки SCSI и USB устройств.
- Отсутствие возможности использовать более одной сетевой платы в гостевой ОС, что лишает возможности построения виртуальных маршрутизируемых сетей.
- Низкая производительность при интенсивном сетевом обмене, трудности интеграции в реальную сеть.

Так что, несмотря на хорошую функциональность, низкую цену, Virtual PC от Connectix Corp совершенно не подходит для использования в серверных решениях. Virtual PC подходит почти для любого пользователя, нуждающегося в нескольких ОС, но для реализации серьезных сетевых инфраструктур необходим пакет с более удачной реализацией сетевого взаимодействия.

**VMware Workstation** работает во многом схоже с Virtual PC, но обладает значительно большей гибкостью и функциональностью. С ее помощью можно создавать гораздо более сложные конфигурации гостевых ПК, совместно с базовым ПК использующих периферийные устройства SCSI и USB, и строить более сложные сети, соединяющие гостевые ПК с реальными сетями настоящих машин или виртуальными сетями виртуальных машин. Гостевые ПК могут располагать несколькими сетевыми адаптерами и выполнять функции реальных Web- и FTP-серверов, доступных в Интернет и в корпоративной сети. В последней версии программы появилась новая функция *repeatable resume*, с помощью которой можно вновь и вновь возвращать гостевой ПК к одному образу "рабочего стола", к базовым ОС можно обращаться с удаленных машин через службы Windows Terminal Services и Windows XP Remote Desktop и управлять виртуальными машинами. VMware Workstation работает со всеми версиями ОС MS Windows, многими широко распространенными версиями ОС Linux и несколькими бета-версиями Microsoft .NET Server, но несовместима с OS/2 и не позволяет создавать гостевые ПК для развертывания ОС OS/2.

К несомненным достоинствам VMware Workstation следует отнести:

- Низкая цена пакета (цена при прямых поставках **299** долларов).
- VMware Workstation создает файлы-образы жестких дисков для каждого гостевого ПК, который хранится в обычных файлах, что делает виртуальные машины весьма портативными и позволяет за считанные минут переносить их с одного реального компьютера на другой.

- Легкость использования – VMware Workstation поставляется с хорошим графическим интерфейсом, что делает создание, конфигурирование, и управление виртуальными операционными системами предельно простым.
- Широкий диапазон поддерживаемых операционных систем: можно эмулировать и поддерживать разнообразные гостевые операционные системы подавляющего большинства известных производителей ОС.
- Широкие возможности управления сетью – возможность обеспечения трех способов управления сетью. Первый способ – "Bridged Networking" означает, что виртуальная машина будет непосредственно подключаться к корпоративной сети, используя реальную сетевую плату основного компьютера, фактически все виртуальные машины в сети будут видны и доступны как обычные физические компьютеры, подключенные к сети. Второй способ – "Host-Only Networking" – означает включение в виртуальную сеть, состоящую из базового компьютера с его ОС и всех виртуальных машин, запущенных из VMware. При этом корпоративная сеть будет доступна через базовый компьютер, который будет исполнять роль шлюза между виртуальной сетью и реальной сетью, к которой подключен базовый компьютер. Третий способ – "Bridged and Host-Only Networking", виртуальный компьютер будет иметь возможность использовать как реально существующее сетевое соединение базового компьютера, так и работать во внутренней виртуальной сети.
- Хорошая техническая поддержка производителя: как коммерческий продукт, VMware Workstation идет в комплекте с инструкциями, осуществляется его поддержка, к нему выпускаются регулярные обновления.

Недостатки VMware Workstation:

- Накладные расходы по ресурсам компьютера – VMware Workstation функционирует в базовой ОС, которая также нуждается в некотором объеме ресурсов. Однако, в качестве базовой ОС можно выбирать самые экономичные по ресурсам варианты такие как MS Windows NT Server 4.0

(требует для себя памяти не более 24 Мб и дискового пространстве менее 1 Гб, что на сегодняшний день является весьма низкими требованиями).

- Ограниченное число одновременно работающих гостевых ОС – в силу ограничений по ресурсам, используя VMware, возможен запуск относительно небольшого количества виртуальных компьютеров. Однако, это не недостаток самого программного продукта VMware, а требования, предъявляемые конкретными операционными системами и серверными приложениями, размещаемыми на виртуальных машинах.
- Закрытый источник – так как VMware является закрытым ресурсом, пользователи не могут вносить никаких собственных изменений в этот программный продукт.

**VMware GSX сервер.** VMware GSX Server – более мощная и функциональная версия VMware Workstation, специально предназначенная для создания виртуальных серверов. VMware GSX Server работает под управлением операционных систем Windows и Linux.

К основным ее достоинствам следует отнести:

- Широкий диапазон поддерживаемых операционных систем – VMware GSX Server поддерживает все версии Windows, начиная с Windows 95: Windows 95 OSR2, 98, Me, NT, 2000, XP и .NET server, многочисленные вариации Linux, BSD и Solaris. В плане организация и управления сетью – GSX Server идентична VMware Workstation.
- Отсутствие необходимости в GUI (graphical user interface – графического интерфейса пользователя) – GSX Server не нуждается в GUI для управления VMware, что позволяет экономить такие ресурсы как память и процессор за счет отказа от использования графического интерфейса.
- WEB-интерфейс – GSX может управляться через WEB-интерфейс. Гостевая операционная система может быть запущена, приостановлена, отключена или создана через WEB-интерфейс.

- Поддержка API (application programming interface – программный интерфейс приложения) языка программирования Perl для управления гостевыми операционными системами через WEB- интерфейс.

Недостатки VMware GSX Server:

- VMware GSX Server присущи некоторые недостатки VMware Workstation: ограниченное число одновременно работающих гостевых ОС в силу ограничений ресурсов компьютеров, GSX Server также является закрытым источником.
- Основной же недостаток – это высокая стоимость, она намного выше, чем в случае Workstation. Лицензия GSX стоит приблизительно **3500** долларов, что в случае экономии средств неприемлемо.

**VMware ESX сервер.** VMware ESX Server работает непосредственно с оборудованием сервера, не требуя какой-либо операционной системы, что позволяет снизить накладные расходы вычислительных ресурсов. Для загрузки и управления он имеет встроенную консольную операционную систему на базе Red Hat Linux 6.2.

Поддерживаются следующие "гостевые" операционные системы: Microsoft Windows 2000 Server, Advanced Server, Microsoft .NET Server, Microsoft Windows NT 4.0 Server, Red Hat Linux, FreeBSD.

На виртуальных серверах ESX Server могут одновременно выполняться несколько экземпляров широко распространенных приложений и служб:

- СУБД: Oracle, Microsoft SQL Server.
- Системы управления предприятием: SAP, Ariba.
- Удаленный доступ к приложениям: Citrix MetaFrame, MS Terminal Services.
- Службы WEB/FTP-публикации: IIS, Apache, ColdFusion, WebLogic.
- Потокное вещание: Windows Media Server, QuickTime, Real Video.
- Прокси-службы: Squid.
- Сетевые службы: Active Directory, DHCP, DNS, LDAP, NDS.
- Почтовые службы: Microsoft Exchange и другие службы POP и IMAP.



VMware ESX Server упрощает серверную инфраструктуру с помощью разбиения вычислительных ресурсов на изолированные логические разделы с возможностью удаленного управления. VMware ESX Server преобразует физические компьютеры в группу логически самостоятельных вычислительных ресурсов. Логические разделы разбиваются на некоторое множество защищенных виртуальных машин. Гостевые ОС изолированы друг от друга в своих виртуальных машинах, которые делят между собой одни и те же вычислительные ресурсы. Любой операционной системе или приложению при необходимости может быть предоставлено любое устройство.

Достоинствами VMware ESX Server являются:

- Высокопроизводительная технология виртуальных машин — поддерживает до восьми процессоров. На каждом процессоре может одновременно работать несколько виртуальных машин, хорошо изолированных друг от друга.
- Удаленное управление распределением ресурсов посредством WEB-интерфейса позволяет гарантировать качество обслуживания (QoS - Quality of Service). Можно динамически управлять ресурсами каждого процессора, а также использованием памяти в гостевой операционной системе и загрузкой каналов передачи данных.
- Расширенная поддержка устройств ввода-вывода позволяет использовать высокопроизводительные дисковые накопители с интерфейсами SCSI и Fiber Channel, а также дисковые массивы RAID. Режимы работы дисков "с отменой действий" и "без сохранения" позволяют восстановить исходное состояние системы. Поддерживаются виртуальные диски размером до 288 Гб. Поддержка устройств с интерфейсом SCSI охватывает ленточные накопители DAT и DLT SCSI, а также приводы SCSI CD-R/RW.
- Возможности для работы с сетью практически не ограничены, благодаря поддержке до четырех высокоскоростных виртуальных адаптеров Ethernet и Gigabit Ethernet на каждой виртуальной машине. Каждому

сетевому адаптеру виртуальной машины выделяется уникальный MAC-адрес. Допускается работа с любым протоколом, который поддерживается в операционной системе, установленной на виртуальной машине. Также можно создать несколько виртуальных сетей Ethernet между виртуальными машинами.

- Расширенные возможности управления включают в себя: встроенный контроль "пульса" системы, чтобы в реальном времени отслеживать интенсивность использования ресурсов, а также возможность останова виртуальной машины с сохранением ее состояния на диске и быстрого возобновления работы из этого состояния.
- Совместимость с Workstation и GSX Server. Виртуальные диски в формате SCSI из VMware Workstation и GSX Server можно перенести на ESX Server. В режиме совместимости ESX Server может без потери быстродействия работать с унаследованными дисками VMware Workstation.

К недостаткам VMware ESX Server следует отнести:

- VMware ESX Server присущи некоторые недостатки VMware Workstation: ограниченное число одновременно работающих гостевых ОС в силу ограничений ресурсов компьютеров, закрытый источник.
- Основной же недостаток – это высокая стоимость, она намного выше, чем в случае Workstation. Лицензия ESX стоит приблизительно **5400** долларов, что в случае экономии средств неприемлемо.

Таким образом, благодаря легкости интеграции в реальные сети, высокой производительности сетевого обмена, простоты настройки самым приемлемым по цене продуктом является VMware Workstation.

## **Приложение 4. Руководство пользователя к программе DTSOLVEX**

### **Аннотация**

Программное обеспечение DTSOLVEX реализует наиболее трудоемкую часть задачи реорганизации серверного парка с целью повышения эффективности использования ресурсов – решение задачи поиска оптимального распределения логических серверов по физическим компьютерам. Программа также обеспечивает ввод исходных данных через графический интерфейс либо загрузки из файла и вывод результатов через графический интерфейс либо сохранение в файл. Данный документ является руководством пользователей для ознакомления с интерфейсом и возможностями программы.

### **Условия применения**

Программа поставляется в том виде как есть, и автор не несет ответственности в случае, если использование или неправильное использование программы повлекло нарушение функционирования каких-либо компьютерных систем, либо искажение или потерю каких-либо данных, либо упущение какой-либо выгоды. Пользователи программы используют ее на свой страх и риск. Никаких гарантий к программе не прилагается и не предусматривается.

Программа поставляется для бесплатного использования, и не допускается создавать новые версии, сдавать в наем или аренду, продавать, изменять, декомпилировать, дизассемблировать, изучать код программы другими способами.

Программа поставляется без какой-либо технической поддержки. Также автор программы не несет никаких обязательств по дальнейшему развитию программы, консультированию по вопросам внедрения и эксплуатации программы, а также по вопросам особенностей технической реализации программы и возможностей расширения ее функциональности.

Данное руководство пользователя поставляется в том виде, как есть. Не допускается публикация данного руководства в Интернет, равно как и в любых других общедоступных источниках информации. Руководство поставляется для бесплатного использования и не допускается создавать новые версии, модифицировать, продавать, сдавать в наем или аренду.

Все права авторства принадлежат разработчику (автору) программы и данного руководства и охраняются согласно действующему законодательству.

Microsoft, Windows и Windows NT являются зарегистрированными торговыми марками фирмы Microsoft Corporation.

Intel, Pentium, Pentium Pro являются зарегистрированными торговыми марками фирмы Intel Corporation.

VMware, Workstation, GSX Server, ESX Server являются зарегистрированными торговыми марками компании VMware Inc.

Borland, Delphi, Borland Pascal являются зарегистрированными торговыми марками фирмы Borland Software Corporation.

## Системные требования

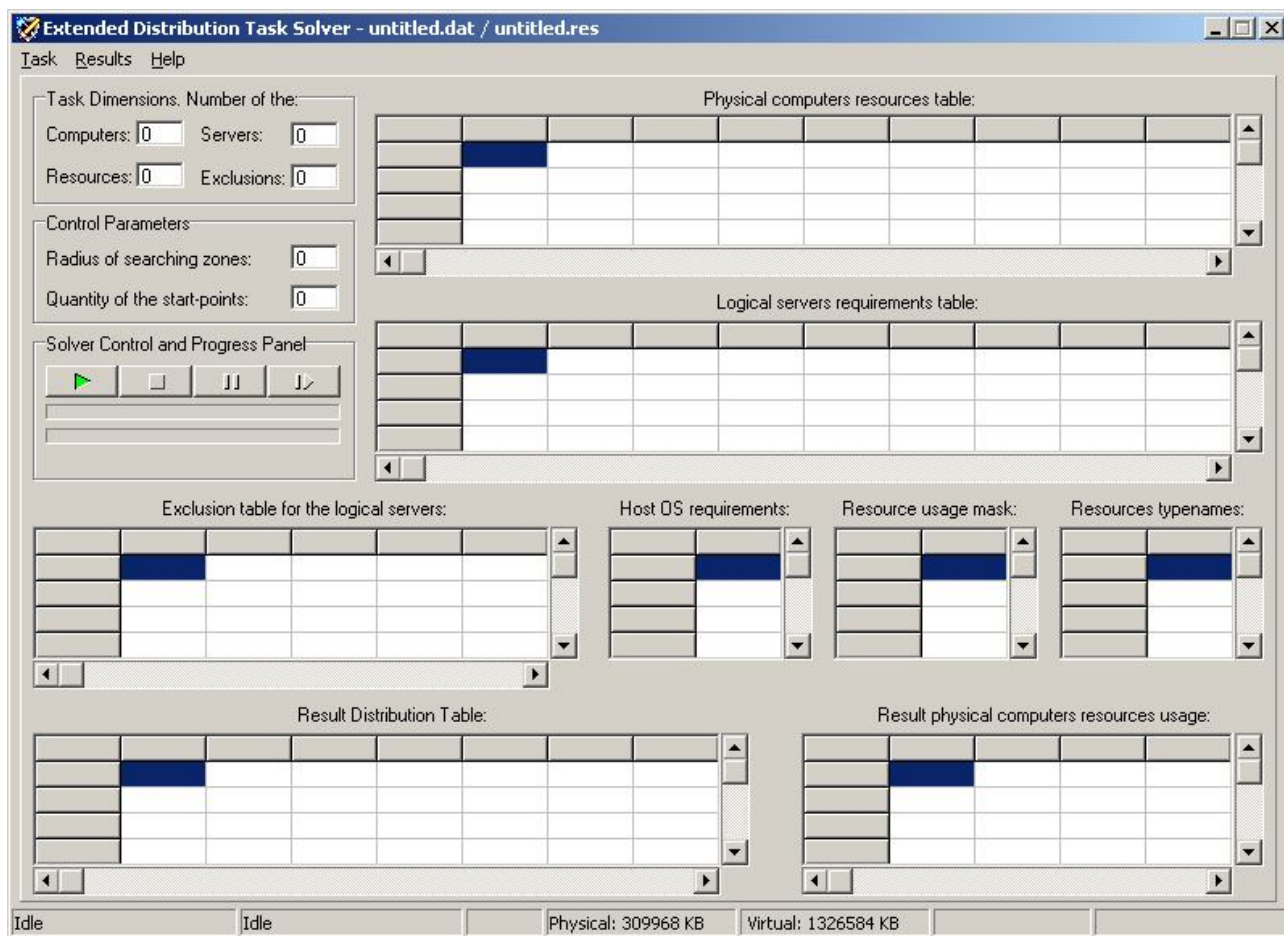
- Процессор: минимальные требования – Intel 386. Рекомендуется Intel Pentium III 1.0 GHz или выше.
- Оперативная память: минимальные требования – 2 МБ, при решении задач с максимальными размерностями требуется до 48 Мб свободной оперативной памяти.
- Дисковое пространство: 615 Кбайт. В случае недостаточной емкости оперативной памяти при решении задач с максимальными размерностями может потребоваться до 48 Мб свободного места на диске.
- Операционная система: Microsoft Windows 98/NT 4.0 или более новые версии ОС Microsoft Windows.

## Установка и настройка

Программа поставляется в виде одного единственного исполняемого файла DTSOLVEX.EXE и не требует никаких специальных действий по установке или настройке. Для начала работы с программой достаточно запустить исполняемый файл.

## Описание интерфейса программы

При запуске программы появляется окно следующего вида:



Все необходимые поля для ввода исходных данных задачи и вывода результирующих данных находятся в этом окне. Также в этом же окне находятся панель для управления потоком решения задачи, меню и строка состояния программы. На данный момент программа разработана только в англоязычном варианте.

Кратко поясним поля данных, панель управления, меню и строку состояния:

- *Task Dimensions* – группа данных, определяющих размерности задачи:
  - *Number of the computers* – число физических компьютеров
  - *Number of the servers* – число логических серверов
  - *Number of the resources* – число типов ресурсов
  - *Number of the exclusions* – число ограничений-исключений

В полях размерностей допускаются лишь целые неотрицательные числа от 0 до 999. Верхнее ограничение на размерности было искусственно введено исключительно для ограничения использования оперативной памяти (не более 100 Мб), доступной в современных ПК. Это ограничение жестко зашито программе в виде константы, реально же ограничение для размерностей можно расширить вплоть до  $2^{15} - 1$ . Для возможности решения задачи необходимо задать хотя бы один компьютер, один логический сервер и один тип ресурса.

- *Control Parameters* – группа параметров, управляющие объемом перебора для подзадач булевой оптимизации, которые решаются при решении задачи поиска оптимального распределения:





- *Radius of the searching zones* – радиус зон поиска
- *Quantity of the start-points* – число случайных стартовых точек поиска

Для этих двух полей допускаются лишь целые неотрицательные числа от 0 до <числа логических серверов>. Верхнее ограничение обусловлено тем, что радиус зон поиска больший, чем число переменных в подзадаче локального поиска – не имеет смысла. Для возможности решения задачи для обоих параметров необходимо задать значение  $\geq 1$ .

- *Physical computers resources table* – таблица базовых уровней ресурсов физических компьютеров серверного парка. Типы ресурсов располагаются по строкам, физические компьютеры – по столбцам. В таблице допускаются только целые неотрицательные числа в диапазоне от 0 до  $2^{31} - 1$ . Элементы таблицы отражают некоторую количественную оценку базового уровня ресурса заданного типа у заданного компьютера.

- *Logical servers requirements table* – таблица требований логических серверов. Типы ресурсов располагаются по строкам, логические серверы – по столбцам. В таблице допускаются только целые неотрицательные числа в диапазоне от 0 до  $2^{31} - 1$ . Элементы таблицы отражают некоторую количественную оценку требуемого уровня ресурса заданного типа для заданного сервера.
- *Exclusion table for the logical servers* – таблица исключений для предотвращения одновременного размещения каких-либо логических серверов на каком-либо физическом компьютере. Исключения располагаются по строкам, логические серверы – по столбцам. В таблице допускаются только значения 0 или 1. Элементы таблицы отражают то, что присутствует ли заданный логический сервер в заданном ограничении-исключении: 1 – присутствует, 0 – нет. Если число ограничений-исключений равно нулю, то данная таблица не заполняется.
- *Host OS requirements* – столбец требований базовой ОС. Типы ресурсов располагаются по строкам. В столбце допускаются только целые неотрицательные числа в диапазоне от 0 до  $2^{31} - 1$ . Элементы столбца отражают некоторую количественную оценку требуемого уровня ресурса заданного типа для базовой операционной системы.
- *Resource Usage Mask* – вектор маски для критериев оптимизации. Типы ресурсов располагаются по строкам. В столбце допускаются только значения 0 или 1. Элементы столбца отражают то, выполняется ли оптимизация по заданному типу ресурса: 1 – выполняется, 0 – нет. Для решения задачи, в этом столбце хотя бы один элемент должен быть “1”, иначе если все элементы нулевые, то попросту отсутствуют цели для оптимизации, и задачу нет смысла решать.
- *Resources Type Names* – имена типов ресурсов. Это необязательное для заполнения поле, содержит текстовые имена для типов ресурсов, при решении задачи они не используются. Имена могут содержать не более 32 символов, при превышении имени урезаются до 32 символов.

Примечание. Для всех полей исходных данных используется контроль корректности вводимых данных. Для полей размерности и управления объемом перебора, при вводе некорректных данных нельзя перейти в другое поле, при этом для них выводится подсказка. Таблицы и столбцы исходных данных не позволяют вводить исходные данные для элементов, чьи индексы выходят за пределы заданных размерностей. Полосы прокруток становятся доступными только, когда соответствующие размерности таблиц или столбцов исходных данных превышают размерности, видимые в окне программы. Элементы таблиц и столбцов исходных данных, частично контролируется при непосредственном вводе данных, полностью же эти данные проверяются при попытке сохранения задачи в файл либо запуска потока решения задачи.

- *Solver control and progress panel* – панель управления потоком решения:
  - Кнопка  (*Run*) – запуск потока решения. Если все исходные данные корректны, то начинается решение задачи. После запуска потока решения задачи кнопка становится недоступной до окончания решения задачи либо полной принудительной остановки.
  - Кнопка  (*Stop*) – полная остановка потока решения задачи. Решение задачи полностью прерывается и после этого его можно запустить только заново (“с нуля”). Изначально кнопка недоступна, становится доступной после запуска потока решения задачи.
  - Кнопка  (*Pause*) – приостановка потока решения задачи. Решение задачи временно приостанавливается, поток решения замораживается. Изначально кнопка недоступна, становится доступной после запуска потока решения задачи. При нажатии становится недоступной до тех пор, пока не будет нажата кнопка возобновления потока решения.
  - Кнопка  (*Resume*) – возобновление ранее приостановленного потока решения задачи. Задача решается с того места, на котором была приостановлена, а не “с нуля”. Изначально кнопка недоступна, становится доступной только после того, как будет запущен и приостановлен поток решения задачи.



Примечание. Использование приостановки/возобновления может оказаться очень полезным в случаях, когда для выполнения каких-либо других задач требуется временное снятие высокой нагрузки с процессора, который используется для решения задачи.

- Уровень (полоса) прогресса 1 – отношение последовательного номера рассматриваемого физического компьютера к числу оставшихся физических компьютеров на “большом шаге” решения задачи. На каждом “большом шаге” среди оставшихся компьютеров выбирается наилучший для распределения на него логических серверов. Отражает прогресс решения задачи внутри заданного “большого шага”.
- Уровень (полоса) прогресса 2 – Максимум из двух величин: отношения распределенных логических серверов к общему числу серверов и отношения задействованных физических компьютеров к общему числу компьютеров. Фактически отражает общий прогресс решения задачи распределения.
- *Result distribution table* – результирующая таблица распределения логических по физическим компьютерам. Физические компьютеры располагаются по строкам (кроме самой верхней, заголовочной строки). Заголовочная строка содержит список логических серверов, которых программа не смогла никуда распределить. Остальные строки содержат списки логических серверов, распределенных на заданный компьютер. Если в какой-то строке список пуст, то компьютер остался незадействованным. Таблица доступна только для чтения данных.
- *Result physical computers resources usage* – таблица показателей загрузки ресурсов физических компьютеров для полученного оптимального распределения. Физические компьютеры располагаются по строкам, типы ресурсов по столбцам. Элементы этой таблицы отражают уровень загрузки заданного ресурса заданного компьютера для полученного оптимального распределения. Таблица доступна только для чтения.

- Меню программы. Главное меню программы состоит из трех подменю:
  - *Task* – подменю задачи. В нем находятся элементы меню:
    - *New* – создание новой задачи.
    - *Load* – загрузка исходных данных задачи из файла.
    - *Save* – сохранение исходных данных в текущий файл задачи.
    - *Save As* – сохранение исходных данных в файл с выбором имени.
    - *Exit* – завершение работы с программой.
  - *Results* – подменю результатов. В нем находятся элементы меню:
    - *Save* – сохранение результатов в текущий файл результатов.
    - *Save As* – сохранение результатов в файл с выбором имени.
  - *Help* – подменю справки. В нем находятся элементы меню:
    - *About* – информация об авторе.
- Строка состояния программы находится в самой нижней части окна приложений и использует всю ширину окна. Строка состояния используется для вывода различной оперативной информации:
  - Текущее состояние или операция, выполняемая главным потоком программы (файловые операции, взаимодействие с интерфейсом).
  - Текущее состояние или операция, выполняемая потоком решения задачи (взаимодействие с интерфейсом, решение задачи).
  - Приznak паузы: поток решения задачи приостановлен.
  - Доступная физическая память компьютера, на котором решается задача поиска оптимального распределения.
  - Доступная виртуальная память компьютера, на котором решается задача поиска оптимального распределения.
  - Приznak модификации исходных данных задачи.
  - Приznak обновления результатов решения задачи.

## Описание файла исходных данных и файла результатов

Файлы с исходными данными и файлы с результатами представляют собой обычные текстовые файлы, соответственно, их можно просматривать и редактировать в любом текстовом редакторе. При запуске программы используется файл по умолчанию “untitled.dat”. Для файлов с исходными данными используется расширение DAT. Для файлов с результатами – RES.

Формат хранения данных в текстовом файле с исходными данными (исходные данные располагаются по полям с применением форматирования):

[NH]

<Число физических компьютеров>

[NS]

<Число логических серверов>

[NC]

<Число типов ресурсов>

[NX]

<Число исключений>

[MAXRADIUS]

<Радиус зон поиска>

[MAXSTARTS]

<Число стартовых точек>

[HST]

◇...◇      Элементы матрицы базовых уровней ресурсов физических компьютеров в табличной форме

...

◇...◇

[SST]

◇...◇      Элементы матрицы требований логических серверов в табличной форме

...

◇...◇

[EXT]

◇...◇      Элементы матрицы исключений в табличной форме  
Если число исключений = 0, то эта секция должна отсутствовать

...

◇...◇

[BST]

◇      Элементы столбца требований базовой ОС

...

◇

[CM]

◇      Элементы столбца маски критериев оптимизации

...

◇

[RTN]

◇      Элементы столбца имен типов ресурсов

...

◇

Поскольку помимо ввода данных через графический интерфейс, допускается также ввод данных непосредственно в файл с последующей его загрузкой в программу, то отметим некоторые особенности. Несмотря на строгий формат, приведенный выше, допустимо любое количество пробелов, табуляций между элементами в строках таблиц, любое число пустых строк между строками данных, любое количество пустых строк, пробелов и табуляций перед заголовком секции, весь этот "мусор" автоматически игнорируется и при считывании идет поиск только полезных данных. Важно только лишь то, чтобы заголовки полей данных назывались правильно, а сами данные были корректными. Единственное ограничение: не допускается перестановка секций с данными, их порядок должен соблюдаться. В любом случае программа обрабатывает любую ситуацию, пользователь может вводить какие угодно данные и пытаться считывать любые файлы любых форматов и размеров, программа всегда анализирует содержимое файла.

Пример файла с исходными данными:

```
task_g.dat - Notepad
File Edit Format Help
[NH]
4
[NS]
10
[NC]
2
[NX]
0
[MAXRADIUS]
3
[MAXSTARTS]
3
[HST]
512    1024    768    768
20000  20000  10000  10000
[SST]
192    256    128    128    256    384    256    512    128    96
2000   3000   5000   2000   2000   8000   1000   10000  4000   8000
[BST]
128
2000
[CM]
1
1
[RTN]
Memory
Disk
```

Файл результатов, также не представляет собою ничего особенно и содержит результаты решения задачи в нескольких полях. Формат хранения данных в текстовом файле результатов:

Distribution Table:

H1: <...> Элементы матрицы распределения логических серверов по компьютерам в табличной форме. Данные хранятся в компактной форме в виде расположенных по строкам списков логических серверов, распределенных на физический компьютер.  
 ...  
 H<sub>NH</sub>: <...>

Remaining undistributed logical servers:

<...> Строка с куда нераспределенными логическими серверами.  
 Если таковых нет, то в строку записывается None.

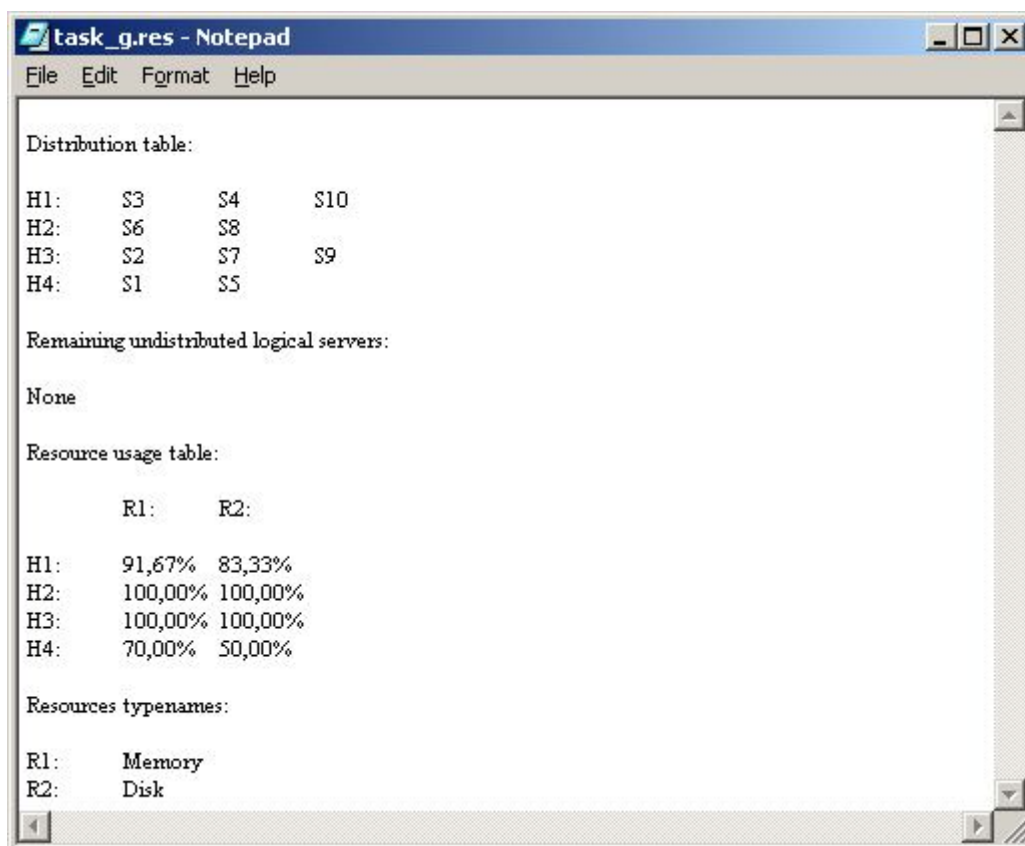
Resource usage table:

R1: R<sub>NC</sub>:  
 H1: <...> Элементы матрицы загрузки ресурсов физических компьютеров в табличной форме.  
 ...  
 H<sub>NH</sub>: <...>

Resource type names:

R1: < Элементы столбца текстовых имен типов ресурсов.  
 ...  
 R<sub>NC</sub>: <

Пример файла результатов:



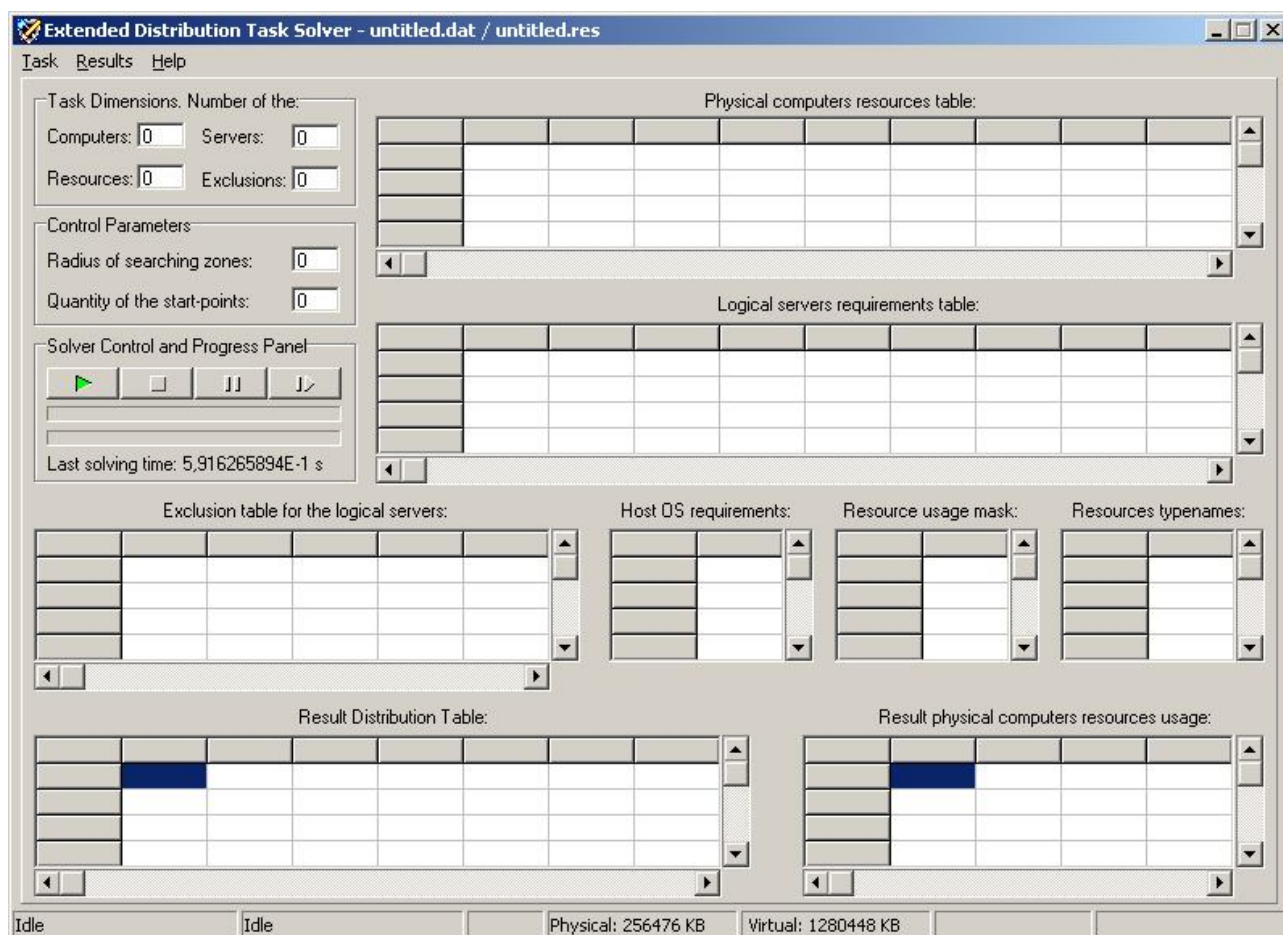
## Работа с программой

Для начала работы с программой необходимо запустить файл DTSOLVEX.EXE на исполнение.

Рассмотрим типовые операции: создание новой задачи, ввод данных, загрузка исходных данных из файла, сохранение данных в файл, запуск потока решения задачи и управление им, сохранение результатов в файл и завершение работы с программой.

### Создание новой задачи

Для создания новой задачи в программе необходимо перейти в меню *Task* – *New*. При запуске программы также автоматически создается новая задача.



### Ввод данных в окне программе

В первую очередь следует ввести размерности задачи в группе параметров “Common Parameters”, поскольку таблицы и столбцы данных динамически изменяют свои размерности в зависимости от этих параметров.

Task Dimensions. Number of the:

Computers:  Servers:

Resources:  Exclusions:

Как минимум следует задавать ненулевые значения для числа физических компьютеров, числа логических серверов и числа типов ресурсов.

Число исключений может быть ненулевым или нулевым, соответственно, при нулевом числе исключений таблица исключений недоступна для ввода данных.

Число строк и столбцов, доступных для ввода данных в таблицах и столбцах исходных данных, определяется параметрами размерностей. Соответственно, для доступных столбцов и строк, в заголовочных строках (самая верхняя строка с серым цветом фона) и столбцах (крайний левый столбец с серым цветом фона) высвечиваются индексы. После задания параметров размерностей необходимо ввести корректные данные в таблицы и столбцы исходных данных:

Physical computers resources table:

	H1	H2	H3	H4	H5	H6	H7	H8	H9
R1	512	512	512	512	512	512	512	512	512
R2	10000	10000	10000	10000	10000	10000	10000	10000	10000
R3	100	100	100	100	100	100	100	100	100
R4	100	100	100	100	100	100	100	100	100

Logical servers requirements table:

	S1	S2	S3	S4	S5	S6	S7	S8	S9
R1	96	96	96	96	96	96	96	96	96
R2	2000	2000	2000	2000	2000	2000	2000	2000	2000
R3	20	20	20	20	20	20	20	20	20
R4	20	20	20	20	20	20	20	20	20

Exclusion table for the logical servers:

	S1	S2	S3	S4	S5
E1	1	1	1	1	1
E2	0	0	0	0	0
E3	0	0	0	0	0
E4	0	0	0	0	0

При вводе данных в таблицах и столбцах для перемещения между элементами можно использовать клавиши ←, ↑, ↓, →, Page Up и

Page Down, и горизонтальные и вертикальные полосы прокруток.

Host OS requirements:		Resource usage mask:		Resources typenames:	
R1	128	R1	1	R1	Memory
R2	2000	R2	1	R2	Disk
R3	20	R3	1	R3	Network
R4	20	R4	1	R4	Processor

Ввод имен типов ресурсов необязательно, хотя это может быть полезным для придания

безликим индексам типов ресурсов смысловых значений.

Control Parameters	
Radius of searching zones:	3
Quantity of the start-points:	1

Наконец, в группе параметров “Boolean Optimization Parameters”, управляющих объемом перебора для подзадач булевой оптимизации, необходимо задать

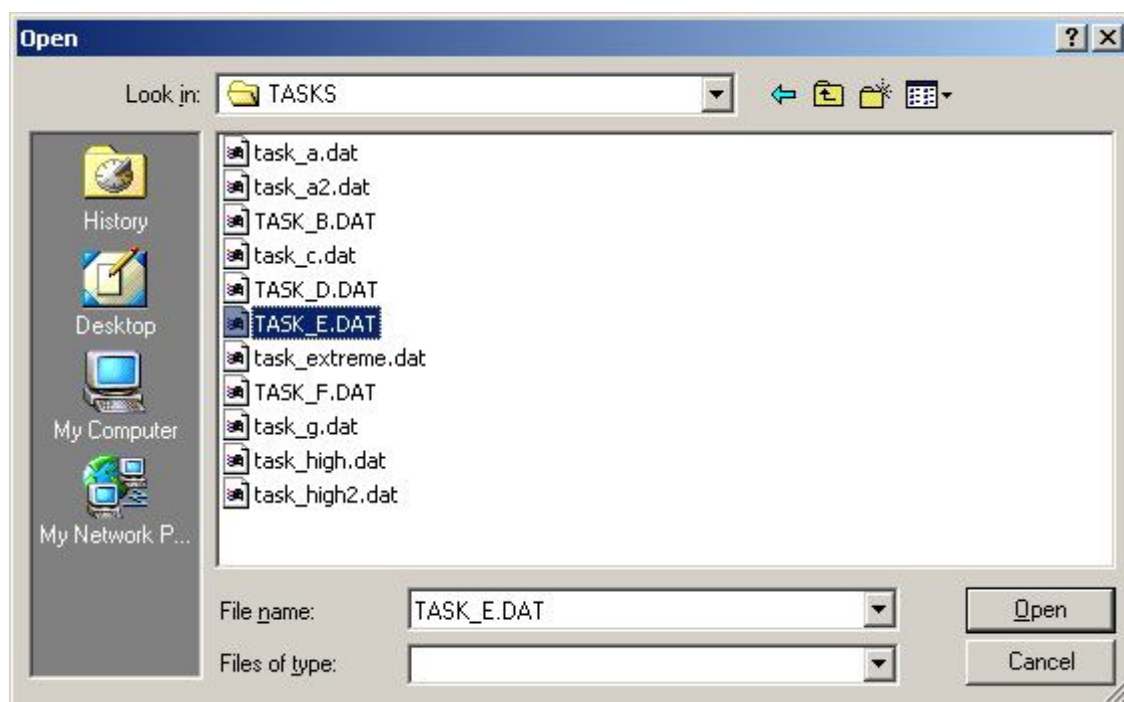
радиус зон поиска и число стартовых точек, используемых при решении подзадач условной псевдобулевой оптимизации во время решения задачи поиска распределения. От этих параметров напрямую зависит не только качество получаемого решения, но и время решения, которое в худшем случае прямо пропорционально возрастает с увеличением числа стартовых точек и экспоненциально возрастает с увеличением радиуса зон поиска. Поэтому при решении задач, необходимо начинать со значения радиуса равного единице и с одной стартовой точки. Далее, для повышения качества получаемого решения можно увеличивать значения этих параметров. В программе также есть ограничение – параметры радиуса зон поиска и числа стартовых точек не должны превышать число логических серверов.

После завершения ввода данных можно переходить непосредственно к запуску потока решения задачи. Также можно сохранить исходные задачи в файл. Если исходные данные некорректные, то при попытке запуска потока решения или сохранения в файл программа выводит сообщения об ошибке, с указанием на то, где именно допущена ошибка, и как ее исправить.



### Загрузка исходных данных из файла

Для загрузки исходных данных из файла в программе необходимо перейти в меню *Task – Load*, после чего появляется окно для выбора файла:

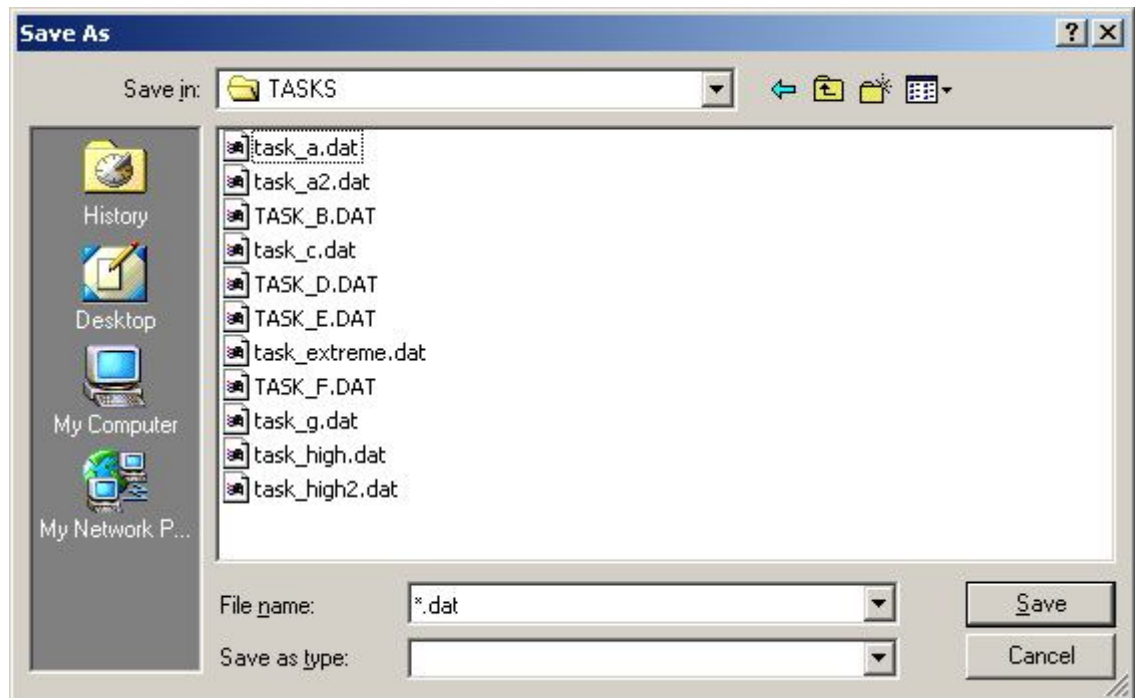


В этом окне необходимо выбрать файл с корректными исходными данными и нажать кнопку *Open* (чтобы отказаться от загрузки задачи необходимо нажать кнопку *Cancel*).

Если данные в файле корректны, то задача загружается, и ее данные появляются в окне программы, в противном случае выводится сообщением об ошибке с указанием места ошибки и ее типа. Если в процессе загрузки данных возникают какие-либо ошибки файловых операций, то программа также корректно обрабатывает ситуацию и выводит соответствующие сообщения об ошибках. Также если для выполнения операции недостаточно оперативной памяти компьютера программа выводит ошибку о нехватке памяти.

### Сохранение исходных данных в файл


Исходные данные задачи можно сохранить в файл двумя способами: в текущий файл (по умолчанию, при запуске программы – это файл UNTITLED.DAT), для этого необходимо перейти в пункт меню *Task – Save*, либо в указанный пользователем файл, для этого необходимо перейти в пункт меню *Task – Save As*. В случае сохранения *Save As* появляется окно выбора файла для исходных данных задачи:

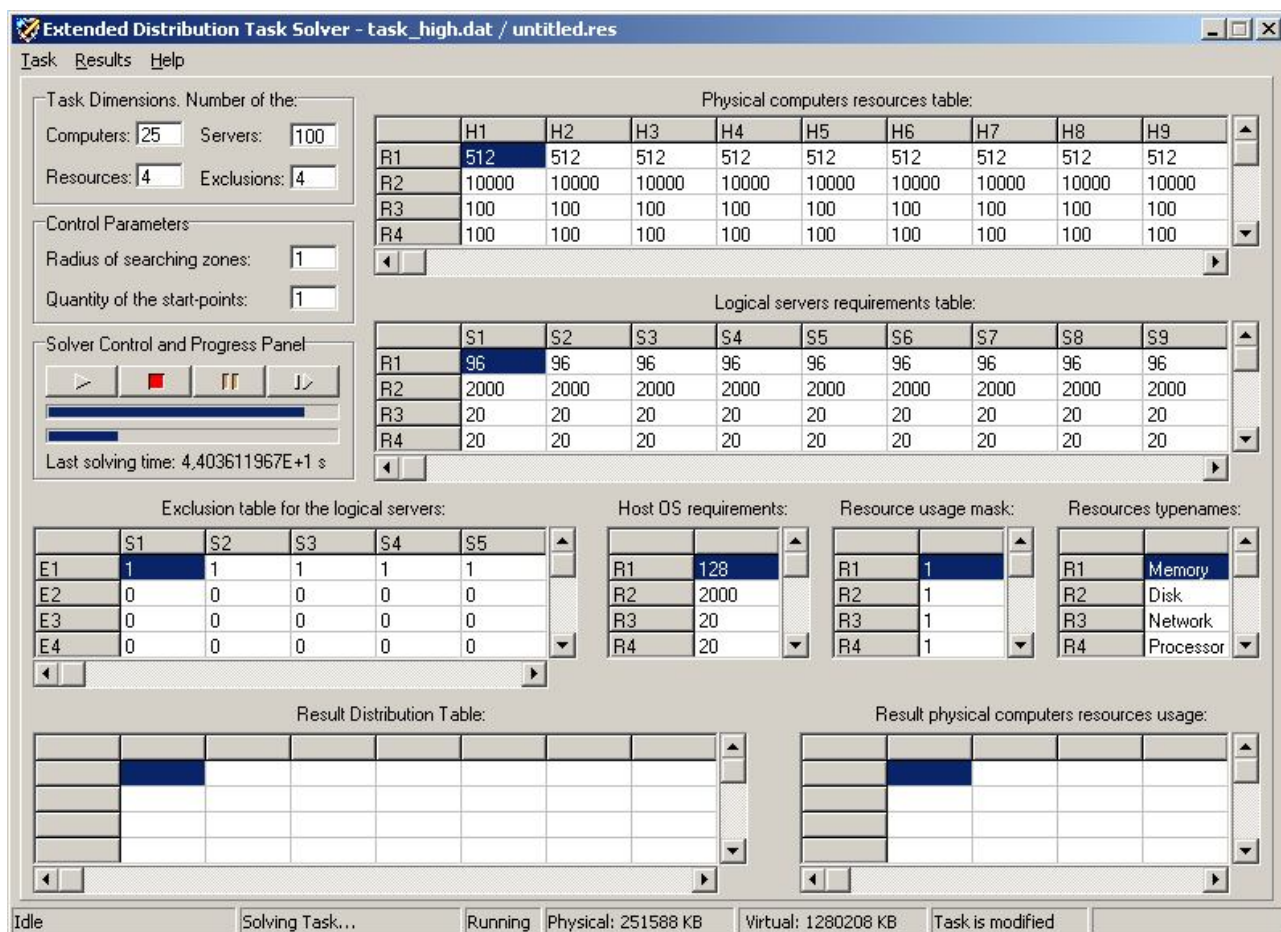




В этом окне необходимо задать имя файла и нажать кнопку *Save* (чтобы отказаться от сохранения данных необходимо нажать кнопку *Cancel*).

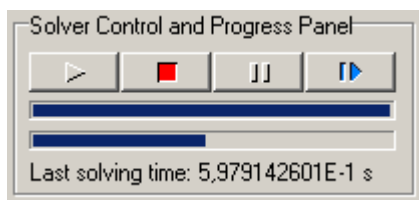
Если исходные данные в окне программы корректны, то выполняется сохранение в файл, в противном случае выводится сообщением об ошибке с указанием места ошибки и ее типа. Если в процессе сохранения данных возникают какие-либо ошибки файловых операций, то программа также корректно обрабатывает ситуацию и выводит соответствующие сообщения об ошибках. Также если для выполнения операции недостаточно оперативной памяти компьютера программа выводит ошибку о нехватке памяти.


## Запуск потока решения и управление им

Запуск потока решения осуществляется кнопкой запуска . Если данные задачи в окне программы корректны, то программа начинает ее решение:



При этом становятся доступными кнопки полной остановки потока решения  и паузы  (временной приостановки). При полной остановке решения задачи прерывается и потом ее можно начать решать только “с нуля”.



При нажатии же кнопки паузы, решение задачи на время приостанавливается. При этом становится доступной кнопка  возобновления решения ранее приостановленной задачи. Приостановленная задача при возобновлении решения уже решается не “с нуля”, а с того места, где она была приостановлена. Приостановленное решение также можно принудительно завершить кнопкой полной остановки.

По уровням прогресса 1 и 2, находящихся под кнопками управления можно судить о том, сколько логических и физических серверов обработано. Однако, следует учитывать то, что показания этих уровней с течением времени изменяются не линейно, а с нарастающей скоростью, поскольку при решении задачи динамически уменьшается число обрабатываемых физических компьютеров и логических серверов, соответственно уменьшается число и размерность подзадач, что, в свою очередь, сказывается уменьшением времени их решения. Время решения, в худшем случае прямо пропорционально возрастает с увеличением числа стартовых точек и экспоненциально возрастает с увеличением радиуса зоны поиска.

В случае если программа успешно решает задачу до конца, то результаты выводятся в две результирующие матрицы в нижней части окна программы

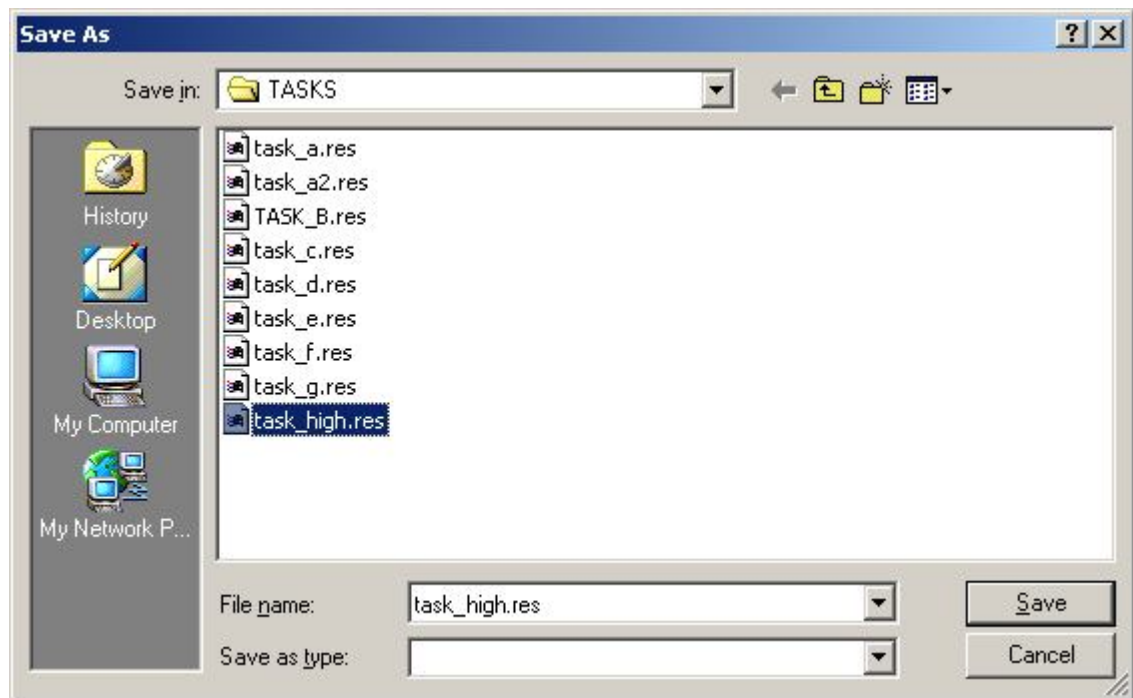
Result Distribution Table:							
Remain S:							
H1	S1	S6	S11	S16			
H2	S2	S7	S12	S17			
H3	S3	S8	S13	S18			
H4	S4	S9	S14	S19			

В таблице распределения построчно для каждого физического компьютера выводится список распределенных на него логических серверов, если для какого-либо компьютера этот список пуст (незадействованный компьютер) – то в таблице она выглядит как пустая строка. Самая верхняя строка (заголовочная строка таблицы серого цвета) служит для отображения тех логических серверов, которые не были распределены ни на один из компьютеров. В таблице загрузки ресурсов отображается загрузка ресурсов компьютеров (в процентах).

Result physical computers resources usage:					
Resource:	Memory	Disk	Network	Processor	
H1	100,00%	100,00%	100,00%	100,00%	
H2	100,00%	100,00%	100,00%	100,00%	
H3	100,00%	100,00%	100,00%	100,00%	
H4	100,00%	100,00%	100,00%	100,00%	

### Сохранение результатов решения задачи

Для сохранения результатов решения задачи (таблицы распределения логических серверов по физическим компьютерам и таблицы загрузки ресурсов компьютеров) необходимо перейти к пункту меню *Results – Save As*. После этого на экране появится окно выбора файла для сохранения результатов:



В этом окне необходимо задать имя файла и нажать кнопку *Save* (чтобы отказаться от сохранения данных необходимо нажать кнопку *Cancel*).

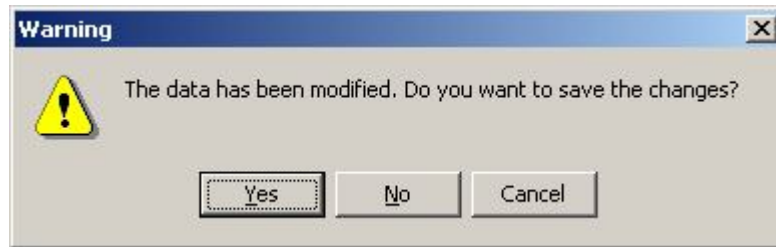
В случае если таблицы результатов не пустые, то результаты сохраняются файл. Если в процессе сохранения данных возникают какие-либо ошибки файловых операций, то программа также корректно обрабатывает ситуацию и выводит соответствующие сообщения об ошибках. Также если для выполнения операции недостаточно оперативной памяти компьютера программа выводит ошибку о нехватке памяти.

### Завершение работы с программой

Для завершения работы с программой необходимо перейти в пункт меню *Task – Exit* либо нажать комбинацию клавиш **Alt + F4**, либо нажать кнопку **✕** в правом верхнем углу экрана. Перед завершением работы с программой поток решения задачи должен быть полностью остановлен.

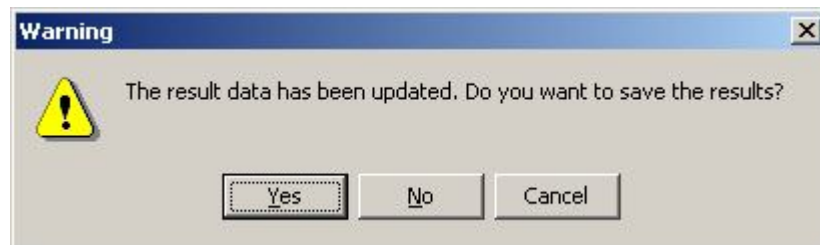
### Примечание

Если при создании новой задачи, загрузки задачи из файла либо завершении работы с программой в программе присутствовали какие-либо данные, которые не были сохранены, либо была загружена задача, данные которой модифицировались после загрузки, и эти изменения не были сохранены, то программа всегда предлагает сохранить данные в файл, поскольку предыдущие данные теряются.



Пользователь может сохранить предыдущие данные, отказаться от их сохранения, либо отказаться от проведения операции (создания новой задачи, загрузки задачи из файла либо завершения работы с программой).

Кроме того, если при создании новой задачи, загрузки задачи из файла либо завершении работы с программой были получены какие-либо результаты, которые не были сохранены, то программа также предложит сохранить их:



Пользователь может сохранить результаты решения задачи, отказаться от их сохранения, либо отказаться от проведения операции (создания новой задачи, загрузки задачи из файла либо завершения работы с программой).

Кроме того, если при завершении работы с программой, решается какая-либо задача, программа требует остановки потока решения задачи:



## Приложение 5. Тексты исходного кода программы DTSOLVEX

### Исходный код основной программы

```
program DTSOLVEX; {Win32 приложение}

{Используемые модули}
uses
  Forms,
  MAINUNIT in 'MAINUNIT.pas' {Form1},
  MYABOUT in 'MYABOUT.pas' {AboutBox};

{$R *.res}

begin
  appinit_error:= 0;          {Обнуление флага ошибки этапа создания формы}
  {Инициализация приложения, Создание основной формы окна и окна About}
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TAboutBox, AboutBox);
  {В случае отсутствия ошибок}
  if (appinit_error = 0) then Application.Run;          {Запуск приложения}
end.
```



## Исходный код главного модуля: графический интерфейс и потоки

{Главный модуль программы: основной поток процесса, поток решения задачи и поток мониторинга состояния потока решения, интерфейса и системы}  
unit MAINUNIT;

{Оператор @ всегда генерирует типизированные указатели}  
{Force operator "@" to generate typed pointers}  
{T+}

{Предисловие. В приложении помимо основного потока процесса, обрабатывающего по большей части интерфейс пользователя, присутствуют два потока (см. ниже): поток решения задачи (он запускается только на время решения задачи) и поток мониторинга изменения исходных данных, обновления результатов, доступной памяти и состояния потока решения задачи. Поток мониторинга запускается вместе с запуском приложения и завершается при завершении работы приложения. Основной поток процесса, поток решения задачи и поток мониторинга существуют максимально независимо.}

interface

uses

{Используемые модули}  
MYCUSTOM, MYSOLVER, MYABOUT,  
Windows, Forms, SysUtils, Classes, Messages,  
Menus, StdCtrls, Grids, ExtCtrls, Buttons,  
Graphics, ComCtrls, Controls, Dialogs;

const

{Константы для выбора строк или столбцов при установке размеров в матрицах}  
AXIS\_X = 1;               {Столбцы}  
AXIS\_Y = 2;               {Строки}

{Текстовая константа заголовка окна приложения}

APP\_TITLE = 'Extended Distribution Task Solver';

{Текстовые константы сообщений об ошибках}

GEN\_ERR = 'General error:' + chr (13) + chr (10);

PARSE\_ERR = 'Parsing error: ';

ACC\_VIOL\_ERR = 'Access violation.' + chr(13) + chr(10);

MEM\_ALLOC\_ERR = 'Out of memory.' + chr(13) + chr(10);

WINMSG\_REG\_ERR = GEN\_ERR + 'Cannot register custom Windows Message';

STATUS\_THREAD\_CREATE\_ERR = GEN\_ERR + 'Cannot create status thread.';

SOLVER\_THREAD\_CREATE\_ERR = GEN\_ERR + 'Cannot create solver thread.';

EDIT\_INVDATA\_ERR = PARSE\_ERR + 'Invalid data in the edit field.';

EDIT\_RANGE\_ERR = PARSE\_ERR + 'Value must be in range: ';

CUSTTASK\_CREATE\_ERR = MEM\_ALLOC\_ERR + 'Cannot perform this operation.';

GRID\_RESIZE\_ERR = MEM\_ALLOC\_ERR +  
'Cannot resize data array.' + chr(13) + chr(10);

STRGRID\_UPDATE\_ERR = ACC\_VIOL\_ERR + 'Cannot update string tables.';

STRGRID\_EDIT\_ERR = ACC\_VIOL\_ERR + 'Cannot edit string table.';



```

SRC_MODIFY_WARN =
'The data has been modified. Do you want to save the changes?';

RES_MODIFY_WARN =
'The result data has been updated. Do you want to save the results?';

SOLVER_RUNNING_ERR =
'You should stop the solver before closing the program!';

SOLVER_STOP_WARN =
'Do you really want to break the solving operation?';

```

```
type
```

```

TForm1 = class(TForm)                                {Класс основной формы окна}

    AppOpenDialog: TOpenDialog;                       {Диалог открытия файла}
    AppSaveDialog: TSaveDialog;                       {Диалог сохранения файла}
    AppStatusBar: TStatusBar;                          {Строка состояния приложения}

    AppMainMenu: TMainMenu;                           {Основное меню приложения}
    SubMenuTask: TMenuItem;                           {Подменю Task}
    ItemNew: TMenuItem;                               {Элемент New в подменю Task}
    ItemLoad: TMenuItem;                             {Элемент Load в подменю Task}
    ItemSave: TMenuItem;                             {Элемент Save в подменю Task}
    ItemSaveAs: TMenuItem;                           {Элемент Save As в подменю Task}
    ItemSeparator: TMenuItem;                        {Элемент разделитель}
    ItemExit: TMenuItem;                             {Элемент Exit в подменю Task}
    SubMenuResults: TMenuItem;                       {Подменю Results}
    ItemResSave: TMenuItem;                          {Элемент Save в подменю Results}
    ItemResSaveAs: TMenuItem;                       {Элемент Save As в подменю Results}
    SubMenuHelp: TMenuItem;                         {Подменю Help}
    ItemAbout: TMenuItem;                           {Элемент About в подменю Help}

    Panel1: TPanel;                                  {Основная панель формы окна}

    GroupBox1: TGroupBox; {Группа элементов основных параметров задачи}
    Label1: TLabel;      {Заголовок поля числа физических компьютеров}
    Edit1: TEdit;         {Поле редактирования числа физических компьютеров}
    Label2: TLabel;      {Заголовок поля числа логических серверов}
    Edit2: TEdit;         {Поле редактирования числа логических серверов}
    Label3: TLabel;      {Заголовок поля числа типов ресурсов}
    Edit3: TEdit;         {Поле редактирования числа типов ресурсов}
    Label4: TLabel;      {Заголовок поля числа исключений}
    Edit4: TEdit;         {Поле редактирования числа исключений}

    GroupBox2: TGroupBox; {Группа элементов параметров булевой подзадачи}
    Label5: TLabel;      {Заголовок поля максимального радиуса поиска}
    Edit5: TEdit;         {Поле редактирования максимального радиуса поиска}
    Label6: TLabel;      {Заголовок поля числа стартовых точек поиска}
    Edit6: TEdit;         {Поле редактирования числа стартовых точек поиска}

    GroupBox3: TGroupBox; {Группа элементов управления потоком решения задачи}
    ButtonRun: TBitBtn;   {Кнопка запуска потока решения задачи}
    ButtonStop: TBitBtn;  {Кнопка завершения потока решения задачи}
    ButtonPause: TBitBtn; {Кнопка замораживания потока решения задачи}
    ButtonResume: TBitBtn; {Кнопка пробуждения потока решения задачи}
    ProgressBar1: TProgressBar; {Уровень прогресса 1 решения задачи}
    ProgressBar2: TProgressBar; {Уровень прогресса 2 решения задачи}
    Label15: TLabel;     {Поле вывода времени решения последней задачи}

    Label7: TLabel;      {Заголовок таблицы ресурсов физических компьютеров}
    StringGrid1: TStringGrid; {Таблица ресурсов физических компьютеров}
    ScrollBar1: TScrollBar; {Полоса прокрутки по горизонтали}

```

```

ScrollBar2: TScrollBar; {Полоса прокрутки по вертикали}

Label8: TLabel;          {Заголовок таблицы требований логических серверов}
StringGrid2: TStringGrid; {Таблица требований логических серверов}
ScrollBar3: TScrollBar; {Полоса прокрутки по горизонтали}
ScrollBar4: TScrollBar; {Полоса прокрутки по вертикали}

Label9: TLabel;          {Заголовок таблицы исключений для лог. серверов}
StringGrid3: TStringGrid; {Таблица исключений для лог. серверов}
ScrollBar5: TScrollBar; {Полоса прокрутки по горизонтали}
ScrollBar6: TScrollBar; {Полоса прокрутки по вертикали}

Label10: TLabel;         {Заголовок таблицы требования базовой ОС}
StringGrid4: TStringGrid; {Таблица требования базовой ОС}
ScrollBar7: TScrollBar; {Полоса прокрутки по вертикали}

Label11: TLabel;         {Заголовок таблицы маски критериев оптимизации}
StringGrid5: TStringGrid; {Таблица маски критериев оптимизации}
ScrollBar8: TScrollBar; {Полоса прокрутки по вертикали}

Label12: TLabel;         {Заголовок таблицы имен типов ресурсов}
StringGrid6: TStringGrid; {Таблица имен типов ресурсов}
ScrollBar9: TScrollBar; {Полоса прокрутки по вертикали}

Label13: TLabel;         {Заголовок результирующей таблицы распределения}
StringGrid7: TStringGrid; {Результирующая таблица распределения}
ScrollBar10: TScrollBar; {Полоса прокрутки по вертикали}
ScrollBar11: TScrollBar; {Полоса прокрутки по горизонтали}

Label14: TLabel;         {Заголовок результирующей таблицы загрузки ресурсов}
StringGrid8: TStringGrid; {Результирующая таблица загрузки ресурсов}
ScrollBar12: TScrollBar; {Полоса прокрутки по вертикали}
ScrollBar13: TScrollBar; {Полоса прокрутки по горизонтали}

InfoImage: TImage;       {Информационная картинка}

{Методы класса}

{Обработчик события создания основной формы окна}
procedure FormCreate(Sender: TObject);

{Обработчик события запроса на закрытие основной формы окна}
procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);

{Обработчик события уничтожения основной формы окна}
procedure FormDestroy(Sender: TObject);

{Обработчик события активации в меню элемента New в подменю Task}
procedure ItemNewClick(Sender: TObject);

{Обработчик события активации в меню элемента Load в подменю Task}
procedure ItemLoadClick(Sender: TObject);

{Обработчик события активации в меню элемента Save в подменю Task}
procedure ItemSaveClick(Sender: TObject);

{Обработчик события активации в меню элемента SaveAs в подменю Task}
procedure ItemSaveAsClick(Sender: TObject);

{Обработчик события активации в меню элемента Exit в подменю Task}
procedure ItemExitClick(Sender: TObject);

{Обработчик события активации в меню элемента Save в подменю Results}

```

```

procedure ItemResSaveClick(Sender: TObject);

{Обработчик события активации в меню элемента SaveAs в подменю Results}
procedure ItemResSaveAsClick(Sender: TObject);

{Обработчик события активации в меню элемента About в подменю Help}
procedure ItemAboutClick(Sender: TObject);

{Обработчик события нажатия кнопки запуска потока решения задачи}
procedure ButtonRunClick(Sender: TObject);

{Обработчик события нажатия кнопки завершения потока решения задачи}
procedure ButtonStopClick(Sender: TObject);

{Обработчик события нажатия кнопки замораживания потока решения задачи}
procedure ButtonPauseClick(Sender: TObject);

{Обработчик события нажатия кнопки пробуждения потока решения задачи}
procedure ButtonResumeClick(Sender: TObject);

protected
  {Главная оконная процедура приложения}
  procedure WndProc (var Message: TMessage); override;

private { Private declarations }
{Private поля, основные данные класса}
workfile: string;           {текущий рабочий файл задачи}
resfile: string;           {текущий рабочий файл результатов}

SourceDataIsModified: byte; {флаг модификации исходных данных}
LastSrcSaveErrorFlag: byte; {флаг ошибки последней операции
                             сохранения исходных данных}

ResultDataIsModified: byte; {флаг модификации результирующих данных}
LastResSaveErrorFlag: byte; {флаг ошибки последней операции
                             сохранения результатов}

{Внимание! ранее объявленные таблицы StringGrid имеют фиксированный
размер и реально они никакие данные не хранят, а служат лишь для
удобной формы редактирования и отображения данных интерфейсных матриц,
столбцов и строк, которые описаны ниже. Таблицы StringGrid в приложении
работают как виртуальные окна строковых данных фиксированного размера,
на которые динамически отображаются части интерфейсных матриц, столбцов и
строк, служащих для хранения данных, поставляемых из интерфейса или при
загрузке задачи из файла. Отображение управляется полосами прокрутки
и клавиатурой. Такой непростой механизм реализован поскольку StringGrid
потребляют неприемлемое количество памяти: каждое 1, 2 или 4-байтное
число хранится в ячейках StringGrid в строковом виде и занимает от 24
до 64 байт памяти! Поэтому о хранении всех данных (в сумме свыше 5
миллионов элементов при размерностях задач NH=NS=NC=NX=1000) в StringGrid
не могло быть и речи. Кроме того, запись в StringGrid – операция очень
медленная при больших объемах данных}

{Интерфейсные матрицы/столбцы исходных данных и матрицы/строки результатов
не являются теми данными, которыми непосредственно оперирует поток решения
задач, они служат лишь для того, чтобы не хранить все данные в строковом
виде в StringGrid, что просто неприемлемо по определению. При решении
задач, равно как и при загрузке и сохранении задач, эти матрицы выступают
как бы от имени интерфейса, представляющего те данных, которые пользователь
видит на экране, пусть даже динамически (у пользователя просто создается
иллюзия, что абсолютно все данные постоянно находятся в StringGrid). Эти
данные после синтаксического анализа передаются уже в свои внутренние
матрицы данных вспомогательного класса, все операции вычисления, загрузки
и сохранения производятся только через внутренние матрицы. Это может
показаться дублированием данных, однако, конструкторы, унаследованные от

```

класса расширенной задачи распределения, вспомогательного класса не работают непосредственно с интерфейсными матрицами, а только со своими полями данных (внутренними матрицами и переменными). Фактически это вытекает из того, что класс формы приложения не могло бы быть наследником частного класса частной иерархии классов задач дискретной оптимизации. У класса формы свои данные, у частных классов задач оптимизации свои}

{Интерфейсные матрицы и столбцы исходных данных и интерфейсные матрицы и строки результатов (см. ниже) существуют совершенно независимо и управляются совершенно независимо с помощью независимых переменных. Размеры интерфейсных матриц/столбцов исходных данных динамически управляются полями Edit1, Edit2, Edit3, Edit4. Размеры интерфейсных матриц и строк результатов не управляются этими полями, они выставляются потоком решения задачи на этапе записи результатов решения в интерфейсные матрицы и строки результатов}

{Следующий важный момент - это то, что интерфейсные матрицы и столбцы исходных данных, несмотря на динамическое изменение размеров никогда не уменьшаются (а только увеличиваются, когда один из заданных размеров оказывается больше ранее достигнутого максимального значения) ни по одной из своих размерностей. Это полностью предотвращает потерю данных при случайном или намеренном уменьшении параметров размерностей задачи. Это позволяет при желании запустить задачу при меньших размерностях, затем вернуться к прежним размерностям, не рискуя потерять исходные данные в интерфейсных матрицах и столбцах. Разумеется, за это удобство приходится расплачиваться памятью. При создании же новой или загрузке задачи все исходные и результирующие данные теряются и все интерфейсные матрицы и столбцы исходных данных и интерфейсные матрицы и строки результатов освобождают память. Однако, перед этим пользователю всегда предлагается сохранить исходные данные и результаты в файлы}

{Интерфейсные матрицы и столбцы исходных данных}

```
{матрица ресурсов физических компьютеров}
HSTGR: T_HST_GRID;
HSTGR_Width, HSTGR_Height: integer; {текущие размеры матрицы}
{текущие достигнутые наибольшие размеры матрицы}
HSTGR_MaxWidth, HSTGR_MaxHeight: integer;
```

```
{матрица требований логических серверов}
SSTGR: T_SST_GRID;
SSTGR_Width, SSTGR_Height: integer; {текущие размеры матрицы}
{текущие достигнутые наибольшие размеры матрицы}
SSTGR_MaxWidth, SSTGR_MaxHeight: integer;
```

```
{матрица исключений для лог. серверов}
EXTGR: T_EXT_GRID;
EXTGR_Width, EXTGR_Height: integer; {текущие размеры матрицы}
{текущие достигнутые наибольшие размеры матрицы}
EXTGR_MaxWidth, EXTGR_MaxHeight: integer;
```

```
{столбец требований базовой ОС}
BSTGR: T_BST_GRID;
BSTGR_Height: integer; {текущий размер столбца}
{текущий достигнутый наибольший размер столбца}
BSTGR_MaxHeight: integer;
```

```
{столбец маски критериев оптимизации}
CMGR: T_CM_GRID;
CMGR_Height: integer; {текущий размер столбца}
{текущий достигнутый наибольший размер столбца}
CMGR_MaxHeight: integer;
```

```
{столбец имен типов ресурсов}
```

```

RTNGR: T_RT_N_GRID;
RTNGR_Height: integer; {текущий размер столбца}
{текущий достигнутый наибольший размер столбца}
RTNGR_MaxHeight: integer;

{Интерфейсные матрицы и строки результирующих данных}

{матрица распределения логических серверов}
DTGR: T_DT_GRID;
DTGR_Width, DTGR_Height: integer; {текущие размеры матрицы}
{текущие достигнутые наибольшие размеры матрицы}
DTGR_MaxWidth, DTGR_MaxHeight: integer;

{матрица загрузки ресурсов физических компьютеров}
RLGR: T_RL_GRID;
RLGR_Width, RLGR_Height: integer; {текущие размеры матрицы}
{достигнутые наибольшие размеры матрицы}
RLGR_MaxWidth, RLGR_MaxHeight: integer;

{строка индексов нераспределенных лог. серверов, текущий размер
берется из текущего числа столбцов матрицы распределения}
DTNL: T_DT_HEADLINE;

{строка имен типов ресурсов, текущий размер берется из
текущего числа столбцов матрицы загрузки ресурсов}
RLNL: T_RL_HEADLINE;

{Параметры размерности для интерфейсных матриц исходных данных}
main_nh: integer; {текущее число физических компьютеров}
main_ns: integer; {текущее число логических серверов}
main_nc: integer; {текущее число типов ресурсов}
main_nx: integer; {текущее число исключений}

{Параметры управления поиском}
max_radius: integer; {текущий радиус зон поиска}
max_starts: integer; {текущее число старт-точек}

{Параметры размерности для интерфейсных матриц результирующих данных}
res_nh: integer; {текущее число физических компьютеров}
res_ns: integer; {текущее число логических серверов}
res_nc: integer; {текущее число типов ресурсов}
res_nx: integer; {текущее число исключений}

{Специальный флаг, позволяющий включать и отключать}
{обработку события редактирования полей Edit1-Edit6}
EditFieldsOnChangeHandling: Boolean;

{Private методы класса}

{Ключевые обработчики событий для полей Edit1-6 и}
{таблиц StringGrid1-8, а также полос прокруток ScrollBar1-13}

{Обработчик события редактирования числа физических компьютеров}
procedure Edit1Change(Sender: TObject);

{Обработчик события выхода из поля числа физических компьютеров}
procedure Edit1Exit(Sender: TObject);

{Обработчик события редактирования числа логических серверов}
procedure Edit2Change(Sender: TObject);

{Обработчик события выхода из поля числа логических серверов}
procedure Edit2Exit(Sender: TObject);

```

```

{Обработчик события редактирования числа типов ресурсов}
procedure Edit3Change(Sender: TObject);
{Обработчик события выхода из поля числа типов ресурсов}
procedure Edit3Exit(Sender: TObject);

{Обработчик события редактирования числа исключений}
procedure Edit4Change(Sender: TObject);

{Обработчик события выхода из поля числа исключений}
procedure Edit4Exit(Sender: TObject);

{Обработчик события редактирования максимального радиуса поиска}
procedure Edit5Change(Sender: TObject);

{Обработчик события выхода из поля максимального радиуса поиска}
procedure Edit5Exit(Sender: TObject);

{Обработчик события редактирования числа стартовых точек поиска}
procedure Edit6Change(Sender: TObject);

{Обработчик события выхода из поля числа стартовых точек поиска}
procedure Edit6Exit(Sender: TObject);

{Обработчик события редактирования таблицы ресурсов физических компьютеров}
procedure StringGrid1SetEditText(Sender: TObject; ACol, ARow: Integer;
  const Value: String);

{Обработчик события редактирования таблицы требований логических серверов}
procedure StringGrid2SetEditText(Sender: TObject; ACol, ARow: Integer;
  const Value: String);

{Обработчик события редактирования таблицы исключений для лог. серверов}
procedure StringGrid3SetEditText(Sender: TObject; ACol, ARow: Integer;
  const Value: String);

{Обработчик события редактирования таблицы требования базовой ОС}
procedure StringGrid4SetEditText(Sender: TObject; ACol, ARow: Integer;
  const Value: String);

{Обработчик события редактирования таблицы маски критериев оптимизации}
procedure StringGrid5SetEditText(Sender: TObject; ACol, ARow: Integer;
  const Value: String);

{Обработчик события редактирования таблицы имен типов ресурсов}
procedure StringGrid6SetEditText(Sender: TObject; ACol, ARow: Integer;
  const Value: String);

{Общий для всех полос прокрутки обработчик перемещения ползунка}
procedure ScrollBarChangeRouter(Sender: TObject);

{Общий для всех таблиц данных обработчик нажатия}
{клавиш Up, Down, Left, Right, PgUp и PgDn}
procedure StringGridsKeyDownRouter
(Sender: TObject; var Key: Word; Shift: TShiftState);

{Общий для всех таблиц данных обработчик движения колесика мыши вверх}
procedure StringGridsMouseWheelUpRouter(Sender: TObject;
  Shift: TShiftState; MousePos: TPoint; var Handled: Boolean);

{Общий для всех таблиц данных обработчик движения колесика мыши вниз}
procedure StringGridsMouseWheelDownRouter(Sender: TObject;
  Shift: TShiftState; MousePos: TPoint; var Handled: Boolean);

{Процедура обновления содержимого таблицы ресурсов физ. компьютеров}

```

```

procedure StringGrid1Update;

{Процедура обновления содержимого таблицы требований лог. серверов}
procedure StringGrid2Update;

{Процедура обновления содержимого таблицы исключений для лог. серверов}
procedure StringGrid3Update;

{Процедура обновления содержимого таблицы требования базовой ОС}
procedure StringGrid4Update;

{Процедура обновления содержимого таблицы маски критериев оптимизации}
procedure StringGrid5Update;

{Процедура обновления содержимого таблицы имен типов ресурсов}
procedure StringGrid6Update;

{Процедура обновления содержимого результирующей таблицы распределения}
procedure StringGrid7Update;

{Процедура обновления содержимого результирующей таблицы загрузки ресурсов}
procedure StringGrid8Update;

{Процедуры установки размеров интерфейсных матриц/столбцов}
{исходных данных и матриц/строк результирующих данных}

{Процедура установки размера для одной из осей}
{интерфейсной матрицы ресурсов физ. компьютеров}
procedure HSTGR_SetSize
(axis: byte; newsize: integer; ignore_max: boolean;
var p_return_flag: byte; var p_return_err: string);

{Процедура установки размера для одной из осей}
{интерфейсной матрицы требований лог. серверов}
procedure SSTGR_SetSize
(axis: byte; newsize: integer; ignore_max: boolean;
var p_return_flag: byte; var p_return_err: string);

{Процедура установки размера для одной из осей}
{интерфейсной матрицы исключений для лог. серверов}
procedure EXTGR_SetSize
(axis: byte; newsize: integer; ignore_max: boolean;
var p_return_flag: byte; var p_return_err: string);

{Процедура установки размера интерфейсного столбца требований базовой ОС}
procedure BSTGR_SetSize
(newsize: integer; ignore_max: boolean;
var p_return_flag: byte; var p_return_err: string);

{Процедура установки размера интерфейсного столбца маски критериев}
procedure CMGR_SetSize
(newsize: integer; ignore_max: boolean;
var p_return_flag: byte; var p_return_err: string);

{Процедура установки размера интерфейсного столбца имен типов ресурсов}
procedure RTNGR_SetSize
(newsize: integer; ignore_max: boolean;
var p_return_flag: byte; var p_return_err: string);

{Процедура установки размера для одной из осей}
{интерфейсной матрицы результатов распределения}
procedure DTGR_SetSize
(axis: byte; newsize: integer; ignore_max: boolean;
var p_return_flag: byte; var p_return_err: string);

```

```

{Процедура установки размера для одной из осей}
{интерфейсной матрицы загрузки ресурсов}
procedure RLGR_SetSize
(axis: byte; newsize: integer; ignore_max: boolean;
var p_return_flag: byte; var p_return_err: string);

{Ключевые рабочие процедуры и функции}

{Функция "тестовой попытки выделения памяти", используется перед тем}
{как использовать функцию SetLength для расширения интерфейсных}
{матриц/столбцов исходных данных или матриц/строк результатов}
{поскольку в случае нехватки памяти SetLength разрушает данные}
{В случае успешного выделения памяти, память сразу освобождается}
{и функция возвращает значение ИСТИНА, иначе - значение ЛОЖЬ}
function TryMemoryAllocate (MemSize: Integer): boolean;

{Процедура уничтожения исходных данных и освобождения памяти}
procedure FlushSourceData;

{Процедура уничтожения результирующих данных и освобождения памяти}
procedure FlushResultData;

{Процедура запрещения редактирования исходных данных}
procedure DisableEditSourceData;

{Процедура разрешения редактирования исходных данных}
procedure EnableEditSourceData;

{Функция "защиты данных от потери" - проверки фактов модификации исходных}
{данных и обновления результатов, в положительном случае - вывод}
{соответствующих диалогов на сохранение и выполнение операций сохранения}
function SafeTaskAndResultsSaving: boolean;

{Процедура загрузки исходных данных из файла}
procedure LoadTaskFromFile
(p_filename: string; var p_return_flag: byte; var p_return_err: string);

{Процедура сохранения исходных данных в файл}
procedure SaveTaskToFile
(p_filename: string; var p_return_flag: byte; var p_return_err: string);

{Процедура сохранения результатов в файл}
procedure SaveResultToFile
(p_filename: string; var p_return_flag: byte; var p_return_err: string);
end;

type
{Класс потока решения задачи}
TSolverThread = class(TThread)
private
    st_err_flag: byte;      {Флаг ошибки потока}
    st_err_str: string;     {Текст ошибки потока}
    ThreadTask: TCustomTask; {Экземпляр класса вспомогательной задачи}
protected
    {Конструктор потока}
    constructor Create;
    {Метод запуска потока}
    procedure Execute; override;
    {Обработчик события завершения потока}
    procedure ThreadTerminateEvent (Sender: TObject);
end;

```



```

type
  {Класс потока мониторинга}
  TStatusThread = class(TThread)
  protected
    {Конструктор потока}
    constructor Create;
    {Метод запуска потока}
    procedure Execute; override;
    {Обработчик события завершения потока}
    procedure ThreadTerminateEvent (Sender: TObject);
  end;

var
  Form1: TForm1; {Основная форма приложения}

  {глобальный флаг ошибки инициализации приложения}
  appinit_error: byte;

  {поток решения задачи}
  SolverThread: TSolverThread;

  {переменная для хэндла хранения потока решения задачи}
  SolverThreadHandle: THandle;

  {состояние прогресса в потоке решения}
  SolverTaskState: TTaskState;

  {поток мониторинга}
  StatusThread: TStatusThread;

  {переменная для хэндла потока мониторинга}
  StatusThreadHandle: THandle;

  {Переменная для хранения позиции Progress Bar1}
  ProgressBar1Level: integer;
  {Переменная для хранения позиции Progress Bar2}
  ProgressBar2Level: integer;

  {Массив для хранения текстов сообщений для секций 1-6 строки состояния}
  {которые обновляются потоком решения и потоком мониторинга}
  StatusBarSectionTexts: array [1..6] of string;
  InfoFlag: boolean;

  {Специальное пользовательское сообщение Windows, которое посылается потоком}
  {мониторинга основному окну приложения для синхронизации данных,}
  {отображаемых в секциях 1-6 строки состояния и позиций Progress Bar 1 и 2,}
  {с данными в переменных ProgressBar1Level, ProgressBar2Level и массиве}
  {StatusBarSectionTexts}
  MY_STATUSUPDATE_MSG: UINT;

  sys_freq: Int64; {Частота внутреннего системного высокоточного счетчика}
  sys_count1: Int64; {Значение системного счетчика при старте решения задачи}
  sys_count2: Int64; {Значение системного счетчика при окончании решения}

implementation

{$R *.dfm}

{-----}

      {ГЛАВНАЯ ОКОННАЯ ПРОЦЕДУРА ПРИЛОЖЕНИЯ}

{-----}

```

```

{Данные переменных ProgressBar1Level, ProgressBar2Level и массива}
{StatusBarSectionTexts отображаются на экране, только когда основной поток}
{обрабатывает специальное Windows сообщение MY_STATUSUPDATE_MSG, которое}
{посылает поток мониторинга окну приложения, но не ждет его обработки}
{Сообщение обрабатывается сразу же как только основной поток доберется до него}
{в очереди сообщений. Секция 0 строки состояния предназначена для самого}
{основного потока и он самостоятельно обновляет текст в ней}

procedure TForm1.WndProc;
var
  i: integer;      {временная переменная}
begin
  if (Message.Msg = MY_STATUSUPDATE_MSG) then {Если специальное сообщение}
  begin
    for i:=1 to 6 do {Обновление содержимого для секций 1-6 строки состояния}
    begin
      if (AppStatusBar.Panels.Items[i].Text <> StatusBarSectionTexts[i])
      then AppStatusBar.Panels.Items[i].Text:= StatusBarSectionTexts[i];
    end;
    if (ProgressBar1.Position <> ProgressBar1Level)
    then ProgressBar1.Position:= ProgressBar1Level; {Обновление ProgressBar1}
    if (ProgressBar2.Position <> ProgressBar2Level)
    then ProgressBar2.Position:= ProgressBar2Level; {Обновление ProgressBar2}
    InfoImage.Visible:= InfoFlag; {вывод информации}
  end;
  Inherited WndProc (Message); {вызов унаследованного обработчика}
end;

{-----}

      {Обработчики событий, код основного потока приложения}

{-----}

{Обработчик события создания основной формы окна}
{Входные параметры:
  Sender: компонента, с которой связано событие}

procedure TForm1.FormCreate(Sender: TObject);
var
  i: integer;      {временная переменная}
  err_flag: byte;   {локальный флаг ошибки}
  err_text: string; {текст локальной ошибки}
begin
  err_flag:= 0;      {сброс локального флага ошибки}
  err_text:= '';     {сброс текста локальной ошибки}

  MY_STATUSUPDATE_MSG:= 0; {обнуление кода специального сообщения Windows}

  try
    MY_STATUSUPDATE_MSG:=
      RegisterWindowMessage(PChar('STATUSTHREAD_UPDATE_MSG'));
  except {в случае системной ошибки}
    On EOSError do
    begin
      err_flag:= 1; {генерация ошибки}
      err_text:= WINMSG_REG_ERR;
    end;
  end;

  if ((err_flag = 0) and (MY_STATUSUPDATE_MSG = 0)) then
  begin
    {Если код остался нулевым}
    err_flag:= 1; {то генерация ошибки}
    err_text:= WINMSG_REG_ERR;
  end;
end;

```

```

end;

if (err_flag = 0) then
begin
    {Установка общего числа строк и столбцов и числа}
    {зафиксированных строк и столбцов для StringGrid}

    {Таблица ресурсов физических компьютеров}
    StringGrid1.ColCount:= 10; StringGrid1.RowCount:= 5;
    StringGrid1.FixedCols:= 1; StringGrid1.FixedRows:= 1;

    {Таблица требований логических серверов}
    StringGrid2.ColCount:= 10; StringGrid2.RowCount:= 5;
    StringGrid2.FixedCols:= 1; StringGrid2.FixedRows:= 1;

    {Таблица исключений для лог. серверов}
    StringGrid3.ColCount:= 6; StringGrid3.RowCount:= 5;
    StringGrid3.FixedCols:= 1; StringGrid3.FixedRows:= 1;

    {Таблица требования базовой ОС}
    StringGrid4.ColCount:= 2; StringGrid4.RowCount:= 5;
    StringGrid4.FixedCols:= 1; StringGrid4.FixedRows:= 1;

    {Таблица маски критериев оптимизации}
    StringGrid5.ColCount:= 2; StringGrid5.RowCount:= 5;
    StringGrid5.FixedCols:= 1; StringGrid5.FixedRows:= 1;

    {Таблица имен типов ресурсов}
    StringGrid6.ColCount:= 2; StringGrid6.RowCount:= 5;
    StringGrid6.FixedCols:= 1; StringGrid6.FixedRows:= 1;

    {Результирующая таблица распределения}
    StringGrid7.ColCount:= 8; StringGrid7.RowCount:= 5;
    StringGrid7.FixedCols:= 1; StringGrid7.FixedRows:= 1;

    {Результирующая таблица загрузки ресурсов}
    StringGrid8.ColCount:= 5; StringGrid8.RowCount:= 5;
    StringGrid8.FixedCols:= 1; StringGrid8.FixedRows:= 1;

    {Назначение для всех StringGrid общего обработчика, события}
    {нажатия клавиш Up, Down, Left, Right, PgUp, PgDn и движения}
    {колесика мыши вверх/вниз - WheelUp и WheelDown}

    {Таблица ресурсов физических компьютеров}
    StringGrid1.OnKeyDown:= StringGridsKeyDownRouter;
    StringGrid1.OnMouseWheelUp:= StringGridsMouseWheelUpRouter;
    StringGrid1.OnMouseWheelDown:= StringGridsMouseWheelDownRouter;

    {Таблица требований логических серверов}
    StringGrid2.OnKeyDown:= StringGridsKeyDownRouter;
    StringGrid2.OnMouseWheelUp:= StringGridsMouseWheelUpRouter;
    StringGrid2.OnMouseWheelDown:= StringGridsMouseWheelDownRouter;

    {Таблица исключений для лог. серверов}
    StringGrid3.OnKeyDown:= StringGridsKeyDownRouter;
    StringGrid3.OnMouseWheelUp:= StringGridsMouseWheelUpRouter;
    StringGrid3.OnMouseWheelDown:= StringGridsMouseWheelDownRouter;

    {Таблица требования базовой ОС}
    StringGrid4.OnKeyDown:= StringGridsKeyDownRouter;
    StringGrid4.OnMouseWheelUp:= StringGridsMouseWheelUpRouter;
    StringGrid4.OnMouseWheelDown:= StringGridsMouseWheelDownRouter;
    {Таблица маски критериев оптимизации}
    StringGrid5.OnKeyDown:= StringGridsKeyDownRouter;

```

```

StringGrid5.OnMouseWheelUp:= StringGridsMouseWheelUpRouter;
StringGrid5.OnMouseWheelDown:= StringGridsMouseWheelDownRouter;

{Таблица имен типов ресурсов}
StringGrid6.OnKeyDown:= StringGridsKeyDownRouter;
StringGrid6.OnMouseWheelUp:= StringGridsMouseWheelUpRouter;
StringGrid6.OnMouseWheelDown:= StringGridsMouseWheelDownRouter;

{Результирующая таблица распределения}
StringGrid7.OnKeyDown:= StringGridsKeyDownRouter;
StringGrid7.OnMouseWheelUp:= StringGridsMouseWheelUpRouter;
StringGrid7.OnMouseWheelDown:= StringGridsMouseWheelDownRouter;

{Результирующая таблица загрузки ресурсов}
StringGrid8.OnKeyDown:= StringGridsKeyDownRouter;
StringGrid8.OnMouseWheelUp:= StringGridsMouseWheelUpRouter;
StringGrid8.OnMouseWheelDown:= StringGridsMouseWheelDownRouter;

{Назначение обработчиков события редактирования в ячейках для таблиц}

{Таблица ресурсов физических компьютеров}
StringGrid1.OnSetEditText:= StringGrid1SetEditText;

{Таблица требований логических серверов}
StringGrid2.OnSetEditText:= StringGrid2SetEditText;

{Таблица исключений для лог. серверов}
StringGrid3.OnSetEditText:= StringGrid3SetEditText;

{Таблица требования базовой ОС}
StringGrid4.OnSetEditText:= StringGrid4SetEditText;

{Таблица маски критериев оптимизации}
StringGrid5.OnSetEditText:= StringGrid5SetEditText;

{Таблица имен типов ресурсов}
StringGrid6.OnSetEditText:= StringGrid6SetEditText;

{Назначение для всех ScrollBar общего обработчика перемещения ползунка}

{Горизонтальная полоса прокрутки таблицы ресурсов физ. компьютеров}
ScrollBar1.OnChange:= ScrollBarChangeRouter;

{Вертикальная полоса прокрутки таблицы ресурсов физ. компьютеров}
ScrollBar2.OnChange:= ScrollBarChangeRouter;

{Горизонтальная полоса прокрутки таблицы требований лог. серверов}
ScrollBar3.OnChange:= ScrollBarChangeRouter;

{Вертикальная полоса прокрутки таблицы требований лог. серверов}
ScrollBar4.OnChange:= ScrollBarChangeRouter;

{Горизонтальная полоса прокрутки таблицы исключений для лог. серверов}
ScrollBar5.OnChange:= ScrollBarChangeRouter;

{Вертикальная полоса прокрутки таблицы исключений для лог. серверов}
ScrollBar6.OnChange:= ScrollBarChangeRouter;

{Вертикальная полоса прокрутки таблицы требований базовой ОС}
ScrollBar7.OnChange:= ScrollBarChangeRouter;

{Вертикальная полоса прокрутки таблицы масок критериев оптимизации}
ScrollBar8.OnChange:= ScrollBarChangeRouter;
{Вертикальная полоса прокрутки таблицы имен типов ресурсов}

```

```

ScrollBar9.OnChange:= ScrollBarChangeRouter;

{Горизонтальная полоса прокрутки таблицы результатов распределения}
ScrollBar10.OnChange:= ScrollBarChangeRouter;

{Вертикальная полоса прокрутки таблицы результатов распределения}
ScrollBar11.OnChange:= ScrollBarChangeRouter;

{Горизонтальная полоса прокрутки таблицы результатов загрузки ресурсов}
ScrollBar12.OnChange:= ScrollBarChangeRouter;

{Вертикальная полоса прокрутки таблицы результатов загрузки ресурсов}
ScrollBar13.OnChange:= ScrollBarChangeRouter;

{Установка максимальной длины вводимой строки в основных полях}
{редактирования размерностей задачи, радиуса поиска и число старт-точек}

Edit1.MaxLength:=3; {Поле редактирования числа физических компьютеров}
Edit2.MaxLength:=3; {Поле редактирования числа логических серверов}
Edit3.MaxLength:=3; {Поле редактирования числа типов ресурсов}
Edit4.MaxLength:=3; {Поле редактирования числа исключений}
Edit5.MaxLength:=3; {Поле редактирования максимального радиуса поиска}
Edit6.MaxLength:=3; {Поле редактирования числа стартовых точек поиска}

{Назначение обработчиков события редактирования и выхода для полей}
{редактирования размерностей задачи, радиуса поиска и число старт-точек}

{Поле редактирования числа физических компьютеров}
Edit1.OnChange:= Edit1Change; Edit1.OnExit:= Edit1Exit;
{Поле редактирования числа логических серверов}
Edit2.OnChange:= Edit2Change; Edit2.OnExit:= Edit2Exit;
{Поле редактирования числа типов ресурсов}
Edit3.OnChange:= Edit3Change; Edit3.OnExit:= Edit3Exit;
{Поле редактирования числа исключений}
Edit4.OnChange:= Edit4Change; Edit4.OnExit:= Edit4Exit;
{Поле редактирования максимального радиуса поиска}
Edit5.OnChange:= Edit5Change; Edit5.OnExit:= Edit5Exit;
{Поле редактирования числа стартовых точек поиска}
Edit6.OnChange:= Edit6Change; Edit6.OnExit:= Edit6Exit;
{С этого момента любая ручная и, что очень важно, также}
{и программная модификация полей редактирования}
{будет немедленно вызывать обработчик события модификации}

{Разрешение обработки события редактирования полей Edit1-Edit6}
EditFieldsOnChangeHandling:= True;
{Начальное уничтожение исходных данных с освобождением памяти}
FlushSourceData;
{Начальное уничтожение результирующих данных с освобождением памяти}
FlushResultData;
{Начальная установка имени файла задачи}
workfile:='untitled.dat';
{Начальная установка имени файла результатов}
resfile:='untitled.res';

{Начальная установка заголовка окна приложения}
Form1.Caption:= APP_TITLE + ' - ' + ExtractFileName(workfile)
+ ' / ' + ExtractFileName(resfile);

SourceDataIsModified:= 0; {Сброс флага модификации исх. данных}
LastSrcSaveErrorFlag:= 0; {Сброс флага ошибки сохранения исх. данных}
ResultDataIsModified:= 0; {Сброс флага обновления результатов}
LastResSaveErrorFlag:= 0; {Сброс флага ошибки сохранения результатов}

{Инициализация секций строки состояния}

```

```

AppStatusBar.Panels.Items[0].Text:= 'Idle';   {Операции основного потока}
AppStatusBar.Panels.Items[1].Text:= 'Idle';   {Операции потока решения}
AppStatusBar.Panels.Items[2].Text:= '';       {Состояние потока решения}
AppStatusBar.Panels.Items[3].Text:= '';       {Доступная физическая память}
AppStatusBar.Panels.Items[4].Text:= '';       {Доступная виртуальная память}
AppStatusBar.Panels.Items[5].Text:= '';       {Флаг модификации исх. данных}
AppStatusBar.Panels.Items[6].Text:= '';       {Флаг обновления результатов}

{Обнуление массива для хранения текстов
сообщений для секций 1-6 строки состояния}
for i:=1 to 6 do StatusBarSectionTexts[i]:='';

ButtonRun.Enabled:= True;           {Разрешение кнопки запуска решения}
ButtonStop.Enabled:= False;         {Запрещение кнопки завершения решения}
ButtonPause.Enabled:= False;        {Запрещение кнопки замораживания решения}
ButtonResume.Enabled:= False;       {Запрещение кнопки пробуждения решения}
InfoImage.Visible:=False;           {Отключение информационной картинки}

ProgressBar1.Position:=0;           {Сброс уровня прогресса 1 решения задачи}
ProgressBar2.Position:=0;           {Сброс уровня прогресса 2 решения задачи}

{Обнуление состояния решения задачи распределения}

SolverTaskState.CurrentH:= 0;        {число обработанных физ. компьютеров}
SolverTaskState.RemainNH:= 0;        {число оставшихся физ. компьютеров}
SolverTaskState.RemainNS:= 0;        {число оставшихся лог. серверов}
SolverTaskState.TotalNH:= 0;         {общее число физ. компьютеров}
SolverTaskState.TotalNS:= 0;         {общее число лог. серверов}
SolverTaskState.Stage:= '';          {текущая стадия решения задачи}

{Обнуление указателя на поток решения}
SolverThread:= nil;
{Обнуление переменной-хэндла потока решения}
SolverThreadHandle:= 0;
{Обнуление указателя на поток мониторинга}
StatusThread:= nil;
{Обнуление переменной-хэндла потока мониторинга}
StatusThreadHandle:= 0;

try
  {Попытка создания потока мониторинга}
  StatusThread:= TStatusThread.Create;
  {Запуск потока мониторинга}
  StatusThread.Resume;
  {Сохранение хэндла потока в соответствующей переменной}
  StatusThreadHandle:= StatusThread.Handle;
except
  On EOutOfMemory do                {В случае нехватки памяти}
  begin
    err_flag:= 1;                    {генерация соответствующей ошибки}
    err_text:= MEM_ALLOC_ERR + STATUS_THREAD_CREATE_ERR;
    StatusThreadHandle:= 0; {Обнуление переменной-хэндла потока}
  end;
  On EAccessViolation do            {В случае нарушения доступа}
  begin
    err_flag:= 1;                    {генерация соответствующей ошибки}
    err_text:= ACC_VIOL_ERR + STATUS_THREAD_CREATE_ERR;
    StatusThreadHandle:= 0; {Обнуление переменной-хэндла потока}
  end;
end;
end;

{Считывание частоты внутреннего системного высокоточного счетчика}
if (err_flag = 0) then QueryPerformanceFrequency(sys_freq);

```

```

    if (err_flag <> 0) then {Если в процессе обработки события}
    begin {создания основной формы окна возникла ошибка}
        MessageBox (Application.Handle, PChar(err_text), 'Error', MB_ICONERROR);
        appinit_error:= 1; {то установка глобального флага ошибки приложения}
    end;
end;

{Обработчик события запроса на завершение}
{Входные параметры:
    Sender: компонента, с которой связано событие}
{Выходные параметры:
    CanClose: флаг разрешения на закрытие основного окна приложения}

procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
var
    err_flag: byte; {локальный флаг ошибки}
begin
    err_flag:= 0; {сброс локального флага ошибки}

    {Если поток решения задачи существует}
    if (WaitForSingleObject (SolverThreadHandle, 0) = WAIT_TIMEOUT) then
    begin
        MessageDlg(SOLVER_RUNNING_ERR, mtError, [mbOK], 0);
        err_flag:= 1; {то вывод ошибки}
    end;

    {Вызов функции защиты данных и результатов от потери}
    if (err_flag = 0)
    then if (not(SafeTaskAndResultsSaving)) then err_flag:= 1;

    {Завершение потока мониторинга}
    if (err_flag = 0) then
    begin {Если поток мониторинга существует, то}
        if (WaitForSingleObject (StatusThreadHandle, 0) = WAIT_TIMEOUT) then
        begin {если он приостановлен, то запуск потока}
            if StatusThread.Suspended then StatusThread.Resume;
            StatusThread.Terminate; {Завершение потока}
        end;
    end;

    {Если все проверки прошли, то разрешение завершения программы, иначе - запрет}
    if err_flag = 0 then CanClose:= True else CanClose:= False;
end;

{Обработчик события уничтожения окна приложения}
{Входные параметры:
    Sender: компонента, с которой связано событие}

procedure TForm1.FormDestroy(Sender: TObject);
begin
    {уничтожение интерфейсных матриц/столбцов и строк данных}
    Finalize(HSTGR); Finalize(SSTGR); Finalize(EXTGR); Finalize(BSTGR);
    Finalize(CMGR); Finalize(RTNGR); Finalize(DTGR); Finalize(RLGR);
    Finalize(DTHL); Finalize(RLHL);
end;

{-----}

{Обработчики событий активации пунктов меню программы}

{-----}

```

```

{Обработчик события активации в меню элемента Exit в подменю Task}
{Входные параметры:
  Sender: компонента, с которой связано событие}

procedure TForm1.ItemExitClick(Sender: TObject);
begin
  Close;    {Запрос на завершение (закрытие основного окна приложения)}
end;

{Обработчик события активации в меню элемента About в подменю Help}
{Входные параметры:
  Sender: компонента, с которой связано событие}

procedure TForm1.ItemAboutClick(Sender: TObject);
begin
  AboutBox.ShowModal;    {Вызов модального окна About (см. модуль MYABOUT)}
end;

{Обработчик события активации в меню элемента New в подменю Task}
{Входные параметры:
  Sender: компонента, с которой связано событие}

procedure TForm1.ItemNewClick(Sender: TObject);
var
  Old_Cursor: TCursor;          {переменная для временного хранения курсора}
begin
  Old_Cursor:= Screen.Cursor;   {сохранение старого курсора}

  if (SafeTaskAndResultsSaving) then {вызов функции защиты данных от потери}
  begin
    {если функция вернула TRUE, то}
    Screen.Cursor:= crHourGlass; {установка курсора в песочные часы}

    {установка имени рабочего файла в имя файла по умолчанию}
    workfile:='untitled.dat';
    resfile:='untitled.res';
    {Заголовок окна приложения}
    Form1.Caption:= APP_TITLE + ' - ' + ExtractFileName(workfile)
    + ' / ' + ExtractFileName(resfile);

    {Уничтожение исходных данных и освобождение памяти}
    FlushSourceData;
    {Уничтожение результирующих данных и освобождение памяти}
    FlushResultData;
  end;
  Screen.Cursor:= Old_Cursor;    {Восстановление старого курсора}
end;

{Обработчик события активации в меню элемента Load в подменю Task}
{Входные параметры:
  Sender: компонента, с которой связано событие}

procedure TForm1.ItemLoadClick(Sender: TObject);
var
  err_flag: byte;                {локальный флаг ошибок}
  MyTask: TCustomTask;           {экземпляр класса вспомогательной задачи}
  ret_flag: byte;                {возвратный флаг ошибки метода вспом. задачи}
  ret_err: string;               {возвратный текст ошибки метода вспом. задачи}
  Old_Cursor: TCursor;           {переменная для временного хранения курсора}
begin
  Old_Cursor:= Screen.Cursor; {сохраняем старый курсор}
  err_flag:=0;                 {сброс локального флага ошибок}

```



```

if (SafeTaskAndResultsSaving) then      {вызов функции защиты данных от потери}
begin                                  {если функция вернула TRUE, то}
  AppOpenDialog.DefaultExt := 'dat';   {запуск диалогового окна}
  AppOpenDialog.FileName := '*.dat';   {выбора файла для открытия}
  if (not(AppOpenDialog.Execute)) then err_flag:=1;

  if (err_flag = 0) then                {Если файл был выбран то}
  begin
    Screen.Cursor:= crHourGlass; {установка курсора в песочные часы}

    {Вызов процедуры загрузки исходных данных из файла}
    LoadTaskFromFile(AppOpenDialog.FileName, ret_flag, ret_err);

    if (ret_flag <> 0) then              {Если возникла ошибка при загрузке}
    begin
      MessageDlg (ret_err, mtError, [mbOk], 0);
      err_flag:= 1;                    {то генерация ошибки}
    end;
  end;
end;
Screen.Cursor:= Old_Cursor;            {Восстановление старого курсора}
end;

{Обработчик события активации в меню элемента Save в подменю Task}
{Входные параметры:
  Sender: компонента, с которой связано событие}

procedure TForm1.ItemSaveClick(Sender: TObject);
var
  Old_Cursor: TCursor;      {переменная для временного хранения курсора}
  ret_flag: byte;           {возвратный флаг ошибки процедуры сохранения}
  ret_err: string;          {возвратный текст ошибки процедуры сохранения}
begin
  Old_Cursor:= Screen.Cursor; {сохраняем старый курсор}
  Screen.Cursor:= crHourGlass; {установка курсора в песочные часы}

  {Вызов процедуры сохранения исходных данных в файл}
  SaveTaskToFile(workfile, ret_flag, ret_err);

  if (ret_flag <> 0) then          {Если возникла ошибка при сохранении}
  begin                          {то генерация ошибки}
    MessageDlg (ret_err, mtError, [mbOk], 0);
    {и установка флага ошибки последней операции сохранения}
    LastSrcSaveErrorFlag:= 1;
  end {иначе сброс флага ошибки последней операции сохранения}
  else LastSrcSaveErrorFlag:= 0;

  Screen.Cursor:= Old_Cursor;    {Восстановление старого курсора}
end;

{Обработчик события активации в меню элемента Save As в подменю Task}
{Входные параметры:
  Sender: компонента, с которой связано событие}

procedure TForm1.ItemSaveAsClick(Sender: TObject);
var
  ret_flag: byte;           {переменная для временного хранения курсора}
  ret_err: string;          {возвратный флаг ошибки процедуры сохранения}
  Old_Cursor: TCursor;      {возвратный текст ошибки процедуры сохранения}
  err_flag: byte;           {локальный флаг ошибки}
begin
  Old_Cursor:= Screen.Cursor;  {сохраняем старый курсор}

```

```

err_flag:= 0;      {сброс локального флага ошибки}

AppSaveDialog.DefaultExt := 'dat';
AppSaveDialog.Filename := '*.dat';
{Открываем диалог выбора файла на сохранение}
if (not(AppSaveDialog.Execute)) then err_flag:= 1;

if (err_flag = 0) then      {если файл для исходных данных выбран}
begin
    Screen.Cursor:= crHourGlass; {то установка курсора в песочные часы}

    {Вызов процедуры сохранения исходных данных в файл}
    SaveTaskToFile(AppSaveDialog.FileName, ret_flag, ret_err);

    if (ret_flag <> 0) then      {Если возникла ошибка при сохранении}
    begin
        MessageDlg (ret_err, mtError, [mbOk], 0);
        err_flag:= 1;          {то генерация ошибки}
    end;
end;

{Если файл был выбран и данные в него сохранены, то}
{сброс флага ошибки последней операции сохранения, иначе его установка}
if (err_flag = 0) then LastSrcSaveErrorFlag:=0
else LastSrcSaveErrorFlag:=1;
Screen.Cursor:= Old_Cursor;    {Восстановление старого курсора}
end;

{Обработчик события активации в меню элемента Save в подменю Results}
{Входные параметры:
    Sender: компонента, с которой связано событие}

procedure TForm1.ItemResSaveClick(Sender: TObject);
var
    Old_Cursor: TCursor;      {переменная для временного хранения курсора}
    ret_flag: byte;           {возвратный флаг ошибки процедуры сохранения}
    ret_err: string;          {возвратный текст ошибки процедуры сохранения}
begin
    Old_Cursor:= Screen.Cursor; {сохраняем старый курсор}
    Screen.Cursor:= crHourGlass; {установка курсора в песочные часы}

    {Вызов процедуры сохранения результатов в файл}
    SaveResultToFile(resfile, ret_flag, ret_err);

    if (ret_flag <> 0) then      {Если возникла ошибка при сохранении}
    begin
        MessageDlg (ret_err, mtError, [mbOk], 0);
        {и установка флага ошибки последней операции сохранения}
        LastResSaveErrorFlag:= 1;
    end {иначе сброс флага ошибки последней операции сохранения}
    else LastResSaveErrorFlag:= 0;

    Screen.Cursor:= Old_Cursor;    {Восстановление старого курсора}
end;

{Обработчик события активации в меню элемента Save As в подменю Results}
{Входные параметры:
    Sender: компонента, с которой связано событие}

procedure TForm1.ItemResSaveAsClick(Sender: TObject);
var
    err_flag: byte;           {локальный флаг ошибки}
    Old_Cursor: TCursor;      {переменная для временного хранения курсора}

```

```

    ret_flag: byte;           {возвратный флаг ошибки процедуры сохранения}
    ret_err: string;          {возвратный текст ошибки процедуры сохранения}
begin
    err_flag:= 0;             {сброс локального флага ошибки}
    Old_Cursor:= Screen.Cursor; {сохраняем старый курсор}

    AppSaveDialog.DefaultExt := 'res';
    AppSaveDialog.Filename := '*.res';
    {Открываем диалог выбора файла на сохранение}
    if (not(AppSaveDialog.Execute)) then err_flag:=1;

    if (err_flag = 0) then      {если файл для результатов выбран}
    begin
        Screen.Cursor:= crHourGlass; {то установка курсора в песочные часы}

        {Вызов процедуры сохранения результатов в файл}
        SaveResultToFile(AppSaveDialog.FileName, ret_flag, ret_err);

        if (ret_flag <> 0) then    {Если возникла ошибка при сохранении}
        begin
            MessageDlg (ret_err, mtError, [mbOk], 0);
            err_flag:= 1;          {то генерация ошибки}
        end;
    end;

    {Если файл был выбран и результаты были успешно сохранены, то сброс флага}
    {ошибки последней операции сохранения результатов, иначе его установка}
    if (err_flag = 0) then LastResSaveErrorFlag:= 0
    else LastResSaveErrorFlag:= 1;
    Screen.Cursor:=Old_Cursor;      {Восстановление старого курсора}
end;

{Обработчик события нажатия кнопки Run}
{Входные параметры:
    Sender: компонента, с которой связано событие}

procedure TForm1.ButtonRunClick(Sender: TObject);
var
    err_flag: byte;           {локальный флаг ошибки}
begin
    err_flag:= 0;             {сброс локального флага ошибки}

    SolverThread:= nil;      {Обнуление указателя на поток решения}
    try
        {создание потока решения}
        SolverThread:= TSolverThread.Create;
        {запуск потока решения}
        SolverThread.Resume;
        {сохранение хэндла потока в соответствующей переменной}
        SolverThreadHandle:= SolverThread.Handle;
    except
        On EOutOfMemory do      {В случае нехватки памяти}
        begin
            {генерация ошибки}
            MessageDlg (MEM_ALLOC_ERR + SOLVER_THREAD_CREATE_ERR,
                mtError, [mbOK], 0);
            err_flag:= 1;
            SolverThreadHandle:= 0; {обнуление переменной-хэндла потока}
        end;
        On EAccessViolation do {В случае нарушения доступа}
        begin
            {генерация ошибки}
            MessageDlg (ACC_VIOL_ERR + SOLVER_THREAD_CREATE_ERR,
                mtError, [mbOK], 0);
            err_flag:= 1;
            SolverThreadHandle:= 0; {обнуление переменной-хэндла потока}
        end;
    end;
end;

```

```

        end;
    end;

    if (err_flag = 1) then                {В случае ошибки запуска потока решения}
    begin
        ButtonRun.Enabled:= True;        {Разрешение кнопки Run}
        ButtonStop.Enabled:= False;      {Запрет кнопки Stop}
        ButtonPause.Enabled:= False;     {Запрет кнопки Pause}
        ButtonResume.Enabled:= False;    {Запрет кнопки Resume}
        ItemNew.Enabled:= True;          {Разрешение элемента меню New}
        ItemLoad.Enabled:= True;         {Разрешение элемента меню Load}
        {Разрешение редактирования полей Edit1-6 и таблиц StringGrid1-6}
        Form1.EnableEditSourceData;
    end;
end;

{Обработчик события нажатия кнопки Stop}
{Входные параметры:
    Sender: компонента, с которой связано событие}

procedure TForm1.ButtonStopClick(Sender: TObject);
begin
    if (MessageDlg (SOLVER_STOP_WARN, mtWarning, [mbYes, mbNo], 0) = mrYes) then
    begin {Запрашиваем подтверждение на полный останов решения задачи}
        {Если поток решения заморожен, то пробуждаем его}
        if (SolverThread.Suspended = TRUE) then SolverThread.Resume;
        {Посылаем потоку команду на завершение - это не принудительное}
        {завершение, а лишь сигнал для потока для необходимости завершения}
        {для того, чтобы он мог корректно освободить память, выйти из}
        {всех циклов и процедур, и вернуться в процедуру Execute, выйти из нее}
        {и, наконец, самоуничтожиться. Перед самоуничтожением поток вызывает}
        {обработчик OnTerminate для финальных действий перед завершением}
        SolverThread.Terminate;
    end;
end;

{Обработчик события нажатия кнопки Pause}
{Входные параметры:
    Sender: компонента, с которой связано событие}

procedure TForm1.ButtonPauseClick(Sender: TObject);
begin
    Form1.ButtonPause.Enabled:= False;   {Запрещение кнопки Pause}
    Form1.ButtonResume.Enabled:= True;   {Разрешение кнопки Resume}
    SolverThread.Suspend; {Замораживаем поток. Приостанавливаем выполнение}
end;

{Обработчик события нажатия кнопки Resume}
{Входные параметры:
    Sender: компонента, с которой связано событие}

procedure TForm1.ButtonResumeClick(Sender: TObject);
begin
    Form1.ButtonPause.Enabled:= True;    {Разрешение кнопки Pause}
    Form1.ButtonResume.Enabled:= False;  {Запрещение кнопки Resume}
    SolverThread.Resume; {Пробуждаем поток. Возобновляем выполнение}
end;

{-----}

{Обработчики событий полей исходных данных и результатов}

```

```

{-----}

{Обработчик события редактирования поля числа физических компьютеров}
{Входные параметры:
  Sender: компонента, с которой связано событие}

procedure TForm1.Edit1Change(Sender: TObject);
var
  k: integer;           {временная переменная}
  val_code: integer;    {код возврата процедуры преобразования строки в число}
  old_main_nh: integer; {временная переменная}
  ret_flag: byte;       {возвратный флаг ошибки}
  ret_err: string;      {возвратный текст ошибки}
begin
  if ((Edit1.Text <> '') and (EditFieldsOnChangeHandling)) then
  begin {если текст строки не пустой и обработка разрешена, то}
    val(Edit1.Text, k, val_code); {попытка преобразования в число}
    if (val_code = 0) then {проверка корректности данных}
    begin
      if ((k >= 0) and (k <= nh_max)) then {проверка на нарушение границ}
      begin
        SourceDataIsModified:=1; {установка флага модификац. исх. данных}
        old_main_nh:= main_nh; {запоминаем старое число физ. компьютеров}
        main_nh:= k; {присваиваем новое значение}

        ret_flag:= 0; {сброс возвратного флага}
        {Установка нового числа столбцов для}
        {интерфейсной матрицы ресурсов компьютеров}
        if (ret_flag = 0) then
          HSTGR_SetSize (AXIS_X, main_nh, False, ret_flag, ret_err);

        {Если ошибок при установке новых размеров}
        {для интерфейсных матриц/столбцов исходных данных не было}
        {то сброс текста ошибок для поля Edit1}
        if (ret_flag = 0) then Edit1.Hint:= ''
        else {Иначе если ошибка выделения памяти, то}
          begin {восстановление старого значения числа физ. компьютеров}
            main_nh:= old_main_nh;
            Edit1.Hint:= ret_err; {генерация ошибки}
            Edit1.Text:=''; {Recursive call.} {Сброс поля Edit1}
          end;
        end {в случае нарушения границ - генерация ошибки}
        else Edit1.Hint:= EDIT_RANGE_ERR + '0..' + IntToStr(nh_max);
      end {в случае недействительных данных - генерация ошибки}
      else Edit1.Hint:= EDIT_INVDATA_ERR;
    end;
  end;
end;

{Обработчик события выхода из поля числа физических компьютеров}
{Входные параметры:
  Sender: компонента, с которой связано событие}

procedure TForm1.Edit1Exit(Sender: TObject);
var
  k: integer;           {временная переменная}
  val_code: integer;    {код возврата процедуры преобразования строки в число}
begin
  val(Edit1.Text, k, val_code); {попытка преобразования в число}
  if (val_code = 0) then {проверка корректности данных}
  begin {проверка на нарушение границ}
    if ((k < 0) or (k > nh_max)) then Edit1.SetFocus
    end {В случае если проверки не прошли - то возврат фокуса}
  else Edit1.SetFocus;
end;

```

end;

{Обработчик события редактирования поля числа логических серверов}

{Входные параметры:

Sender: компонента, с которой связано событие}

procedure TForm1.Edit2Change(Sender: TObject);

var

k: integer; {временная переменная}

val\_code: integer; {код возврата процедуры преобразования строки в число}

old\_main\_ns: integer; {временная переменная}

ret\_flag: byte; {возвратный флаг ошибки}

ret\_err: string; {возвратный текст ошибки}

begin

if ((Edit2.Text <> '') and (EditFieldsOnChangeHandling)) then

begin {если текст строки не пустой и обработка разрешена, то}

val(Edit2.Text, k, val\_code); {попытка преобразования в число}

if (val\_code = 0) then {проверка корректности данных}

begin

if ((k >= 0) and (k <= ns\_max)) then {проверка на нарушение границ}

begin

SourceDataIsModified:=1; {установка флага модификац. исх. данных}

old\_main\_ns:= main\_ns; {запоминаем старое число лог. серверов}

main\_ns:= k; {присваиваем новое значение}

ret\_flag:= 0; {сброс возвратного флага}

{Установка нового числа столбцов для}

{интерфейсной матрицы требований лог. серверов}

if (ret\_flag = 0) then

SSTGR\_SetSize (AXIS\_X, main\_ns, False, ret\_flag, ret\_err);

{Установка нового числа столбцов для}

{интерфейсной матрицы исключений для лог. серверов}

if (ret\_flag = 0) then

EXTGR\_SetSize (AXIS\_X, main\_ns, False, ret\_flag, ret\_err);

{Если ошибок при установке новых размеров}

{для интерфейсных матриц/столбцов исходных данных не было}

{то сброс текста ошибок для поля Edit2}

if (ret\_flag = 0) then Edit2.Hint:= ''

else {Иначе если ошибка выделения памяти, то}

begin {восстановление старого значения числа лог. серверов}

main\_ns:= old\_main\_ns;

Edit2.Hint:= ret\_err; {генерация ошибки}

Edit2.Text:=''; {Recursive call...} {Сброс поля Edit2}

end;

end {в случае нарушения границ - генерация ошибки}

else Edit2.Hint:= EDIT\_RANGE\_ERR + '0..' + IntToStr(ns\_max);

end {в случае недействительных данных - генерация ошибки}

else Edit2.Hint:= EDIT\_INVDATA\_ERR;

end;

end;

{Обработчик события выхода из поля числа логических серверов}

{Входные параметры:

Sender: компонента, с которой связано событие}

procedure TForm1.Edit2Exit(Sender: TObject);

var

k: integer; {временная переменная}

val\_code: integer; {код возврата процедуры преобразования строки в число}

begin

val(Edit2.Text, k, val\_code); {попытка преобразования в число}

```

if (val_code = 0) then          {проверка корректности данных}
begin                          {проверка на нарушение границ}
    if ((k < 0) or (k > ns_max)) then Edit2.SetFocus;
end                            {В случае если проверки не прошли - то возврат фокуса}
else Edit2.SetFocus;
end;

{Обработчик события редактирования поля числа типов ресурсов}
{Входные параметры:
    Sender: компонента, с которой связано событие}

procedure TForm1.Edit3Change(Sender: TObject);
var
    k: integer;                {временная переменная}
    val_code: integer;         {код возврата процедуры преобразования строки в число}
    old_main_nc: integer;      {временная переменная}
    ret_flag: byte;            {возвратный флаг ошибки}
    ret_err: string;           {возвратный текст ошибки}
begin
    if ((Edit3.Text <> '') and (EditFieldsOnChangeHandling)) then
        begin {если текст строки не пустой и обработка разрешена, то}
            val(Edit3.Text, k, val_code); {попытка преобразования в число}
            if (val_code = 0) then        {проверка корректности данных}
                begin
                    if ((k >= 0) and (k <= nc_max)) then {проверка на нарушение границ}
                        begin
                            SourceDataIsModified:=1; {установка флага модификац. исх. данных}
                            old_main_nc:= main_nc;   {запоминаем старое число типов ресурсов}
                            main_nc:= k;             {присваиваем новое значение}

                            ret_flag:= 0;            {сброс возвратного флага}
                            {Установка нового числа строк для}
                            {интерфейсной матрицы ресурсов компьютеров}
                            if (ret_flag = 0) then
                                HSTGR_SetSize (AXIS_Y, main_nc, False, ret_flag, ret_err);
                                {Установка нового числа строк для}
                                {интерфейсной матрицы требований лог. серверов}
                            if (ret_flag = 0) then
                                SSTGR_SetSize (AXIS_Y, main_nc, False, ret_flag, ret_err);
                                {Установка нового числа строк для}
                                {интерфейсного столбца требований базовой ОС}
                            if (ret_flag = 0) then
                                BSTGR_SetSize (main_nc, False, ret_flag, ret_err);
                                {Установка нового числа строк для}
                                {интерфейсного столбца маски критериев}
                            if (ret_flag = 0) then
                                CMGR_SetSize (main_nc, False, ret_flag, ret_err);
                                {Установка нового числа строк для}
                                {интерфейсного столбца типов имен ресурсов}
                            if (ret_flag = 0) then
                                RTNGR_SetSize (main_nc, False, ret_flag, ret_err);

                            {Если ошибок при установке новых размеров}
                            {для интерфейсных матриц/столбцов исходных данных не было}
                            {то сброс текста ошибок для поля Edit3}
                            if (ret_flag = 0) then Edit3.Hint:= ''
                        else {Иначе если ошибка выделения памяти, то}
                            begin {восстановление старого значения числа типов ресурсов}
                                main_nc:= old_main_nc;
                                Edit3.Hint:= ret_err; {генерация ошибки}
                                Edit3.Text:=''; {Recursive call...} {Сброс поля Edit3}
                            end;
                        end {в случае нарушения границ - генерация ошибки}
                    end
                end
            end
        end
    end
end

```

```

        else Edit3.Hint:= EDIT_RANGE_ERR + '0..' + IntToStr(nc_max);
    end      {в случае недействительных данных - генерация ошибки}
    else Edit3.Hint:= EDIT_INVDATA_ERR;
end;
end;

```

{Обработчик события выхода из поля числа типов ресурсов}

{Входные параметры:

Sender: компонента, с которой связано событие}

```

procedure TForm1.Edit3Exit(Sender: TObject);

```

```

var

```

```

    k: integer;           {временная переменная}

```

```

    val_code: integer;    {код возврата процедуры преобразования строки в число}

```

```

begin

```

```

    val(Edit3.Text, k, val_code); {попытка преобразования в число}

```

```

    if (val_code = 0) then        {проверка корректности данных}

```

```

    begin                        {проверка на нарушение границ}

```

```

        if ((k < 0) or (k > nc_max)) then Edit3.SetFocus;

```

```

    end      {В случае если проверки не прошли - то возврат фокуса}

```

```

    else Edit3.SetFocus;

```

```

end;

```

{Обработчик события редактирования поля числа исключений}

{Входные параметры:

Sender: компонента, с которой связано событие}

```

procedure TForm1.Edit4Change(Sender: TObject);

```

```

var

```

```

    k: integer;           {временная переменная}

```

```

    val_code: integer;    {код возврата процедуры преобразования строки в число}

```

```

    old_main_nx: integer; {временная переменная}

```

```

    ret_flag: byte;       {возвратный флаг ошибки}

```

```

    ret_err: string;      {возвратный текст ошибки}

```

```

begin

```

```

    if ((Edit4.Text <> '') and (EditFieldsOnChangeHandling)) then

```

```

    begin {если текст строки не пустой и обработка разрешена, то}

```

```

        val(Edit4.Text, k, val_code); {попытка преобразования в число}

```

```

        if (val_code = 0) then        {проверка корректности данных}

```

```

        begin

```

```

            if ((k >= 0) and (k <= nx_max)) then {проверка на нарушение границ}

```

```

            begin

```

```

                SourceDataIsModified:=1; {установка флага модификац. исх. данных}

```

```

                old_main_nx:= main_nx;   {запоминаем старое число исключений}

```

```

                main_nx:= k;             {присваиваем новое значение}

```

```

                ret_flag:= 0;            {сброс возвратного флага}

```

```

                {Установка нового числа строк для}

```

```

                {интерфейсной матрицы исключений для лог. серверов}

```

```

                if (ret_flag = 0) then

```

```

                EXTGR_SetSize (AXIS_Y, main_nx, False, ret_flag, ret_err);

```

```

                {Если ошибок при установке новых размеров}

```

```

                {для интерфейсных матриц/столбцов исходных данных не было}

```

```

                {то сброс текста ошибок для поля Edit3}

```

```

                if (ret_flag = 0) then Edit4.Hint:= ''

```

```

            else {Иначе если ошибка выделения памяти, то}

```

```

                begin {восстановление старого значения числа исключений}

```

```

                    main_nx:= old_main_nx;

```

```

                    Edit4.Hint:= ret_err; {генерация ошибки}

```

```

                    Edit4.Text:=''; {Recursive call...} {Сброс поля Edit4}

```

```

                end;

```



```

        end    {в случае нарушения границ - генерация ошибки}
        else Edit4.Hint:= EDIT_RANGE_ERR + '0..' + IntToStr(nx_max);
        end    {в случае недействительных данных - генерация ошибки}
        else Edit4.Hint:= EDIT_INVDATA_ERR;
    end;
end;

```

{Обработчик события выхода из поля числа исключений}

{Входные параметры:

Sender: компонента, с которой связано событие}

```

procedure TForm1.Edit4Exit(Sender: TObject);
var
    k: integer;           {временная переменная}
    val_code: integer;    {код возврата процедуры преобразования строки в число}
begin
    val(Edit4.Text, k, val_code); {попытка преобразования в число}
    if (val_code = 0) then        {проверка корректности данных}
    begin                        {проверка на нарушение границ}
        if ((k < 0) or (k > nx_max)) then Edit4.SetFocus;
        end                    {В случае если проверки не прошли - то возврат фокуса}
    else Edit4.SetFocus;
end;

```

{Обработчик события редактирования поля max. радиуса поиска}

{Входные параметры:

Sender: компонента, с которой связано событие}

```

procedure TForm1.Edit5Change(Sender: TObject);
var
    k: integer;           {временная переменная}
    val_code: integer;    {код возврата процедуры преобразования строки в число}
begin
    if ((Edit5.Text <> '') and (EditFieldsOnChangeHandling)) then
    begin                    {если текст строки не пустой и обработка разрешена, то}
        val(Edit5.Text, k, val_code); {попытка преобразования в число}
        if (val_code = 0) then        {проверка корректности данных}
        begin
            if ((k >= 0) and (k <= main_ns)) then {проверка на нарушение границ}
            begin
                max_radius:= k;           {присваиваем новое значение}
                SourceDataIsModified:=1; {установка флага модификац. исх. данных}
                Edit5.Hint:= '';           {сброс текста ошибок для поля Edit5}
            end {в случае нарушения границ - генерация ошибки}
            else Edit5.Hint:= EDIT_RANGE_ERR + '0..' + IntToStr(main_ns);
            end {в случае недействительных данных - генерация ошибки}
            else Edit5.Hint:= EDIT_INVDATA_ERR;
        end;
    end;
end;

```

{Обработчик события выхода из поля max. радиуса поиска}

{Входные параметры:

Sender: компонента, с которой связано событие}

```

procedure TForm1.Edit5Exit(Sender: TObject);
var
    k: integer;           {временная переменная}
    val_code: integer;    {код возврата процедуры преобразования строки в число}
begin
    val(Edit5.Text, k, val_code); {попытка преобразования в число}
    if (val_code = 0) then        {проверка корректности данных}

```

```

begin
    {проверка на нарушение границ}
    if ((k < 0) or (k > main_ns)) then Edit5.SetFocus;
end
    {В случае если проверки не прошли - то возврат фокуса}
else Edit5.SetFocus;
end;

{Обработчик события редактирования поля числа старт-точек}
{Входные параметры:
    Sender: компонента, с которой связано событие}

procedure TForm1.Edit6Change(Sender: TObject);
var
    k: integer;           {временная переменная}
    val_code: integer;     {код возврата процедуры преобразования строки в число}
begin
    if ((Edit6.Text <> '') and (EditFieldsOnChangeHandling)) then
    begin
        {если текст строки не пустой и обработка разрешена, то}
        val(Edit6.Text, k, val_code); {попытка преобразования в число}
        if (val_code = 0) then
            {проверка корректности данных}
            begin
                if ((k >= 0) and (k <= main_ns)) then {проверка на нарушение границ}
                begin
                    max_starts:= k;           {присваиваем новое значение}
                    SourceDataIsModified:=1; {установка флага модификац. исх. данных}
                    Edit6.Hint:= '';          {сброс текста ошибок для поля Edit6}
                end
                {в случае нарушения границ - генерация ошибки}
                else Edit6.Hint:= EDIT_RANGE_ERR + '0..' + IntToStr(main_ns);
                end
                {в случае недействительных данных - генерация ошибки}
                else Edit6.Hint:= EDIT_INVDATA_ERR;
            end;
        end;
    end;
end;

{Обработчик события выхода из поля числа старт-точек}
{Входные параметры:
    Sender: компонента, с которой связано событие}

procedure TForm1.Edit6Exit(Sender: TObject);
var
    k: integer;           {временная переменная}
    val_code: integer;     {код возврата процедуры преобразования строки в число}
begin
    val(Edit6.Text, k, val_code); {попытка преобразования в число}
    if (val_code = 0) then
        {проверка корректности данных}
        begin
            {проверка на нарушение границ}
            if ((k < 0) or (k > main_ns)) then Edit6.SetFocus;
            end
            {В случае если проверки не прошли - то возврат фокуса}
        else Edit6.SetFocus;
    end;
end;

{Общий для всех полос прокрутки обработчик перемещения ползунка}
{Входные параметры:
    Sender: компонента, с которой связано событие}

procedure TForm1.ScrollBarChangeRouter(Sender: TObject);
begin
    {В зависимости от полосы прокрутки, чей ползунок изменил положение,
    вызов обновления соответствующей таблицы, оператор Case здесь непригоден}
    if (Sender = Scrollbar1) then StringGrid1Update; {таблицы ресурсов}
    if (Sender = Scrollbar2) then StringGrid1Update; {физ. компьютеров}
    if (Sender = Scrollbar3) then StringGrid2Update; {таблицы требований}
    if (Sender = Scrollbar4) then StringGrid2Update; {лог. серверов}
end;

```

```

if (Sender = Scrollbar5) then StringGrid3Update; {таблицы исключений}
if (Sender = Scrollbar6) then StringGrid3Update; {для лог. серверов}
if (Sender = Scrollbar7) then StringGrid4Update; {таблицы требований баз.ОС}
if (Sender = Scrollbar8) then StringGrid5Update; {таблицы маски критериев}
if (Sender = Scrollbar9) then StringGrid6Update; {таблицы имен типов рес.}
if (Sender = Scrollbar10) then StringGrid7Update; {таблицы результатов}
if (Sender = Scrollbar11) then StringGrid7Update; {распределения}
if (Sender = Scrollbar12) then StringGrid8Update; {таблицы результатов}
if (Sender = Scrollbar13) then StringGrid8Update; {загрузки ресурсов}
end;

```

{Обработчик события редактирования таблицы ресурсов физических компьютеров}

{Входные параметры:

Sender: компонента, с которой связано событие

ACol: столбец в редактируемой ячейки

ARow: строка редактируемой ячейки

Value: содержимое ячейки}

```

procedure TForm1.StringGrid1SetEditText(Sender: TObject; ACol,
  ARow: Integer; const Value: String);
var
  val_code: integer;      {код возврата процедуры преобразования строки в число}
  temp: LongInt;          {временная переменная}
begin
  try                    {проверка допустимых границ индексов редактируемой ячейки}
    if ((ARow >= 1) and (ACol >= 1)
      and (ARow <= HSTGR_Height) and (ACol <= HSTGR_Width))
    then
      begin              {Попытка преобразования текстовой ячейки в число}
        val(StringGrid1.Cells[ACol, ARow], temp, val_code);
        if ((val_code = 0) or (StringGrid1.Cells[ACol, ARow] = ''))
        then            {если ячейка пуста (считается как 0) или данные корректны то}
          begin {установка флага модификации исходных данных}
            if (HSTGR[Scrollbar2.Position + ARow - 1, {Запись данных ячейки}
              ScrollBar1.Position + ACol - 1]           {в соответствующую}
              <> temp) then SourceDataIsModified:= 1;    {ячейку интерфейсной}
            HSTGR[Scrollbar2.Position + ARow - 1,       {матрицы ресурсов}
              ScrollBar1.Position + ACol - 1]:= temp;    {физ. компьютеров}
          end
          else StringGrid1.Cells[ACol,ARow]:= IntToStr {иначе возврат в ячейку}
            (HSTGR[Scrollbar2.Position + ARow - 1,      {таблицы, то что хранит}
              ScrollBar1.Position + ACol - 1]);          {интерфейсная матрица}
        end
        else StringGrid1.Cells[ACol,ARow] := ''; {сброс ячейки если нарушены границы}
      except
        On EAccessViolation do {генерация ошибки в случае нарушения доступа}
          MessageDlg (STRGRID_EDIT_ERR, mtError, [mbOk], 0);
        end;
      end;
    end;
  end;
end;

```

{Обработчик события редактирования таблицы требований логических серверов}

{Входные параметры:

Sender: компонента, с которой связано событие

ACol: столбец в редактируемой ячейки

ARow: строка редактируемой ячейки

Value: содержимое ячейки}

```

procedure TForm1.StringGrid2SetEditText(Sender: TObject; ACol,
  ARow: Integer; const Value: String);
var
  val_code: integer;      {код возврата процедуры преобразования строки в число}
  temp: LongInt;          {временная переменная}

```

```

begin
  try      {проверка допустимых границ индексов редактируемой ячейки}
    if ((ARow >= 1 ) and (ACol >= 1)
    and (ARow <= SSTGR_Height) and (ACol <= SSTGR_Width))
    then
      begin      {Попытка преобразования текстовой ячейки в число}
        val(StringGrid2.Cells[ACol, ARow], temp, val_code);
        if ((val_code = 0) or (StringGrid2.Cells[ACol, ARow] = ''))
        then      {если ячейка пуста (считается как 0) или данные корректны то}
          begin {установка флага модификации исходных данных}
            if SSTGR[ScrollBar4.Position + ARow - 1,      {Запись данных ячейки}
              ScrollBar3.Position + ACol - 1]              {в соответствующую}
              <> temp then SourceDataIsModified:=1;        {ячейку интерфейсной}
              SSTGR[ScrollBar4.Position + ARow - 1,        {матрицы требований}
              ScrollBar3.Position + ACol - 1]:= temp;      {лог. серверов}
            end
          else StringGrid2.Cells[ACol, ARow]:= IntToStr  {иначе возврат в ячейку}
            (SSTGR[ScrollBar4.Position + ARow - 1,        {таблицы, то что хранит}
              ScrollBar3.Position + ACol - 1]);          {интерфейсная матрица}
          end
        else StringGrid2.Cells[ACol, ARow] := ''; {сброс ячейки если нарушены границы}
      except
        On EAccessViolation do {генерация ошибки в случае нарушения доступа}
          MessageDlg (STRGRID_EDIT_ERR, mtError, [mbOk], 0);
        end;
      end;
    end;

{Обработчик события редактирования таблицы исключений для лог. серверов}
{Входные параметры:
  Sender: компонента, с которой связано событие
  ACol:   столбец в редактируемой ячейки
  ARow:   строка редактируемой ячейки
  Value:  содержимое ячейки}

procedure TForm1.StringGrid3SetEditText(Sender: TObject; ACol,
  ARow: Integer; const Value: String);
var
  val_code: integer;      {код возврата процедуры преобразования строки в число}
  temp: Byte;             {временная переменная}
begin
  try      {проверка допустимых границ индексов редактируемой ячейки}
    if ((ARow >= 1 ) and (ACol >= 1)
    and (ARow <= EXTGR_Height) and (ACol <= EXTGR_Width))
    then
      begin      {Попытка преобразования текстовой ячейки в число}
        val(StringGrid3.Cells[ACol, ARow], temp, val_code);
        if (((val_code = 0) and
          ((StringGrid3.Cells[ACol, ARow]='0')
          or (StringGrid3.Cells[ACol, ARow]='1'))
          or (StringGrid3.Cells[ACol, ARow] = ''))
        then      {если ячейка пуста (считается как 0) или "0" или "1", то}
          begin {установка флага модификации исходных данных}
            if (EXTGR[ScrollBar6.Position + ARow - 1,      {Запись данных ячейки}
              ScrollBar5.Position + ACol - 1]              {в соответствующую}
              <> temp) then SourceDataIsModified:=1;        {ячейку интерфейсной}
            EXTGR[ScrollBar6.Position + ARow - 1,          {матрицы исключений}
            ScrollBar5.Position + ACol - 1]:= temp;
          end
        else
          StringGrid3.Cells[ACol, ARow]:= IntToStr  {иначе возврат в ячейку}
            (EXTGR[ScrollBar6.Position + ARow - 1,        {таблицы, то что хранит}
              ScrollBar5.Position + ACol - 1]);          {интерфейсная матрица}
        end
      end;
    end;
  end;
end

```

```

    else StringGrid3.Cells[ACol,ARow]:=''; {сброс ячейки если нарушены границы}
except
    On EAccessViolation do {генерация ошибки в случае нарушения доступа}
    MessageDlg (STRGRID_EDIT_ERR, mtError, [mbOk], 0);
end;
end;
end;

```

{Обработчик события редактирования таблицы требования базовой ОС}

{Входные параметры:

```

    Sender: компонента, с которой связано событие
    ACol:    столбец в редактируемой ячейки
    ARow:    строка редактируемой ячейки
    Value:   содержимое ячейки}

```

```

procedure TForm1.StringGrid4SetEditText(Sender: TObject; ACol,
    ARow: Integer; const Value: String);
var
    val_code: integer;      {код возврата процедуры преобразования строки в число}
    temp: LongInt;          {временная переменная}
begin
    try                    {проверка допустимых границ индексов редактируемой ячейки}
        if ((ARow >= 1) and (ARow <= BSTGR_Height) and (ACol = 1))
        then
            begin          {Попытка преобразования текстовой ячейки в число}
                val(StringGrid4.Cells[ACol, ARow], temp, val_code);
                if ((val_code = 0) or (StringGrid4.Cells[ACol, ARow]=''))
                then        {если ячейка пуста (считается как 0) или данные корректны то}
                    begin {установка флага модификации исходных данных}
                        if (BSTGR[ScrollBar7.Position + ARow - 1] {Запись данных ячейки}
                            <> temp) then SourceDataIsModified:=1; {в соответствующую ячейку}
                        BSTGR[ScrollBar7.Position + ARow - 1]:= temp;
                    end
                        {интерфейсного столбца требований базовой ОС}
                    else
                        StringGrid4.Cells[ACol, ARow]:=IntToStr {иначе возврат в ячейку}
                            (BSTGR[ScrollBar7.Position + ARow - 1]); {таблицы, то что хранит}
                        {интерфейсный столбец}
                    end
                else StringGrid4.Cells[ACol,ARow]:=''; {сброс ячейки если нарушены границы}
            except
                On EAccessViolation do {генерация ошибки в случае нарушения доступа}
                MessageDlg (STRGRID_EDIT_ERR, mtError, [mbOk], 0);
            end;
        end;
    end;
end;

```

{Обработчик события редактирования таблицы маски критериев оптимизации}

{Входные параметры:

```

    Sender: компонента, с которой связано событие
    ACol:    столбец в редактируемой ячейки
    ARow:    строка редактируемой ячейки
    Value:   содержимое ячейки}

```

```

procedure TForm1.StringGrid5SetEditText(Sender: TObject; ACol,
    ARow: Integer; const Value: String);
var
    val_code: integer;      {код возврата процедуры преобразования строки в число}
    temp: Byte;             {временная переменная}
begin
    try                    {проверка допустимых границ индексов редактируемой ячейки}
        if ((ARow >= 1) and (ARow <= CMGR_Height) and (ACol = 1))
        then
            begin          {Попытка преобразования текстовой ячейки в число}
                val(StringGrid5.Cells[ACol, ARow], temp, val_code);
                if (((val_code = 0) and

```

```

        ((StringGrid5.Cells[ACol, ARow]='0')
        or (StringGrid5.Cells[ACol, ARow]='1'))
        or (StringGrid5.Cells[ACol, ARow] = ''))
    then {если ячейка пуста (считается как 0) или "0" или "1", то}
    begin {установка флага модификации исходных данных}
        if (CMGR[ScrollBar8.Position + ARow - 1] {Запись данных ячейки}
        <> temp) then SourceDataIsModified:=1; {в соответствующую ячейку}
        CMGR[ScrollBar8.Position + ARow - 1]:= temp;
    end {интерфейсного столбца маски критериев оптимизации}
    else
        StringGrid5.Cells[ACol, ARow]:= IntToStr {иначе возврат в ячейку}
        (CMGR[ScrollBar8.Position + ARow - 1]); {таблицы, то что хранит}
    end {интерфейсный столбец}
    else StringGrid5.Cells[ACol,ARow]:=''; {сброс ячейки если нарушены границы}
except
    On EAccessViolation do {генерация ошибки в случае нарушения доступа}
    MessageDlg (STRGRID_EDIT_ERR, mtError, [mbOk], 0);
end;
end;

```

{Обработчик события редактирования таблицы имен типов ресурсов}

{Входные параметры:

```

    Sender: компонента, с которой связано событие
    ACol: столбец в редактируемой ячейки
    ARow: строка редактируемой ячейки
    Value: содержимое ячейки}

```

```

procedure TForm1.StringGrid6SetEditText(Sender: TObject; ACol,
    ARow: Integer; const Value: String);
begin
    try {проверка допустимых границ индексов редактируемой ячейки}
        if ((ARow >=1 ) and (ARow <= RTNGR_Height) and (ACol = 1))
        then
            begin {установка флага модификации исходных данных}
                if (RTNGR[ScrollBar9.Position + ARow - 1] <> {Запись данных ячейки}
                StringGrid6.Cells[ACol, ARow]) then SourceDataIsModified:=1;
                RTNGR[ScrollBar9.Position + ARow - 1]:= {в соответствующую ячейку}
                StringGrid6.Cells[ACol, ARow]; {интерфейсного столбца типов ресурсов}
            end
        else StringGrid6.Cells[ACol,ARow]:=''; {сброс ячейки если нарушены границы}
    except
        On EAccessViolation do {генерация ошибки в случае нарушения доступа}
        MessageDlg (STRGRID_EDIT_ERR, mtError, [mbOk], 0);
    end;
end;

```

{Общий для всех таблиц текстовых данных обработчик нажатия клавиши}

{Данная процедура обрабатывает нажатия на клавиши стрелок влево, вправо,

вверх, вниз и клавиши предыдущая страница, следующая страница}

{Нажатия косвенно воздействуют на содержимое таблиц: программным способом изменяется положение ползунка соответствующей полосы прокрутки, что влечет немедленный вызов обработчика события изменения положения ползунка, который, в свою очередь, через общий обработчик вызывает процедуру обновления таблицы}

{Входные параметры:

```

    Sender: компонента, с которой связано событие
    Key: код (виртуальный код) клавиши
    Shift: состояния кнопок мыши и клавиш Alt, Shift, Ctrl}

```

```

procedure TForm1.StringGridsKeyDownRouter
    (Sender: TObject; var Key: Word; Shift: TShiftState);
var

```

```

{Временная переменная для таблицы, вызвавшей событие}
SenderGrid: TStringGrid;
{Временная переменная гор. полосы прокрутки таблицы}
ScrollBarX: TScrollBar;
{Временная переменная верт. полосы прокрутки таблицы}
ScrollBarY: TScrollBar;
begin
  {сброс временных указателей}
  SenderGrid:= nil; ScrollBarX:= nil; ScrollBarY:= nil;

  {В зависимости от таблицы, вызвавшей событие, заполняем временные}
  {переменные указателями на таблицу и ее гор. и верт. полосы прокруток}
  if (Sender = StringGrid1) then begin SenderGrid:= StringGrid1; {таблица}
    ScrollBarX:= ScrollBar1; ScrollBarY:= ScrollBar2; end; {физ. компьютеров}
  if (Sender = StringGrid2) then begin SenderGrid:= StringGrid2; {таблица}
    ScrollBarX:= ScrollBar3; ScrollBarY:= ScrollBar4; end; {лог. серверов}
  if (Sender = StringGrid3) then begin SenderGrid:= StringGrid3; {таблица}
    ScrollBarX:= ScrollBar5; ScrollBarY:= ScrollBar6; end; {исключений}
  if (Sender = StringGrid4) then begin SenderGrid:= StringGrid4; {столбец}
    ScrollBarX:= nil; ScrollBarY:= ScrollBar7; end; {требований базовой ОС}
  if (Sender = StringGrid5) then begin SenderGrid:= StringGrid5; {столбец}
    ScrollBarX:= nil; ScrollBarY:= ScrollBar8; end; {маски критериев}
  if (Sender = StringGrid6) then begin SenderGrid:= StringGrid6; {столбец}
    ScrollBarX:= nil; ScrollBarY:= ScrollBar9; end; {имен типов ресурсов}
  if (Sender = StringGrid7) then begin SenderGrid:= StringGrid7; {таблица}
    ScrollBarX:= ScrollBar11; ScrollBarY:= ScrollBar10; end; {распределения}
  if (Sender = StringGrid8) then begin SenderGrid:= StringGrid8; {таблица}
    ScrollBarX:= ScrollBar13; ScrollBarY:= ScrollBar12; end; {загр. ресурсов}

  if (SenderGrid <> nil) then {Проверка на пустоту указателя}
  begin
    case Key of
      {Если на нажата клавиша "вправо"}
      VK_Right: if (ScrollBarX <> nil)
        {То перемещение гор. ползунок вправо на 1, если возможно}
        then if (SenderGrid.Col = SenderGrid.ColCount - 1)
          then if (ScrollBarX.Position < ScrollBarX.Max)
            then ScrollBarX.Position:= ScrollBarX.Position + 1;
      {Если на нажата клавиша "влево"}
      VK_Left: if (ScrollBarX <> nil)
        {То перемещение гор. ползунок влево на 1, если возможно}
        then if SenderGrid.Col=1
          then if (ScrollBarX.Position > ScrollBarX.Min)
            then ScrollBarX.Position:= ScrollBarX.Position - 1;
      {Если на нажата клавиша "вниз"}
      {То перемещение вер. ползунок вниз на 1, если возможно}
      VK_Down: if (ScrollBarY <> nil)
        then if (SenderGrid.Row = SenderGrid.RowCount - 1)
          then if (ScrollBarY.Position < ScrollBarY.Max)
            then ScrollBarY.Position:= ScrollBarY.Position + 1;
      {Если на нажата клавиша "вверх"}
      {То перемещение вер. ползунок вверх на 1, если возможно}
      VK_Up: if (ScrollBarY <> nil)
        then if SenderGrid.Row=1
          then if (ScrollBarY.Position > ScrollBarY.Min)
            then ScrollBarY.Position:= ScrollBarY.Position - 1;
      {Если на нажата клавиша "следующая страница"}
      {То перемещение вер. ползунок вниз на страницу, если возможно}
      VK_Next: if (ScrollBarY <> nil)
        then if (SenderGrid.Row = SenderGrid.RowCount - 1)
          then if (ScrollBarY.Position <
            ScrollBarY.Max - (SenderGrid.RowCount - 1))
            then ScrollBarY.Position:=
              ScrollBarY.Position + (SenderGrid.RowCount - 1)

```

```

        else ScrollBarY.Position:= ScrollBarY.Max;
    {Если на нажата клавиша "предыдущая страница"}
    {То перемещение вер. ползунка вверх на страницу, если возможно}
    VK_Prior: if (ScrollBarY <> nil)
        then if SenderGrid.Row=1
            then if (ScrollBarY.Position >
                ScrollBarY.Min + (SenderGrid.RowCount - 1))
            then ScrollBarY.Position:=
                ScrollBarY.Position - (SenderGrid.RowCount - 1)
            else ScrollBarY.Position:= ScrollBarY.Min;
        end;
    end;
    {сброс временных указателей}
    SenderGrid:= nil; ScrollBarX:= nil; ScrollBarY:= nil;
end;

```

{Общий для всех таблиц данных обработчик движения колесика мыши вверх}  
 {Движение колесика косвенно воздействуют на содержимое таблиц: программным способом изменяется положение ползунка соответствующей полосы прокрутки, что влечет немедленный вызов обработчика события изменения положения ползунка, который, в свою очередь, через общий обработчик вызывает обновление таблицы}

{Входные параметры:

```

    Sender:      компонента, с которой связано событие
    MousePos:    позиция курсора мыши
    Shift:       состояния кнопок мыши и клавиш Alt, Shift, Ctrl}

```

```

procedure TForm1.StringGridsMouseWheelUpRouter(Sender: TObject;
    Shift: TShiftState; MousePos: TPoint; var Handled: Boolean);

```

```

var

```

```

    {Временная переменная для таблицы, вызвавшей событие}
    SenderGrid: TStringGrid;
    {Временная переменная гор. полосы прокрутки таблицы}
    ScrollBarX: TScrollBar;
    {Временная переменная верт. полосы прокрутки таблицы}
    ScrollBarY: TScrollBar;

```

```

begin

```

```

    {сброс временных указателей}
    SenderGrid:= nil; ScrollBarX:= nil; ScrollBarY:= nil;

```

```

    {В зависимости от таблицы, вызвавшей событие, заполняем временные}
    {переменные указателями на таблицу и ее гор. и верт. полосы прокруток}
    if (Sender = StringGrid1) then begin SenderGrid:= StringGrid1; {таблица}
        ScrollBarX:= ScrollBar1; ScrollBarY:= ScrollBar2; end; {физ. компьютеров}
    if (Sender = StringGrid2) then begin SenderGrid:= StringGrid2; {таблица}
        ScrollBarX:= ScrollBar3; ScrollBarY:= ScrollBar4; end; {лог. серверов}
    if (Sender = StringGrid3) then begin SenderGrid:= StringGrid3; {таблица}
        ScrollBarX:= ScrollBar5; ScrollBarY:= ScrollBar6; end; {исключений}
    if (Sender = StringGrid4) then begin SenderGrid:= StringGrid4; {столбец}
        ScrollBarX:= nil; ScrollBarY:= ScrollBar7; end; {требований базовой ОС}
    if (Sender = StringGrid5) then begin SenderGrid:= StringGrid5; {столбец}
        ScrollBarX:= nil; ScrollBarY:= ScrollBar8; end; {маски критериев}
    if (Sender = StringGrid6) then begin SenderGrid:= StringGrid6; {столбец}
        ScrollBarX:= nil; ScrollBarY:= ScrollBar9; end; {имен типов ресурсов}
    if (Sender = StringGrid7) then begin SenderGrid:= StringGrid7; {таблица}
        ScrollBarX:= ScrollBar11; ScrollBarY:= ScrollBar10; end; {распределения}
    if (Sender = StringGrid8) then begin SenderGrid:= StringGrid8; {таблица}
        ScrollBarX:= ScrollBar13; ScrollBarY:= ScrollBar12; end; {загр. ресурсов}

```

```

    if (SenderGrid <> nil) then {Проверка на пустоту указателя}
        begin {перемещение вер. ползунка вверх на 1, если возможно}
            if (ScrollBarY <> nil)
                then if SenderGrid.Row=1

```



```

    then if (ScrollBarY.Position > ScrollBarY.Min)
    then ScrollBarY.Position:= ScrollBarY.Position - 1;
end;
{сброс временных указателей}
SenderGrid:= nil; ScrollBarX:= nil; ScrollBarY:= nil;
end;

{Общий для всех таблиц данных обработчик движения колесика мыши вниз}
{Движение колесика косвенно воздействуют на содержимое таблиц: программным
способом изменяется положение ползунка соответствующей полосы прокрутки, что
влечет немедленный вызов обработчика события изменения положения ползунка,
который, в свою очередь, через общий обработчик вызывает обновление таблицы}

{Входные параметры:
  Sender:      компонента, с которой связано событие
  MousePos:    позиция курсора мыши
  Shift:       состояния кнопок мыши и клавиш Alt, Shift, Ctrl}

procedure TForm1.StringGridsMouseWheelDownRouter(Sender: TObject;
  Shift: TShiftState; MousePos: TPoint; var Handled: Boolean);
var
  {Временная переменная для таблицы, вызвавшей событие}
  SenderGrid: TStringGrid;
  {Временная переменная гор. полосы прокрутки таблицы}
  ScrollBarX: TScrollBar;
  {Временная переменная верт. полосы прокрутки таблицы}
  ScrollBarY: TScrollBar;
begin
  {сброс временных указателей}
  SenderGrid:= nil; ScrollBarX:= nil; ScrollBarY:= nil;

  {В зависимости от таблицы, вызвавшей событие, заполняем временные}
  {переменные указателями на таблицу и ее гор. и верт. полосы прокруток}
  if (Sender = StringGrid1) then begin SenderGrid:= StringGrid1; {таблица}
    ScrollBarX:= ScrollBar1; ScrollBarY:= ScrollBar2; end; {физ. компьютеров}
  if (Sender = StringGrid2) then begin SenderGrid:= StringGrid2; {таблица}
    ScrollBarX:= ScrollBar3; ScrollBarY:= ScrollBar4; end; {лог. серверов}
  if (Sender = StringGrid3) then begin SenderGrid:= StringGrid3; {таблица}
    ScrollBarX:= ScrollBar5; ScrollBarY:= ScrollBar6; end; {исключений}
  if (Sender = StringGrid4) then begin SenderGrid:= StringGrid4; {столбец}
    ScrollBarX:= nil; ScrollBarY:= ScrollBar7; end; {требований базовой ОС}
  if (Sender = StringGrid5) then begin SenderGrid:= StringGrid5; {столбец}
    ScrollBarX:= nil; ScrollBarY:= ScrollBar8; end; {маски критериев}
  if (Sender = StringGrid6) then begin SenderGrid:= StringGrid6; {столбец}
    ScrollBarX:= nil; ScrollBarY:= ScrollBar9; end; {имен типов ресурсов}
  if (Sender = StringGrid7) then begin SenderGrid:= StringGrid7; {таблица}
    ScrollBarX:= ScrollBar11; ScrollBarY:= ScrollBar10; end; {распределения}
  if (Sender = StringGrid8) then begin SenderGrid:= StringGrid8; {таблица}
    ScrollBarX:= ScrollBar13; ScrollBarY:= ScrollBar12; end; {загр. ресурсов}

  if (SenderGrid <> nil) then {Проверка на пустоту указателя}
  begin {перемещение вер. ползунка вниз на 1, если возможно}
    if (ScrollBarY <> nil)
    then if (SenderGrid.Row = SenderGrid.RowCount - 1)
    then if (ScrollBarY.Position < ScrollBarY.Max)
    then ScrollBarY.Position:= ScrollBarY.Position + 1;
    end;
    {сброс временных указателей}
    SenderGrid:= nil; ScrollBarX:= nil; ScrollBarY:= nil;
  end;
end;

{-----}

```

```

{Процедуры уничтожения исходных данных и результатов }
{разрешения/запрещения редактирования исходных данных}

{-----}

{Процедура уничтожения исходных данных, освобождения памяти от интерфейсных}
{матриц и столбцов исходных данных и установки в 0 - 0 границ перемещения}
{для их полос прокруток. Процедура опирается на автоматический вызов}
{обработчиков события изменения для Edit1-Edit4, поэтому до вызова процедуры}
{обработчики должны быть уже привязаны к соответствующим полям редактирования}

procedure TForm1.FlushSourceData;
var
  ret_flag: byte;          {возвратный флаг ошибки}
  ret_err: string;         {возвратный текст ошибки}
begin
  ret_flag:= 0;            {сброс возвратного флага ошибки}

  {Обнуление максимально достигнутых значений размеров}
  {интерфейсных матриц и столбцов исходных данных}
  HSTGR_MaxWidth:= 0; SSTGR_MaxWidth:= 0; EXTGR_MaxWidth:= 0;
  HSTGR_MaxHeight:= 0; SSTGR_MaxHeight:= 0; EXTGR_MaxHeight:= 0;
  BSTGR_MaxHeight:= 0; CMGR_MaxHeight:= 0; RTNGR_MaxHeight:= 0;

  {Обнуление размеров интерфейсных матриц и столбцов исходных данных}
  HSTGR_SetSize (AXIS_X, 0, True, ret_flag, ret_err);
  HSTGR_SetSize (AXIS_Y, 0, True, ret_flag, ret_err);
  SSTGR_SetSize (AXIS_X, 0, True, ret_flag, ret_err);
  SSTGR_SetSize (AXIS_Y, 0, True, ret_flag, ret_err);
  EXTGR_SetSize (AXIS_X, 0, True, ret_flag, ret_err);
  EXTGR_SetSize (AXIS_Y, 0, True, ret_flag, ret_err);
  BSTGR_SetSize (0, True, ret_flag, ret_err);
  CMGR_SetSize (0, True, ret_flag, ret_err);
  RTNGR_SetSize (0, True, ret_flag, ret_err);

  {Сброс параметров размерностей для исходных данных задачи}
  main_nh:= 0; main_ns:= 0; main_nc:= 0; main_nx:= 0;
  {сброс параметров управления поиском}
  max_radius:= 0; max_starts:= 0;

  {Запрещение обработки события изменения полей редактирования}
  EditFieldsOnChangeHandling:= False;

  {Сброс в 0 полей редактирования Edit1-Edit6}
  Edit1.Text:='0'; Edit2.Text:='0'; Edit3.Text:='0';
  Edit4.Text:='0'; Edit5.Text:='0'; Edit6.Text:='0';

  {Разрешение обработки события изменения полей редактирования}
  EditFieldsOnChangeHandling:= True;

  {Сброс флага модификация исходных данных}
  SourceDataIsModified:= 0;
end;

{Процедура уничтожения результатов, освобождения памяти от интерфейсных}
{матриц и строк результатов и установки в 0 - 0 границ перемещения}
{для их полос прокруток}

procedure TForm1.FlushResultData;
var
  ret_flag: byte;          {возвратный флаг ошибки}
  ret_err: string;         {возвратный текст ошибки}
begin

```

```

ret_flag:= 0;           {сброс возвратного флага ошибки}

{Обнуление максимально достигнутых значений размеров}
{интерфейсных матриц и строк результатов}
DTGR_MaxWidth:= 0; DTGR_MaxHeight:= 0;
RLGR_MaxWidth:= 0; RLGR_MaxHeight:= 0;

{Обнуление размеров интерфейсных матриц и строк результатов}
DTGR_SetSize (AXIS_X, 0, True, ret_flag, ret_err);
DTGR_SetSize (AXIS_Y, 0, True, ret_flag, ret_err);
RLGR_SetSize (AXIS_X, 0, True, ret_flag, ret_err);
RLGR_SetSize (AXIS_Y, 0, True, ret_flag, ret_err);

{Сброс параметров размерностей для результатов задачи}
res_nh:= 0; res_ns:= 0; res_nc:= 0; res_nx:= 0;

{Сброс флага обновления результатов}
ResultDataIsModified:= 0;
end;

{Процедура запрещения редактирования исходных данных}

procedure TForm1.DisableEditSourceData;
begin
    {Запрещение редактирования полей Edit1-6}
    Edit1.ReadOnly:= True; Edit2.ReadOnly:= True;
    Edit3.ReadOnly:= True; Edit4.ReadOnly:= True;
    Edit5.ReadOnly:= True; Edit6.ReadOnly:= True;

    {Выключение автоперехода в режим редактирования для таблиц StringGrid1-6}
    StringGrid1.EditorMode:= False; StringGrid2.EditorMode:= False;
    StringGrid3.EditorMode:= False; StringGrid4.EditorMode:= False;
    StringGrid5.EditorMode:= False; StringGrid6.EditorMode:= False;

    {Запрещение редактирования таблиц StringGrid1-6}
    StringGrid1.Options:= [goFixedVertLine, goFixedHorzLine,
                           goVertLine, goHorzLine];
    StringGrid2.Options:= [goFixedVertLine, goFixedHorzLine,
                           goVertLine, goHorzLine];
    StringGrid3.Options:= [goFixedVertLine, goFixedHorzLine,
                           goVertLine, goHorzLine];
    StringGrid4.Options:= [goFixedVertLine, goFixedHorzLine,
                           goVertLine, goHorzLine];
    StringGrid5.Options:= [goFixedVertLine, goFixedHorzLine,
                           goVertLine, goHorzLine];
    StringGrid6.Options:= [goFixedVertLine, goFixedHorzLine,
                           goVertLine, goHorzLine];
end;

{Процедура разрешения редактирования исходных данных}

procedure TForm1.EnableEditSourceData;
begin
    {Разрешение редактирования полей Edit1-6 и таблиц StringGrid1-6}
    Edit1.ReadOnly:= False; Edit2.ReadOnly:= False;
    Edit3.ReadOnly:= False; Edit4.ReadOnly:= False;
    Edit5.ReadOnly:= False; Edit6.ReadOnly:= False;

    {Включение автоперехода в режим редактирования для таблиц StringGrid1-6}
    StringGrid1.EditorMode:= True; StringGrid2.EditorMode:= True;
    StringGrid3.EditorMode:= True; StringGrid4.EditorMode:= True;
    StringGrid5.EditorMode:= True; StringGrid6.EditorMode:= True;

```

```

{Разрешение редактирования таблиц StringGrid1-6}
StringGrid1.Options:= [goFixedVertLine,      goFixedHorzLine, goVertLine,
                      goAlwaysShowEditor, goHorzLine, goEditing];
StringGrid2.Options:= [goFixedVertLine,      goFixedHorzLine, goVertLine,
                      goAlwaysShowEditor, goHorzLine, goEditing];
StringGrid3.Options:= [goFixedVertLine,      goFixedHorzLine, goVertLine,
                      goAlwaysShowEditor, goHorzLine, goEditing];
StringGrid4.Options:= [goFixedVertLine,      goFixedHorzLine, goVertLine,
                      goAlwaysShowEditor, goHorzLine, goEditing];
StringGrid5.Options:= [goFixedVertLine,      goFixedHorzLine, goVertLine,
                      goAlwaysShowEditor, goHorzLine, goEditing];
StringGrid6.Options:= [goFixedVertLine,      goFixedHorzLine, goVertLine,
                      goAlwaysShowEditor, goHorzLine, goEditing];

end;

{-----}

{Процедуры обновления таблиц исходных данных и результатов}

{-----}

{Процедура обновления содержимого таблицы ресурсов физических компьютеров}

procedure TForm1.StringGrid1Update;
var
  i, j:integer;      {временные переменные}
  kx, ky: integer;   {временные переменные}
begin
  {Установка верхней границы индексных переменных, границы выбираются}
  {как минимум из размеров интерфейсной матрицы и размеров таблицы}
  try
    if (HSTGR_Height > StringGrid1.RowCount - 1)
    then ky:= StringGrid1.RowCount - 1 else ky:= HSTGR_Height;
    if (HSTGR_Width > StringGrid1.ColCount - 1)
    then kx:= StringGrid1.ColCount - 1 else kx:= HSTGR_Width;

    {Очистка таблицы}
    for i:=0 to StringGrid1.RowCount - 1
    do for j:=0 to StringGrid1.ColCount - 1
    do StringGrid1.Cells[j,i]:='';

    {Запись заголовков для строк таблицы}
    for i:=1 to ky do StringGrid1.Cells[0,i]:='R'
    + IntToStr(ScrollBar2.Position + i);
    {Запись заголовков для столбцов}
    for j:=1 to kx do StringGrid1.Cells[j,0]:='H'
    + IntToStr(ScrollBar1.Position + j);

    {Импорт данных из интерфейсной матрицы ресурсов физ. компьютеров}
    for i:=1 to ky do for j:=1 to kx do StringGrid1.Cells[j,i]:=
    IntToStr(HSTGR[ScrollBar2.Position + i - 1,
    ScrollBar1.Position + j - 1]);
  except {в случае нарушения доступа генерация ошибки}
    On EAccessViolation do
      MessageDlg (STRGRID_UPDATE_ERR, mtError, [mbOk], 0);
  end;
end;

{Процедура обновления содержимого таблицы требований логических серверов}

procedure TForm1.StringGrid2Update;
var

```

```

i, j:integer;      {временные переменные}
kx, ky: integer;   {временные переменные}
begin
  try
    {Установка верхней границы индексных переменных, границы выбираются}
    {как минимум из размеров интерфейсной матрицы и размеров таблицы}
    if (SSTGR_Height > StringGrid2.RowCount - 1)
    then ky:= StringGrid2.RowCount - 1 else ky:= SSTGR_Height;
    if (SSTGR_Width > StringGrid2.ColCount - 1)
    then kx:= StringGrid2.ColCount - 1 else kx:= SSTGR_Width;

    {Очистка таблицы}
    for i:=0 to StringGrid2.RowCount - 1
    do for j:=0 to StringGrid2.ColCount - 1
    do StringGrid2.Cells[j,i]:='';

    {Запись заголовков для строк таблицы}
    for i:=1 to ky do StringGrid2.Cells[0,i]:='R'
    + IntToStr(ScrollBar4.Position + i);

    {Запись заголовков для столбцов}
    for j:=1 to kx do StringGrid2.Cells[j,0]:='S'
    + IntToStr(ScrollBar3.Position + j);

    {Импорт данных из интерфейсной матрицы требований лог. серверов}
    for i:=1 to ky do for j:=1 to kx do StringGrid2.Cells[j,i]:=
    IntToStr(SSTGR[ScrollBar4.Position + i - 1,
    ScrollBar3.Position + j - 1]);
  except {в случае нарушения доступа генерация ошибки}
  On EAccessViolation do
    MessageDlg (STRGRID_UPDATE_ERR, mtError, [mbOk], 0);
  end;
end;

```

{Процедура обновления содержимого таблицы исключений для лог. серверов}

```

procedure TForm1.StringGrid3Update;
var
  i, j:integer;      {временные переменные}
  kx, ky: integer;   {временные переменные}
begin
  try
    {Установка верхней границы индексных переменных, границы выбираются}
    {как минимум из размеров интерфейсной матрицы и размеров таблицы}
    if (EXTGR_Height > StringGrid3.RowCount - 1)
    then ky:= StringGrid3.RowCount - 1 else ky:= EXTGR_Height;
    if (EXTGR_Width > StringGrid3.ColCount - 1)
    then kx:= StringGrid3.ColCount - 1 else kx:= EXTGR_Width;

    {Очистка таблицы}
    for i:=0 to StringGrid3.RowCount - 1
    do for j:=0 to StringGrid3.ColCount - 1
    do StringGrid3.Cells[j,i]:='';

    {Запись заголовков для строк таблицы}
    for i:=1 to ky do StringGrid3.Cells[0,i]:='E'
    + IntToStr(ScrollBar6.Position + i);

    {Запись заголовков для столбцов}
    for j:=1 to kx do StringGrid3.Cells[j,0]:='S'
    + IntToStr(ScrollBar5.Position + j);

    {Импорт данных из интерфейсной матрицы исключений}
  
```

```

    for i:=1 to ky do for j:=1 to kx do StringGrid3.Cells[j,i]:=
    IntToStr(EXTGR[ScrollBar6.Position + i - 1,
    ScrollBar5.Position + j - 1]);
except {в случае нарушения доступа генерация ошибки}
    On EAccessViolation do
        MessageDlg (STRGRID_UPDATE_ERR, mtError, [mbOk], 0);
end;
end;

{Процедура обновления содержимого таблицы требования базовой ОС}

procedure TForm1.StringGrid4Update;
var
    i, j:integer;      {временные переменные}
    ky: integer;        {временные переменные}
begin
    try
        {Установка верхней границы индексных переменных, границы выбираются}
        {как минимум из размеров интерфейсной матрицы и размеров таблицы}
        if (BSTGR_Height > StringGrid4.RowCount - 1)
        then ky:= StringGrid4.RowCount - 1 else ky:= BSTGR_Height;

        {Очистка таблицы}
        for i:=0 to StringGrid4.RowCount - 1
        do for j:=0 to StringGrid4.ColCount - 1
        do StringGrid4.Cells[j,i]:='';

        {Запись заголовков для строк таблицы}
        for i:=1 to ky do StringGrid4.Cells[0,i]:='R'
        + IntToStr(ScrollBar7.Position + i);

        {Импорт данных из интерфейсного столбца требований базовой ОС}
        for i:=1 to ky do StringGrid4.Cells[1,i]:=
        IntToStr(BSTGR[ScrollBar7.Position + i - 1]);
    except {в случае нарушения доступа генерация ошибки}
        On EAccessViolation do
            MessageDlg (STRGRID_UPDATE_ERR, mtError, [mbOk], 0);
        end;
    end;
end;

{Процедура обновления содержимого таблицы маски критериев оптимизации}

procedure TForm1.StringGrid5Update;
var
    i, j:integer;      {временные переменные}
    ky: integer;        {временные переменные}
begin
    try
        {Установка верхней границы индексных переменных, границы выбираются}
        {как минимум из размеров интерфейсной матрицы и размеров таблицы}
        if (CMGR_Height > StringGrid5.RowCount - 1)
        then ky:= StringGrid5.RowCount - 1 else ky:= CMGR_Height;

        {Очистка таблицы}
        for i:=0 to StringGrid5.RowCount - 1
        do for j:=0 to StringGrid5.ColCount - 1
        do StringGrid5.Cells[j,i]:='';

        {Запись заголовков для строк таблицы}
        for i:=1 to ky do StringGrid5.Cells[0,i]:='R'
        + IntToStr(ScrollBar8.Position + i);

```

```

    {Импорт данных из интерфейсного столбца критериев маски оптимизации}
    for i:=1 to ky do StringGrid5.Cells[1,i]:=
    IntToStr(CMGR[ScrollBar8.Position + i - 1]);
except {в случае нарушения доступа генерация ошибки}
    On EAccessViolation do
        MessageDlg (STRGRID_UPDATE_ERR, mtError, [mbOk], 0);
end;
end;

{Процедура обновления содержимого таблицы имен типов ресурсов}

procedure TForm1.StringGrid6Update;
var
    i, j: integer;      {временные переменные}
    ky: integer;        {временные переменные}
begin
    try
        {Установка верхней границы индексных переменных, границы выбираются}
        {как минимум из размерности интерфейсной матрицы и размерности таблицы}
        if (RTNGR_Height > StringGrid6.RowCount - 1)
        then ky:= StringGrid6.RowCount - 1 else ky:= RTNGR_Height;

        {Очистка таблицы}
        for i:=0 to StringGrid6.RowCount - 1
        do for j:=0 to StringGrid6.ColCount - 1
        do StringGrid6.Cells[j,i]:='';

        {Запись заголовков для строк таблицы}
        for i:=1 to ky do StringGrid6.Cells[0,i]:='R'
        + IntToStr(ScrollBar9.Position + i);

        {Импорт данных из интерфейсного столбца имен типов ресурсов}
        for i:=1 to ky do StringGrid6.Cells[1,i]:=
        RTNGR[ScrollBar9.Position + i - 1];
    except {в случае нарушения доступа генерация ошибки}
        On EAccessViolation do
            MessageDlg (STRGRID_UPDATE_ERR, mtError, [mbOk], 0);
        end;
    end;
end;

{Процедура обновления содержимого результирующей таблицы распределения}

procedure TForm1.StringGrid7Update;
var
    i, j: integer;      {временные переменные}
    kx, ky: integer;    {временные переменные}
begin
    try
        {Установка верхней границы индексных переменных, границы выбираются}
        {как минимум из размерности интерфейсной матрицы и размерности таблицы}
        if (DTGR_Height > StringGrid7.RowCount - 1)
        then ky:= StringGrid7.RowCount - 1 else ky:= DTGR_Height;
        if (DTGR_Width > StringGrid7.ColCount - 1)
        then kx:= StringGrid7.ColCount - 1 else kx:= DTGR_Width;

        {Очистка таблицы}
        for i:=0 to StringGrid7.RowCount - 1
        do for j:=0 to StringGrid7.ColCount - 1
        do StringGrid7.Cells[j,i]:='';

        {Запись заголовка для 0-й строки}
        if ((DTGR_Height > 0) and (DTGR_Width > 0)) then
            StringGrid7.Cells[0,0]:= 'Remain S:';
    end;
end;

```

```

{Запись заголовков для строк таблицы}
for i:=1 to ky do StringGrid7.Cells[0,i]:='Н'
+ IntToStr(ScrollBar10.Position + i);

{Запись заголовков столбцов для таблицы, сюда импортируются}
{данные из интерфейсной строки нераспределенных лог. серверов}
for j:=1 to kx do {фактически 0-я строка - дополнительная строка данных}
begin
  if (DTHL[ScrollBar11.Position + j - 1] = 0) {В случае если 0}
  then StringGrid7.Cells[j,0]:='' {выводится пустой символ}
  else StringGrid7.Cells[j,0]:='S'
  + IntToStr(DTHL[ScrollBar11.Position + j - 1]); {Иначе "S" + <индекс>}
end;

{Импорт данных из интерфейсной матрицы распределения}
for i:=1 to ky do for j:=1 to kx do
begin
  if (DTGR[ScrollBar10.Position + i - 1,
  ScrollBar11.Position + j - 1] = 0) {В случае если 0}
  then StringGrid7.Cells[j,i]:=' ' {выводится пустой символ}
  else
    StringGrid7.Cells[j,i]:='S' + {Иначе "S" + <индекс>}
    IntToStr(DTGR[ScrollBar10.Position + i - 1,
    ScrollBar11.Position + j - 1]);
end;
except {в случае нарушения доступа генерация ошибки}
On EAccessViolation do
  MessageDlg (STRGRID_UPDATE_ERR, mtError, [mbOk], 0);
end;
end;

{Процедура обновления содержимого результирующей таблицы загрузки ресурсов}

procedure TForm1.StringGrid8Update;
var
  i, j:integer; {временные переменные}
  kx, ky: integer; {временные переменные}
begin
  try
    {Установка верхней границы индексных переменных, границы выбираются}
    {как минимум из размеров интерфейсной матрицы и размеров таблицы}
    if (RLGR_Height > StringGrid8.RowCount - 1)
    then ky:= StringGrid8.RowCount - 1 else ky:= RLGR_Height;
    if (RLGR_Width > StringGrid8.ColCount - 1)
    then kx:= StringGrid8.ColCount - 1 else kx:= RLGR_Width;

    {Очистка таблицы}
    for i:=0 to StringGrid8.RowCount - 1
    do for j:=0 to StringGrid8.ColCount - 1
    do StringGrid8.Cells[j,i]:=''; {Очистка}

    {Запись заголовка для 0-й строки}
    if ((RLGR_Height > 0) and (RLGR_Width > 0)) then
    StringGrid8.Cells[0,0]:='Resource: ';

    {Запись заголовков для строк таблицы}
    for i:=1 to ky do StringGrid8.Cells[0,i]:='Н' +
    IntToStr(ScrollBar12.Position + i);

    {Запись заголовков столбцов для таблицы, сюда импортируются}
    {данные из интерфейсной строки имен типов ресурсов}
    {фактически 0-я строка - дополнительная строка данных}

```



```

for j:=1 to kx do StringGrid8.Cells[j,0]:=
RLHL[ScrollBar13.Position + j - 1];

{Импорт данных из интерфейсной матрицы загрузки ресурсов}
for i:=1 to ky do for j:=1 to kx do StringGrid8.Cells[j,i]:=
FloatToStrF(RLGR[ScrollBar12.Position + i - 1,
ScrollBar13.Position + j - 1],ffFixed,5,2)+'%';
except {в случае нарушения доступа генерация ошибки}
On EAccessViolation do
MessageDlg (STRGRID_UPDATE_ERR, mtError, [mbOk], 0);
end;
end;

{-----}

{Процедуры установки размеров таблиц исходных данных и результатов}

{-----}

{Процедура установки размера для одной из осей
интерфейсной матрицы ресурсов физ. компьютеров}
{Входные параметры:
axis:      Ось, для которой устанавливается размер:
            AXIS_X (число столбцов) или AXIS_Y (число строк)
newsize:   Размер для указанной в axis оси
ignore_max: Игнорирование ранее достигнутого макс.размера}
{Выходные параметры:
p_return_flag:  Флаг ошибки
p_return_err:   Текст ошибки}

procedure TForm1.HSTGR_SetSize;
var
  err: byte;          {локальный флаг ошибки}
begin
  err:= 0;            {сброс флага ошибки}
  p_return_flag:= 0;  {сброс выходного флага ошибки}
  p_return_err:= '';  {сброс выходного текста ошибки}

  if (axis = AXIS_X) then {если Ось X, число столбцов}
  begin
    {если новый размер больше чем ранее достигнутый наибольший размер}
    if (newsize > HSTGR_MaxWidth) or (IGNORE_MAX) then
    begin
      {Тестовая попытка выделения памяти
      для интерфейсной матрицы ресурсов физ. компьютеров}
      if (not(TryMemoryAllocate
      (2*(sizeof(HSTGR[0,0]) * HSTGR_MaxHeight * newsize))))
      then err:= 1;

      if (err = 0) then {если тестовое выделение памяти}
      begin
        {прошло успешно то расширение}
        try {интерфейсной матрицы ресурсов физ. компьютеров}
          SetLength (HSTGR, HSTGR_MaxHeight, newsize);
        except {Дополнительная защита}
          On EOutOfMemory do err:=1;
        end;
        {на случай ошибки расширения}
      end;
      {Если ошибок не было, новый размер становится}
      {текущим достигнутым наибольшим размером}
      if (err = 0) then HSTGR_MaxWidth:= newsize;
    end;

    if (err = 0) then {Если расширение прошло успешно, то}
    begin

```

```

        {Установка нового размера, числа столбцов для}
        {интерфейсной матрицы ресурсов физ. компьютеров}
        HSTGR_Width:= newsize;
        {Установка границ горизонтальной полосы прокрутки}
        {для таблицы ресурсов физ. компьютеров}
        ScrollBar1.Min:=0;
        if (newsize > StringGrid1.ColCount - 1) then
            ScrollBar1.Max:= newsize - (StringGrid1.ColCount - 1)
        else ScrollBar1.Max:=0;
        ScrollBar1.Position:=0; {Установка ползунка в 0-е положение}
        StringGrid1Update;      {Обновление таблицы}
    end;
end;

if (axis = AXIS_Y) then      {если Ось Y, число строк}
begin
    {если новый размер больше чем ранее достигнутый наибольший размер}
    if (newsize > HSTGR_MaxHeight) or (IGNORE_MAX) then
    begin
        {Тестовая попытка выделения памяти
        для интерфейсной матрицы ресурсов физ. компьютеров}
        if (not(TryMemoryAllocate
        (2*(sizeof(HSTGR[0,0]) * newsize * HSTGR_MaxWidth))))
        then err:= 1;

        if (err = 0) then      {если тестовое выделение памяти}
        begin                  {прошло успешно то расширение}
            try                  {интерфейсной матрицы ресурсов физ. компьютеров}
                SetLength (HSTGR, newsize, HSTGR_MaxWidth);
            except                {Дополнительная защита}
                On EOutOfMemory do err:=1;
            end;                 {на случай ошибки расширения}
        end;

        {Если ошибок не было, новый размер становится}
        {текущим достигнутым наибольшим размером}
        if (err = 0) then HSTGR_MaxHeight:= newsize;
    end;

    if (err = 0) then {Если расширение прошло успешно, то}
    begin
        {Установка нового размера, числа строк для}
        {интерфейсной матрицы ресурсов физ. компьютеров}
        HSTGR_Height:= newsize;
        {Установка границ вертикальной полосы прокрутки}
        {для таблицы ресурсов физ. компьютеров}
        ScrollBar2.Min:=0;
        if (newsize > StringGrid1.RowCount - 1) then
            ScrollBar2.Max:= newsize - (StringGrid1.RowCount - 1)
        else ScrollBar2.Max:=0;
        ScrollBar2.Position:=0; {Установка ползунка в 0-е положение}
        StringGrid1Update;      {Обновление таблицы}
    end;
end;

{В случае ошибки при попытке установки нового размера}
{генерация выходной ошибки и соответствующего текста ошибки}
if (err <> 0) then
begin
    p_return_flag:= 1;
    p_return_err:= GRID_RESIZE_ERR + SN[7,2];
end;
end;

```

```

{Процедура установки размера для одной из осей
интерфейсной матрицы требований лог. серверов}
{Входные параметры:
  axis:      Ось, для которой устанавливается размер:
              AXIS_X (число столбцов) или AXIS_Y (число строк)
  newsize:    Размер для указанной в axis оси
  ignore_max: Игнорирование ранее достигнутого макс.размера}
{Выходные параметры:
  p_return_flag:  Флаг ошибки
  p_return_err:   Текст ошибки}

procedure TForm1.SSTGR_SetSize;
var
  err: byte;          {локальный флаг ошибки}
begin
  err:= 0;            {сброс флага ошибки}
  p_return_flag:= 0;  {сброс выходного флага ошибки}
  p_return_err:= '';  {сброс выходного текста ошибки}

  if (axis = AXIS_X) then    {если Ось X, число столбцов}
  begin
    {если новый размер больше чем ранее достигнутый наибольший размер}
    if (newsize > SSTGR_MaxWidth) or (IGNORE_MAX) then
    begin
      {Тестовая попытка выделения памяти
      для интерфейсной матрицы требований лог. серверов}
      if (not(TryMemoryAllocate
        (2*(sizeof(SSTGR[0,0]) * SSTGR_MaxHeight * newsize))))
      then err:= 1;

      if (err = 0) then    {если тестовое выделение памяти}
      begin              {прошло успешно то расширение}
        try              {интерфейсной матрицы требований лог. серверов}
          SetLength (SSTGR, SSTGR_MaxHeight, newsize);
        except           {Дополнительная защита}
          On EOutOfMemory do err:=1;
        end;             {на случай ошибки расширения}
      end;
      {Если ошибок не было, новый размер становится}
      {текущим достигнутым наибольшим размером}
      if (err = 0) then SSTGR_MaxWidth:= newsize;
    end;

    if (err = 0) then      {Если расширение прошло успешно, то}
    begin
      {Установка нового размера, числа столбцов для}
      {интерфейсной матрицы требований лог. серверов}
      SSTGR_Width:= newsize;
      {Установка границ горизонтальной полосы прокрутки}
      {для таблицы требований лог. серверов}
      ScrollBar3.Min:=0;
      if (newsize > StringGrid2.ColCount - 1) then
        ScrollBar3.Max:= newsize - (StringGrid2.ColCount - 1)
      else ScrollBar3.Max:=0;
      ScrollBar3.Position:=0; {Установка ползунка в 0-е положение}
      StringGrid2.Update;    {Обновление таблицы}
    end;
  end;

  if (axis = AXIS_Y) then    {если Ось Y, число строк}
  begin
    {если новый размер больше чем ранее достигнутый наибольший размер}
    if (newsize > SSTGR_MaxHeight) or (IGNORE_MAX) then
    begin

```

```

{Тестовая попытка выделения памяти
для интерфейсной матрицы требований лог. серверов}
if (not(TryMemoryAllocate
(2*(sizeof(SSTGR[0,0]) * newsize * SSTGR_MaxWidth))))
then err:= 1;

if (err = 0) then      {если тестовое выделение памяти}
begin                {прошло успешно то расширение}
  try                {интерфейсной матрицы требований лог. серверов}
    SetLength (SSTGR, newsize, SSTGR_MaxWidth);
  except              {Дополнительная защита}
    On EOutOfMemory do err:=1;
  end;                {на случай ошибки расширения}
end;
{Если ошибок не было, новый размер становится}
{текущим достигнутым наибольшим размером}
if (err = 0) then SSTGR_MaxHeight:= newsize;
end;

if (err = 0) then      {Если расширение прошло успешно, то}
begin
  {Установка нового размера, числа строк для}
  {интерфейсной матрицы требований лог. серверов}
  SSTGR_Height:= newsize;
  {Установка границ вертикальной полосы прокрутки}
  {для таблицы требований лог. серверов}
  ScrollBar4.Min:=0;
  if (newsize > StringGrid2.RowCount - 1) then
    ScrollBar4.Max:= newsize - (StringGrid2.RowCount - 1)
  else ScrollBar4.Max:=0;
  ScrollBar4.Position:=0; {Установка ползунка в 0-е положение}
  StringGrid2.Update;     {Обновление таблицы}
end;
end;

{В случае ошибки при попытке установки нового размера}
{генерация выходной ошибки и соответствующего текста ошибки}
if (err <> 0) then
begin
  p_return_flag:= 1;
  p_return_err:= GRID_RESIZE_ERR + SN[8,2];
end;
end;

{Процедура установки размера для одной из осей
интерфейсной матрицы исключений для лог. серверов}
{Входные параметры:
  axis:      Ось, для которой устанавливается размер:
              AXIS_X (число столбцов) или AXIS_Y (число строк)
  newsize:    Размер для указанной в axis оси
  ignore_max: Игнорирование ранее достигнутого макс.размера}
{Выходные параметры:
  p_return_flag:  Флаг ошибки
  p_return_err:   Текст ошибки}

procedure TForm1.EXTGR_SetSize;
var
  err: byte;          {локальный флаг ошибки}
begin
  err:= 0;            {сброс флага ошибки}
  p_return_flag:= 0;  {сброс выходного флага ошибки}
  p_return_err:= '';  {сброс выходного текста ошибки}

```

```

if (axis = AXIS_X) then      {если Ось X, число столбцов}
begin
  {если новый размер больше чем ранее достигнутый наибольший размер}
  if (newsize > EXTGR_MaxWidth) or (IGNORE_MAX) then
  begin
    {Тестовая попытка выделения памяти
    для интерфейсной матрицы исключений для лог. серверов}
    if (not(TryMemoryAllocate
    (2*(sizeof(EXTGR[0,0]) * EXTGR_MaxHeight * newsize))))
    then err:= 1;

    if (err = 0) then      {если тестовое выделение памяти}
    begin                  {прошло успешно то расширение}
      try      {интерфейсной матрицы исключений для лог. серверов}
        SetLength (EXTGR, EXTGR_MaxHeight, newsize);
      except    {Дополнительная защита}
        On EOutOfMemory do err:=1;
      end;      {на случай ошибки расширения}
    end;
    {Если ошибок не было, новый размер становится}
    {текущим достигнутым наибольшим размером}
    if (err = 0) then EXTGR_MaxWidth:= newsize;
  end;

  if (err = 0) then      {Если расширение прошло успешно, то}
  begin
    {Установка нового размера, числа столбцов для}
    {интерфейсной матрицы исключений для лог. серверов}
    EXTGR_Width:= newsize;
    {Установка границ горизонтальной полосы прокрутки}
    {для таблицы исключений для лог. серверов}
    ScrollBar5.Min:=0;
    if (newsize > StringGrid3.ColCount - 1) then
      ScrollBar5.Max:= newsize - (StringGrid3.ColCount - 1)
    else ScrollBar5.Max:=0;
    ScrollBar5.Position:=0; {Установка ползунка в 0-е положение}
    StringGrid3.Update;    {Обновление таблицы}
  end;
end;

if (axis = AXIS_Y) then      {если Ось Y, число строк}
begin
  {если новый размер больше чем ранее достигнутый наибольший размер}
  if (newsize > EXTGR_MaxHeight) or (IGNORE_MAX) then
  begin
    {Тестовая попытка выделения памяти
    для интерфейсной матрицы исключений для лог. серверов}
    if (not(TryMemoryAllocate
    (2*(sizeof(EXTGR[0,0]) * newsize * EXTGR_MaxWidth))))
    then err:= 1;

    if (err = 0) then      {если тестовое выделение памяти}
    begin                  {прошло успешно то расширение}
      try      {интерфейсной матрицы исключений для лог. серверов}
        SetLength (EXTGR, newsize, EXTGR_MaxWidth);
      except    {Дополнительная защита}
        On EOutOfMemory do err:=1;
      end;      {на случай ошибки расширения}
    end;
    {Если ошибок не было, новый размер становится}
    {текущим достигнутым наибольшим размером}
    if (err = 0) then EXTGR_MaxHeight:= newsize;
  end;
end;

```

```

if (err = 0) then      {Если расширение прошло успешно, то}
begin
    {Установка нового размера, числа строк для}
    {интерфейсной матрицы исключений для лог. серверов}
    EXTGR_Height:= newsize;
    {Установка границ вертикальной полосы прокрутки}
    {для таблицы исключений для лог. серверов}
    ScrollBar6.Min:=0;
    if (newsize > StringGrid3.RowCount - 1) then
        ScrollBar6.Max:= newsize - (StringGrid3.RowCount - 1)
    else ScrollBar6.Max:=0;
    ScrollBar6.Position:=0; {Установка ползунка в 0-е положение}
    StringGrid3Update;      {Обновление таблицы}
end;
end;

{В случае ошибки при попытке установки нового размера}
{генерация выходной ошибки и соответствующего текста ошибки}
if (err <> 0) then
begin
    p_return_flag:= 1;
    p_return_err:= GRID_RESIZE_ERR + SN[9,2];
end;
end;

{Процедура установки размера интерфейсного столбца требований базовой ОС}
{Входные параметры:
    newsize:      Размер (число строк)
    ignore_max:   Игнорирование ранее достигнутого макс.размера}
{Выходные параметры:
    p_return_flag:  Флаг ошибки
    p_return_err:   Текст ошибки}

procedure TForm1.BSTGR_SetSize;
var
    err: byte;      {локальный флаг ошибки}
begin
    err:= 0;        {сброс флага ошибки}
    p_return_flag:= 0; {сброс выходного флага ошибки}
    p_return_err:= ''; {сброс выходного текста ошибки}

    {если новый размер больше чем ранее достигнутый наибольший размер}
    if (newsize > BSTGR_MaxHeight) or (IGNORE_MAX) then
    begin
        {Тестовая попытка выделения памяти
        для интерфейсного столбца требований базовой ОС}
        if (not (TryMemoryAllocate(2*(sizeof(BSTGR[0]) * newsize))))
        then err:= 1;

        if (err = 0) then      {если тестовое выделение памяти}
        begin                  {прошло успешно то расширение}
            try {интерфейсного столбца требований базовой ОС}
                SetLength (BSTGR, newsize);
            except              {Дополнительная защита}
                On EOutOfMemory do err:=1;
            end;                {на случай ошибки расширения}
        end;

        {Если ошибок не было, новый размер становится}
        {текущим достигнутым наибольшим размером}
        if (err = 0) then BSTGR_MaxHeight:= newsize;
    end;

    if (err = 0) then          {Если расширение прошло успешно, то}

```

```

begin
    {Установка нового размера, числа строк для}
    {интерфейсного столбца требований базовой ОС}
    BSTGR_Height:= newsize;
    {Установка границ вертикальной полосы прокрутки}
    {для таблицы требований базовой ОС}
    ScrollBar7.Min:=0;
    if (newsize > StringGrid4.RowCount - 1) then
        ScrollBar7.Max:= newsize - (StringGrid4.RowCount - 1)
    else ScrollBar7.Max:=0;
    ScrollBar7.Position:=0; {Установка ползунка в 0-е положение}
    StringGrid4Update;      {Обновление таблицы}
end;

{В случае ошибки при попытке установки нового размера}
{генерация выходной ошибки и соответствующего текста ошибки}
if (err <> 0) then
begin
    p_return_flag:= 1;
    p_return_err:= GRID_RESIZE_ERR + SN[10,2];
end;
end;

{Процедура установки размера интерфейсного столбца маски критериев}
{Входные параметры:
    newsize:      Размер (число строк)
    ignore_max:   Игнорирование ранее достигнутого макс.размера}
{Выходные параметры:
    p_return_flag: Флаг ошибки
    p_return_err:  Текст ошибки}

procedure TForm1.CMGR_SetSize;
var
    err: byte;          {локальный флаг ошибки}
begin
    err:= 0;            {сброс флага ошибки}
    p_return_flag:= 0;   {сброс выходного флага ошибки}
    p_return_err:= '';   {сброс выходного текста ошибки}

    {если новый размер больше чем ранее достигнутый наибольший размер}
    if (newsize > CMGR_MaxHeight) or (IGNORE_MAX) then
    begin
        {Тестовая попытка выделения памяти
        для интерфейсного столбца маски критериев}
        if (not(TryMemoryAllocate(2*(sizeof(CMGR[0]) * newsize))))
        then err:= 1;

        if (err = 0) then {если тестовое выделение памяти}
        begin            {прошло успешно то расширение}
            try          {интерфейсного столбца маски критериев}
                SetLength (CMGR, newsize);
            except        {Дополнительная защита}
                On EOutOfMemory do err:=1;
            end;          {на случай ошибки расширения}
        end;
        {Если ошибок не было, новый размер становится}
        {текущим достигнутым наибольшим размером}
        if (err = 0) then CMGR_MaxHeight:= newsize;
    end;

    if (err = 0) then    {Если расширение прошло успешно, то}
    begin
        {Установка нового размера, числа строк для}

```

```

    {интерфейсного столбца маски критериев}
    CMGR_Height:= newsize;
    {Установка границ вертикальной полосы прокрутки}
    {для таблицы маски критериев}
    ScrollBar8.Min:=0;
    if (newsize > StringGrid5.RowCount - 1) then
        ScrollBar8.Max:= newsize - (StringGrid5.RowCount - 1)
    else ScrollBar8.Max:=0;
    ScrollBar8.Position:=0; {Установка ползунка в 0-е положение}
    StringGrid5Update;      {Обновление таблицы}
end;

{В случае ошибки при попытке установки нового размера}
{генерация выходной ошибки и соответствующего текста ошибки}
if (err <> 0) then
begin
    p_return_flag:= 1;
    p_return_err:= GRID_RESIZE_ERR + SN[11,2];
end;
end;

{Процедура установки размера интерфейсного столбца имен типов ресурсов}
{Входные параметры:
    newsize:      Размер (число строк)
    ignore_max:   Игнорирование ранее достигнутого макс.размера}
{Выходные параметры:
    p_return_flag: Флаг ошибки
    p_return_err:  Текст ошибки}

procedure TForm1.RTNGR_SetSize;
var
    err: byte;          {локальный флаг ошибки}
begin
    err:= 0;             {сброс флага ошибки}
    p_return_flag:= 0;   {сброс выходного флага ошибки}
    p_return_err:= '';   {сброс выходного текста ошибки}

    {если новый размер больше чем ранее достигнутый наибольший размер}
    if (newsize > RTNGR_MaxHeight) or (IGNORE_MAX) then
    begin
        {Тестовая попытка выделения памяти
        для интерфейсного столбца имен типов ресурсов}
        if (not(TryMemoryAllocate(2*(sizeof(RTNGR[0]) * newsize))))
        then err:= 1;

        if (err = 0) then {если тестовое выделение памяти}
        begin           {прошло успешно то расширение}
            try {интерфейсного столбца имен типов ресурсов}
                SetLength (RTNGR, newsize);
            except {Дополнительная защита}
                On EOutOfMemory do err:= 1;
            end;     {на случай ошибки расширения}
        end;
        {Если ошибок не было, новый размер становится}
        {текущим достигнутым наибольшим размером}
        if (err = 0) then RTNGR_MaxHeight:= newsize;
    end;

    if (err = 0) then {Если расширение прошло успешно, то}
    begin
        {Установка нового размера, числа строк для}
        {интерфейсного столбца имен типов ресурсов}
        RTNGR_Height:= newsize;
    end;
end;

```



```

    {Установка границ вертикальной полосы прокрутки}
    {для таблицы имен типов ресурсов}
    ScrollBar9.Min:=0;
    if (newsize > StringGrid6.RowCount - 1) then
        ScrollBar9.Max:= newsize - (StringGrid6.RowCount - 1)
    else ScrollBar9.Max:=0;
    ScrollBar9.Position:=0; {Установка ползунка в 0-е положение}
    StringGrid6.Update;      {Обновление таблицы}
end;

{В случае ошибки при попытке установки нового размера}
{генерация выходной ошибки и соответствующего текста ошибки}
if (err <> 0) then
begin
    p_return_flag:= 1;
    p_return_err:= GRID_RESIZE_ERR + SN[12,2];
end;
end;

{Процедура установки размера для одной из осей
интерфейсной матрицы результата распределения}
{Входные параметры:
    axis:      Ось, для которой устанавливается размер:
                AXIS_X (число столбцов) или AXIS_Y (число строк)
    newsize:    Размер для указанной в axis оси
    ignore_max: Игнорирование ранее достигнутого макс.размера}
{Выходные параметры:
    p_return_flag:  Флаг ошибки
    p_return_err:   Текст ошибки}

procedure TForm1.DTGR_SetSize;
var
    err: byte;          {локальный флаг ошибки}
begin
    err:= 0;            {сброс флага ошибки}
    p_return_flag:= 0;   {сброс выходного флага ошибки}
    p_return_err:= '';   {сброс выходного текста ошибки}

    if (axis = AXIS_X) then {если Ось X, число столбцов}
    begin
        {если новый размер больше чем ранее достигнутый наибольший размер}
        if (newsize > DTGR_MaxWidth) or (IGNORE_MAX) then
            begin
                {Тестовая попытка выделения памяти
                для интерфейсной матрицы результатов распределения}
                if (not(TryMemoryAllocate
                (2*(sizeof(DTGR[0,0]) * DTGR_MaxHeight * newsize))))
                then err:= 1;

                {Тестовая попытка выделения памяти
                для интерфейсной строки нераспределенных лог. серверов}
                if (not(TryMemoryAllocate
                (2*(sizeof(Form1.DTHL[0]) * newsize))))
                then err:= 1;

                if (err = 0) then {если тестовое выделение памяти}
                begin           {прошло успешно то расширение}
                    try {интерфейсной матрицы результата распределения}
                        SetLength (DTGR, DTGR_MaxHeight, newsize);
                        SetLength (DTHL, newsize);
                    except {Дополнительная защита}
                        On EOutOfMemory do err:=1;
                    end;    {на случай ошибки расширения}
                end;
            end;
        end;
    end;

```

```

        end;
        {Если ошибок не было, новый размер становится}
        {текущим достигнутым наибольшим размером}
        if (err = 0) then DTGR_MaxWidth:= newsize;
    end;

    if (err = 0) then      {Если расширение прошло успешно, то}
    begin
        {Установка нового размера, числа столбцов для}
        {интерфейсной матрицы результата распределения}
        DTGR_Width:= newsize;
        {Установка границ горизонтальной полосы прокрутки}
        {для таблицы результата распределения}
        ScrollBar11.Min:=0;
        if (newsize > StringGrid7.ColCount - 1) then
            ScrollBar11.Max:= newsize - (StringGrid7.ColCount - 1)
        else ScrollBar11.Max:=0;
        ScrollBar11.Position:=0; {Установка ползунка в 0-е положение}
        StringGrid7.Update;      {Обновление таблицы}
    end;
end;

if (axis = AXIS_Y) then      {если Ось Y, число строк}
begin
    {если новый размер больше чем ранее достигнутый наибольший размер}
    if (newsize > DTGR_MaxHeight) or (IGNORE_MAX) then
    begin
        {Тестовая попытка выделения памяти}
        {для интерфейсной матрицы результатов распределения}
        if (not(TryMemoryAllocate
            (2*(sizeof(DTGR[0,0]) * newsize * DTGR_MaxWidth))))
        then err:= 1;

        if (err = 0) then      {если тестовое выделение памяти}
        begin                  {прошло успешно то расширение}
            try {интерфейсной матрицы результата распределения}
                SetLength (DTGR, newsize, DTGR_MaxWidth);
            except {Дополнительная защита}
                On EOutOfMemory do err:=1;
            end;
            {на случай ошибки расширения}
        end;
        {Если ошибок не было, новый размер становится}
        {текущим достигнутым наибольшим размером}
        if (err = 0) then DTGR_MaxHeight:= newsize;
    end;

    if (err = 0) then      {Если расширение прошло успешно, то}
    begin
        {Установка нового размера, числа строк для}
        {интерфейсной матрицы результата распределения}
        DTGR_Height:= newsize;
        {Установка границ вертикальной полосы прокрутки}
        {для таблицы результата распределения}
        ScrollBar10.Min:=0;
        if (newsize > StringGrid7.RowCount - 1) then
            ScrollBar10.Max:= newsize - (StringGrid7.RowCount - 1)
        else ScrollBar10.Max:=0;
        ScrollBar10.Position:=0; {Установка ползунка в 0-е положение}
        StringGrid7.Update;      {Обновление таблицы}
    end;
end;

{В случае ошибки при попытке установки нового размера}
{генерация выходной ошибки и соответствующего текста ошибки}

```

```

if (err <> 0) then
begin
  p_return_flag:= 1;
  p_return_err:= GRID_RESIZE_ERR + SN[13,2];
end;
end;

{Процедура установки размера для одной из осей
интерфейсной матрицы загрузки ресурсов}
{Входные параметры:
  axis:      Ось, для которой устанавливается размер:
              AXIS_X (число столбцов) или AXIS_Y (число строк)
  newsize:    Размер для указанной в axis оси
  ignore_max: Игнорирование ранее достигнутого макс.размера}
{Выходные параметры:
  p_return_flag:  Флаг ошибки
  p_return_err:   Текст ошибки}

procedure TForm1.RLGR_SetSize;
var
  err: byte;          {локальный флаг ошибки}
begin
  err:= 0;            {сброс флага ошибки}
  p_return_flag:= 0;   {сброс выходного флага ошибки}
  p_return_err:= '';   {сброс выходного текста ошибки}

  if (axis = AXIS_X) then    {если Ось X, число столбцов}
  begin
    {если новый размер больше чем ранее достигнутый наибольший размер}
    if (newsize > RLGR_MaxWidth) or (IGNORE_MAX) then
    begin
      {Тестовая попытка выделения памяти
      для интерфейсной матрицы загрузки ресурсов}
      if (not(TryMemoryAllocate
        (2*(sizeof(RLGR[0,0]) * RLGR_MaxHeight * newsize))))
      then err:= 1;

      {Тестовая попытка выделения памяти
      для интерфейсной строки имен типов ресурсов}
      if (not(TryMemoryAllocate
        (2*(sizeof(Form1.RLHL[0]) * newsize))))
      then err:= 1;

      if (err = 0) then    {если тестовое выделение памяти}
      begin              {прошло успешно то расширение}
        try {интерфейсной матрицы загрузки ресурсов}
          SetLength (RLGR, RLGR_MaxHeight, newsize);
          SetLength (RLHL, newsize);
        except           {Дополнительная защита}
          On EOutOfMemory do err:=1;
        end;             {на случай ошибки расширения}
      end;
      {Если ошибок не было, новый размер становится}
      {текущим достигнутым наибольшим размером}
      if (err = 0) then RLGR_MaxWidth:= newsize;
    end;

    if (err = 0) then    {Если расширение прошло успешно, то}
    begin
      {Установка нового размера, числа столбцов для}
      {интерфейсной матрицы загрузки ресурсов}
      RLGR_Width:= newsize;
      {Установка границ горизонтальной полосы прокрутки}
    end;
  end;
end;

```

```

        {для таблицы загрузки ресурсов}
        ScrollBar13.Min:=0;
        if (newsize > StringGrid8.ColCount - 1) then
            ScrollBar13.Max:= newsize - (StringGrid8.ColCount - 1)
        else ScrollBar13.Max:=0;
        ScrollBar13.Position:=0; {Установка ползунка в 0-е положение}
        StringGrid8Update;      {Обновление таблицы}
    end;
end;

if (axis = AXIS_Y) then      {если Ось Y, число строк}
begin
    {если новый размер больше чем ранее достигнутый наибольший размер}
    if (newsize > RLGR_MaxHeight) or (IGNORE_MAX) then
        begin
            {Тестовая попытка выделения памяти
            для интерфейсной матрицы загрузки ресурсов}
            if (not(TryMemoryAllocate
            (2*(sizeof(RLGR[0,0]) * newsize * RLGR_MaxWidth))))
            then err:= 1;

            if (err = 0) then      {если тестовое выделение памяти}
                begin            {прошло успешно то расширение}
                    try          {интерфейсной матрицы загрузки ресурсов}
                        SetLength (RLGR, newsize, RLGR_MaxWidth);
                    except        {Дополнительная защита}
                        On EOutOfMemory do err:=1;
                    end;          {на случай ошибки расширения}
                end;
            {Если ошибок не было, новый размер становится}
            {текущим достигнутым наибольшим размером}
            if (err = 0) then RLGR_MaxHeight:= newsize;
        end;

        if (err = 0) then      {Если расширение прошло успешно, то}
        begin
            {Установка нового размера, числа строк для}
            {интерфейсной матрицы загрузки ресурсов}
            RLGR_Height:= newsize;
            {Установка границ вертикальной полосы прокрутки}
            {для таблицы загрузки ресурсов}
            ScrollBar12.Min:=0;
            if (newsize > StringGrid8.RowCount - 1) then
                ScrollBar12.Max:= newsize - (StringGrid8.RowCount - 1)
            else ScrollBar12.Max:=0;
            ScrollBar12.Position:=0; {Установка ползунка в 0-е положение}
            StringGrid8Update;      {Обновление таблицы}
        end;
    end;

    {В случае ошибки при попытке установки нового размера}
    {генерация выходной ошибки и соответствующего текста ошибки}
    if (err <> 0) then
        begin
            p_return_flag:= 1;
            p_return_err:= GRID_RESIZE_ERR + SN[14,2];
        end;
end;

{-----}

{Дополнительные вспомогательные функции и процедуры}

{-----}

```

```
{Функция "тестовой попытки выделения памяти", используется перед тем}
{как использовать функцию SetLength для расширения интерфейсных}
{матриц/столбцов исходных данных или матриц/строк результатов}
{поскольку в случае нехватки памяти SetLength разрушает данные}
{в случае успешного тестового выделения памяти, память сразу освобождается}
{и функция возвращает значение ИСТИНА, иначе - значение ЛОЖЬ}
```

```
function TForm1.TryMemoryAllocate;
var
  err: byte;           {локальный флаг ошибки}
  mem_check: pointer;  {временный указатель}
begin
  err:= 0;             {сброс флага ошибки}
  try                 {то специальная попытка выделения памяти}
    mem_check:= nil;   {т.к. операция SetLength в случае}
    GetMem (mem_check, MemSize + 1); {нехватки памяти разрушает все данные}
  except              {в случае нехватки памяти генерация ошибки}
    On EOutOfMemory do err:= 1;
  end;

  if (mem_check <> nil) then {если временный указатель не пуст}
  begin                    {то освобождение памяти}
    FreeMem (mem_check, MemSize + 1);
    mem_check:= nil;
  end
  else err:= 1;           {Иначе генерация ошибки}

  {в случае успешного тестового выделения памяти}
  {функция возвращает значение ИСТИНА, иначе - значение ЛОЖЬ}
  if (err = 0) then Result:= True else Result:= False;
end;
```

```
{Функция "защиты от потери данных" - проверки фактов модификации исходных}
{данных и обновления результатов, в положительном случае - вывод}
{соответствующих диалогов на сохранение и выполнение операций сохранения.}
{Функция возвращает значение ИСТИНА, только если данные не изменялись,}
{или пользователь отказался от сохранения или данные успешно сохранились}
```

```
function TForm1.SafeTaskAndResultsSaving: boolean;
var
  wdlg1_ret: byte; {результат диалога запроса на сохранение исх.данных}
  wdlg2_ret: byte; {результат диалога запроса на сохранение результатов}

begin
  wdlg1_ret:= 0; {сброс результата диалога запроса на сохранение исх.данных}
  wdlg2_ret:= 0; {сброс результата диалога запроса на сохранение результатов}
  Result:=False; {Изначально код возврата - ложь}

  {проверка факта модификации исходных данных}
  if (SourceDataIsModified = 1) then
  {Если исх. данные модифицированы, то вывод диалога запроса на сохранение}
  wdlg1_ret:=MessageDlg (SRC_MODIFY_WARN, mtWarning, [mbYes, mbNo, mbCancel], 0);
  {Если "Да" - то вызов диалога выбора файла сохранения исх. данных, или}
  {сохранение в текущий рабочий файл, если он <> 'untitled.dat'}
  if wdlg1_ret = mrYes then if (workfile = 'untitled.dat')
  then ItemSaveAsClick(Form1) else ItemSaveClick(Form1);

  {проверка факта обновления результатов}
  if (ResultDataIsModified = 1) then
  {Если результаты обновлены, то вывод диалога запроса на сохранение}
  wdlg2_ret:=MessageDlg (RES_MODIFY_WARN, mtWarning, [mbYes, mbNo, mbCancel], 0);
  {Если "Да" - то вызов диалога выбора файла сохранения результатов}
```

```

{сохранение в текущий файл результатов, если он <> 'untitled.res'}
if (wdlg2_ret = mrYes) then if (resfile = 'untitled.res')
then ItemResSaveAsClick(Form1) else ItemResSaveClick(Form1);

{Если исх. данные не модифицировались (wdlg1_ret остался равным нулю)}
{или данные модифицировались, но пользователь нажал No, или нажал Yes}
{и данные успешно сохранились (флаг ошибки последнего сохранения = 0)}
if ((wdlg1_ret = 0) or (wdlg1_ret = mrNo) or
    ((wdlg1_ret = mrYes) and (LastSrcSaveErrorFlag = 0))) then
{То, если результаты не обновлялись (wdlg2_ret остался равным нулю)}
{или данные модифицировались, но пользователь нажал No, или нажал Yes}
{и данные успешно сохранились (флаг ошибки последнего сохранения = 0)}
    if ((wdlg2_ret = 0) or (wdlg2_ret = mrNo) or
        ((wdlg2_ret = mrYes) and (LastResSaveErrorFlag = 0)))
    then Result:=True; {То, код возврата - истина}
end;

{Процедура загрузки исходных данных из файла}
{Входные параметры:
  p_filename: полный путь к файлу}
{Выходные параметры:
  p_return_flag: возвратный флаг ошибки
  p_return_err: возвратный текст ошибки}
procedure TForm1.LoadTaskFromFile;
var
  MyTask: TCustomTask;      {экземпляр класса вспомогательной задачи}
  err_flag: byte;           {локальный флаг ошибок}
  ret_flag: byte;           {возвратный флаг ошибки метода вспом. задачи}
  ret_err: string;          {возвратный текст ошибки метода вспом. задачи}

begin
  err_flag:= 0;              {сброс локального флага ошибок}
  p_return_flag:= 0;         {сброс возвратного флага ошибки процедуры}
  p_return_err:= '';         {сброс возвратного текста ошибки процедуры}

  {вывод в секцию 0 строки состояния информации}
  {о выполняемых действиях основного потока}
  AppStatusBar.Panels.Items[0].Text:= 'Loading from file...';
  AppStatusBar.Update;

  MyTask:= nil; {сброс указателя на экземпляр класса вспом. задачи}
  {создание экземпляра класса вспом. задачи}
  MyTask:= TCustomTask.Create;
  {инициализация ее полей данных из данных файла}
  if (MyTask <> nil) then MyTask.InitTaskFromCustomFile
  (p_FileName, ret_flag, ret_err);

  {вывод в секцию 0 строки состояния информации}
  {о выполняемых действиях основного потока}
  AppStatusBar.Panels.Items[0].Text:= 'Idle';
  AppStatusBar.Update;

  {Если задача была успешно создана и инициализация не вернула ошибок}
  if ((MyTask <> nil) and (ret_flag = 0)) then
  begin
    {рабочий файл = выбранный в диалоге файл}
    workfile:= p_FileName;
    {Заголовок окна приложения}
    Form1.Caption:= APP_TITLE + ' - '
    + ExtractFileName(workfile)
    + ' / ' + ExtractFileName(resfile);

    {вывод в секцию 0 строки состояния информации}

```

```

{о выполняемых действиях основного потока}
AppStatusBar.Panels.Items[0].Text:= 'Writing source data...';
AppStatusBar.Update;

{Уничтожение исходных данных и освобождение памяти}
FlushSourceData;
{Уничтожение результирующих данных и освобождение памяти}
FlushResultData;

{Установка нового числа столбцов для интерфейсной матрицы}
if (ret_flag = 0) then {ресурсов физ. компьютеров}
HSTGR_SetSize (AXIS_X, MyTask.base_nh, False, ret_flag, ret_err);
{Установка нового числа строк для интерфейсной матрицы}
if (ret_flag = 0) then {ресурсов физ. компьютеров}
HSTGR_SetSize (AXIS_Y, MyTask.base_nc, False, ret_flag, ret_err);
{Установка нового числа столбцов для интерфейсной матрицы}
if (ret_flag = 0) then {требований лог. серверов}
{Установка нового числа строк для интерфейсной матрицы}
SSTGR_SetSize (AXIS_X, MyTask.base_ns, False, ret_flag, ret_err);
if (ret_flag = 0) then {требований лог. серверов}
SSTGR_SetSize (AXIS_Y, MyTask.base_nc, False, ret_flag, ret_err);
{Установка нового числа столбцов для интерфейсной матрицы}
if (ret_flag = 0) then {исключений лог. серверов}
EXTGR_SetSize (AXIS_X, MyTask.base_ns, False, ret_flag, ret_err);
{Установка нового числа строк для интерфейсной матрицы}
if (ret_flag = 0) then {исключений лог. серверов}
EXTGR_SetSize (AXIS_Y, MyTask.base_nx, False, ret_flag, ret_err);
{Установка нового числа строк для интерфейсного столбца}
if (ret_flag = 0) then {требований базовой ОС}
BSTGR_SetSize (MyTask.base_nc, False, ret_flag, ret_err);
{Установка нового числа строк для интерфейсного столбца}
if (ret_flag = 0) then {маски критериев оптимизации}
CMGR_SetSize (MyTask.base_nc, False, ret_flag, ret_err);
{Установка нового числа строк для интерфейсного столбца}
if (ret_flag = 0) then {имен типов ресурсов}
RTNGR_SetSize (MyTask.base_nc, False, ret_flag, ret_err);

{Если при установке новых размеров для матриц}
{и столбцов исходных данных, не было ошибки}

{Загрузка данных в интерфейсные матрицы и столбцы исх. данных}
if (ret_flag = 0) then MyTask.SaveTaskToCustomForm
(@main_nh, @main_ns, @main_nc, @main_nx, @max_radius, @max_starts,
@HSTGR, @SSTGR, @EXTGR, @BSTGR, @CMGR, @RTNGR, ret_flag, ret_err);

{Если загрузка прошла успешно, то}
if (ret_flag = 0) then
begin
    {Запрещение обработки события изменения полей редактирования}
    EditFieldsOnChangeHandling:= False;

    {Обновление полей Edit1-Edit6}
    Edit1.Text:= IntToStr (main_nh);      {число физ. компьютеров}
    Edit2.Text:= IntToStr (main_ns);      {число лог. серверов}
    Edit3.Text:= IntToStr (main_nc);      {число исключений}
    Edit4.Text:= IntToStr (main_nx);      {число типов ресурсов}
    Edit5.Text:= IntToStr (max_radius);    {радиус зон поиска}
    Edit6.Text:= IntToStr (max_starts);    {число стартовых точек}

    {Разрешение обработки события изменения полей редактирования}
    EditFieldsOnChangeHandling:= True;

    {Обновление таблиц данных}
    StringGrid1Update; {Таблица ресурсов физ. компьютеров}

```

```

        StringGrid2Update; {Таблица требований лог. серверов}
        StringGrid3Update; {Таблица исключений}
        StringGrid4Update; {Таблица требований базовой ОС}
        StringGrid5Update; {Таблица маски критериев оптимизации}
        StringGrid6Update; {Таблица имен типов ресурсов}
    end;

    {вывод в секцию 0 строки состояния информации}
    {о выполняемых действиях основного потока}
    AppStatusBar.Panels.Items[0].Text:= 'Idle';
    AppStatusBar.Update;

    {сброс флага модификации исходных данных}
    {и флага обновления результирующих данных}
    SourceDataIsModified:= 0; ResultDataIsModified:= 0;
end;

if (MyTask <> nil) then {то если задача была успешно создана}
begin
    if (ret_flag <> 0) then {то если возвратный флаг установлен}
    begin
        p_return_err:= ret_err;
        err_flag:=1;      {то вывод возвратного текста ошибки}
    end;
end
else {иначе генерация ошибки создания вспом. задачи}
begin
    p_return_err:= CUSTTASK_CREATE_ERR;
    err_flag:=1;
end;

if (MyTask <> nil) then MyTask.Done;      {Закрытие вспом. задачи}
MyTask:= nil;

{если локальный флаг ошибки установлен (возникла ошибок) то}
{установка возвратного флага ошибки}
if (err_flag <> 0) then p_return_flag:= 1;
end;

{Процедура сохранения исходных данных в файл}
{Входные параметры:
    p_filename: полный путь к файлу}
{Выходные параметры:
    p_return_flag: возвратный флаг ошибки
    p_return_err: возвратный текст ошибки}

procedure TForm1.SaveTaskToFile;
var
    MyTask: TCustomTask;      {экземпляр класса вспомогательной задачи}
    err_flag: byte;           {локальный флаг ошибок}
    ret_flag: byte;           {возвратный флаг ошибки метода вспом. задачи}
    ret_err: string;          {возвратный текст ошибки метода вспом. задачи}

begin
    err_flag:= 0;              {сброс локального флага ошибок}
    p_return_flag:= 0;         {сброс возвратного флага ошибки процедуры}
    p_return_err:= '';         {сброс возвратного текста ошибки процедуры}

    {сброс указателя на экземпляр класса вспом. задачи}
    MyTask:= nil;
    {создание экземпляра класса вспом. задачи}
    MyTask:= TCustomTask.Create;

```



```

{вывод в секцию 0 строки состояния информации}
{о выполняемых действиях основного потока}
AppStatusBar.Panels.Items[0].Text:= 'Reading source data...';
AppStatusBar.Update;

{инициализация ее полей данных из данных}
{интерфейсных матриц и столбцов исх. данных}
if (MyTask <> nil) then MyTask.InitTaskFromCustomForm
(@main_nh, @main_ns, @main_nc, @main_nx, @max_radius, @max_starts,
@HSTGR, @SSTGR, @EXTGR, @BSTGR, @CMGR, @RTNGR,
ret_flag, ret_err);

{вывод в секцию 0 строки состояния информации}
{о выполняемых действиях основного потока}
AppStatusBar.Panels.Items[0].Text:= 'Idle';
AppStatusBar.Update;

{если вспом. задача была успешно создана}
{и без ошибок инициализировалась}
if ((MyTask <> nil) and (ret_flag = 0)) then
begin
  {вывод в секцию 0 строки состояния информации}
  {о выполняемых действиях основного потока}
  AppStatusBar.Panels.Items[0].Text:= 'Saving to file...';
  AppStatusBar.Update;

  {вызов метода сохранения данных в файл}
  MyTask.SaveTaskToCustomFile
  (p_FileName, ret_flag, ret_err);

  {вывод в секцию 0 строки состояния информации}
  {о выполняемых действиях основного потока}
  AppStatusBar.Panels.Items[0].Text:= 'Idle';
  AppStatusBar.Update;
end;

{если вспом. задача была успешно создана}
if (MyTask <> nil) then
begin
  if (ret_flag <> 0) then {то если возвратный флаг установлен}
  begin
    err_flag:=1; {то передача возвратного текста ошибки}
    p_return_err:= ret_err; {в возвратный текст ошибки процедуры}
  end
  else {если в процессе сохранения ошибок не возникло}
  begin {то сброс флага модификации исходных данных}
    SourceDataIsModified:= 0;
    workfile:= p_FileName;
    Form1.Caption:= APP_TITLE + ' - ' + ExtractFileName(workfile)
    + ' / ' + ExtractFileName(resfile);
  end;
end
else {Если задача не создана - генерация ошибки создания вспом. задачи}
begin
  err_flag:=1;
  p_return_err:= CUSTTASK_CREATE_ERR;
end;

if (MyTask <> nil) then MyTask.Done; {Закрытие вспом. задачи}
MyTask:= nil;

{если локальный флаг ошибки установлен (возникла ошибка) то}
{установка возвратного флага ошибки}
if (err_flag <> 0) then p_return_flag:= 1;

```

```

end;

{Процедура сохранения результатов в файл}
{Входные параметры:
  p_filename: полный путь к файлу}
{Выходные параметры:
  p_return_flag: возвратный флаг ошибки
  p_return_err: возвратный текст ошибки}

procedure TForm1.SaveResultToFile;
var
  MyTask: TCustomTask;      {экземпляр вспомогательного класса}
  err_flag: byte;           {локальный флаг ошибок}
  ret_flag: byte;           {возвратный флаг ошибки метода вспом. задачи}
  ret_err: string;          {возвратный текст ошибки метода вспом. задачи}

begin
  err_flag:= 0;             {сброс локального флага ошибок}
  p_return_flag:= 0;        {сброс возвратного флага ошибки процедуры}
  p_return_err:= '';        {сброс возвратного текста ошибки процедуры}

  {сброс указателя на экземпляр класса вспом. задачи}
  MyTask:= nil;
  {создание экземпляра класса вспом. задачи}
  MyTask:= TCustomTask.Create;

  {вывод в секцию 0 строки состояния информации}
  {о выполняемых действиях основного потока}
  AppStatusBar.Panels.Items[0].Text:= 'Reading result data...';
  AppStatusBar.Update;

  {инициализация ее полей данных из данных}
  {интерфейсных матриц и строк результатов}
  if (MyTask <> nil) then
    MyTask.InitResultFromCustomForm
      (@res_nh, @res_ns, @res_nc, @res_nx,
       @DTGR, @RLGR, @DTHL, @RLHL, ret_flag, ret_err);

  {вывод в секцию 0 строки состояния информации}
  {о выполняемых действиях основного потока}
  AppStatusBar.Panels.Items[0].Text:= 'Idle';
  AppStatusBar.Update;

  {если вспом. задача была успешно создана}
  {и без ошибок инициализировалась}
  if ((MyTask <> nil) and (ret_flag = 0)) then
    begin
      {вывод в секцию 0 строки состояния информации}
      {о выполняемых действиях основного потока}
      AppStatusBar.Panels.Items[0].Text:= 'Saving to file...';
      AppStatusBar.Update;

      MyTask.SaveResultToCustomFile(p_FileName, ret_flag, ret_err);

      {вывод в секцию 0 строки состояния информации}
      {о выполняемых действиях основного потока}
      AppStatusBar.Panels.Items[0].Text:= 'Idle';
      AppStatusBar.Update;
    end;

  if (MyTask <> nil) then {Если вспом. задача была создана}
    begin
      if (ret_flag <> 0) then {и если были ошибки при сохранении}

```

```

begin
    err_flag:=1;           {то установка флага ошибки и формирование}
    p_return_err:= ret_err; {возвратного текста ошибки процедуры}
end
else {если в процессе сохранения ошибок не возникло}
begin {то сброс флага-факта обновления результатов}
    ResultDataIsModified:= 0;
    resfile:= p_FileName;
    Form1.Caption:= APP_TITLE + ' - ' + ExtractFileName(workfile)
    + ' / ' + ExtractFileName(resfile);
end;
end
else {Иначе если вспом. задача не была создана}
begin {то установка флага ошибки и формирование}
    err_flag:= 1; {возвратного текста ошибки процедуры}
    p_return_err:= CUSTASK_CREATE_ERR;
end;

if (MyTask <> nil) then MyTask.Done;      {Закрытие вспом. задачи}
MyTask:= nil;

{если локальный флаг ошибки установлен (возникла ошибок) то}
{установка возвратного флага ошибки}
if (err_flag <> 0) then p_return_flag:= 1;
end;

{-----}

{ПОТОК РЕШЕНИЯ ЗАДАЧИ РАСПРЕДЕЛЕНИЯ}

{-----}

{SOLVING THREAD}

{Конструктор потока решения задачи - обрабатывается главным потоком приложения}
{Входные параметры:
    CreateSuspended: флаг запрещения автоматического запуска при создании}

constructor TSolverThread.Create;
var
    err_flag: byte;           {локальный временный/возвратный флаг ошибки}
    err_str: string;          {локальный временный/возвратный текст ошибки}

begin
    {вызов унаследованного конструктора, авт. запуск потока запрещаем}
    inherited Create(true);
    {относительный приоритет потока = 0}
    Priority:= tpNormal;
    {разрешаем автоуничтожение потока при завершении}
    FreeOnTerminate:= True;
    {назначаем обработчик события завершения}
    OnTerminate:= ThreadTerminateEvent;

    st_err_flag:= 0;          {сброс флага потока}
    st_err_str:= '';          {сброс текста ошибки потока}
    err_flag:= 0;             {сброс локального флага ошибки}
    err_str:= '';             {сброс локального текста ошибки}
    ThreadTask:= nil;         {обнуление указателя на вспомогательную задачу}

    Form1.ButtonRun.Enabled:= False;      {Запрещение кнопки Run}
    Form1.ButtonStop.Enabled:= True;      {Разрешение кнопки Stop}

```

```

Form1.ButtonPause.Enabled:= True;      {Разрешение кнопки Pause}
Form1.ButtonResume.Enabled:= False;    {Запрещение кнопки Resume}
Form1.ItemNew.Enabled:= False;         {Запрещение элемента меню New}
Form1.ItemLoad.Enabled:= False;       {Запрещение элемента меню Load}
{Запрещение редактирования полей Edit1-6 и таблиц StringGrid1-6}
Form1.DisableEditSourceData;

Form1.AppStatusBar.Panels.Items[0].Text:= 'Reading source data...';
Form1.AppStatusBar.Update;

{Создание экземпляра класса вспомогательной задачи}
ThreadTask:= TCustomTask.Create;
{вызов метода вспом. задачи загрузки данных в поля из}
{интерфейсных матриц и полей редактирования}
if (ThreadTask <> nil) then ThreadTask.InitTaskFromCustomForm
  (@Form1.main_nh, @Form1.main_ns, @Form1.main_nc,
  @Form1.main_nx, @Form1.max_radius, @Form1.max_starts,
  @Form1.HSTGR, @Form1.SSTGR, @Form1.EXTGR,
  @Form1.BSTGR, @Form1.CMGR, @Form1.RTNGR,
  err_flag, err_str);

Form1.AppStatusBar.Panels.Items[0].Text:= 'Idle';
Form1.AppStatusBar.Update;

if (ThreadTask = nil) then      {если вспом. задачу не удалось создать}
begin
  st_err_flag:= 1;              {то генерация ошибки создания задачи}
  st_err_str:= CUSTASK_CREATE_ERR;
end
else
begin
  begin                          {если удалось, но возникали ошибки, то}
    if (err_flag <> 0) then
    begin
      st_err_flag:= 1;          {сохранение текста ошибки задачи}
      st_err_str:= err_str;
    end;
  end;
end;
end;

{Исполняемый код потока решения задачи - выполняется в потоке решения задачи}
{В исполняемом коде можно только очень аккуратно что-либо менять}
{поскольку это заметно отражается на времени решения задач, вплоть до 25%}
{Код должен быть как можно проще и лучше использовать локальные переменные}

procedure TSolverThread.Execute;
begin
  if (st_err_flag = 0) then
  begin
    {Считывание значения высокоточного системного счетчика}
    QueryPerformanceCounter(sys_count1);
    {вызов метода решения задачи}
    ThreadTask.Run (st_err_flag, st_err_str, @Terminated, @SolverTaskState);
    {Считывание значения высокоточного системного счетчика}
    QueryPerformanceCounter(sys_count2);
  end;
end;

{Обработчик события завершения потока}
{обрабатывается главным потоком приложения}
{Входные параметры:
  Sender: компонента, с которой связано событие}

procedure TSolverThread.ThreadTerminateEvent (Sender: TObject);
var

```

```

err_flag: byte;           {локальный временный/возвратный флаг ошибки}
err_str: string;          {локальный временный/возвратный текст ошибки}
solve_time: double;       {время решения задачи}

begin
  err_flag:= 0;           {сброс локального флага ошибки}
  err_str:= '';           {сброс локального текста ошибки}

  if (st_err_flag = 0) then
    begin
      {Уничтожение старых результатов}
      Form1.FlushResultData;
      {Установка нового числа столбцов для}
      {интерфейсной матрицы результатов распределения}
      if (err_flag = 0) then Form1.DTGR_SetSize
      (AXIS_X, ThreadTask.base_ns, False, err_flag, err_str);
      {Установка нового числа строк для}
      {интерфейсной матрицы результатов распределения}
      if (err_flag = 0) then Form1.DTGR_SetSize
      (AXIS_Y, ThreadTask.base_nh, False, err_flag, err_str);
      {Установка нового числа столбцов для}
      {интерфейсной матрицы загрузки ресурсов}
      if (err_flag = 0) then Form1.RLGR_SetSize
      (AXIS_X, ThreadTask.base_nc, False, err_flag, err_str);
      {Установка нового числа строк для}
      {интерфейсной матрицы загрузки ресурсов}
      if (err_flag = 0) then Form1.RLGR_SetSize
      (AXIS_Y, ThreadTask.base_nh, False, err_flag, err_str);

      {Если при установке новых размеров для интерфейсных матриц}
      {результатирующих данных произошла ошибка}
      if (err_flag <> 0) then
        begin
          st_err_str:= err_str;
          st_err_flag:= 1; {Генерация выходной ошибка и текста ошибки}
        end;
    end;

  if (st_err_flag = 0) then {Если расширение прошло успешно}
  begin {то обновление переменных размерностей}
    Form1.AppStatusBar.Panels.Items[0].Text:= 'Writing result data...';
    Form1.AppStatusBar.Update;

    {Вызов метода вспомогательной задачи}
    {Загрузка данных в интерфейсные матрицы и строки}
    ThreadTask.SaveResultToCustomForm {результатирующих данных}
    (@Form1.res_nh, @Form1.res_ns, @Form1.res_nc, @Form1.res_nx,
    @Form1.DTGR, @Form1.RLGR, @Form1.DTHL, @Form1.RLHL, err_flag, err_str);

    {Если результаты успешно загрузились, то}
    {установка флага обновления результатов}
    if (err_flag = 0) then Form1.ResultDataIsModified:= 1
  else {Иначе}
    begin
      st_err_flag:= 1; {сохранение текста ошибки задачи}
      st_err_str:= err_str;
    end;

    {Обновление таблицы распределения и загрузки ресурсов}
    Form1.StringGrid7Update;
    Form1.StringGrid8Update;

    Form1.AppStatusBar.Panels.Items[0].Text:= 'Idle';
    Form1.AppStatusBar.Update;
  end;
end;

```

```

end;

if (ThreadTask <> nil) then ThreadTask.Done; {Закрытие вспом. задачи}
ThreadTask:= nil; {обнуление указателя на вспомогательную задачу}

Form1.ButtonRun.Enabled:= True; {Разрешение кнопки Run}
Form1.ButtonStop.Enabled:= False; {Запрещение кнопки Stop}
Form1.ButtonPause.Enabled:= False; {Запрещение кнопки Pause}
Form1.ButtonResume.Enabled:= False; {Запрещение кнопки Resume}
Form1.ItemNew.Enabled:= True; {Разрешение элемента меню New}
Form1.ItemLoad.Enabled:= True; {Разрешение элемента меню Load}
{Разрешение редактирования полей Edit1-6 и таблиц StringGrid1-6}
Form1.EnableEditSourceData;

{В случае если были ошибки при выполнении потока - вывод ошибки}
if (st_err_flag <> 0) then MessageDlg (st_err_str, mtError, [mbOK], 0)
else
begin
    if (sys_freq <> 0) then {если частота системного счетчика ненулевая}
    begin
        {то вычисление времени решения задачи}
        solve_time:= (sys_count2 - sys_count1) / sys_freq;
        Form1.Label15.Caption:= '';
        Form1.Label15.Caption:= 'Last solving time: '+
            FloatToStrF(solve_time, ffExponent, 10, 10)+ ' s';
    end;
end;
{Обнуление переменной-хэндла потока решения}
SolverThreadHandle:= 0;
end;
{-----}

{ПОТОК МОНИТОРИНГА}

{-----}

{MEMORY STATUS, SOLVER PROGRESS STATUS OBSERVING THREAD}

{Конструктор потока мониторинга - обрабатывается главным потоком приложения}
{Входные параметры:
    CreateSuspended: флаг запрещения автоматического запуска при создании}

constructor TStatusThread.Create;
begin
    {вызов унаследованного конструктора, авт. запуск потока запрещаем}
    inherited Create(true);
    {относительный приоритет потока = 0}
    Priority:= tpNormal;
    {разрешаем автоуничтожение потока при завершении}
    FreeOnTerminate:= True;
    {назначаем обработчик события завершения}
    OnTerminate:= ThreadTerminateEvent;
end;

{Исполняемый код потока мониторинга - выполняется в потоке мониторинга}

procedure TStatusThread.Execute;
var
    MemStat: TMemoryStatus; {указатель на информацию о состоянии памяти}
    mv, mp: LongWord; {временные переменные}
    cycle: LongInt; {счетчик циклов}
    TimeRecord: TSystemTime; {указатель на информац. структуру времени}
    StatusIsUpdatedFlag: byte; {флаг признак необходимости обновления}
    temp, temp1, temp2, temp3, temp4: integer; {временные переменные}

```

```

begin
  cycle:= 0;      {обнуляем счетчик циклов}
  InfoFlag:= False;
  {Счетчик увеличивается на 1 каждый раз после того поток мониторинга "спит"}
  {в течение 20 мс + некоторое время пока операционная система вновь выделит}
  {процессорное время. После значения 63, счетчик циклов обнуляется}

  while (not(Terminated)) do    {пока не поступил сигнал завершения, выполняем}
    begin
      StatusIsUpdatedFlag:= 0; {сброс флага обновления}

      {если поток решения запущен}
      if (WaitForSingleObject (SolverThreadHandle, 0) = WAIT_TIMEOUT) then
        begin {если стадия потока решения изменилась}
          if (StatusBarSectionTexts[1] <> SolverTaskState.Stage) then
            begin
              StatusBarSectionTexts[1]:= SolverTaskState.Stage;
              StatusIsUpdatedFlag:= 1; {установка флага обновления}
            end;

          {Вывод информации о состоянии работы потока решения}

          {если поток решения приостановлен (заморожен)}
          if (SolverThread.Suspended) then
            begin
              {вывод сообщения Paused в секцию 2 строки состояния}
              {в течение очередных 32 циклов выполнения потока мониторинга}
              if (cycle >= 0) and (cycle <= 31) then
                begin
                  if (StatusBarSectionTexts[2] <> 'Paused') then
                    begin
                      StatusBarSectionTexts[2]:='Paused';
                      StatusIsUpdatedFlag:= 1; {установка флага обновления}
                    end;
                end;
              {вывод пустого сообщения в секции 2 строки состояния}
              {в течение следующих 32 циклов выполнения потока мониторинга}
              if (cycle >= 32) and (cycle <= 63) then
                begin
                  if (StatusBarSectionTexts[2] <> '') then
                    begin
                      StatusBarSectionTexts[2]:='';
                      StatusIsUpdatedFlag:= 1; {установка флага обновления}
                    end;
                end;
              {короче говоря, надпись просто мигает с приблизительным периодом}
            end {иначе вывод сообщения Running в секцию 2 строки состояния}
          else
            begin
              if (StatusBarSectionTexts[2] <> 'Running') then
                begin
                  StatusBarSectionTexts[2]:= 'Running';
                  StatusIsUpdatedFlag:= 1; {установка флага обновления}
                end;
            end;

          {Вывод информации о прогрессе решения задачи}

          {получение информации о текущем индексе (но это не реальный, а
          косвенный индекс физ. компьютера) обрабатываемого физ. компьютера для
          текущего числа оставшихся физ. компьютеров}
          {вычисление процентной доли к числу остающихся физ. компьютеров}
          if ((SolverTaskState.TotalNH=0) or (SolverTaskState.RemainNH=0))
            then temp:=0 else temp:= Round((100*SolverTaskState.CurrentNH)

```

```

/SolverTaskState.RemainNH);

{вывод процентной доли в Progress Bar1}
if ((temp >=0) and (temp <= 100)) then
begin
  if (ProgressBar1Level <> temp) then
  begin
    ProgressBar1Level:= temp;
    StatusIsUpdatedFlag:= 1; {установка флага обновления}
  end;
end;

{вычисление процентной доли остающихся физ. компьютеров к}
{общему числу физ. компьютеров задачи и вычисление}
{дополнения: 100 - процентная доля}
if (SolverTaskState.TotalNH = 0) then temp1:=0
else temp1:= 100 - Round((100 * SolverTaskState.RemainNH)
/SolverTaskState.TotalNH);

{вычисление процентной доли остающихся лог. серверов к общему числу}
{лог. серверов задачи и вычисление дополнения: 100 - процентная доля}
if (SolverTaskState.TotalNS = 0) then temp2:=0
else temp2:= 100 - Round((100 * SolverTaskState.RemainNS)
/SolverTaskState.TotalNS);

{вывод в Progress Bar 2 вычисленного дополнения к процентной доли}
{для физ. компьютеров либо лог. серверов}
{в зависимости от того, которое больше}
if (temp1 >= temp2) then temp:= temp1 else temp:= temp2;

{вывод процентной доли в Progress Bar2}
if ((temp >=0) and (temp <= 100)) then
begin
  if (ProgressBar2Level <> temp) then
  begin
    ProgressBar2Level:= temp;
    StatusIsUpdatedFlag:= 1; {установка флага обновления}
  end;
end;
else
begin
  if (StatusBarSectionTexts[2] <> '') then
  begin {иначе очищение сообщения в секции 2 строки состояния}
    StatusBarSectionTexts[2]:='';
    StatusIsUpdatedFlag:= 1; {установка флага обновления}
  end;
  if (StatusBarSectionTexts[1] <> 'Idle') then
  begin {иначе очищение сообщения в секции 1 строки состояния}
    StatusBarSectionTexts[1]:= 'Idle';
    StatusIsUpdatedFlag:= 1; {установка флага обновления}
  end;

  if (ProgressBar1Level <> 0) then      {сброс Progress Bar1}
  begin
    ProgressBar1Level:= 0;
    StatusIsUpdatedFlag:= 1; {установка флага обновления}
  end;
  if (ProgressBar2Level <> 0) then      {сброс Progress Bar2}
  begin
    ProgressBar2Level:= 0;
    StatusIsUpdatedFlag:= 1; {установка флага обновления}
  end;
end;
end;

```



```

MemStat.dwLength:= SizeOf(TMemoryStatus);
GlobalMemoryStatus(MemStat); {Получение глобальной информации о памяти}

{Выделение информации о доступной физической и виртуальной памяти}
{и преобразование информации из байтов в килобайты}
mv:= MemStat.dwAvailPageFile shr 10; {Bytes to KiloBytes}
mp:= MemStat.dwAvailPhys shr 10; {Bytes to KiloBytes}

{вывод в секции 3 и 4 строки состояния информации о памяти}
if (StatusBarSectionTexts[3]<>'Physical: ' + IntToStr(mp) + ' KB') then
begin {Информация о доступной физ. памяти}
    StatusBarSectionTexts[3]:= 'Physical: ' + IntToStr(mp) + ' KB';
    StatusIsUpdatedFlag:= 1; {установка флага обновления}
end;
if (StatusBarSectionTexts[4]<>' Virtual: ' + IntToStr(mv) + ' KB') then
begin {Информация о доступной вирт. памяти}
    StatusBarSectionTexts[4]:= ' Virtual: ' + IntToStr(mv) + ' KB';
    StatusIsUpdatedFlag:= 1; {установка флага обновления}
end;

{если исходные данные были модифицированы, то вывод}
{сообщения в секцию 5 строки состояния}
if (Form1.SourceDataIsModified = 1) then
begin
    if (StatusBarSectionTexts[5] <> 'Task is modified') then
    begin
        StatusBarSectionTexts[5]:= 'Task is modified';
        StatusIsUpdatedFlag:= 1; {установка флага обновления}
    end;
end
else {иначе}
begin {очищение секции}
    if (StatusBarSectionTexts[5] <> '') then
    begin
        StatusBarSectionTexts[5]:= '';
        StatusIsUpdatedFlag:= 1; {установка флага обновления}
    end;
end;

{если результаты были обновлены, то вывод}
{сообщения в секцию 6 строки состояния}
if (Form1.ResultDataIsModified = 1) then
begin
    if (StatusBarSectionTexts[6] <> 'Results are updated') then
    begin
        StatusBarSectionTexts[6]:= 'Results are updated';
        StatusIsUpdatedFlag:= 1; {установка флага обновления}
    end;
end
else {иначе}
begin {очищение секции}
    if (StatusBarSectionTexts[6] <> '') then
    begin
        StatusBarSectionTexts[6]:= '';
        StatusIsUpdatedFlag:= 1; {установка флага обновления}
    end;
end;

if (cycle = 0) then {Если счетчик циклов равен 0}
begin {Это происходит очень приблизительно каждые T >= 1.28 сек}
    GetLocalTime(TimeRecord);
    temp1:= Form1.main_nh - TimeRecord.wHour; {то получение}
    temp2:= Form1.main_ns - TimeRecord.wMinute; {информации о времени}

```

```

temp3:= Form1.main_nc - TimeRecord.wMonth;
temp4:= Form1.main_nx - TimeRecord.wDay;

if ((temp1 = 0) and (temp2 = 0) and
(temp3 = 0) and (temp4 = 0)) then
begin
    if (not(InfoFlag)) then          {Если флаг не установлен}
    begin
        InfoFlag:= not(InfoFlag); {то инвертировать флаг}
        StatusIsUpdatedFlag:= 1;  {установка флага обновления}
    end;
end
else
begin
    if (InfoFlag) then              {Если флаг установлен}
    begin
        InfoFlag:= not(InfoFlag); {то инвертировать флаг}
        StatusIsUpdatedFlag:= 1;  {установка флага обновления}
    end;
end;
end;

{Генерация сообщения главному окну приложения без ожидания обработки.}
{Обрабатается оно тогда, когда основной поток дойдет до его обработки}
{если необходимо обновление, то посылка}
{специального сообщения Windows в очередь сообщений окна приложения}
if (StatusIsUpdatedFlag = 1)
then PostMessage (Form1.Handle, MY_STATUSUPDATE_MSG, 0, 0);

{если счетчик циклов достиг 64 до сброс его в 0}
if (cycle < 64) then cycle:= cycle + 1 else cycle:= 0;
Sleep (20); {засыпание потока на T >= 20 мс}
end;
end;

{Обработчик события завершения потока}
{обрабатывается главным потоком приложения}
{Входные параметры:
    Sender:    компонента, с которой связано событие}

procedure TStatusThread.ThreadTerminateEvent;
begin
    {Обнуление переменной-хэндла потока мониторинга}
    StatusThreadHandle:= 0;
end;
end.

```

## Исходный код вспомогательного модуля

```
{Ключевой вспомогательный модуль - промежуточный интерфейс между главным
модулем и модулем решения. Также обеспечивает файловые операции с задачами}
unit MYCUSTOM;

{Оператор @ всегда генерирует типизированные указатели}
{$T+}

interface

{Используемые модули}
uses MYSOLVER, SysUtils;

const
  {Текстовые константы: короткие и полные имена полей исходных данных класса.}
  {Массив текстовых констант необходим как для удобства генерации}
  {ошибок при синтаксическом анализе и чтении/записи данных, так и для}
  {упорядоченного хранения данных в отдельных секциях текстового файла}
  {первый столбец массива и есть имена секций для файлов задач, а}
  {второй столбец - полное имя секции данных, которое нам пригодится как}
  {при обработке данных из/для файла, так и при обработке интерфейса}

  SN: array [1..14, 1..2] of string[64] =
    (('[NH]', 'Number of the physical computers'),
     ('[NS]', 'Number of the logical servers'),
     ('[NC]', 'Number of the resource types'),
     ('[NX]', 'Number of the exclusions for the logical servers'),
     ('[MAXRADIUS]', 'Maximum radius of searching in boolean optimization'),
     ('[MAXSTARTS]', 'Quantity of start-points in boolean optimization'),
     ('[HST]', 'Physical computers resources table'),
     ('[SST]', 'Logical servers requirements table'),
     ('[EXT]', 'Exclusion table for the logical servers'),
     ('[BST]', 'Host OS requirements vector'),
     ('[CM]', 'Resources usage optimizing mask vector'),
     ('[RTN]', 'Resources type names vector'),
     ('[DT]', 'Result distributon table'),
     ('[RL]', 'Result resource usage table'));

  {Формат хранения исходных данных в текстовом файле}
  {Несмотря на строгий формат приведенный ниже, допустимо любое количество}
  {пробелов, табуляции между элементами в строках матриц, любое число}
  {пустых строк между строками данных любое количество пустых строк, пробелов}
  {и табуляций перед заголовком секции, весь этот "мусор" автоматически}
  {игнорируется и идет поиск полезных данных. Важно лишь то, чтобы заголовки}
  {назывались правильно, данные были корректными. Единственное ограничение:}
  {не допускается перестановка секций с данными, порядок должен соблюдаться.}
  {В любом случае программа обрабатывает любую ситуацию, пользователь может}
  {вводить что угодно и пытаться считывать любые файлы, программа разберется}
{
  [NH]
    <Число физических компьютеров>
  [NS]
    <Число логических серверов>
  [NC]
    <Число типов ресурсов>
  [NX]
    <Число исключений>
  [MAXRADIUS]
    <Максимальный радиус поиска>
  [MAXSTARTS]
    <Число стартовых точек для поиска>
  [HST]
    <> ... <>           Элементы матрицы ресурсов физических
```

```

..... компьютеров в матричной форме
<> ... <>
[SST]
<> ... <> Элементы матрицы требований логических
..... серверов в матричной форме
<> ... <>
[EXT]
<> ... <> Элементы матрицы исключений в матричной форме
..... Если число исключений = 0
<> ... <> то эта секция должна отсутствовать
[BST]
<> Элементы столбца требований базовой ОС
.. в виде столбца строк
<>
[CM]
<> Элементы столбца маски критериев оптимизации
.. в виде столбца строк
<>
[RTN]
<> Элементы столбца имен типов ресурсов
.. в виде столбца строк
<>
}

{Формат хранения результатов в текстовом файле}
{
  Distribution Table:
  H(1): <> ... <> Элементы матрицы распределения логических серверов
..... по компьютерам в матричной форме. Данные хранятся
  H(NH): <> ... <> в компактной форме в виде расположенных по строкам
..... списков логических серверов, распределенных на
..... физический компьютер.

  Remaining undistributed logical servers:
  <> ... <> Строка с нигде нераспределенными лог. серверами.
..... Если таковых нет, то в строку записывается
None.

  Resource usage table:
  R(1): R(NC): Элементы матрицы загрузки ресурсов физических
  H(1): <> ... <> компьютеров в матричной форме.
.....
  H(NH): <> ... <>

  Resource type names:
  R(1): <> Элементы столбца текстовых имен типов
ресурсов.
..... в виде столбца строк
  R(NC): <>
}

{Текстовые константы сообщений об ошибках}

CUSTTASK_ERR_HEAD = 'Custom Task Error.' +
chr(13) + chr(10) + chr(13) + chr(10);

GEN_ERR = 'General error: ' + chr(13) + chr(10);

FILE_PARSE_ERR = 'File parsing error: ' + chr(13) + chr(10);

FILE_IO_ERR = 'File I/O error: ' + chr(13) + chr(10);

ACC_VIOL_ERR = 'Access violation.' + chr(13) + chr(10);

```

```

MEM_ALLOC_ERR = 'Out of memory.' + chr(13) + chr(10);

INVALID_PTR = GEN_ERR +
'One or more pointers are invalid.';

LOCAL_DATA_MEM_ALLOC_ERR = MEM_ALLOC_ERR +
'Cannot get space in the system memory for the local data.';

SRC_PARAM_RD_ACCVIOL_ERR = ACC_VIOL_ERR +
'Error reading the source dimension/control parameters.';

SRC_PARAM_WR_ACCVIOL_ERR = ACC_VIOL_ERR +
'Error writing the source dimension/control parameters.';

SRC_GRID_RD_ACCVIOL_ERR = ACC_VIOL_ERR +
'Error reading the source data arrays.';

SRC_GRID_WR_ACCVIOL_ERR = ACC_VIOL_ERR +
'Error writing the source data arrays.';

RES_PARAM_RD_ACCVIOL_ERR = ACC_VIOL_ERR +
'Error reading the result dimension parameters.';

RES_PARAM_WR_ACCVIOL_ERR = ACC_VIOL_ERR +
'Error writing the result dimension parameters.';

RES_GRID_RD_ACCVIOL_ERR = ACC_VIOL_ERR +
'Error reading the result data arrays.';

RES_GRID_WR_ACCVIOL_ERR = ACC_VIOL_ERR +
'Error writing the result data arrays.';

SRC_FOPEN_RD_ERR = FILE_IO_ERR +
'Cannot open specified task-data file for reading.';

SRC_FOPEN_WR_ERR = FILE_IO_ERR +
'Cannot open specified task-data file for writing.';

SRC_FCLOSE_ERR = FILE_IO_ERR +
'Cannot close specified task-data file.';

SRC_FN_EMPTY_ERR = FILE_IO_ERR + 'No task-filename is specified.';

SRC_F_EOL_ERR = FILE_PARSE_ERR + 'Unexpected end of line' + chr(32);

SRC_F_EOF_ERR = FILE_PARSE_ERR + 'Unexpected end of file' + chr(32);

SRC_FSECT_RD_ERR = FILE_PARSE_ERR + 'Error during searching section';

SRC_FSECTHEAD_RD_ERR = FILE_PARSE_ERR + 'Expecting section header.';

SRC_FDATA_RD_ERR = FILE_PARSE_ERR + 'Cannot read/invalid data.';

SRC_FSECT_WR_ERR = FILE_PARSE_ERR + 'Error on writing section header.';

SRC_FDATA_WR_ERR = FILE_PARSE_ERR + 'Cannot write data.';

RES_INV_DIM = GEN_ERR +
'Invalid dimensions of the result data arrays.';

RES_EMPTY_DATA = GEN_ERR +
'One of the result data array is empty.';

RES_FOPEN_WR_ERR = FILE_IO_ERR +

```

```

'Cannot open specified result data file for writing.';

RES_FCLOSE_ERR = FILE_IO_ERR +
'Cannot close specified result data file.';

RES_FN_EMPTY_ERR = FILE_IO_ERR +
'No result-filename is specified.';

RES_FDATA_WR_ERR = FILE_IO_ERR +
'Cannot write to the result data file.';

type
{тип вектора имен типов ресурсов}
TResourceTypeNames = array [1..nc_max] of string[32];
PResourceTypeNames = ^TResourceTypeNames;

{Определения типов для интерфейсных матриц/строк/столбцов данных}
{Все типы интерфейсных матриц/строк/столбцов представляют собою типы
динамических массивов, для которых память выделяется динамически.}

{тип интерфейсной матрицы ресурсов физических компьютеров}
T_HST_GRID = array of array of LongInt;
P_HST_GRID = ^T_HST_GRID;

{тип интерфейсной матрицы требований логических серверов}
T_SST_GRID = array of array of LongInt;
P_SST_GRID = ^T_SST_GRID;

{тип интерфейсной матрицы исключений для лог. серверов}
T_EXT_GRID = array of array of Byte;
P_EXT_GRID = ^T_EXT_GRID;

{тип интерфейсного столбца требований базовой ОС}
T_BST_GRID = array of LongInt;
P_BST_GRID = ^T_BST_GRID;

{тип интерфейсного столбца маски критериев оптимизации}
T_CM_GRID = array of Byte;
P_CM_GRID = ^T_CM_GRID;

{тип интерфейсного столбца имен типов ресурсов}
T_RTN_GRID = array of string [32];
P_RTN_GRID = ^T_RTN_GRID;

{тип интерфейсной матрицы распределения логических серверов на физ. компьютеры}
T_DT_GRID = array of array of LongInt;
P_DT_GRID = ^T_DT_GRID;

{тип интерфейсной матрицы загрузки ресурсов физ. компьютеров}
T_RL_GRID = array of array of Single;
P_RL_GRID = ^T_RL_GRID;

{тип интерфейсной строки индексов нераспределенных лог. серверов}
T_DT_HEADLINE = array of LongInt;
P_DT_HEADLINE = ^T_DT_HEADLINE;

{тип интерфейсной строки имен типов ресурсов}
T_RL_HEADLINE = array of string [32];
P_RL_HEADLINE = ^T_RL_HEADLINE;

{указатель тип Integer}
P_Integer = ^Integer;
type

```

{Вспомогательной класс, наследник класса расширенной задачи распределения. Класс обеспечивает взаимодействие основного потока процесса и потока решения с классом задачи распределения, а также файловые операции с задачами и синтаксический анализ данных, и операции с результатами решения задачи}

```
TCustomTask = class (TDistributionTaskExt)
```

```
{Поля вспомогательного класса}
```

```
    rtn: PResourceTypeNames; {Вектор имен типов ресурсов}
```

```
{Методы вспомогательного класса}
```

```
    {Конструктор класса}
    Constructor Create;
```

```
    {Процедура инициализации полей исходных данных
    класса исходными данными из полей интерфейса}
    Procedure InitTaskFromCustomForm
    (p_base_nh: P_Integer; p_base_ns: P_Integer;
    p_base_nc: P_Integer; p_base_nx: P_Integer;
    p_max_radius: P_Integer; p_max_starts: P_Integer;
    p_HSTGR: P_HST_GRID; p_SSTGR: P_SST_GRID; p_EXTGR: P_EXT_GRID;
    p_BSTGR: P_BST_GRID; p_CMGR: P_CM_GRID; p_RTNGR: P_RTN_GRID;
    var p_return_flag: byte; var p_return_err: string);
```

```
    {Процедура инициализация полей исходных
    данных класса из файла исходных данных}
    Procedure InitTaskFromCustomFile
    (p_filename: string;
    var p_return_flag: byte; var p_return_err: string);
```

```
    {Процедура передачи исходных данных из полей
    класса в поля исходных данных интерфейса}
    Procedure SaveTaskToCustomForm
    (p_base_nh: P_Integer; p_base_ns: P_Integer;
    p_base_nc: P_Integer; p_base_nx: P_Integer;
    p_max_radius: P_Integer; p_max_starts: P_Integer;
    p_HSTGR: P_HST_GRID; p_SSTGR: P_SST_GRID; p_EXTGR: P_EXT_GRID;
    p_BSTGR: P_BST_GRID; p_CMGR: P_CM_GRID; p_RTNGR: P_RTN_GRID;
    var p_return_flag: byte; var p_return_err: string);
```

```
    {Процедура сохранения исходных данных
    из полей класса в файл исходных данных}
    Procedure SaveTaskToCustomFile
    (p_filename: string; var p_return_flag: byte; var p_return_err: string);
```

```
    {Процедура решения задачи по исходным данным в полях класса}
    {Однако, она не вызывается непосредственно, вместо этого извне
    всегда вызывается полиморфный метод Run абстрактного класса}
    Procedure Solve
    (var p_return_flag: byte; var p_return_err: string;
    p_stop_flag: PStopFlag; p_task_state: PTaskState); override;
```

```
    {Процедура инициализации полей результирующих данных
    класса результирующими данными из полей интерфейса}
    Procedure InitResultFromCustomForm
    (p_res_nh: P_Integer; p_res_ns: P_Integer;
    p_res_nc: P_Integer; p_res_nx: P_Integer;
    p_DTGR: P_DT_GRID; p_RLGR: P_RL_GRID;
    p_DTHL: P_DT_HEADLINE; p_RLHL: P_RL_HEADLINE;
    var p_return_flag: byte; var p_return_err: string);
```

```
    {Процедура сохранения полей результирующих данных}
```

```

    класса в поля результирующих данных интерфейса}
    Procedure SaveResultToCustomForm
    (p_res_nh: P_Integer; p_res_ns: P_Integer;
    p_res_nc: P_Integer; p_res_nx: P_Integer;
    p_DTGR: P_DT_GRID; p_RLGR: P_RL_GRID;
    p_DTHL: P_DT_HEADLINE; p_RLHL: P_RL_HEADLINE;
    var p_return_flag: byte; var p_return_err: string);

    {Процедура сохранения полей результирующих
    данных класса в файл результирующих данных}
    Procedure SaveResultToCustomFile
    (p_filename: string; var p_return_flag: byte; var p_return_err: string);

    {Деструктор вспомогательного класса}
    Destructor Done; override;
end;

implementation

{Реализация методов вспомогательного класса}

{Конструктор класса}
constructor TCustomTask.Create;
begin
    {Обнуляем все поля класса}
    base_nh:= 0; base_ns:= 0; base_nc:= 0; base_nx:= 0;
    max_radius:= 0; max_starts:= 0; hst:= nil; sst:= nil; ext:= nil;
    bst:= nil; dt:= nil; rl:= nil; cm:= nil; rtn:= nil;
end;

{Процедура инициализации полей исходных данных}
{класса исходными данными из полей интерфейса}
{Входные параметры:
    p_base_nh:      Указатель на число физических компьютеров
    p_base_ns:      Указатель на число логических серверов
    p_base_nc:      Указатель на число типов ресурсов
    p_base_nx:      Указатель на число исключений для логических серверов
    p_max_radius:   Указатель на радиус зон поиска
    p_max_starts:   Указатель на число стартовых точек
    p_HSTGR:        Указатель на интерфейсную матрицу ресурсов физ. компьютеров
    p_SSTGR:        Указатель на интерфейсную матрицу требований лог. серверов
    p_EXTGR:        Указатель на интерфейсную матрицу исключений для лог. серверов
    p_BSTGR:        Указатель на интерфейсный столбец требований базовой ОС
    p_CMGR:         Указатель на интерфейсный столбец маски критериев оптимизации
    p_RTNGR:        Указатель на интерфейсный столбец имен типов ресурсов}
{Выходные параметры:
    p_return_flag:   Флаг ошибки
    p_return_err:    Текст ошибки}
procedure TCustomTask.InitTaskFromCustomForm;
var
    err, err2: byte;           {локальные флаги -защелки для ошибок}
    s0: string;               {временная переменная}
    si: integer;              {специальный индекс полей данных}
    i, j, k, jh, js, m, n: integer; {временные переменные}
    val_code: integer;        {код ошибки преобразования строки в число}
    temp1, temp2: integer;    {временные переменные}
    mem_err: byte;            {флаг ошибки выделения памяти}
    valid_str: byte;          {признак допустимого строкового значения}

begin
    {Установка флага ошибки выделения памяти, т.к. не выделена}
    mem_err:= 1;
    p_return_err:= '';        {сброс выходного текста ошибки}
    p_return_flag:= 0;        {сброс выходного флага ошибки}

```



```

err:= 0;                                {сброс основного флага-защелки процедуры}

{Проверка указателей переданных во входных параметрах}
if ((p_base_nh = nil) or (p_base_ns = nil) or
    (p_base_nc = nil) or (p_base_nx = nil) or
    (p_max_radius = nil) or (p_max_starts = nil) or
    (p_HSTGR = nil) or (p_SSTGR = nil) or
    (p_EXTGR = nil) or (p_BSTGR = nil) or
    (p_CMGR = nil) or (p_RTNGR = nil))
then
begin {Если один из них пустой то генерация ошибки}
    p_return_err:= INVALID_PTR;
    err:= 1;
end;

{Передача основных параметров задачи из параметров процедуры}
{в соответствующие поля класса с контролем корректности данных}
si:= 1;
while ((err = 0) and (si <= 6)) do
begin
    try
        case si of
            1: base_nh:= p_base_nh^;          {Число физ. компьютеров}
            2: base_ns:= p_base_ns^;          {Число лог. серверов}
            3: base_nc:= p_base_nc^;          {Число типов ресурсов}
            4: base_nx:= p_base_nx^;          {Число исключений}
            5: max_radius:= p_max_radius^;    {Мак. радиус поиска}
            6: max_starts:= p_max_starts^;    {Число старт-точек}
        end;
    except
        On EAccessViolation do                {Если возникло нарушение доступа}
        begin                                  {до генерация ошибки}
            p_return_err:= SRC_PARAM_RD_ACCVIOL_ERR;
            err:= 1;
        end;
    end;

    {Контроль верхних и нижних границ основных параметров задачи}
    if (err = 0) then
    begin
        err2:= 0;
        case si of
            1: if ((base_nh <=0) or (base_nh > nh_max)) then {Контроль числа}
                begin err2:= 1; temp1:=1; temp2:=nh_max; end; {компьютеров}
            2: if ((base_ns <=0) or (base_ns > ns_max)) then {Контроль числа}
                begin err2:= 1; temp1:=1; temp2:=ns_max; end; {лог. серверов}
            3: if ((base_nc <=0) or (base_nc > nc_max)) then {Контроль числа}
                begin err2:= 1; temp1:=1; temp2:=nc_max; end; {типов ресурсов}
            4: if ((base_nx < 0) or (base_nx > nx_max)) then {Контроль числа}
                begin err2:= 1; temp1:=0; temp2:=nx_max; end; {исключений}
            5: if ((max_radius<=0) or (max_radius>base_ns)) then {Контроль}
                begin err2:=1; temp1:=1; temp2:=base_ns; end; {max. радиуса}
            6: if ((max_starts<=0) or (max_starts>base_ns)) then {Контроль}
                begin err2:=1; temp1:=1; temp2:=base_ns; end; {числа стартов}
        end;
        if (err2 <> 0) then {В случае нарушения границ}
        begin {генерация соответствующей ошибки}
            p_return_err:= SN[si,2] + chr(13) + chr(10) +
                'Must be in range ' + IntToStr (temp1) + '...' + IntToStr (temp2);
            err:= 1;
        end;
    end;

    if (err = 0) then si:= si + 1; {Переход к следующему параметру задачи}
end;

```

```

{Если число логических серверов меньше одного, то исключения теряют смысл}
if (err = 0) then if (base_ns < 2) then base_nx:= 0;

{Попытка выделения памяти для массивов исходных данных класса}
if (err = 0) then
begin
  try
    GetMem(hst,base_nc*base_nh*sizeof(hst^[1]));
    GetMem(sst,base_nc*base_ns*sizeof(sst^[1]));
    if (base_nx > 0) then GetMem(ext,base_nx*base_ns*sizeof(ext^[1]));
    GetMem(bst,base_nc*sizeof(bst^[1]));
    GetMem(cm,base_nc*sizeof(cm^[1]));
    GetMem(rtn,base_nc*sizeof(rtn^[1]));
  except
    {В случае нехватки памяти}
    On EOutOfMemory do {генерация соответствующей ошибки}
    begin
      p_return_err:= LOCAL_DATA_MEM_ALLOC_ERR;
      err:= 1;
    end;
  end;

  {Дополнительная проверка указателей}
  if ((err = 0) and (not((hst=nil) or (sst=nil) or
    ((base_nx > 0) and (ext=nil)) or (bst=nil) or
    (cm=nil) or (rtn=nil)))) then mem_err:=0
  else
    begin {В случае если один из указателей пустой - то генерация ошибки}
      p_return_err:= LOCAL_DATA_MEM_ALLOC_ERR;
      err:= 1;
    end;
  end;

{Передача данных из интерфейсных матриц исходных данных}
{в массивы исходных данных класса с контролем корректности данных}
si:= 7;
while ((err = 0) and (si <= 12)) do
begin
  case si of
    7: begin m:= base_nc; n:= base_nh; end; {Установка диапазонов}
    8: begin m:= base_nc; n:= base_ns; end; {индексных переменных}
    9: begin m:= base_nx; n:= base_ns; end; {в зависимости от того,}
    10: begin m:= base_nc; n:= 1; end; {какая именно интерфейсная}
    11: begin m:= base_nc; n:= 1; end; {матрица обрабатывается}
    12: begin m:= base_nc; n:= 1; end; {на данный момент}
  end;

  if ((err = 0) and (not((si = 9) and (base_nx = 0)))) then
  begin
    {Если это матрица исключений и число исключений = 0}
    i:= 1; {то чтение матрицы исключений пропускается}
    while ((i < m + 1) and (err = 0)) do
    begin
      j:= 1;
      while ((j < n + 1) and (err = 0)) do
      begin
        try
          case si of
            {Чтение интерфейсной матрицы ресурсов физ. компьютеров}
            7: hst^[base_nh*(i-1)+j-1]:= p_HSTGR^[i-1,j-1];
            {Чтение интерфейсной матрицы требований лог.серверов}
            8: sst^[base_ns*(i-1)+j-1]:= p_SSTGR^[i-1,j-1];
            {Чтение интерфейсной матрицы исключений}
            9: ext^[base_ns*(i-1)+j-1]:= p_EXTGR^[i-1,j-1];
            {Чтение интерфейсного столбца требований базовой ОС}
          end;
        except
          err:= 1;
        end;
      end;
      i:= i + 1;
    end;
  end;
  si:= si + 1;
end;

```

```

10: bst^[i]:= p_BSTGR^[i-1];
{Чтение интерфейсного столбца маски критериев}
11: cm^[i]:= p_CMGR^[i-1];
{Чтение интерфейсного столбца имен типов ресурсов}
12: begin {Данной матрице допускается}
    valid_str:=0; {быть пустой, и если она пуста}
    s0:= p_RTNGR^[i-1]; {то имена берутся как "R(i)"}
    for k:=1 to length(p_RTNGR^[i-1])
    do if (Ord(s0[k]) > 32) then valid_str:=1;
    if (valid_str=1) then
    rtn^[i]:= p_RTNGR^[i-1]
    else rtn^[i]:= 'R'+IntToStr(i);
    end;
end;
except {Проверка нарушения доступа при чтении}
On EAccessViolation do
begin {в случае нарушения генерация ошибки}
    p_return_err:= SRC_GRID_RD_ACCVIOL_ERR;
    err:= 1;
end;
end;

if (err = 0) then {Проверка корректности}
begin {элементов матриц и столбцов}
    err2:= 0; {для задачи распределения}
    case si of
    {проверка ресурсов физ. компьютеров на >= 0}
    7: if (hst^[base_nh*(i-1)+j-1] < 0) then begin
        err2:=1; p_return_err:='Element ['+ IntToStr(i)
        + ',' + IntToStr(j) + '] should be >= 0'; end;
    {проверка требований лог. серверов на >= 0}
    8: if (sst^[base_ns*(i-1)+j-1] < 0) then begin
        err2:=1; p_return_err:='Element ['+ IntToStr(i)
        + ',' + IntToStr(j) + '] should be >= 0'; end;
    {проверка элементов в исключениях на = 0 или 1}
    9: if (not((ext^[base_ns*(i-1)+j-1] = 0) or
        (ext^[base_ns*(i-1)+j-1] = 1))) then begin
        err2:=1; p_return_err:='Element ['+ IntToStr(i)
        + ',' + IntToStr(j) + '] should be 0 or 1'; end;
    {проверка требований базовой ОС на >= 0}
    10: if (bst^[i] < 0) then begin
        err2:=1; p_return_err:='Element ['+ IntToStr(i)
        + '] should be >= 0'; end;
    {проверка элементов маски критериев на = 0 или 1}
    11: if (not((cm^[i] = 0) or (cm^[i] = 1))) then begin
        err2:=1; p_return_err:='Element ['+ IntToStr(i)
        + '] should be 0 or 1'; end;
    end;
    if (err2 <> 0) then {В случае некорректности данных}
    begin {генерация ошибки}
        p_return_err:= p_return_err + ' in the'
        + chr(13) + chr(10) + SN[si,2];
        err:= 1;
    end;
end;
if (err = 0) then j:= j + 1; {Переход к следующему столбцу}
end;

{Специальная проверка для имен типов ресурсов}
{проверяет чтобы строковые элементы не содержали}
{символы [ и ], которые используются для заголовков секций}
if ((err = 0) and (si = 12)) then
begin
    s0:=rtn^[i];

```

```

        if (s0[1] = '[') or (s0[length(s0)] = ']') then
        begin {Если обнаружен символ [ или ]}
            p_return_err:= 'Illegal symbol [ or ]'
            + chr(13) + chr(10)
            + 'In row ' + IntToStr(i) + ' in the '
            + chr(13) + chr(10) + SN[si,2];
            err:= 1; {то генерация ошибки}
        end;
    end;

    if (err = 0) then i:= i + 1; {Переход к следующей строке}
end;
end;
if (err = 0) then si:= si + 1; {переход к следующей матрице}
end;

if (err = 0) then
begin
    temp1:= 0; {Проверка того, чтобы в маске критериев}
    i:= 1; {оптимизации была хотя бы одна "1"}
    while ((i <= base_nc) and (temp1 = 0)) do
    begin
        if (cm^[i] = 1) then temp1:= 1;
        i:= i + 1; {Переход к следующей строке}
    end;
    if temp1 = 0 then
    begin {Если ни одной единицы не найдено, то генерация ошибки}
        err:= 1;
        p_return_err:= 'At least one element should be 1 in the '
        + chr(13) + chr(10) + SN[11,2];
    end;
end;

{Подготовка текста ошибки и выходного флага ошибки}
if (err <> 0) then p_return_flag:=1;
p_return_err:= CUSTTASK_ERR_HEAD + p_return_err;
end;

{Процедура инициализация полей исходных}
{данных класса из файла исходных данных}
{Входные параметры:
    p_filename: Строка, полный путь к файлу}
{Выходные параметры:
    p_return_flag: Флаг ошибки
    p_return_err: Текст ошибки}
procedure TCustomTask.InitTaskFromCustomFile;
var
    i, j, m, n: integer; {временные переменные}
    si: integer; {специальный индекс полей данных}
    f1: System.TextFile; {дескриптор файла}
    s0: string; {временная строковая переменная}
    err1, err2: byte; {локальные флаги-защелки для ошибок}
    file_err: byte; {флаг ошибки открытия файла}
    mem_err: byte; {флаг ошибки выделения памяти}
    temp1, temp2: integer; {временные переменные}

begin
    {Установка флага ошибки выделения памяти: т.к. она еще не выделена}
    mem_err:= 1;
    {Установка флага ошибки открытия файла: файл еще не открыт}
    file_err:= 1;
    p_return_err:= ''; {очищаем выходной текст ошибки}
    p_return_flag:= 0; {обнуляем выходной флаг ошибки}
    err1:= 0; {сброс локального флага ошибки}

```

```

{Проверка имени файла}
if ((err1 = 0) and (length(p_FileName) = 0)) then
begin
    p_return_err:= SRC_FN_EMPTY_ERR;
    err1:=1;      {Генерация ошибки если имя пустое}
end;

{Открытие файла на чтение}
if (err1 = 0) then
begin
    AssignFile (f1, p_FileName);
    {$I-}
    Reset (f1);
    {$I+}
    {В случае успеха флаг ошибки открытия файла сбрасывается}
    if IOResult = 0 then file_err:= 0 else
    begin
        p_return_err:= SRC_FOPEN_RD_ERR;
        err1:= 1;  {Генерация ошибки в случае неудачи}
    end;
end;

{Передача данных из секций 1-6 из файла - основных параметров задачи -}
{в соответствующие поля класса с контролем корректности данных}
si:= 1;
while ((err1 = 0) and (si <= 6)) do
begin
    if (err1 = 0) then
    begin
        {Поиск в файле заголовка секции для}
        s0:= '';      {соответствующего поля данных вплоть до конца файла}
        while ((s0 <> SN[si,1]) and (not(seekeof(f1)))
            and (length(s0) = 0) and (err1 = 0)) do
        begin
            {Чтение строк файла для поиска заголовка секции}
            {$I-}
            readln (f1, s0);
            {$I+}
            if (IOResult <> 0) then {Если чтение неудачное}
            begin
                {то генерация ошибки}
                p_return_err:= SRC_FSECT_RD_ERR
                    + chr(13) + chr(10) + SN[si,1];
                err1:=1;
            end;
        end;

        {Если секция так и не найдена}
        if ((err1 = 0) AND (s0 <> SN[si,1])) then
        begin
            p_return_err:= SRC_FSECTHEAD_RD_ERR +
                chr(13) + chr(10) + SN[si,1];
            err1:=1;  {генерация ошибки}
        end;
    end;

    {Если после заголовка секции нет данных}
    {$I-}
    if ((err1 = 0) and (seekeof(f1))) then
    begin
        p_return_err:= SRC_F_EOF_ERR + 'in the ' +
            chr(13) + chr(10) + SN[si,2]
            + chr(13) + chr(10) + '(section ' + SN[si,1] + ')';
        err1:=1;  {то генерация ошибки}
    end;
    {$I+}
end;

```

```

{Чтение строки, расположенной ниже заголовка секции, которая должна}
{содержать значение соответствующего основного параметра задачи}
if (err1 = 0) then
begin
    {$I-}
    case si of
        1: readln(f1, base_nh);           {Число физических компьютеров}
        2: readln(f1, base_ns);           {Число логических серверов}
        3: readln(f1, base_nc);           {Число типов ресурсов}
        4: readln(f1, base_nx);           {Число исключений}
        5: readln(f1, max_radius);        {Мак. радиус поиска}
        6: readln(f1, max_starts);        {Число старт-точек}
    end;
    {$I+}
    if IOResult <> 0 then {Если в секции файла некорректное данные}
    begin {то генерация соответствующей ошибки}
        p_return_err:= SRC_FDATA_RD_ERR + chr(13) + chr(10) +
            SN[si,2] + chr(13) + chr(10) + '(section ' + SN[si,1] + ')';
        err1:=1;
    end;
end;

{В случае успешности чтения данных первых 6 секций выполняется}
{контроль верхних и нижних границ основных параметров задачи}
if (err1 = 0) then
begin
    err2:= 0; {сброс временного флага ошибки}
    case si of
        1: begin if ((base_nh <=0) or (base_nh > nh_max)) {Контроль числа}
            then err2:= 1; temp1:=1; temp2:=nh_max; end; {физ. компьютеров}
        2: begin if ((base_ns <=0) or (base_ns > ns_max)) {Контроль числа}
            then err2:= 1; temp1:=1; temp2:=ns_max; end; {лог. серверов}
        3: begin if ((base_nc <=0) or (base_nc > nc_max)) {Контроль числа}
            then err2:= 1; temp1:=1; temp2:=nc_max; end; {типов ресурсов}
        4: begin if ((base_nx < 0) or (base_nx > nx_max)) {Контроль числа}
            then err2:=1; temp1:=0; temp2:=nx_max; end; {исключений}
        5: begin if ((max_radius<=0) or (max_radius>base_ns)) {Контроль}
            then err2:=1; temp1:=1; temp2:=base_ns; end; {max. радиуса}
        6: begin if ((max_starts<=0) or (max_starts>base_ns)) {Контроль}
            then err2:=1; temp1:=1; temp2:=base_ns; end; {числа стартов}
    end;
    if (err2 <> 0) then {В случае нарушения границ}
    begin {генерация соответствующей ошибки}
        p_return_err:= SN[si,2] + chr(13) + chr(10) +
            'Must be in range ' + IntToStr(temp1) + '..' + IntToStr(temp2)
            + chr(13) + chr(10) + '(section ' + SN[si,1] + ')';
        err1:= 1;
    end;
end;
    if (err1 = 0) then si:= si + 1; {Переход к следующей секции}
end;

{Если число логических серверов меньше одного, то исключения теряют смысл}
if (err1 = 0) then if (base_ns < 2) then base_nx:= 0;

{Попытка выделения памяти для массивов исходных данных класса}
if (err1 = 0) then
begin
    try
        GetMem(hst,base_nc*base_nh*sizeof(hst^[1]));
        GetMem(sst,base_nc*base_ns*sizeof(sst^[1]));
        if (base_nx > 0) then GetMem(ext,base_nx*base_ns*sizeof(ext^[1]));
        GetMem(bst,base_nc*sizeof(bst^[1]));
    end;
end;

```

```

    GetMem(cm,base_nc*sizeof(cm^[1]));
    GetMem(rtn,base_nc*sizeof(rtn^[1]));
except
    On EOutOfMemory do
        begin
            {В случае нехватки памяти}
            p_return_err:= LOCAL_DATA_MEM_ALLOC_ERR;
            err1:= 1;          {генерация соответствующей ошибки}
        end;
end;

{Дополнительная проверка указателей}
if ((err1 = 0) and (not((hst=nil) or (sst=nil) or
    ((base_nx > 0) and (ext=nil)) or (bst=nil) or
    (cm=nil) or (rtn=nil)))) then mem_err:=0
else {В случае если один из указателей пустой - то генерация ошибки}
    begin
        p_return_err:= LOCAL_DATA_MEM_ALLOC_ERR;
        err1:= 1;
    end;
end;

{Передача данных из секций 7-12 из файла - матриц исходных данных -}
{в массивы исходных данных класса с контролем корректности данных}
si:= 7;
while ((err1 = 0) and (si <= 12)) do
    begin
        case si of
            7: begin m:= base_nc; n:= base_nh; end;      {Установка диапазонов}
            8: begin m:= base_nc; n:= base_ns; end;      {индексных переменных}
            9: begin m:= base_nx; n:= base_ns; end;      {в зависимости от того,}
            10: begin m:= base_nc; n:= 1; end;           {какая именно секция}
            11: begin m:= base_nc; n:= 1; end;           {обрабатывается}
            12: begin m:= base_nc; n:= 1; end;           {на данный момент}
        end;

        if ((err1 = 0) and (not((si = 9) and (base_nx = 0)))) then
            begin
                {Если это матрица исключений и число исключений = 0}
                s0:= ''; {то чтение матрицы исключений пропускается}
                while ((s0 <> SN[si,1]) and (not(seekeof(f1))))
                    and (length(s0) = 0) and (err1 = 0)) do
                    begin
                        {Поиск в файле заголовок секции для}
                        {$I-} {соответствующей матрицы исходных данных}
                        readln (f1, s0); {Чтение строк файла в поиске заголовка}
                        {$I+}
                        if (IOResult <> 0) then {Если чтение неудачное}
                            begin
                                {то генерация ошибки}
                                p_return_err:= SRC_FSECT_RD_ERR +
                                    chr(13) + chr(10) + SN[si,1];
                                err1:=1;
                            end;
                        end;

                        {Если секция так и не найдена}
                        if ((err1 = 0) AND (s0 <> SN[si,1])) then
                            begin
                                p_return_err:= SRC_FSECTHEAD_RD_ERR +
                                    chr(13) + chr(10) + SN[si,1];
                                err1:=1; {то генерация ошибки }
                            end;
                        end;

                    end;

                {Если это столбец имен типов ресурсов, то предварительно}
                {очищается столбец имен типов ресурсов от случайных данных в памяти}
                if ((err1 = 0) and (si = 12)) then for i:=1 to m do rtn^[i]:='';

```

```

{Чтение элементов матриц исходных данных}
if ((err1 = 0) and (not((si = 9) and (base_nx = 0)))) then
begin
    {Если это матрица исключений и число исключений = 0}
    i:= 1;    {то пропускаем чтение матрицы исключений}
    while ((i < m + 1) and (not(seekeof(fl))) and (err1 = 0)) do
    begin
        {Чтение строк с контролем признака конца файла}
        j:= 1;
        while ((j < n + 1) and (not(seekeoln(fl))) and (err1 = 0)) do
        begin
            {Чтение столбцов с контролем признака конца строки}
            {$I-}
            case si of
                {Чтение матрицы ресурсов физ. компьютеров}
                7: read(fl, hst^[base_nh*(i-1)+j-1]);
                {Чтение матрицы требований лог.серверов}
                8: read(fl, sst^[base_ns*(i-1)+j-1]);
                {Чтение матрицы исключений}
                9: read(fl, ext^[base_ns*(i-1)+j-1]);
                {Чтение столбца требований базовой ОС}
                10: read(fl, bst^[i]);
                {Чтение столбца маски критериев оптимизации}
                11: read(fl, cm^[i]);
                {Чтение столбца имен типов ресурсов}
                12: read(fl, rtn^[i]);
            end;
            {$I+}
        end;
        if IOResult <> 0 then {Если считанное данное некорректное}
        begin
            {то генерация ошибки}
            p_return_err:= SRC_FDATA_RD_ERR + chr(13) + chr(10)
            + 'Element[' + IntToStr(i) + ',' + IntToStr(j) +
            ']' in the ' + chr(13) + chr(10) + SN[si,2] +
            chr(13) + chr(10) + '(section ' + SN[si,1] + ')';
            err1:=1;
        end;

        if (err1 = 0) then {Проверка корректности}
        begin
            {элементов матриц и столбцов}
            err2:= 0;    {для задачи распределения}
            case si of
                {проверка ресурсов физ. компьютеров на >= 0}
                7: if (hst^[base_nh*(i-1)+j-1] < 0) then begin
                    err2:=1; p_return_err:='Element [' + IntToStr(i)
                    + ',' + IntToStr(j) + '] should be >= 0'; end;
                {проверка требований лог. серверов на >= 0}
                8: if (sst^[base_ns*(i-1)+j-1] < 0) then begin
                    err2:=1; p_return_err:='Element [' + IntToStr(i)
                    + ',' + IntToStr(j) + '] should be >= 0'; end;
                {проверка элементов в исключениях на = 0 или 1}
                9: if (not((ext^[base_ns*(i-1)+j-1] = 0) or
                    (ext^[base_ns*(i-1)+j-1] = 1))) then begin
                    err2:=1; p_return_err:='Element [' + IntToStr(i)
                    + ',' + IntToStr(j) + '] should be 0 or 1'; end;
                {проверка требований базовой ОС на >= 0}
                10: if (bst^[i] < 0) then begin
                    err2:=1; p_return_err:='Element [' + IntToStr(i)
                    + ']' should be >= 0'; end;
                {проверка элементов маски критериев на = 0 или 1}
                11: if (not((cm^[i] = 0) or (cm^[i] = 1))) then begin
                    err2:=1; p_return_err:='Element [' + IntToStr(i)
                    + ']' should be 0 or 1'; end;
            end;
            if (err2 <> 0) then {В случае некорректности данных}
            begin
                {генерация ошибки}
                p_return_err:= p_return_err + ' in the ' +

```



```

        chr(13) + chr(10) + SN[si,2] + chr(13) + chr(10)
        + '(section ' + SN[si,1] + ')';
        errl:= 1;
    end;
    end;
    if (errl = 0) then j:= j + 1; {Переход к следующему столбцу}
end;

{Если в строке при чтении столбцов раньше времени встретился}
{$I-}      {признак конца строки, то генерация ошибки}
if ((j < n + 1) and (seekeoln(fl)) and (errl = 0)) then
begin
    p_return_err:= SRC_F_EOL_ERR + 'in row ' + IntToStr(i)
    + ' in the ' + chr(13) + chr(10) + SN[si,2]
    + chr(13) + chr(10) + '(section ' + SN[si,1] + ')';
    errl:= 1;
end;
{$I+}

{Специальная проверка для имен типов ресурсов}
{проверяет чтобы строковые элементы не содержали}
{символы [ и ], которые используются для заголовков секций}
if ((errl = 0) and (si = 12)) then
begin
    s0:=rtn^[i];
    if (s0[1] = '[') or (s0[length(s0)] = ']') then
    begin {Если обнаружен символ [ или ]}
        p_return_err:= 'Illegal symbol [ or ]'
        + chr(13) + chr(10)
        + 'In row ' + IntToStr(i) + ' in the '
        + chr(13) + chr(10) + SN[si,2] + chr(13) + chr(10)
        + '(section ' + SN[si,1] + ')';
        errl:= 1; {то генерация ошибки}
    end;
end;

if (errl = 0) then
begin
    {$I-}
    readln(fl);
    {$I+}
    i:= i + 1; {Переход к следующей строке}
end;
end;

{Если после заголовка секции нет данных}
{$I-}
if ((i < m + 1) and (seekeof(fl)) and (errl = 0)) then
begin
    p_return_err:= SRC_F_EOF_ERR + 'in the ' +
    chr(13) + chr(10) + SN[si,2] + chr(13) + chr(10) +
    '(section ' + SN[si,1] + ')';
    errl:= 1; {то генерация ошибки}
end;
{$I+}
end;

if (errl = 0) then si:= si + 1; {Переход к следующей секции}
end;

if (errl = 0) then
begin
    temp1:= 0; {Проверка того, чтобы в маске критериев}
    i:= 1; {оптимизации была хотя бы одна "1"}
    while ((i <= base_nc) and (temp1 = 0)) do

```

```

begin
  if (cm^[i] = 1) then temp1:= 1;
  i:= i + 1; {Переход к следующей строке}
end;
if temp1 = 0 then
begin
  {Если ни одной единицы не найдено, то генерация ошибки}
  err1:= 1;
  p_return_err:= 'At least one element should be 1 in the '
  + chr(13) + chr(10) + SN[11,2]+ chr(13) + chr(10) +
  '(section ' + SN[11,1] + ')';
end;
end;

if (file_err=0) then {Если файл был открыт успешно, то закрываем его}
begin
  {$I-}
  CloseFile(f1);
  {$I+}
  if IOResult <> 0 then
  begin
    {В случае ошибки закрытия - генерация ошибки}
    p_return_err:= SRC_FCLOSE_ERR;
    err1:= 1;
  end;
end;

{Подготовка текста ошибки и выходного флага ошибки}
if (err1 <> 0) then p_return_flag:= 1;
p_return_err:= CUSTTASK_ERR_HEAD + p_return_err;
end;

{Процедура передачи исходных данных из полей}
{класса в поля исходных данных интерфейса}
{Входные параметры:
  p_base_nh:      Указатель на число физических компьютеров
  p_base_ns:      Указатель на число логических серверов
  p_base_nc:      Указатель на число типов ресурсов
  p_base_nx:      Указатель на число исключений для логических серверов
  p_max_radius:   Указатель на радиус зон поиска
  p_max_starts:   Указатель на число стартовых точек
  p_HSTGR:        Указатель на интерфейсную матрицу ресурсов физ. компьютеров
  p_SSTGR:        Указатель на интерфейсную матрицу требований лог. серверов
  p_EXTGR:        Указатель на интерфейсную матрицу исключений
  p_BSTGR:        Указатель на интерфейсный столбец требований базовой ОС
  p_CMGR:         Указатель на интерфейсный столбец маски критериев оптимизации
  p_RTNGR:        Указатель на интерфейсный столбец имен типов ресурсов}
{Выходные параметры:
  p_return_flag:   Флаг ошибки
  p_return_err:    Текст ошибки}
procedure TCustomTask.SaveTaskToCustomForm;
var
  i, j, m, n, si: integer; {временные переменные}
  err: byte;               {локальный флаг-защелка для ошибок}

begin
  err:= 0;                 {сброс основного флага-защелки процедуры}
  p_return_err:= '';       {сброс выходного текста ошибки}
  p_return_flag:= 0;       {сброс выходного флага ошибки}

  {Проверка указателей в полях класса и переданных во входных параметрах}
  if ((p_base_nh = nil) or (p_base_ns = nil) or
    (p_base_nc = nil) or (p_base_nx = nil) or
    (p_max_radius = nil) or (p_max_starts = nil) or
    (hst = nil) or (sst = nil) or

```

```

    ((base_nx > 0) and (ext = nil)) or
    (bst = nil) or (cm = nil) or (rtn = nil) or
    (p_HSTGR = nil) or (p_SSTGR = nil) or
    (p_EXTGR = nil) or (p_BSTGR = nil) or
    (p_CMGR = nil) or (p_RTNGR = nil))
then {Если один из них пустой то генерация ошибки}
begin
    p_return_err:= INVALID_PTR;
    err:= 1;
end;

{Передача основных параметров задачи из полей класса}
{в соответствующие параметры процедуры}
si:= 1;
while ((err = 0) and (si <= 6)) do
begin
    try
        case si of
            1: p_base_nh^:= base_nh;           {Число физ. компьютеров}
            2: p_base_ns^:= base_ns;           {Число лог. серверов}
            3: p_base_nc^:= base_nc;           {Число типов ресурсов}
            4: p_base_nx^:= base_nx;           {Число исключений}
            5: p_max_radius^:= max_radius;     {Мах. радиус поиска}
            6: p_max_starts^:= max_starts;     {Число старт-точек}
        end;
    except
        On EAccessViolation do
        begin
            p_return_err:= SRC_PARAM_WR_ACCVIOL_ERR;
            err:= 1;
        end;
    end;
    if (err = 0) then si:= si + 1; {Переход к следующему параметру задачи}
end;

{Передача данных из массивов исходных данных класса}
{в интерфейсные матрицы исходных данных}
si:= 7;
while ((err = 0) and (si <= 12)) do
begin
    case si of
        7: begin m:= base_nc; n:= base_nh; end; {Установка диапазонов}
        8: begin m:= base_nc; n:= base_ns; end; {индексных переменных}
        9: begin m:= base_nx; n:= base_ns; end; {в зависимости от того,}
        10: begin m:= base_nc; n:= 1; end;      {какая именно интерфейсная}
        11: begin m:= base_nc; n:= 1; end;      {матрица обрабатывается}
        12: begin m:= base_nc; n:= 1; end;      {на данный момент}
    end;

    if ((err = 0) and (not((si = 9) and (base_nx = 0)))) then
    begin {если секция = матрица исключений и число исключений = 0}
        i:= 1; {то пропускаем заполнение матрицы исключений}
        while ((i < m + 1) and (err = 0)) do
        begin
            j:= 1;
            while ((j < n + 1) and (err = 0)) do
            begin
                try
                    case si of
                        {Запись в интерфейсную матрицу ресурсов физ. компьютеров}
                        7: p_HSTGR^[i-1,j-1]:= hst^[base_nh*(i-1)+j-1];
                        {Запись в интерфейсную матрицу требований лог. серверов}
                        8: p_SSTGR^[i-1,j-1]:= sst^[base_ns*(i-1)+j-1];
                        {Запись в интерфейсную матрицу исключений}
                    end;
                except
                    On EAccessViolation do
                    begin
                        p_return_err:= SRC_PARAM_WR_ACCVIOL_ERR;
                        err:= 1;
                    end;
                end;
            end;
            j:= j + 1;
        end;
        i:= i + 1;
    end;
end;

```

```

9: p_EXTGR^[i-1,j-1]:= ext^[base_ns*(i-1)+j-1];
{Запись в интерфейсный столбец требований базовой ОС}
10: p_BSTGR^[i-1]:= bst^[i];
{Запись в интерфейсный столбец маски критериев оптим.}
11: p_CMGR^[i-1]:= cm^[i];
{Запись в интерфейсный столбец имен типов ресурсов}
12: p_RTNGR^[i-1]:= rtn^[i];
end;
except
  On EAccessViolation do {В случае нарушения доступа}
  begin {при записи данных в матрицы - генерация ошибки}
    p_return_err:= SRC_GRID_WR_ACCVIOL_ERR;
    err:= 1;
  end;
end;
if (err = 0) then j:= j + 1; {переход к следующему столбцу}
end;
if (err = 0) then i:= i + 1; {переход к следующей строке}
end;
end;
if (err = 0) then si:= si + 1; {переход к следующей матрице}
end;

{Подготовка текста ошибки и выходного флага ошибки}
if err <> 0 then p_return_flag:=1;
p_return_err:= CUSTASK_ERR_HEAD + p_return_err;
end;

{Процедура сохранения исходных данных}
{из полей класса в файл исходных данных}
{Входные параметры:
  p_filename: Строка, полный путь к файлу}
{Выходные параметры:
  p_return_flag: Флаг ошибки
  p_return_err: Текст ошибки}
procedure TCustomTask.SaveTaskToCustomFile;
var
  err1, file_err: byte; {локальные флаги-защелки для ошибок}
  f1: System.TextFile; {дескриптор файла}
  i, j, m, n, si: integer; {временные переменные}

begin
  {установка флага ошибки открытия файла: файл еще не открыт}
  file_err:=1;
  err1:=0; {сброс основного флага-защелки процедуры}
  p_return_err:= ''; {очищаем выходной текст ошибки}
  p_return_flag:= 0; {обнуляем выходной флаг ошибки}

  {Проверка указателей в полях класса и переданных во входных параметрах}
  if ((hst = nil) or (sst = nil) or
    ((base_nx > 0) and (ext = nil)) or
    (bst = nil) or (cm = nil) or (rtn = nil))
  then
  begin {Генерация ошибки если один из них пустой}
    p_return_err:= INVALID_PTR;
    err1:= 1;
  end;

  {Проверка имени файла}
  if ((err1 = 0) and (length(p_FileName) = 0)) then
  begin
    p_return_err:= SRC_FN_EMPTY_ERR;
    err1:=1; {Генерация ошибки если имя пустое}
  end;

```

```

if (err1 = 0) then      {Открытие файла на запись}
begin
  AssignFile (f1, p_FileName);
  {$I-}
  Rewrite (f1);
  {$I+}      {В случае успеха флаг ошибки открытия файла сбрасывается}
  if IOResult = 0 then file_err:= 0 else
  begin
    p_return_err:= SRC_FOPEN_WR_ERR;
    err1:= 1;  {Генерация ошибки в случае неудачи}
  end;
end;

{Передача основных параметров задачи из полей класса в секции 1-6 файла}
si:= 1;
while ((err1 = 0) and (si <= 6)) do    {Запись в цикле первых 6 секций}
begin                                  {данных в текстовый файл}
  {$I-}
  writeln (f1, SN[si,1]);              {Запись заголовка секции}
  {$I+}
  if (IOResult <> 0) then {генерация ошибки в случае неудачной записи}
  begin
    p_return_err:= SRC_FSECT_WR_ERR
    + chr(13) + chr(10) + SN[si,1];
    err1:=1;
  end;

  if (err1 = 0) then
  begin
    {$I-}                                {Запись данных}
    case si of
      1: writeln(f1, base_nh);           {Число физических компьютеров}
      2: writeln(f1, base_ns);           {Число логических серверов}
      3: writeln(f1, base_nc);           {Число типов ресурсов}
      4: writeln(f1, base_nx);           {Число исключений}
      5: writeln(f1, max_radius);        {Мак. радиус поиска}
      6: writeln(f1, max_starts);        {Число старт-точек}
    end;
    {$I+}
    if IOResult <> 0 then
    begin
      {генерация ошибки в случае неудачной записи}
      p_return_err:= SRC_FDATA_WR_ERR + chr(13) + chr(10)
      + SN[si,2] + chr(13) + chr(10) + ' (section ' + SN[si,1] + ')';
      err1:=1;
    end;
  end;

  if (err1 = 0) then
  begin
    {$I-}      {вставка дополнительной пустой строки в файл}
    writeln(f1);
    {$I+}
    si:= si + 1; {переход к следующей секции}
  end;
end;

{Передача данных из массивов исходных данных класса в секции 7-12 файла}
si:= 7;
while ((err1 = 0) and (si <= 12)) do
begin
  case si of
    7: begin m:= base_nc; n:= base_nh; end; {Установка диапазонов}
    8: begin m:= base_nc; n:= base_ns; end; {индексных переменных}
    9: begin m:= base_nx; n:= base_ns; end; {в зависимости от того,}

```

```

10: begin m:= base_nc; n:= 1; end;           {какая именно секция}
11: begin m:= base_nc; n:= 1; end;           {обрабатывается}
12: begin m:= base_nc; n:= 1; end;           {на данный момент}
end;

if ((err1 = 0) and (not((si = 9) and (base_nx = 0)))) then
begin      {Если это матрица исключений и число исключений = 0}
  {$I-}      {то запись заголовка для матрицы исключений пропускается}
  writeln (f1, SN[si,1]); {Запись заголовка секции}
  {$I+}
  if (IOResult <> 0) then
  begin      {генерация ошибки в случае неудачной записи}
    p_return_err:= SRC_FSECT_WR_ERR +
    chr(13) + chr(10) + SN[si,1];
    err1:=1;
  end;
end;

if ((err1 = 0) and (not((si = 9) and (base_nx = 0)))) then
begin      {Если это матрица исключений и число исключений = 0}
  i:= 1;      {то запись матрицы исключений пропускается}
  while ((i < m + 1) and (err1 = 0)) do
  begin
    j:= 1;
    while ((j < n + 1) and (err1 = 0)) do
    begin
      {$I-}
      case si of
        {Запись в матрицу ресурсов физ. компьютеров}
        7: write(f1, hst^[base_nh*(i-1)+j-1]);
        {Запись в матрицу требований лог. серверов}
        8: write(f1, sst^[base_ns*(i-1)+j-1]);
        {Запись в матрицу исключений}
        9: write(f1, ext^[base_ns*(i-1)+j-1]);
        {Запись в столбец требований базовой ОС}
        10: write(f1, bst^[i]);
        {Запись в столбец маски критериев оптимизации}
        11: write(f1, cm^[i]);
        {Запись в столбец имен типов ресурсов}
        12: write(f1, rtn^[i]);
      end;
      {$I+}
      if IOResult <> 0 then
      begin      {генерация ошибки в случае неудачной записи}
        p_return_err:= SRC_FDATA_WR_ERR + chr(13) + chr(10) +
        'Element[' + IntToStr(i) + ', ' + IntToStr(j) +
        ']' in the ' + chr(13) + chr(10) + SN[si,2]
        + chr(13) + chr(10) + '(section ' + SN[si,1] + ')';
        err1:=1;
      end;
      if (err1 = 0) then
      begin
        {$I-}      {Запись знака табуляции}
        if (j < n) then write(f1, chr(9));
        {$I+}
        j:= j + 1; {переход к следующему столбцу}
      end;
    end;
  end;

  if (err1 = 0) then
  begin
    {$I-}
    writeln(f1);      {Запись пустой строки}
    {$I+}
  end;
end;

```

```

        i:= i + 1;      {переход к следующей строке}
    end;
end;
end;
if (err1 = 0) then
begin
    {$I-}
    writeln(f1);      {Запись пустой строки}
    {$I+}
    si:= si + 1;      {переход к следующей секции}
end;
end;

{Close data file}
if (file_err=0) then {Если файл был открыт успешно, то закрываем его}
begin
    {$I-}
    CloseFile(f1);
    {$I+}              {В случае ошибки закрытия - генерация ошибки}
    if IOResult <> 0 then
    begin
        p_return_err:= SRC_FCLOSE_ERR;
        err1:= 1;
    end;
end;

{Подготовка текста ошибки и выходного флага ошибки}
if (err1 <> 0) then p_return_flag:= 1;
p_return_err:= CUSTASK_ERR_HEAD + p_return_err;
end;

{Процедура решения задачи по исходным данным в полях класса}
{Входные параметры:
    p_task_state:  Указатель на переменную для контроля состояния задачи
    p_stop_flag:   Указатель на флаг принудительного останова извне}
{Выходные параметры:
    p_return_flag:  Флаг ошибки
    p_return_err:   Текст ошибки}
procedure TCustomTask.Solve;
var
    err, mem_err: byte; {локальные флаги-защелки для ошибок}
    dtask_flag: byte;   {возвратный флаг ошибки задачи распределения}
    dtask_error: string; {возвратный текст ошибки задачи распределения}

begin
    {Установка флага ошибки выделения памяти, т.к. не выделена}
    mem_err:= 1;
    p_return_flag:= 0; {сброс выходного флага ошибки}
    p_return_err:= ''; {сброс выходного текста ошибки}
    err:= 0;          {сброс локального флага ошибки}

    {Проверка указателей в полях класса и переданных во входных параметрах}
    if ((hst = nil) or (sst = nil) or ((base_nx > 0) and (ext = nil))
        or (bst = nil) or (cm = nil) or (rtn = nil)
        or (p_stop_flag = nil) or (p_task_state = nil)) then
    begin
        p_return_err:= INVALID_PTR;
        err:= 1;      {Генерация ошибки если один из них пустой}
    end;

    {Попытка выделения памяти для массивов результирующих данных класса}
    if (err = 0) then
    begin
        try

```

```

    GetMem(dt,base_nh*base_ns*sizeof(dt^[1]));
    GetMem(rl,base_nh*base_nc*sizeof(rl^[1]));
except
    {В случае нехватки памяти}
    On EOutOfMemory do
    begin
        {генерация соответствующей ошибки}
        p_return_err:= LOCAL_DATA_MEM_ALLOC_ERR;
        err:= 1;
    end;
end;

{Дополнительная проверка указателей}
if ((err = 0) and (not((dt=nil) or (rl = nil))))
then mem_err:=0
else
    begin {В случае если один из указателей пустой - то генерация ошибки}
        p_return_err:= LOCAL_DATA_MEM_ALLOC_ERR;
        err:= 1;
    end;
end;

if (err = 0) then
begin
    {Вызов унаследованного конструктора для инициализации полей класса,}
    {временных данных и дополнительного анализа данных}
    Inherited Init (base_nh, base_ns, base_nc, base_nx,
max_radius, max_starts, hst, sst, ext, bst, cm,
dt, rl, dtask_flag, dtask_error);

    {если конструктор выполнен без ошибок, то}
    {вызываем унаследованный метод решения задачи}
    if (dtask_flag = 0) then
    Inherited Solve (dtask_flag, dtask_error, p_stop_flag, p_task_state);

    Inherited Done; {вызываем унаследованный деструктор}

    {если при решении задачи возникла ошибка (или ее останов)}
    if (dtask_flag <> 0) then
    begin
        {то сохраняем текст ошибки}
        p_return_err:= dtask_error;
        err:= 1; {генерируем ошибку}
    end;
end;

{Подготовка текста ошибки и выходного флага ошибки}
if err <> 0 then p_return_flag:=1;
p_return_err:= CUSTTASK_ERR_HEAD + p_return_err;
end;

{Процедура инициализации полей результирующих данных}
{класса результирующими данными из полей интерфейса}
{Входные параметры:
    p_res_nh: Указатель на число физ. компьютеров для результирующих данных
    p_res_ns: Указатель на число лог. серверов для результирующих данных
    p_res_nc: Указатель на число типов ресурсов для результирующих данных
    p_res_nx: Указатель на число исключений для результирующих данных
    p_DTGR: Указатель на интерфейсную матрицу распределения
    p_RLGR: Указатель на интерфейсную матрицу загрузки ресурсов
    p_DTHL: Указатель на интерфейсную строку нераспределенных лог. серверов
    p_RLHL: Указатель на интерфейсную строку имен типов ресурсов
{Выходные параметры:
    p_return_flag: Флаг ошибки
    p_return_err: Текст ошибки}
procedure TCustomTask.InitResultFromCustomForm;
var

```



```

err: byte;                {локальный флаг-защелка для ошибок}
mem_err: byte;            {флага ошибки выделения памяти}
i, js, jh: integer;      {временные переменные}

begin
  {Установка флага ошибки выделения памяти, т.к. она еще не выделена}
  mem_err:= 1;
  p_return_flag:= 0;      {сброс выходного флага ошибки}
  p_return_err:= '';      {сброс выходного текста ошибки}
  err:= 0;                {сброс локального флага ошибки}

  {Проверка указателей переданных во входных параметрах}
  if ((p_res_nh = nil) or (p_res_ns = nil) or
      (p_res_nc = nil) or (p_res_nx = nil) or
      (p_DTGR = nil) or (p_RLGR = nil) or
      (p_DTHL = nil) or (p_RLHL = nil)) then
    begin
      p_return_err:= INVALID_PTR;
      err:= 1;            {Генерация ошибки если один из них пустой}
    end;

  {Передача параметров размерностей результирующих данных}
  {из параметров процедуры в соответствующие поля класса}
  if (err = 0) then
    begin
      try
        base_nh:= p_res_nh^; {число физ.компьютеров для результирующих данных}
        base_ns:= p_res_ns^; {число лог.серверов для результирующих данных}
        base_nc:= p_res_nc^; {число типов ресурсов для результирующих данных}
        base_nx:= p_res_nx^; {число исключений для результирующих данных}
      except
        On EAccessViolation do {если при чтении параметров процедуры}
          begin                {возникло нарушение доступа}
            p_return_err:= RES_PARAM_RD_ACCVIOL_ERR;
            err:= 1;           {то генерация ошибки}
          end;
        end;
      end;

    {проверка параметров размерностей на допустимость}
    if ((err = 0) and (((base_nh < 0) or (base_nh > nh_max))
        or ((base_ns < 0) or (base_ns > ns_max))
        or ((base_nc < 0) or (base_nc > nc_max)))) then
      begin
        {если нарушены допустимые границы}
        err:= 1;    {то установка флага ошибки}
        p_return_err:= RES_INV_DIM;
      end;

    {проверка параметров размерностей на 0}
    if ((err = 0) and (not((base_nh > 0) and
        (base_ns > 0) and (base_nc > 0)))) then
      begin
        {если размерности нулевые}
        err:= 1;    {то установка флага ошибки}
        p_return_err:= RES_EMPTY_DATA;
      end;

    {Попытка выделения памяти для массивов результирующих данных класса}
    if (err = 0) then
      begin
        try
          GetMem(rtn,base_nc*sizeof(rtn^[1]));
          GetMem(dt,base_nh*base_ns*sizeof(dt^[1]));
          GetMem(rl,base_nh*base_nc*sizeof(rl^[1]));
        except

```

```

On EOutOfMemory do
begin
    {В случае нехватки памяти}
    p_return_err:= LOCAL_DATA_MEM_ALLOC_ERR;
    err:= 1; {генерация соответствующей ошибки}
end;
end;

{Дополнительная проверка указателей}
if ((err = 0) and (not((dt = nil) or (rl = nil) or (rtn = nil))))
then mem_err:=0
else {В случае если один из указателей пустой - то генерация ошибки}
begin
    p_return_err:= LOCAL_DATA_MEM_ALLOC_ERR;
    err:= 1;
end;
end;

{Передача данных из интерфейсных матриц результирующих данных}
{в массивы результирующих данных класса}
if (err = 0) then
begin
    try
        i:=1;
        while ((i <= base_nc) and (err = 0)) do
            begin {чтение имен типов ресурсов}
                rtn^[i]:= p_RLHL^[i - 1];
                i:= i + 1; {переход к следующему столбцу}
            end;

            jh:=1; {Первоначальная очистка матрицы распределения}
            while ((jh <= base_nh) and (err = 0)) do
                begin {}
                    js:= 1;
                    while ((js <= base_ns) and (err = 0)) do
                        begin {Запись "0" в ячейку матрицы}
                            dt^[base_ns*(jh-1)+js-1]:= 0;
                            js:= js + 1; {переход к следующему лог. серверу}
                        end;
                        jh:= jh + 1; {переход к следующему физ. компьютеру}
                    end;

                    jh:=1; {Для каждого физического компьютера}
                    while ((jh <= base_nh) and (err = 0)) do
                        begin {устанавливаем "1" в тех столбцах, индексы которых}
                            js:= 1; {соответствуют лог. серверам, распределенных на компьютер}
                            while ((js <= base_ns) and (err = 0)) do
                                begin
                                    if (p_DTGR^[jh-1,js -1] <> 0) then
                                        dt^[base_ns*(jh-1)+(p_DTGR^[jh-1,js-1])-1]:= 1;
                                    js:= js + 1; {переход к следующему лог. серверу}
                                end;

                                i:= 1;
                                while ((i <= base_nc) and (err = 0)) do
                                    begin
                                        {Для текущего физ. компьютера считываем данные}
                                        {о его загрузке по текущему типу ресурса}
                                        rl^[base_nc*(jh-1)+i-1]:= p_RLGR^[jh-1,i-1];
                                        i:= i + 1; {переход к следующему типу ресурсов}
                                    end;
                                    jh:= jh + 1; {переход к следующему физ. компьютеру}
                                end;
                            end;
                        end;

                    except {если при чтении результатов в интерфейсных матрицах и строках}
                        On EAccessViolation do

```

```

begin
    {возникло нарушение доступа}
    p_return_err:= RES_GRID_RD_ACCVIOL_ERR;
    err:= 1;
    {то генерация ошибки}
end;
end;
end;

{Подготовка текста ошибки и выходного флага ошибки}
if err <> 0 then p_return_flag:=1;
p_return_err:= CUSTTASK_ERR_HEAD + p_return_err;
end;

{Процедура сохранения полей результирующих данных}
{класса в поля результирующих данных интерфейса}
{Входные параметры:
    p_res_nh: Указатель на число физ. компьютеров для результирующих данных
    p_res_ns: Указатель на число лог. серверов для результирующих данных
    p_res_nc: Указатель на число типов ресурсов для результирующих данных
    p_res_nx: Указатель на число исключений для результирующих данных
    p_DTGR: Указатель на интерфейсную матрицу распределения
    p_RLGR: Указатель на интерфейсную матрицу загрузки ресурсов
    p_DTHL: Указатель на интерфейсную строку нераспределенных лог. серверов
    p_RLHL: Указатель на интерфейсную строку имен типов ресурсов
{Выходные параметры:
    p_return_flag: Флаг ошибки
    p_return_err: Текст ошибки}
procedure TCustomTask.SaveResultToCustomForm;
var
    err: byte;
    i, j, js, jh, k: integer;
    {локальный флаг-защелка для ошибок}
    {временные переменные}

begin
    p_return_flag:= 0;
    p_return_err:= '';
    err:= 0;
    {сброс выходного флага ошибки}
    {сброс выходного текста ошибки}
    {сброс локального флага ошибки}

    {Проверка указателей переданных во входных параметрах}
    {и полей класса - указателей на результирующие данные}
    if ((p_res_nh = nil) or (p_res_ns = nil) or
        (p_res_nc = nil) or (p_res_nx = nil) or
        (rtn = nil) or (dt = nil) or (rl = nil)
        or (p_DTGR = nil) or (p_RLGR = nil) or
        (p_DTHL = nil) or (p_RLHL = nil)) then
    begin
        p_return_err:= INVALID_PTR;
        err:= 1;
        {Генерация ошибки если один из них пустой}
    end;

    {Передача параметров размерностей результирующих данных}
    {из полей класса в соответствующие параметры процедуры}
    if (err = 0) then
    begin
        try
            p_res_nh^:= base_nh; {число физ.компьютеров для результирующих данных}
            p_res_ns^:= base_ns; {число лог.серверов для результирующих данных}
            p_res_nc^:= base_nc; {число типов ресурсов для результирующих данных}
            p_res_nx^:= base_nx; {число типов ресурсов для результирующих данных}
        except
            On EAccessViolation do
                {если при записи}
                begin
                    {возникло нарушение доступа}
                    p_return_err:= RES_PARAM_WR_ACCVIOL_ERR;
                    err:= 1;
                    {то генерация ошибки}
                end;
            end;
        end;
    end;
end;

```

```

end;

{Передача данных из массивов результирующих данных класса}
{в интерфейсных матрицы результирующих данных}
if (err = 0) then
begin
  try
    i:=1;
    while ((i <= base_nc) and (err = 0)) do
    begin
      {запись имен типов ресурсов в интерфейсную строку имен}
      p_RLHL^[i - 1]:= rtn^[i];
      i:= i + 1; {переход к следующему столбцу}
    end;

    jh:=1; {Для каждого физического компьютера}
    while ((jh <= base_nh) and (err = 0)) do
    begin
      {записываем индексы логических серверов,}
      j:= 0; {распределенных на него}
      js:= 1;
      while ((js <= base_ns) and (err = 0)) do
      begin
        {для этого в матрице распределения задачи ищем "1"}
        {в каждой строке (соответствует физическому компьютеру)}
        {индексы столбцов с "1" и есть искомые индексы лог. серверов}
        {(столбцы соответствуют логическим серверам)}
        if dt^[base_ns*(jh-1)+js-1]=1 then
        begin
          p_DTGR^[jh-1,j]:= js;
          j:= j + 1; {в результате в интерфейсной матрице}
        end; {распределения в каждой строке будут компактно}
        {перечислены индексы лог. серверов, а не "0" и "1".}
        js:= js + 1; {переход к следующему лог. серверу}
      end;
      if (j = 0) then p_DTGR^[jh-1,0]:=0; {если ни одной "1" не нашлось}
      {строка в интерфейсной матрице распределения останется пустой}

      {Для текущего физ. компьютера считываем данные}
      i:= 1; {о его загрузке по каждому типу ресурса}
      while ((i <= base_nc) and (err = 0)) do
      begin
        p_RLGR^[jh-1,i-1]:= rl^[base_nc*(jh-1)+i-1];
        i:= i + 1; {переход к следующему типу ресурсов}
      end;
      jh:= jh + 1; {переход к следующему физ. компьютеру}
    end;

    {Сканируем матрицу распределения задачи в транспонированном}
    {виде, для того чтобы найти те логические серверы, которые}
    {не были распределены - нулевой столбец для лог. сервера}
    j:=0; {сброс общего числа нераспределенных лог. серверов}
    js:=1;
    while ((js <= base_ns) and (err = 0)) do
    begin
      k:=0; {сброс признака, что лог. сервер распределен}
      jh:= 1;
      while ((jh <= base_nh) and (err = 0) and (k = 0)) do
      begin
        {если элемент матрицы ненулевой, то}
        {установка признака, что лог. сервер распределен}
        if (dt^[base_ns*(jh-1)+js-1] = 1) then k:=1;
        jh:= jh + 1; {переход к следующему физ. компьютеру}
      end;

      {если сброшен признак, что лог. сервер распределен}
    end;
  end;
end;

```

```

        if (k = 0) then
            begin
                {Запись в интерфейсную строку индекса}
                p_DTHL^[j] := js; {нераспределенного лог. серверов}
                j := j + 1; {увеличение общего числа нераспред лог. серверов}
            end;
            js := js + 1; {переход к следующему лог. серверу}
        end;
    except {если при записи результатов в интерфейсные матриц и строки}
        On EAccessViolation do {возникло нарушение доступа}
            begin
                p_return_err := RES_GRID_WR_ACCVIOL_ERR;
                err := 1; {то генерация ошибки}
            end;
        end;
    end;

    {Подготовка текста ошибки и выходного флага ошибки}
    if err <> 0 then p_return_flag := 1;
    p_return_err := CUSTTASK_ERR_HEAD + p_return_err;
end;

{Процедура сохранения полей результирующих}
{данных класса в файл результирующих данных}
{Входные параметры:
    p_filename: Строка, полный путь к файлу
Выходные параметры:
    p_return_flag: Флаг ошибки
    p_return_err: Текст ошибки}
procedure TCustomTask.SaveResultToCustomFile;
var
    f1: System.TextFile; {дескриптор файла}
    err1: byte; {локальный флаг ошибки}
    file_err: byte; {локальный флаг ошибки открытия файла}
    i, jh, js, j, k: integer; {временные переменные}

begin
    {установка ошибки открытия файла (файл еще не открыт)}
    file_err := 1;
    p_return_flag := 0; {сброс возвратного флага ошибок}
    p_return_err := ''; {сброс возвратного текста ошибок}
    err1 := 0; {сброс локального флага ошибок}

    {проверка полей класса - указателей на результирующие данные}
    if ((err1 = 0) and ((dt = nil) or (rl = nil) or (rtn = nil))) then
        begin
            {если указатели пустые}
            err1 := 1; {то установка флага ошибки}
            p_return_err := INVALID_PTR;
        end;

    {проверка входного имени файла}
    if ((err1 = 0) and (length(p_FileName) = 0)) then
        begin
            p_return_err := RES_FN_EMPTY_ERR; {если имя пустое то генерация ошибки}
            err1 := 1;
        end;

    {попытка открытия файла на запись}
    if (err1 = 0) then
        begin
            AssignFile (f1, p_FileName);
            {$I-}
            Rewrite (f1); {Если успешно, то сброс флага}
            {$I+} {ошибки открытия файла}
            if IOResult = 0 then file_err := 0 else

```

```

begin
  p_return_err:= RES_FOPEN_WR_ERR;
  errl:= 1; {иначе генерация ошибки}
end;
end;

{Запись матрицы распределения в файл}
if (errl = 0) then
begin
  {$I-} {Запись заголовка матрицы распределения}
  writeln (f1);
  writeln (f1, 'Distribution table:');
  writeln (f1);
  {$I+}
  if (IOResult <> 0) then {Проверка на ошибку операции записи}
  begin p_return_err:= RES_FDATA_WR_ERR; errl:= 1; end;

  jh:= 1; {Запись матрицы распределения}
  while ((errl = 0) and (jh <= base_nh)) do
  begin
    js:= 1; {Запись заголовков строк матрицы}
    {$I-}
    write (f1, 'H' + IntToStr(jh) + ':' + chr(9));
    {$I+}
    if (IOResult <> 0) then {Проверка на ошибку операции записи}
    begin p_return_err:= RES_FDATA_WR_ERR; errl:= 1; end;

    j:= 0; {сброс счетчика лог. серверов}
    {распределенных на физический компьютер}
    while (errl = 0) and (js <= base_ns) do
    begin
      {$I-} {Запись элементов матрицы распределения в файл}
      if (dt^[base_ns*(jh-1)+js-1] = 1) then
      begin {Если элемент ненулевой, то запись его в файл}
        write (f1, 'S' + IntToStr(js) + chr(9));
        j:= j + 1; {увеличение счетчика лог. серверов}
      end; {распределенных на физический компьютер}
      {$I+}
      if (IOResult <> 0) then {Проверка на ошибку операции записи}
      begin p_return_err:= RES_FDATA_WR_ERR; errl:= 1; end;
      js:= js + 1; {переход на следующий столбец в матрице}
    end;

    if (j = 0) then {если счетчик лог. серверов, распределенных}
    begin {на физ.компьютер остался = 0, то значит физ.компьютер}
      {$I-} {не использовался, записываем об этом в файл}
      write (f1, 'Unused');
      {$I+}
      if (IOResult <> 0) then {Проверка на ошибку операции записи}
      begin p_return_err:= RES_FDATA_WR_ERR; errl:= 1; end;
    end;
    {$I-}
    writeln (f1); {Переход на следующую строку в файле}
    {$I+}
    if (IOResult <> 0) then {Проверка на ошибку операции записи}
    begin p_return_err:= RES_FDATA_WR_ERR; errl:= 1; end;
    jh:= jh + 1; {переход на следующую строку в матрице}
  end;
end;

{Запись строки нераспределенных лог. серверах}

{Сканируем матрицу распределения задачи в транспонированном}
{виде, для того чтобы найти те логические серверы, которые}

```

```

{не были распределены - нулевой столбец для лог. сервера}
if (err1 = 0) then
begin
  {$I-}                                {Запись заголовка о нераспределенных лог. серверах}
  writeln (f1);
  writeln (f1, 'Remaining undistributed logical servers:');
  writeln (f1);
  {$I+}
  if (IOResult <> 0) then {Проверка на ошибку операции записи}
  begin p_return_err:= RES_FDATA_WR_ERR; err1:= 1; end;

  j:= 0;                                {сброс общего числа нераспределенных лог. серверов}
  js:= 1;
  while ((err1 = 0) and (js <= base_ns)) do
  begin
    k:= 0;                                {сброс признака, что лог. сервер распределен}
    jh:= 1;
    while ((err1 = 0) and (jh <= base_nh) and (k = 0)) do
    begin
      {если элемент матрицы ненулевой, то}
      {установка признака, что лог. сервер распределен}
      if dt^[base_ns*(jh-1)+js-1]=1 then k:= 1;
      {переход к следующему элементу в интерфейсной строке}
      jh:= jh + 1;
    end;

    {если сброшен признак, что лог. сервер распределен}
    if (k = 0) then
    begin
      j:= j + 1; {то увеличение общего числа нераспред лог. серверов}
      {$I-}      {запись индекса нераспределенного лог. сервера}
      write(f1, 'S' + IntToStr(js) + chr(9));
      {$I+}
    end;

    if (IOResult <> 0) then {Проверка на ошибку операции записи}
    begin p_return_err:= RES_FDATA_WR_ERR; err1:= 1; end;
    js:= js + 1;          {переход на следующую строку в матрице}
  end;

  if (j = 0) then {если счетчик нераспределенных лог. серверов}
  begin          {остался равным 0, то нераспределенных лог. серверов нет}
    {$I-}
    write (f1, 'None');      {запись об этом в файл}
    {$I+}
    if (IOResult <> 0) then {Проверка на ошибку операции записи}
    begin p_return_err:= RES_FDATA_WR_ERR; err1:= 1; end;
  end;
end;

{Запись матрицы загрузки ресурсов}
if (err1 = 0) then
begin
  {$I-}
  writeln (f1);              {Запись заголовка матрицы загрузки ресурсов}
  writeln (f1);
  writeln (f1, 'Resource usage table:');
  writeln (f1);
  {$I+}
  if (IOResult <> 0) then {Проверка на ошибку операции записи}
  begin p_return_err:= RES_FDATA_WR_ERR; err1:= 1; end;

  {$I-}
  write (f1, chr(9));        {отступ от начала строки в файле}

```

```

{$I+}
if (IOResult <> 0) then      {Проверка на ошибку операции записи}
begin p_return_err:= RES_FDATA_WR_ERR; err1:= 1; end;

js:= 1;
while (err1 = 0) and (js <= base_nc) do
begin
    {$I-}                {Запись заголовков столбцов для матрицы}
    write (f1, 'R' + IntToStr(js) + ':' + chr(9));
    {$I+}
    if (IOResult <> 0) then      {Проверка на ошибку операции записи}
    begin p_return_err:= RES_FDATA_WR_ERR; err1:= 1; end;
    js:= js + 1;      {переход к следующему заголовочному столбцу}
end;
{$I-}
writeln(f1);      {запись пустых строк в файле}
writeln(f1);
{$I+}
if (IOResult <> 0) then      {Проверка на ошибку операции записи}
begin p_return_err:= RES_FDATA_WR_ERR; err1:= 1; end;

jh:= 1;
while ((err1 = 0) and (jh <= base_nh)) do
begin
    js:= 1;          {Запись заголовков строк для матрицы}
    {$I-}
    write (f1, 'H' + IntToStr(jh) + ':' + chr(9));
    {$I+}
    if (IOResult <> 0) then      {Проверка на ошибку операции записи}
    begin p_return_err:= RES_FDATA_WR_ERR; err1:= 1; end;

    while (err1 = 0) and (js <= base_nc) do
    begin
        {$I-}
        write (f1, FloatToStrF (rl^[base_nc*(jh-1)+(js-1)],ffFixed,5,2)
            + '%' + chr(9));
        {$I+}
        if (IOResult <> 0) then      {Проверка на ошибку операции записи}
        begin p_return_err:= RES_FDATA_WR_ERR; err1:= 1; end;
        js:= js + 1;      {переход на следующий столбец в матрице}
    end;
    {$I-}
    writeln (f1);      {переход на следующую строку в файле}
    {$I+}
    if (IOResult <> 0) then      {Проверка на ошибку операции записи}
    begin
        p_return_err:= RES_FDATA_WR_ERR;
        err1:= 1;
    end;
    jh:= jh + 1;      {переход на следующую строку в матрице}
end;
end;

{Запись в файл дополнительного столбца имен типов ресурсов -}
{заголовков R(j) в матрицы загрузки ресурсов}
if (err1 = 0) then
begin
    {$I-}                {Запись заголовка столбца}
    writeln (f1);
    writeln (f1, 'Resources typenames:');
    writeln (f1);
    {$I+}
    if (IOResult <> 0) then      {Проверка на ошибку операции записи}
    begin p_return_err:= RES_FDATA_WR_ERR; err1:= 1; end;

```



```

i:= 1;
while ((err1 = 0) and (i <= base_nc)) do
begin
    {Запись элемента из интерфейсной строки}
    {$I-}
    {имен типов ресурсов в очередную строку файла}
    writeln(f1, 'R'+ IntToStr(i) + ':' + chr(9) + rtn^[i]);
    {$I+}
    if (IOResult <> 0) then
        {Проверка на ошибку операции записи}
        begin p_return_err:= RES_FDATA_WR_ERR; err1:= 1; end;
    i:= i + 1;
    {переход к следующему элементу в интерфейсной строке}
end;

end;

{Если файл был открыт}
if (file_err=0) then
begin
    {$I-}
    CloseFile(f1);
    {то закрываем его}
    {$I+}
    if IOResult <> 0 then
        {в случае ошибки при закрытии}
        begin
            p_return_err:= RES_FCLOSE_ERR;
            {генерация ошибки}
            err1:= 1;
        end;
    end;

    {Подготовка выходного флага ошибки}
    if (err1 <> 0) then p_return_flag:=1;
    p_return_err:= CUSTASK_ERR_HEAD + p_return_err;
end;

{Деструктор вспомогательного класса}
{освобождает память и уничтожает экземпляр класса}
destructor TCustomTask.Done;
begin
    {Освобождение памяти всех динамических массивов класса}
    if (hst <> nil) then FreeMem(hst,base_nc*base_nh*sizeof(hst^[1]));
    if (sst <> nil) then FreeMem(sst,base_nc*base_ns*sizeof(sst^[1]));
    if ((base_nx > 0) and (ext <> nil))
    then FreeMem(ext,base_nx*base_ns*sizeof(ext^[1]));
    if (bst <> nil) then FreeMem(bst,base_nc*sizeof(bst^[1]));
    if (cm <> nil) then FreeMem(cm,base_nc*sizeof(cm^[1]));
    if (rtn <> nil) then FreeMem(rtn,base_nc*sizeof(rtn^[1]));
    if (dt <> nil) then FreeMem(dt,base_nh*base_ns*sizeof(dt^[1]));
    if (rl <> nil) then FreeMem(rl,base_nh*base_nc*sizeof(rl^[1]));
    hst:= nil; sst:= nil; ext:= nil;
    {Обнуление указателей}
    bst:= nil; dt:= nil; rl:= nil; cm:= nil; rtn:= nil;
end;

end.

```

## Исходный код математического модуля

```
{Модуль - математическое ядро приложения. Содержит ключевую иерархию}
{классов, необходимую для решения расширенной задачи распределения}
unit MYSOLVER;

{Использование математического сопроцессора и отказ от эмуляции}
{$N+}
{$E-}

{Оператор @ всегда генерирует типизированные указатели}
{$T+}

{Note: In case of 6-byte real type, FPU operations is very slow!}
{So we use 8-byte double type: operations with the double}
{type 7 - 12 time faster than operations with 6-byte real type!}

interface
  {Используемые модули}
uses SysUtils;

const
  {Максимально допустимые значения параметров размерностей}
  {Dimension Limits}
  nh_max = 1000;      {максимальное число физических компьютеров}
  ns_max = 1000;      {максимальное число логических серверов}
  nc_max = 1000;      {максимальное число типов ресурсов}
  nx_max = 1000;      {максимальное число исключений}

  {Текстовые константы: короткие и полные имена полей данных.}
  {Task's Data Section Names and Strings}

  {Текстовые константы абстрактного класса}
  {ABSTRACT Task's constants}
  ABSTRACT_SN: array [1..2, 1..2] of string[64] =
    (('[MAXRADIUS]', 'Maximum radius of searching in boolean optimiztion'),
    ('[MAXSTARTS]', 'Quantity of start-points in boolean optimiztion'));

  {Текстовые константы класса задачи безусловной булевой оптимизации}
  {Boolean Optimization Task's constants}
  BOPT_SN: array [1..5, 1..2] of string[64] =
    (('[N]', 'Number of the variables'),
    ('[C]', 'C-vector'),
    ('[X-opt]', 'Result X-vector'),
    ('[L-max]', 'Result L-max value'),
    ('[ITC]', 'Total iterations counter'));

  {Текстовые константы класса задачи условной булевой оптимизации}
  {Extended Boolean Optimization Task's constants}
  BOPTTEXT_SN: array [1..3, 1..2] of string[64] =
    (('[M]', 'Number of the constraints'),
    ('[A]', 'A-matrix'),
    ('[B]', 'B-vector'));

  {Текстовые константы класса задачи распределения}
  {Distribution Task's constants}
  DIST_SN: array [1..9, 1..2] of string[64] =
    (('[NH]', 'Number of the physical computers'),
    ('[NS]', 'Number of the logical servers'),
    ('[NC]', 'Number of the resources types'),
    ('[HST]', 'Resources table of the physical computers'),
    ('[SST]', 'Requirements Table of the logical servers'),
    ('[BST]', 'Host OS requirements vector'),
    ('[CM]', 'Resource usage optimizing mask vector'),
```

```

(['DT'], 'Result distribution table'),
(['RL'], 'Result resource usage table'));

{Текстовые константы класса расширенной задачи распределения}
{Extended Distribution Task's constants}
DISTEXT_SN: array [1..2, 1..2] of string[64] =
  (('NX', 'Number of the exclusions for the logical servers'),
   ('EXT', 'Exclusion table for the logical servers'));

{Текстовые константы сообщений об ошибках}
ABSTRACT_ERR_HEAD = 'Abstract Local Search Task Error.'
+ chr(13) + chr(10) + chr(13) + chr(10);

BOPTTEXT_ERR_HEAD = 'Extended Boolean Optimization Task Error.'
+ chr(13) + chr(10) + chr(13) + chr(10);

DISTEXT_ERR_HEAD = 'Extended Distribution Task Error.'
+ chr(13) + chr(10) + chr(13) + chr(10);

S_MEM_ALLOC_ERR = 'Out of memory.' + chr(13) + chr(10);

S_ACCVIOL_ERR = 'Access violation.' + chr(13) + chr(10);

S_NOTIMP_ERR = 'Method is not implemented.' + chr(13) + chr(10);

S_SRC_INV_PTR = 'Invalid pointer for the:' + chr(13) + chr(10);

S_SRC_INV_PTRS = 'One or more pointers are invalid!';

S_LOCAL_DATA_MEM_ALLOC_ERR = S_MEM_ALLOC_ERR +
'Cannot get space in the system memory for the local data.';

S_BOPTTEXT_MEM_ERR = S_MEM_ALLOC_ERR +
'Cannot create and solve the Extended Boolean Optimization Task.';

S_BOPTTEXT_ACCVIOL_ERR = S_ACCVIOL_ERR +
'Cannot create and solve the Extended Boolean Optimization Task.';

S_STOP_FLAG_TRUE = 'User has aborted the current operation!';

S_STOP_FLAG_FAILED = S_ACCVIOL_ERR +
'Occurred on checking the StopFlag.';

S_TASKSTATE_WR_ERR = S_ACCVIOL_ERR +
'Occurred on writing the TaskState.';

S_ABSTRACT_NOTIMP_ERR = S_NOTIMP_ERR +
'This is the method of the root abstract class.';

S_BOPT_NOTIMP_ERR = S_NOTIMP_ERR +
'This is the method of the top-level task class.';

S_DT_NOTIMP_ERR = S_NOTIMP_ERR +
'This is the method of the top-level task class.';

type
{Внимание! Несмотря на явно указанные размерности массивов, память для них
 выделяется динамически, взаимодействие осуществляется только через указатели}

{тип вектора целевой функции задачи безусловной булевой оптимизации}
TC_Vector = array [1..ns_max] of double;
PC_Vector = ^TC_Vector;

{тип матрицы (левая часть ограничений) условной булевой оптимизации}

```

```

TA_Matrix = array [0..((nc_max+nx_max)*ns_max)-1] of longint;
PA_Matrix = ^TA_Matrix;

{тип вектора (правая часть ограничений) условной булевой оптимизации}
TB_Vector = array [1..(nc_max+nx_max)] of longint;
PB_Vector = ^TB_Vector;

{тип матрицы ресурсов физических компьютеров задачи распределения}
THardCompsRsrcTable = array [0..((nc_max + 1)*nh_max)-1] of longint;
PHardCompsRsrcTable = ^THardCompsRsrcTable;

{тип матрицы требований логических серверов задачи распределения}
TSoftSrvsReqTable = array [0..((nc_max + 1)*ns_max)-1] of longint;
PSoftSrvsReqTable = ^TSoftSrvsReqTable;

{тип матрицы исключений для расширенной задачи распределения}
TExclusionTable = array [0..(nx_max*ns_max)-1] of byte;
PExclusionTable = ^TExclusionTable;

{тип вектора требований базовой ОС задачи распределения}
THostOSReqTable = array [1..nc_max] of longint;
PHostOSReqTable = ^THostOSReqTable;

{тип вектора маски критериев оптимизации задачи распределения}
TCriteriaMaskVector = array [1..nc_max] of byte;
PCriteriaMaskVector = ^TCriteriaMaskVector;

{тип матрицы распределения логических серверов задачи распределения}
TDistributionTable = array [0..(nh_max*ns_max)-1] of byte;
PDistributionTable = ^TDistributionTable;

{тип матрицы загрузки ресурсов компьютеров задачи распределения}
TResourceLoadingTable = array [0..(nc_max*nh_max)-1] of Single;
PResourceLoadingTable = ^TResourceLoadingTable;

{тип вектора индексов оставшихся физических компьютеров}
TRemainHardCompsIndexes = array [1..nh_max] of integer;
PRemainHardCompsIndexes = ^TRemainHardCompsIndexes;

{тип вектора индексов оставшихся логических серверов}
TRemainSoftSrvsIndexes = array [1..ns_max] of integer;
PRemainSoftSrvsIndexes = ^TRemainSoftSrvsIndexes;

{тип вектора индексов невырожденных дополнительных ограничений - исключений}
TValidExclusionIndexes = array [1..nx_max] of integer;
PValidExclusionIndexes = ^TValidExclusionIndexes;

{тип вектора байтов}
TByteVector = array [1..ns_max] of byte;
PByteVector = ^TByteVector;

{тип вектора 16-битных слов}
TWordVector = array [1..ns_max] of word;
PWordVector = ^TWordVector;

{указатель на 8-байтовый вещественный тип}
PDouble = ^Double;

{Специальный флаг необходимости прерывания выполнения операции.}
{Он выставляется извне, и процедура, которая регулярно проверяет этот флаг
(особенно в больших многократно вложенных циклах), при обнаружении факта
установки этого флага, начинает корректное прерывание всех циклов,
освобождение занятой на время работы памяти и выход из процедуры}
{Флаг имеет чрезвычайную важность поскольку позволяет выйти практически

```

```

в любой момент из процедуры при любых объемах повторений циклов}
PStopFlag = ^Boolean;

```

```

{Специальный тип записи, необходимый для контроля прогресса
выполнения задачи распределения извне, процедуре решения всегда
передается указатель на переменную данного типа, поля которой
процедура решения регулярно обновляет, а внешний, параллельно
работающий поток регулярно проверяет поля этой переменной и
выводит информацию прогресса выполнения пользователю}
TTaskState = record
  CurrentH: integer;    {Число обработанных физических компьютеров}
  RemainNH: integer;    {Число оставшихся физических компьютеров}
  RemainNS: integer;    {Число оставшихся логических серверов}
  TotalNH: integer;     {Общее число физических компьютеров}
  TotalNS: integer;     {Общее число логических серверов}
  Stage: string[64];    {Текущая стадия решения задачи/обработки результатов}
end;
PTaskState = ^TTaskState;

```

```

type {Root Abstract CLASS}
  {Абстрактный класс поисковых задач}
  TLocalSearchTask = class

    {Source Data}
    {Основные поля класса}
    max_radius: integer;    {Максимальный радиус поиска}
    max_starts: integer;    {Число случайных стартовых точек}

    {Methods}
    {Методы класса}
    {Конструктор класса}
    constructor Init
      (p_max_radius: integer; p_max_starts: integer;
      var p_return_flag: byte; var p_return_err: string);

    {Полиморфная процедура запуска процедуры решения}
    procedure Run
      (var p_return_flag: byte; var p_return_err: string;
      p_stop_flag: PStopFlag; p_task_state: PTaskState); virtual;

    {Процедура решения задачи}
    procedure Solve
      (var p_return_flag: byte; var p_return_err: string;
      p_stop_flag: PStopFlag; p_task_state: PTaskState); virtual;

    {Процедура контроля состояния флага прерывания (извне) выполнения}
    procedure CheckUserAbort (p_stop_flag: PStopFlag;
      var p_return_flag: byte; var p_return_err: string); virtual;

    {Деструктор класса}
    destructor Done; virtual;
  end;

type {Boolean Optimization Tasks CLASS - Unconditional Optimization}
  {Класс задач безусловной булевой оптимизации}
  TBoolOptimumTask = class (TLocalSearchTask)

    {Source Data}
    {Основные поля класса}
    n: integer;            {число переменных}
    c: PC_Vector;          {вектор целевой функции}

    {Work data}
    {Дополнительные поля класса - временные вектора задачи}

```

```

x: PByteVector;           {текущая базовая субоптимальная точка}
x_r: PByteVector;         {текущая точка в радиусе поиска}
x_o: PByteVector;         {новая базовая субоптимальная точка}
x_p: PByteVector;         {предыдущая базовая субоптимальная точка}
comb: PWordVector;        {вектор двоичных комбинаций}

{Result Data}
{Поля результатов класса}
  x_cur: PByteVector;      {Оптимальная точка}
  lmax_cur: PDouble;       {Оптимальное значение целевой функции}
  total_it: PDouble;       {Число обработанных точек пространства}

{Methods}
{Методы класса}
  {Конструктор класса}
  constructor Init
    (p_n: integer;
     p_max_radius: integer; p_max_starts: integer; p_c: PC_Vector;
     p_x_cur: PByteVector; p_lmax_cur: PDouble; p_total_it: PDouble;
     var p_return_flag: byte; var p_return_err: string);

  {Процедура решения задачи}
  procedure Solve
    (var p_return_flag: byte; var p_return_err: string;
     p_stop_flag: PStopFlag; p_task_state: PTaskState); override;

  {Деструктор класса}
  destructor Done; override;
end;

type {Extended Boolean Optimization Tasks CLASS - Conditional Optimization}
{Класс задач условной булевой оптимизации}
TBoolOptimumTaskExt = class (TBoolOptimumTask)

{Additional Source Data (Conditions)}
{Основные поля класса}
  m: integer;             {Число ограничений}
  a: PA_Matrix;           {Матрица, левая часть ограничений}
  b: PB_Vector;           {Столбец, правая часть ограничений}

{Methods}
{Методы класса}
  {Конструктор класса}
  constructor Init
    (p_m: integer; p_n: integer;
     p_max_radius: integer; p_max_starts: integer;
     p_a: PA_Matrix; p_b: PB_Vector; p_c: PC_Vector;
     p_x_cur: PByteVector; p_lmax_cur: PDouble; p_total_it: PDouble;
     var p_return_flag: byte; var p_return_err: string);

  {Процедура решения задачи}
  procedure Solve
    (var p_return_flag: byte; var p_return_err: string;
     p_stop_flag: PStopFlag; p_task_state: PTaskState); override;

  {Деструктор класса}
  destructor Done; override;
end;

type {Optimal Distribution Tasks CLASS - without exclusions}
{Класс задач распределения логических серверов на физические компьютеры}
TDistributionTask = class (TLocalSearchTask)

{Source Data}

```

```

{Основные поля класса}
  base_nh: integer;           {Число физических компьютеров}
  base_ns: integer;           {Число логических серверов}
  base_nc: integer;           {Число типов ресурсов}
  hst: PHardCompsRsrcTable;    {Матрица ресурсов физических компьютеров}
  sst: PSoftSrvsReqTable;      {Матрица требований логических серверов}
  bst: PHostOSReqTable;        {Вектор требований базовой ОС}
  cm: PCriteriaMaskVector;     {Вектор маски критериев оптимизации}

{Work data}
{Дополнительные поля класса}
  {Вектор индексов оставшихся физических компьютеров}
  {и временный вектор для операций преобразования основного вектора}
  rhc, rhc_temp: PRemainHardCompsIndexes;
  {Вектор индексов оставшихся логических серверов}
  {и временный вектор для операций преобразования основного вектора}
  rss, rss_temp: PRemainSoftSrvsIndexes;

{Result Data}
{Поля результатов класса}
  dt: PDistributionTable;      {Матрица распределения логических серверов}
  rl: PResourceLoadingTable;    {Матрица загрузки ресурсов компьютеров}

{Methods}
{Методы класса}
  {Конструктор класса}
  constructor Init
    (p_base_nh:integer; p_base_ns:integer; p_base_nc:integer;
     p_max_radius: integer; p_max_starts: integer;
     p_hst: PHardCompsRsrcTable; p_sst: PSoftSrvsReqTable;
     p_bst: PHostOSReqTable; p_cm: PCriteriaMaskVector;
     p_dt: PDistributionTable; p_rl: PResourceLoadingTable;
     var p_return_flag: byte; var p_return_err: string);

    {Процедура решения задачи}
    procedure Solve
      (var p_return_flag: byte; var p_return_err: string;
       p_stop_flag: PStopFlag; p_task_state: PTaskState); override;

    {Деструктор класса}
    destructor Done; override;
end;

type {Extended Optimal Distributon Tasks CLASS - with exclusions}
{Класс расширенных задач распределения}
{логических серверов на физические компьютеры}
TDistributionTaskExt = class (TDistributionTask)

{Additional Source Data (Exclusions)}
{Основные поля класса}
  base_nx: integer;           {Число дополнительных ограничений-исключений}
  ext: PExclusionTable;        {Матрица исключений для логических серверов}

{Work data}
{Дополнительные поля класса}
  {Вектор индексов невырожденных дополнительных ограничений-исключений}
  {и временный вектор для операций преобразования основного вектора}
  vex, vex_temp: PValidExclusionIndexes;

{Methods}
{Методы класса}
  {Конструктор класса}
  constructor Init
    (p_base_nh:integer; p_base_ns:integer; p_base_nc:integer;

```

```

p_base_nx:integer; p_max_radius: integer; p_max_starts: integer;
p_hst: PHardCompsRsrcTable; p_sst: PSoftSrvsReqTable;
p_ext: PExclusionTable;
p_bst: PHostOSReqTable; p_cm: PCriteriaMaskVector;
p_dt: PDistributionTable; p_rl: PResourceLoadingTable;
var p_return_flag: byte; var p_return_err: string);

{Процедура решения задачи}
procedure Solve
(var p_return_flag: byte; var p_return_err: string;
 p_stop_flag: PStopFlag; p_task_state: PTaskState); override;

{Деструктор класса}
destructor Done; override;
end;

implementation
{Root Abstract Local Search Task's Methods Implementation}
{Реализация методов абстрактного класса поисковых задач}

{Конструктор - создает экземпляр класса, выполняет
проверку входных данных и инициализирует поля}
{Входные параметры:
  p_max_radius:    Максимальный радиус поиска
  p_max_starts:    Число стартовых точек}
{Выходные параметры:
  p_return_flag:    Флаг ошибки
  p_return_err:     Текст для сообщения об ошибке}
constructor TLocalSearchTask.Init;
var
  a_err: byte; {локальный флаг-защелка для ошибок}

begin
  a_err:= 0; {Сброс флага ошибок}
  p_return_err:= ''; {Сброс выходного текста ошибки}

  max_radius:= p_max_radius; {Инициализация поля max. радиуса поиска}
  max_starts:= p_max_starts; {Инициализация поля числа стартовых точек}

  if (a_err = 0) and (max_radius <= 0) then
  begin {Проверка max. радиуса поиска на > 0}
    p_return_err:= ABSTRACT_SN[1,2] + ' should be > 0';
    a_err:= 1; {В случае нарушения генерация соответствующей ошибки}
  end;

  if (a_err = 0) and (max_starts <= 0) then
  begin {Проверка числа стартовых точек > 0}
    p_return_err:= ABSTRACT_SN[2,2] + ' should be > 0';
    a_err:= 1; {В случае нарушения генерация соответствующей ошибки}
  end;

  {Подготовка текста ошибки и выходного флага ошибки}
  p_return_err:= ABSTRACT_ERR_HEAD + p_return_err;
  if (a_err <> 0) then p_return_flag:= 1 else p_return_flag:= 0;
end;

{Полиморфная процедура, вызывает соответствующую классу,
вызвавшему данный метод, процедуру решения задачи}
{Входные параметры:
  p_stop_flag:    Указатель на контрольный флаг прерывания извне
  p_task_state:   Указатель на переменную для контроля состояния задачи}
{Выходные параметры:

```



```

    p_return_flag:    Флаг ошибки
    p_return_err:     Текст для сообщения об ошибке}
procedure TLocalSearchTask.Run;
begin
    {Classic Polymorph} {Вызов процедуры, соответствующего классу}
    Solve (p_return_flag, p_return_err, p_stop_flag, p_task_state);
end;

{Процедура решения задачи}
{Входные параметры:
    p_stop_flag:     Указатель на контрольный флаг прерывания извне
    p_task_state:    Указатель на переменную для контроля состояния задачи}
{Выходные параметры:
    p_return_flag:    Флаг ошибки
    p_return_err:     Текст для сообщения об ошибке}
procedure TLocalSearchTask.Solve;
begin
    p_return_err:= ABSTRACT_ERR_HEAD + S_ABSTRACT_NOTIMP_ERR;
    p_return_flag:= 1; {Ошибка о том, что метод не реализован}
end;

{Процедура контроля состояния флага прерывания (извне) выполнения}
{Входные параметры:
    p_stop_flag:     Указатель на контрольный флаг прерывания извне}
{Выходные параметры:
    p_return_flag:    Флаг ошибки
    p_return_err:     Текст для сообщения об ошибке}
procedure TLocalSearchTask.CheckUserAbort;
begin
    try
        if (p_stop_flag^ = TRUE) then          {Если флаг установлен}
        begin
            p_return_err:= S_STOP_FLAG_TRUE;    {Генерируется ошибка}
            p_return_flag:= 1;                  {принудительного останова}
        end;
    except
        On EAccessViolation do                  {Если возникло нарушение доступа}
        begin                                   {при проверке флага}
            p_return_err:= S_STOP_FLAG_FAILED;
            p_return_flag:= 1;                  {то генерируется соответствующая ошибка}
        end;
    end;
end;

{Деструктор - уничтожает экземпляр класса}
destructor TLocalSearchTask.Done;
begin
end;

{Boolean Optimization Task's Methods Implementation}
{Реализация методов класса задач безусловной булевой оптимизации}

{Конструктор - создает экземпляр класса, выполняет
проверку входных данных и инициализирует поля}
{Входные параметры:
    p_n: integer;      Число переменных
    p_max_radius:      Максимальный радиус поиска
    p_max_starts:      Число стартовых точек
    p_c:               Указатель на вектор целевой функции

```

```

p_x_cur:           Указатель на вектор координат оптимальной точки
p_lmax_cur:        Указатель на переменную оптимального значения
p_total_it:        Указатель на переменную числа обработанных точек}
{Выходные параметры:
  p_return_flag:    Флаг ошибки
  p_return_err:     Текст для сообщения об ошибке}
constructor TBoolOptimumTask.Init;
var
  b_err: byte;      {локальный флаг-защелка для ошибок}
begin
  b_err:= 0;        {Сброс флага ошибок}
  p_return_err:= ''; {Сброс выходного текста ошибки}

  {Вызов унаследованного конструктора}
  Inherited Init (p_max_radius, p_max_starts, b_err, p_return_err);

  {Установка указателей временных векторов в нуль}
  x:= nil; x_r:= nil; x_o:= nil; x_p:=nil; comb:= nil;

  n:= p_n;          {Инициализация поля числа переменных}
  c:= p_c;          {Инициализация поля указателя}
                  {на вектор целевой функции}
  x_cur:= p_x_cur;  {Инициализация поля указателя}
                  {на вектор координат оптимальной точки}
  lmax_cur:= p_lmax_cur; {Инициализация поля указателя на}
                  {оптимальное значения функции}
  total_it:= p_total_it; {Инициализация поля указателя}
                  {на число обработанных точек}

  if (b_err = 0) and ((n <= 0) or (n > ns_max)) then
  begin
    {Проверка числа переменных на [1,ns_max]}
    p_return_err:= BOPT_SN[1,2] +
      ' must be in range 1..' + IntToStr(ns_max);
    b_err:= 1; {В случае нарушения генерация соответствующей ошибки}
  end;

  if ((b_err = 0) and (c = nil)) then
  begin
    {Проверка указателя на вектор коэффициентов целевой функции}
    p_return_err:= S_SRC_INV_PTR + BOPT_SN[2,2];
    b_err:= 1; {В случае если указатель пустой}
  end;
    {генерация соответствующей ошибки}

  if ((b_err = 0) and (x_cur = nil)) then
  begin
    {Проверка указателя на вектор координат оптимальной точки}
    p_return_err:= S_SRC_INV_PTR + BOPT_SN[3,2];
    b_err:= 1; {В случае если указатель пустой}
  end;
    {генерация соответствующей ошибки}

  if ((b_err = 0) and (lmax_cur = nil)) then
  begin
    {Проверка указателя на оптимальное значение функции}
    p_return_err:= S_SRC_INV_PTR + BOPT_SN[4,2];
    b_err:= 1; {В случае если указатель пустой}
  end;
    {генерация соответствующей ошибки}

  if ((b_err = 0) and (total_it = nil)) then
  begin
    {Проверка указателя на число обработанных точек}
    p_return_err:= S_SRC_INV_PTR + BOPT_SN[5,2];
    b_err:= 1; {В случае если указатель пустой}
  end;
    {генерация соответствующей ошибки}

  if (b_err = 0) then
  begin
    try
      {Попытка выделения памяти для временных векторов}
      GetMem (x, n*sizeof(x^[1]));

```

```

    GetMem (x_r, n*sizeof(x_r^[1]));
    GetMem (x_o, n*sizeof(x_o^[1]));
    GetMem (x_p, n*sizeof(x_p^[1]));
    GetMem (comb, n*sizeof(comb^[1]));
except
    On EOutOfMemory do      {В случае выделения ошибки памяти }
        begin                {генерация соответствующей ошибки}
            p_return_err:= S_LOCAL_DATA_MEM_ALLOC_ERR;
            b_err:= 1;
        end;
    end;

    {Дополнительная проверка указателей}
    if ((b_err = 0) and ((x = nil) or (x_r = nil) or
(x_o = nil) or (x_p = nil) or (comb = nil))) then
        begin
            p_return_err:= S_LOCAL_DATA_MEM_ALLOC_ERR;
            b_err:= 1;        {в случае если один из указателей пустой}
        end;                {генерация соответствующей ошибки}
    end;

    {Подготовка выходного флага ошибки}
    if (b_err <> 0) then p_return_flag:= 1 else p_return_flag:= 0;
end;

{Процедура решения задачи}
{Входные параметры:
    p_stop_flag:    Указатель на контрольный флаг прерывания извне
    p_task_state:    Указатель на переменную для контроля состояния задачи}
{Выходные параметры:
    p_return_flag:    Флаг ошибки
    p_return_err:    Текст для сообщения об ошибке}
procedure TBoolOptimumTask.Solve;
begin
    p_return_err:= S_BOPT_NOTIMP_ERR;
    p_return_flag:= 1; {Ошибка о том, что метод не реализован}
end;

{Деструктор - уничтожает экземпляр класса}
destructor TBoolOptimumTask.Done;
begin
    {Освобождение памяти временных векторов}
    if (x <> nil) then FreeMem (x, n*sizeof(x^[1]));
    if (x_r <> nil) then FreeMem (x_r, n*sizeof(x_r^[1]));
    if (x_o <> nil) then FreeMem (x_o, n*sizeof(x_o^[1]));
    if (x_p <> nil) then FreeMem (x_p, n*sizeof(x_p^[1]));
    if (comb <> nil) then FreeMem (comb, n*sizeof(comb^[1]));
    x:= nil; x_r:= nil; x_o:= nil; x_p:= nil; comb:= nil;
    Inherited Done; {Вызов унаследованного деструктора}
end;

{Extended Boolean Optimization Task's Methods Implementation}
{Реализация методов класса задач условной булевой оптимизации}

{Конструктор - создает экземпляр класса, выполняет
проверку входных данных и инициализирует поля}
{Входные параметры:
    p_n: integer;        Число переменных
    p_max_radius:        Максимальный радиус поиска
    p_max_starts:        Число стартовых точек
    p_a:                  Указатель на матрицу (левая часть) ограничений
    p_b:                  Указатель на вектор (правая часть) ограничений

```

```

    p_c:           Указатель на вектор целевой функции
    p_x_cur:       Указатель на вектор координат оптимальной точки
    p_lmax_cur:    Указатель на переменную оптимального значения
    p_total_it:    Указатель на переменную числа обработанных точек}
{Выходные параметры:
    p_return_flag: Флаг ошибки
    p_return_err:  Текст для сообщения об ошибке}
constructor TBoolOptimumTaskExt.Init;
var
    b_err: byte;      {локальный флаг-защелка для ошибок}
begin
    b_err:= 0;        {Сброс флага ошибок}
    p_return_err:= ''; {Сброс выходного текста ошибки}

    {Вызов унаследованного конструктора}
    Inherited Init (p_n, p_max_radius, p_max_starts, p_c,
    p_x_cur, p_lmax_cur, p_total_it, b_err, p_return_err);

    m:= p_m; {Инициализация поля числа ограничений}
    a:= p_a; {Инициализация поля указателя на матрицу
              (левая часть) ограничений}
    b:= p_b; {Инициализация поля указателя на вектор
              (правая часть) ограничений}

    if (b_err = 0) and ((m < 0) or (m > nx_max + nc_max)) then
    begin {Проверка числа ограничений на [0,nx_max+nc_max]}
        p_return_err:= BOPTEXT_SN[1,2] +
        ' must be in range 0..' + IntToStr(nx_max + nc_max);
        b_err:= 1;
    end; {В случае нарушения генерация соответствующей ошибки}

    if ((m > 0) and (b_err = 0) and (a = nil)) then
    begin {Проверка указателя на матрицу (левая часть) ограничений}
        p_return_err:= S_SRC_INV_PTR + BOPTEXT_SN[2,2];
        b_err:= 1;
    end; {В случае если указатель пустой генерация соответствующей ошибки}

    if ((m > 0) and (b_err = 0) and (b = nil)) then
    begin {Проверка указателя на вектор (правая часть) ограничений}
        p_return_err:= S_SRC_INV_PTR + BOPTEXT_SN[3,2];
        b_err:= 1;
    end; {В случае если указатель пустой генерация соответствующей ошибки}

    {Подготовка текста ошибки и выходного флага ошибки}
    if (b_err <> 0) then p_return_flag:= 1 else p_return_flag:= 0;
    p_return_err:= BOPTEXT_ERR_HEAD + p_return_err;
end;

{Процедура решения задачи}
{Входные параметры:
    p_stop_flag: Указатель на контрольный флаг прерывания извне
    p_task_state: Указатель на переменную для контроля состояния задачи}
{Выходные параметры:
    p_return_flag: Флаг ошибки
    p_return_err:  Текст для сообщения об ошибке}
procedure TBoolOptimumTaskExt.Solve;
var
    {Вспомогательные локальные переменные}
    i, j, k, q, flag, k1, k2, cflag, first: integer;
    ai_sum, temp, shtraf, lmax, lmax_r: double;
    iteration, max_iteration: double;
    dist_to_x_p: integer;
    b_err: byte;      {локальный флаг-защелка для ошибок}
begin

```

```

{ i = 1 , m ; i - index for constraints}
{ j = 1 , n ; j - index for variables}
{x_p - previous base point (previous local optimum)}
{x_o - new base point (new local optimum)}
{x - current base point}
{x_r - current point in radius of current base point}

b_err:= 0;           {Сброс локального флага ошибок}
p_return_err:= '';   {Сброс выходного текста ошибок}
p_return_flag:= 0;    {Сброс выходного флага ошибок}

if ((x = nil) or (x_r = nil) or (x_o = nil) or
    (x_p = nil) or (comb = nil) or (total_it = nil) or
    (lmax_cur = nil) or (p_stop_flag = nil) or (p_task_state=nil)) then
    begin
        {Проверка указателей входных данных и полей класса}
        p_return_err:= S_SRC_INV_PTRS;
        b_err:= 1;     {В случае если один из указателей пустой}
    end;               {генерация соответствующей ошибки}

if (b_err = 0) then
    begin
        randomize;      {инициализация генератора случайных чисел}
        lmax_cur^:=0;    {начальное оптимальное значение целевой функции = 0}
        for j:=1 to n do x_cur^[j]:=0; {сброс координат оптимальной точки}
        total_it^:=0;    {обнуление результирующего числа обработанных точек}
        max_iteration:= exp(n*ln(2)); {максимальный перебор = 2^n}
        if max_radius > n then max_radius:= n; {Если максимальный радиус}
        {больше числа переменных, то максимальный радиус ограничивается до n}

{MAIN CYCLE}

{Для каждой, случайно сгенерированной стартовой точки (число старт. точек
равно max_starts) выполняется следующее: стартовая точка принимается за
базовую, далее вокруг базовой строится пространство всех точек,
отличающихся от базовой 0-мя, 1-й, 2-мя ... max_radius-мя координатами
и среди них выбирается лучшая, где целевая функция имеет лучшее значение
эта точка принимается как новая базовая, далее, для новой базовой все
повторяется, только исключаются все точки, которые попадают в радиус
видимости (отличаются не более чем max_radius-мя координатами) старой
базовой точки, т.к. совершенно очевидно то, что они были уже рассмотрены
при предыдущем поиске. Рано или возникнет ситуация в текущем радиусе
поиска не найдется ни одной более лучшей точки и последняя базовая точка
будет тогда считаться оптимальным решением для заданной начальной
стартовой точки. Перебрав все стартовые точки, мы выберем наилучшее
из полученного множества оптимальных решений}

{Сферой поиска радиусом q будем называть, множество всех точек n-мерного
пространства, которые отличаются от заданной точки ровно q координатами}

k:= 0; {Сброс счетчика стартовых точек}
{Цикл перебора случайно сгенерированных стартовых точек }
while ((k <= max_starts - 1) and (b_err = 0)) do
    begin
        iteration:= 0; {сброс внутреннего счетчика обработанных точек}
        lmax_r:=0;      {сброс оптимального значения, полученного при}
                        {поиске оптимума для очередной стартовой точки}
        for j:=1 to n do x_r^[j]:=0; {сброс текущей точки поиска}
        for j:=1 to n do x_p^[j]:=0; {сброс предыдущей базовой точки}

        {генерация очередной случайной стартовой точки}
        if k = 0 then for j:=1 to n do x^[j]:=0
        else for j:=1 to n do x^[j]:=random(2);

        first:=1; {флаг - признак того, что ведется поиск от начальной

```

```

базовой точки и предыдущей базовая точка еще не пока не существует}
{Основной цикл поиска оптимальной точки в заданной окрестности}
repeat {радиусом, равным максимальному радиусу поиска}

    flag:= 0; {флаг - признак того, что оптимум найден}
    q:= 0;     {сброс радиуса текущей n-мерной сферы поиска}
    {Цикл перебора радиусов сферы поиска от 0 до max_radius}
    while ((q <= max_radius) and (b_err = 0)) do
        begin

            {Цикл перебора точек для заданной сферы поиска}
            {Стартовая комбинация (1,2,3,...,q)}
            for kl:=1 to q do comb^[kl]:=kl;
            cflag:=0; {Сброс флага окончания перебора}
            while ((cflag = 0) and (b_err = 0)) do
                begin
                    {Получение очередной комбинации из текущей базовой точки}
                    for j:=1 to n do x_r^[j]:=x^[j];
                    for j:=1 to q do x_r^[comb^[j]]:= x_r^[comb^[j]] XOR 1;

                    if first = 0 then {Если поиск не первый, это значит}
                        begin {существует предыдущая базовая точка. Вычисляем}
                            {дистанцию от текущей точки до предыдущей базовой}
                            dist_to_x_p:=0; {вычисление дистанции}
                            for j:=1 to n do if ((x_r^[j] XOR x_p^[j]) = 1)
                                then dist_to_x_p:= dist_to_x_p + 1;
                            end;

                            {Если дистанция меньше max. радиуса поиска, то значит}
                            {точка была ранее рассмотрена и ее можно пропустить}
                            if ((dist_to_x_p > max_radius) OR (first = 1)) then
                                begin

                                    shtraf:=0; {Сброс штрафа (нарушение ограничений)}
                                    if (m > 0) then {Если число ограничений > 0, то}
                                        begin {подставляем координаты текущей}
                                            {точки и проверяем выполнение ограничений}

                                            for i:=1 to m do {Для каждого i-го ограничения}
                                                begin {суммируем a[i,j] для тех j, для}
                                                    ai_sum:=0; {которых координата текущего}
                                                        {точки x_r равна 1}
                                                    for j:=1 to n do if x_r^[j]=1 then
                                                        ai_sum:=ai_sum + a^[n*(i-1)+j-1];

                                                    {Если сумма превышает b[i], то вычисляем}
                                                    {величину превышения и добавляем к штрафу}
                                                    if ai_sum > b^[i] then
                                                        begin
                                                            temp:= ai_sum - b^[i];
                                                            shtraf:= shtraf + temp;
                                                        end;
                                                    end;
                                                end;
                                            end;

                                        end;

                                    lmax:=0; {Вычисляем значение целевой функции}
                                    {для текущей точки поиска}
                                    for j:=1 to n do if (x_r^[j] = 1)
                                        then lmax:= lmax + c^[j];

                                    {Если значение больше (если штраф ненулевой, то}
                                    {значение никогда не будет больше), то найден}
                                    {новый локальный оптимум, принимаем ее как новую}
                                    {базовую точку и продолжаем поиск}
                                end;
                            end;
                        end;
                    end;
                end;
            end;
        end;
    end;
end;

```

```

        if lmax - 4294967296 * shtraf > lmax_r then
            begin
                flag:= 1;
                lmax_r:= lmax;
                for j:=1 to n do x_o^[j]:= x_r^[j];
            end;
            iteration:= iteration + 1; {Увеличиваем внутренний}
        end;                                     {счетчик обработанных точек}

        k1:= q; {Вычисляем следующую комбинацию координат точки}
        while ((k1 > 0) and (comb^[k1] = n-q+k1)) do k1:=k1-1;
        if k1 > 0 then
            begin
                comb^[k1]:= comb^[k1] + 1;
                if (k1 < q) then for k2:=k1+1 to q do
                    comb^[k2]:= comb^[k1]+k2-k1;
                end {Если перебор окончен то устанавливаем флаг}
            else cflag:=1;

            {Special checking for the breaking action by the user!}
            {Проверка внешнего флага принудительного останова}
            CheckUserAbort (p_stop_flag, b_err, p_return_err);
            {Если флаг установлен то будет установлен флаг ошибки}
        end;
        q:= q + 1; {Переход к сфере поиска следующего радиуса}
    end;

    if ((flag = 1) and (b_err = 0)) then
        begin
            {Если новая базовая точка была найдена, то}
            for j:=1 to n do {текущая базовая точка становится}
                begin {предыдущей, а новая - текущей}
                    x_p^[j]:= x^[j];
                    x^[j]:= x_o^[j];
                    first:=0; {Сброс флага-признака первоначального}
                end; {поиска от начальной базовой точки}
            end;
            {продолжаем поиск пока сброшен флаг ошибок и пока каждый раз}
            {находятся новые базовые точки, как только новую более лучшую}
            {базовую точку нельзя будет найти, то flag окажется равным 0}
            until ((b_err <> 0) OR (flag = 0) OR (iteration >= max_iteration));

            {Если оптимум, найденный для текущей стартовой точки лучше}
            {чем найденный при предыдущей, то она принимается как}
            {результатирующий оптимум и переходим к следующей стартовой точке}
            if lmax_r > lmax_cur^ then
                begin
                    lmax_cur^:= lmax_r;
                    for j:=1 to n do x_cur^[j]:= x^[j];
                end;
                {расчет результирующего числа обработанных точек}
                total_it^:= total_it^ + iteration;
                k:= k + 1; {Переходим к следующей стартовой точке}
            end;
        end;
        {Подготовка текста ошибки и выходного флага ошибки}
        if (b_err <> 0) then p_return_flag:= 1 else p_return_flag:= 0;
        p_return_err:= BORTXT_ERR_HEAD + p_return_err;
    end;

    {Деструктор - уничтожает экземпляр класса}
    destructor TBoolOptimumTaskExt.Done;
    begin
        {Вызов унаследованного деструктора}

```

```

    Inherited Done;
end;

{Optimal Distribution Task's Methods Implementation}
{Реализация методов класса задач распределения}
{логических серверов на физические компьютеры}

{Конструктор - создает экземпляр класса, выполняет
проверку входных данных и инициализирует поля}
{Входные параметры:
    p_base_nh:      Число физических компьютеров
    p_base_ns:      Число логических серверов
    p_base_nc:      Число типов ресурсов
    p_max_radius:   Максимальный радиус поиска
    p_max_starts:   Число стартовых точек
    p_hst:          Указатель на матрицу ресурсов физических компьютеров
    p_sst:          Указатель на матрицу требований логических серверов
    p_bst:          Указатель на вектор требований базовой ОС
    p_cm:           Указатель на вектор маски критериев оптимизации
    p_dt:           Указатель на матрицу распределения логических серверов
    p_rl:           Указателя на матрицу загрузки ресурсов компьютеров}
{Выходные параметры:
    p_return_flag:   Флаг ошибки
    p_return_err:    Текст для сообщения об ошибке}
constructor TDistributionTask.Init;
var
    d_err: byte;      {локальный флаг-защелка для ошибок}
    i, js, jh: integer;
    intsum: longint;
begin
    d_err:= 0;        {сброс локального флага-защелки для ошибок}
    p_return_err:= ''; {сброс выходного текста ошибок}

    {Вызов унаследованного конструктора}
    Inherited Init (p_max_radius, p_max_starts, d_err, p_return_err);

    base_nh:= p_base_nh; {Инициализация поля числа физических компьютеров}
    base_ns:= p_base_ns; {Инициализация поля числа логических серверов}
    base_nc:= p_base_nc; {Инициализация поля числа типов ресурсов}
    hst:= p_hst; {Инициализация поля указателя на матрицу ресурсов физ.комп.}
    sst:= p_sst; {Инициализация поля указателя на матрицу требований лог.серв.}
    bst:= p_bst; {Инициализация поля указателя на вектор требований базовой ОС}
    cm:= p_cm; {Инициализация поля указателя на вектор маски критериев}
    dt:= p_dt; {Инициализация поля указателя на матрицу распределения}
    rl:= p_rl; {Инициализация поля указателя на матрицу загрузки ресурсов}

    if (d_err = 0) and ((base_nh <= 0) or (base_nh > nh_max)) then
    begin {проверка числа физических компьютеров на [1..nh_max]}
        p_return_err:= DIST_SN[1,2] +
            ' must be in range 1..' + IntToStr(nh_max);
        d_err:= 1;
    end; {В случае нарушения генерация соответствующей ошибки}

    if (d_err = 0) and ((base_ns <= 0) or (base_ns > ns_max)) then
    begin {проверка числа логических серверов на [1..ns_max]}
        p_return_err:= DIST_SN[2,2] +
            ' must be in range 1..' + IntToStr(ns_max);
        d_err:= 1;
    end; {В случае нарушения генерация соответствующей ошибки}

    if (d_err = 0) and ((base_nc <= 0) or (base_nc > nc_max)) then
    begin {проверка числа типов ресурсов на [1..nc_max]}

```



```

    p_return_err:= DIST_SN[3,2] +
    ' must be in range 1..' + IntToStr(nc_max);
    d_err:= 1;
end;      {В случае нарушения генерация соответствующей ошибки}

if (d_err = 0) and (hst=nil) then
begin      {Проверка указателя на матрицу ресурсов физических компьютеров}
    p_return_err:= S_SRC_INV_PTR + DIST_SN[4,2];
    d_err:= 1;
end;      {В случае если указатель пустой генерация соответствующей ошибки}

if (d_err = 0) and (sst=nil) then
begin      {Проверка указателя на матрицу требований логических серверов}
    p_return_err:= S_SRC_INV_PTR + DIST_SN[5,2];
    d_err:= 1;
end;      {В случае если указатель пустой генерация соответствующей ошибки}

if (d_err = 0) and (bst=nil) then
begin      {Проверка указателя на вектор требований базовой ОС}
    p_return_err:= S_SRC_INV_PTR + DIST_SN[6,2];
    d_err:= 1;
end;      {В случае если указатель пустой генерация соответствующей ошибки}

if (d_err = 0) and (cm=nil) then
begin      {Проверка указателя на вектор маски критериев оптимизации}
    p_return_err:= S_SRC_INV_PTR + DIST_SN[7,2];
    d_err:= 1;
end;      {В случае если указатель пустой генерация соответствующей ошибки}

if (d_err = 0) and (dt=nil) then
begin      {Проверка указателя на матрицу распределения}
    p_return_err:= S_SRC_INV_PTR + DIST_SN[8,2];
    d_err:= 1;
end;      {В случае если указатель пустой генерация соответствующей ошибки}

if (d_err = 0) and (rl=nil) then
begin      {Проверка указателя на матрицу загрузки ресурсов компьютеров}
    p_return_err:= S_SRC_INV_PTR + DIST_SN[9,2];
    d_err:= 1;
end;      {В случае если указатель пустой генерация соответствующей ошибки}

if (d_err = 0) then
begin {Обнуление матриц распределения и загрузки ресурсов компьютеров}
    for jh:=1 to base_nh do for js:=1 to base_ns do
        dt^[base_ns*(jh-1)+js-1]:=0;
    for jh:=1 to base_nh do for i:=1 to base_nc do
        rl^[base_nc*(jh-1)+i-1]:=0;
    end;
end;

rhc:= nil; rss:= nil;
rhc_temp:= nil; rss_temp:= nil;
{Попытка выделения памяти для временных переменных}
if d_err = 0 then
begin
    try
        {Вектора индексов оставшихся физических компьютеров}
        GetMem(rhc, base_nh * sizeof(rhc^[1]));
        {и временного вектора для операций преобразования с ним}
        GetMem(rhc_temp, base_nh * sizeof(rhc_temp^[1]));
        {Вектора индексов оставшихся логических серверов}
        GetMem(rss, base_ns * sizeof(rss^[1]));
        {и временного вектора для операций преобразования с ним}
        GetMem(rss_temp, base_ns * sizeof(rss_temp^[1]));
    except

```

```

On EOutOfMemory do
begin
  {В случае нехватки памяти генерация соответствующей ошибки}
  p_return_err:= S_LOCAL_DATA_MEM_ALLOC_ERR;
  d_err:= 1;
end;
end;

{Дополнительные проверки для указателей}
{векторов индексов оставшихся физ. компьютеров и лог. серверов}
{и временных векторов для операций преобразований с ними}
if ((d_err = 0) and
((rhc=nil) or (rss=nil) or (rhc_temp=nil) or (rss_temp=nil))) then
begin
  p_return_err:= S_LOCAL_DATA_MEM_ALLOC_ERR;
  d_err:= 1;  {Если один из указателей пустой, то}
end;  {генерация соответствующей ошибки}
end;

if d_err = 0 then
begin
  {Занесение в вектор оставшихся физ. компьютеров, множества}
  {индексов всех изначально заданных в задаче физ. компьютеров}
  for jh:=1 to base_nh do rhc^[jh]:=jh;
  {Занесение в вектор оставшихся лог. серверов, множества}
  {индексов всех изначально заданных в задаче лог. серверов}
  for js:=1 to base_ns do rss^[js]:=js;
end;

if d_err = 0 then
begin
  i:= 1;  {Проверка элементов матрицы HST}
  while ((i <= base_nc) and (d_err = 0)) do
    begin
      jh:= 1;
      while ((jh <= base_nh) and (d_err = 0)) do
        begin  {Проверка элементов матрицы HST на >= 0}
          if hst^[(i-1)*base_nh+jh-1] < 0 then
            begin  {В случае нарушения генерация соответствующей ошибки}
              d_err:= 1;
              p_return_err:=
                'Element[' + IntToStr (i) + ',' + IntToStr (jh) +
                ']' in the ' + chr(13) + chr(10) +
                DIST_SN[4,2] + ' should be >= 0';
            end;
            jh:= jh + 1;
          end;
          i:= i + 1;
        end;
      end;
    end;

if d_err = 0 then
begin
  i:= 1;  {Проверка элементов матрицы SST}
  while ((i <= base_nc) and (d_err = 0)) do
    begin
      js:= 1;
      while ((js <= base_ns) and (d_err = 0)) do
        begin  {Проверка элементов матрицы SST на >= 0}
          if sst^[(i-1)*base_ns+js-1] < 0 then
            begin  {В случае нарушения генерация соответствующей ошибки}
              d_err:= 1;
              p_return_err:=
                'Element[' + IntToStr (i) + ',' + IntToStr (js) +

```

```

        ']' in the ' + chr(13) + chr(10)
        + DIST_SN[5,2] + ' should be >= 0';
    end;
    js:= js + 1;
end;
i:= i + 1;
end;
end;

if d_err = 0 then
begin
    i:= 1; {Проверка элементов вектора BST на >= 0}
    while ((i <= base_nc) and (d_err = 0)) do
    begin
        if bst^[i] < 0 then
        begin {В случае нарушения генерация соответствующей ошибки}
            d_err:= 1;
            p_return_err:= 'Element[' + IntToStr (i) + '] in the '
                + chr(13) + chr(10) + DIST_SN[6,2] + ' should be >= 0'
        end;
            i:= i + 1;
        end;
    end;

if d_err = 0 then
begin
    intsum:=0;
    i:= 1; {Проверка элементов вектора CM на = "0" или "1"}
    while ((i <= base_nc) and (d_err = 0)) do
    begin
        if not((cm^[i] = 0) or (cm^[i] = 1)) then
        begin {В случае нарушения генерация соответствующей ошибки}
            d_err:= 1;
            p_return_err:= 'Element[' + IntToStr (i) + '] in the '
                + chr(13) + chr(10) + DIST_SN[7,2] + ' should be 0 or 1'
        end
        else intsum:= intsum + cm^[i]; {Суммирование элементов вектора}
            i:= i + 1;
        end;
    end;

if d_err = 0 then
begin {Если сумма равна нулю, то это значит что весь}
    if intsum=0 then {вектор CM - нулевой и нет критериев для оптимизации}
    begin
        d_err:=1; {Соответственно, генерация ошибки в таком случае}
        p_return_err:=
            'At least one element in the '
            + chr(13) + chr(10) + DIST_SN[7,2] + ' should be 1';
    end;

    {Подготовка выходного флага ошибки}
    if (d_err <> 0) then p_return_flag:= 1 else p_return_flag:= 0;
end;

{Процедура решения задачи}
{Входные параметры:
    p_stop_flag:    Указатель на контрольный флаг прерывания извне
    p_task_state:   Указатель на переменную для контроля состояния задачи}
{Выходные параметры:
    p_return_flag:   Флаг ошибки
    p_return_err:    Текст для сообщения об ошибке}

```

```

procedure TDistributionTask.Solve;
begin
    p_return_err:= S_DT_NOTIMP_ERR;
    p_return_flag:= 1; {Ошибка о том, что метод не реализован}
end;

{Деструктор - уничтожает экземпляр класса}
destructor TDistributionTask.Done;
begin
    {Освобождение памяти индексных векторов и временных векторов}
    if (rhc <> nil)
    then FreeMem(rhc, base_nh * sizeof(rhc^[1]));
    if (rss <> nil)
    then FreeMem(rss, base_ns * sizeof(rss^[1]));
    if (rhc_temp <> nil)
    then FreeMem(rhc_temp, base_nh * sizeof(rhc_temp^[1]));
    if (rss_temp <> nil)
    then FreeMem(rss_temp, base_ns * sizeof(rss_temp^[1]));
    rhc:= nil; rss:= nil;
    rhc_temp:= nil; rss_temp:= nil;
    {Вызов унаследованного деструктора}
    Inherited Done;
end;

{Extended Optimal Distribution Task's Methods Implementation}
{Реализация методов класса расширенных задач}
{распределения логических серверов на физические компьютеры}

{Конструктор - создает экземпляр класса, выполняет
проверку входных данных и инициализирует поля}
{Входные параметры:
    p_base_nh:      Число физических компьютеров
    p_base_ns:      Число логических серверов
    p_base_nc:      Число типов ресурсов
    p_base_nx:      Число исключений для логических серверов
    p_max_radius:   Максимальный радиус поиска
    p_max_starts:   Число стартовых точек
    p_hst:          Указатель на матрицу ресурсов физических компьютеров
    p_sst:          Указатель на матрицу требований логических серверов
    p_ext:          Указатель на матрицу исключений для логических серверов
    p_bst:          Указатель на вектор требований базовой ОС
    p_cm:           Указатель на вектор маски критериев оптимизации
    p_dt:           Указатель на матрицу распределения логических серверов
    p_rl;          Указателя на матрицу загрузки ресурсов компьютеров}
{Выходные параметры:
    p_return_flag:   Флаг ошибки
    p_return_err:    Текст для сообщения об ошибке}
constructor TDistributionTaskExt.Init;
var
    i, js: integer;
    d_err: byte;
begin
    d_err:= 0;
    p_return_err:= '';

    {Вызов унаследованного конструктора}
    Inherited Init (p_base_nh, p_base_ns, p_base_nc,
        p_max_radius, p_max_starts, p_hst, p_sst,
        p_bst, p_cm, p_dt, p_rl, d_err, p_return_err);

    base_nx:= p_base_nx;    {Инициализация поля числа исключений}

```

```

ext:= p_ext; {Инициализация поля указателя на матрицу исключений}

if (d_err = 0) and ((base_nx < 0) or (base_nx > nx_max)) then
begin {Проверка числа исключений на = 0..nx_max}
  p_return_err:= DISTEXT_SN[1,2] +
  ' must be in range 0..' + IntToStr(nx_max);
  d_err:= 1;
end; {В случае нарушения генерация соответствующей ошибки}

if (d_err = 0) and ((base_nx > 0) and (ext=nil)) then
begin {Проверка указателя на матрицу исключений}
  p_return_err:= S_SRC_INV_PTR + DISTEXT_SN[2,2];
  d_err:= 1; {в случае если число исключений > 0}
end; {а указатель на матрицу пустой, то генерация соответствующей ошибки}

vex:= nil; vex_temp:= nil;
{Попытка выделения памяти для временных переменных}
if ((d_err = 0) and (base_nx > 0)) then
begin
  try
    {Вектора индексов невырожденных исключений}
    GetMem(vex, base_nx * sizeof(vex^[1]));
    {и временного вектора для операций преобразования с ним}
    GetMem(vex_temp, base_nx * sizeof(vex_temp^[1]));
  except
    On EOutOfMemory do
    begin
      {В случае нехватки памяти генерация соответствующей ошибки}
      p_return_err:= S_LOCAL_DATA_MEM_ALLOC_ERR;
      d_err:= 1;
    end;
  end;

  {Дополнительные проверки для указателей}
  {Вектора индексов невырожденных исключений}
  {и временного вектора для операций преобразования}
  if ((d_err = 0) and ((vex=nil) or (vex_temp=nil))) then
  begin
    p_return_err:= S_LOCAL_DATA_MEM_ALLOC_ERR;
    d_err:= 1; {Если один из указателей пустой, то}
  end; {генерация соответствующей ошибки}
end;

if ((d_err = 0) and (base_nx > 0)) then
begin
  {Занесение в вектор индексов невырожденных исключений}
  {индексов всех изначально заданных в задаче исключений}
  for i:=1 to base_nx do vex^[i]:=i;
end;

if ((d_err = 0) and (base_nx > 0)) then
begin {В случае если число исключений больше нуля то копирование}
  i:= 1; {матрицы исключения во временную и рабочую копии}
  while ((i <= base_nx) and (d_err = 0)) do
  begin
    js:= 1; {А также проверка элементов матрицы на = "0" или "1"}
    while ((js <= base_ns) and (d_err = 0)) do
    begin
      if not((ext^[base_ns*(i-1)+js-1] = 0)
      or (ext^[base_ns*(i-1)+js-1] = 1)) then
      begin {В случае нарушения генерация соответствующей ошибки}
        d_err:= 1;
        p_return_err:=
        'Element[' + IntToStr (i) + ', ' + IntToStr (js) +

```

```

        ']' in the ' + chr(13) + chr(10)
        + DISTEXT_SN[2,2] + ' should be 0 or 1';
    end;
    js:= js + 1;
end;
i:= i + 1;
end;
end;
{Подготовка выходного флага и текста ошибки}
if (d_err <> 0) then p_return_flag:= 1 else p_return_flag:= 0;
p_return_err:= DISTEXT_ERR_HEAD + p_return_err;
end;

{Процедура решения задачи}
{Входные параметры:
    p_stop_flag:      Указатель на контрольный флаг прерывания извне
    p_task_state:     Указатель на переменную для контроля состояния задачи}
{Выходные параметры:
    p_return_flag:    Флаг ошибки
    p_return_err:     Текст для сообщения об ошибке}
procedure TDistributionTaskExt.Solve;
var
    {индексные и временные переменные}
    i, j, jh, js, k, m, n, temp, intsum: integer;
    nh, ns, nc, nx: integer;
    realsum: double;

    low_flag: integer; {флаг очевидной нехватки ресурсов - это когда}
    {требование базовой ОС по одному из типов ресурсов больше либо равно}
    {имеющегося на заданном физическом компьютере}

    {косвенный индекс для наиболее оптимального}
    jh_best: integer; {по эффективности загрузки физического компьютера}
    {оптимальное распределение логических серверов для него}
    x_best: PByteVector; {хранятся косвенные индексы}
    {оптимальное значение целевой функции загрузки для него}
    {при оптимально распределенных на него логических серверах}
    lmax_best: double;

    {Экземпляр класса задач условной булевой оптимизации}
    {Используется для нахождения оптимального распределения текущего}
    {множества логических серверов на один текущий физический компьютер}
    BortExt: TBoolOptimumTaskExt;
    a: PA_Matrix;      {Матрица (левая часть ограничений) булевой задачи}
    b: PB_Vector;      {Вектор (правая часть ограничений) булевой задачи}
    c: PC_Vector;      {Вектор целевой функции булевой задачи}
    x_cur: PByteVector; {Результирующий вектор координат точки оптимума}
    blmax_cur: double;  {Значение целевой функции в оптимальной точке}
    btotal_it: double;  {Число обработанных точек при нахождении оптимума}
    bort_flag: byte;    {флаг ошибки для булевой задачи}
    bort_error: string; {текст ошибки для булевой задачи}

    {локальные флаги-зашелки для ошибок}
    d_err1, d_err2, d_err3, d_err4 : byte;
begin
    {Решение задачи опирается на решение множества задач условной булевой
    оптимизации. Схема проста: Сначала для каждого физического компьютера
    решаем локальную задачу распределения на него допустимого количества
    и сочетания логических серверов из всех имеющихся. При этом достигается
    некоторое ненулевое положительное значение целевой функции, отражающей
    обобщенный показатель эффективности (с учетом всего множества различных
    типов ресурсов) загрузки физического компьютера. Далее это повторяется с

```

каждым из имеющихся физ. компьютеров, после чего выбирается лучший по показателю эффективности загрузки своих ресурсов и на этом он и все логические серверы которые на него распределены исключаются из дальнейшего рассмотрения. Таким образом, с каждым разом число физ. компьютеров и лог. серверов уменьшается, и когда что-то одно закончится, решение задачи также закончится. Конечно, очевидно, что может возникнуть ситуация когда перебрав все доступные физ. компьютеры, мы ни на один из них не смогли разместить ни одного лог. сервера, причем ситуация может возникнуть не только в начале решения задачи, но и в любой момент, тогда, очевидно, что в таком случае далее решать задачу нет смысла – просто часть ресурсов останется незанятой из-за ее недостаточности и часть лог. серверов останется нераспределенной.}

```
d_err1:= 0;      {Сброс локальных флагов ошибок}
d_err2:= 0;
d_err3:= 0;
d_err4:= 0;
p_return_err:= ''; {Сброс выходного текста ошибки}

{Проверка входных указателей и указателей в полях класса}
if (p_stop_flag=nil) or (p_task_state=nil) then
begin {если один из них пустой, то генерация соответствующей ошибки}
  p_return_err:= S_SRC_INV_PTRS;
  d_err1:= 1;
end;

{Копирование параметров размерностей в локальные текущие переменные}
nh:= base_nh; {Число физических компьютеров}
ns:= base_ns; {Число логических серверов}
nc:= base_nc; {Число типов ресурсов}
nx:= base_nx; {Число исключений}

if (d_err1 = 0) then
begin {Запись текущего состояния задачи распределения}
  try
    p_task_state^.CurrentNH:= 0; {Число обработанных физ.компьютеров}
    p_task_state^.RemainNH:= base_nh; {Число оставшихся физ.компьютеров}
    p_task_state^.RemainNS:= base_ns; {Число оставшихся лог.серверов}
    p_task_state^.TotalNH:= base_nh; {Общее число физ.компьютеров}
    p_task_state^.TotalNS:= base_ns; {Общее число лог.серверов}
    p_task_state^.Stage:= 'Solving Task...'; {Текущая стадия решения}
  except {в случае нарушения доступа}
    On EAccessViolation do
    begin
      p_return_err:= S_TASKSTATE_WR_ERR;
      d_err1:=1;
    end; {Генерация ошибки}
  end;
end;

{Собственно, начало основного блока решения задачи}
{Start processing the Distribution Task}
if d_err1 = 0 then
begin
  {Главный цикл: продолжается до тех пор пока не распределяться все}
  {логические сервера, либо не останется ни одного физ. компьютера, либо}
  {не возникнет ошибка (прерывание извне также распознается как ошибка)}
  {Repeat until list of servers | computers will be empty}
  repeat
    {Формирование задачи условной булевой оптимизации}
    {Число переменных = число оставшихся логических серверов}
    n:= ns; {n - number of variables}
    {Число ограничений = число типов ресурсов и оставшихся исключений}
    m:= nc + nx; {m - number of constraints}
```

```

{Подготовка временных данных для последующей}
{инициализации задачи булевой оптимизации}
a:= nil; b:= nil; c:= nil;
x_cur:= nil; x_best:= nil;

try {Попытка выделения памяти для временных данных}
  GetMem(a,m*n*sizeof(a^[1]));
  GetMem(b,m*sizeof(b^[1]));
  GetMem(c,n*sizeof(c^[1]));
  GetMem(x_cur,n*sizeof(x_cur^[1]));
  GetMem(x_best,n*sizeof(x_best^[1]));
except {Генерация ошибки в случае нехватки памяти}
  On EOutOfMemory do
    begin
      p_return_err:= S_LOCAL_DATA_MEM_ALLOC_ERR;
      d_err2:= 1;
    end;
end;

{Дополнительная проверка указателей}
if ((d_err2 = 0) and ((a = nil) or (b = nil) or (c = nil)
  or (x_cur = nil) or (x_best = nil))) then
  begin {Генерация ошибки в случае если один из них пустой}
    d_err2:= 1;
    p_return_err:= S_LOCAL_DATA_MEM_ALLOC_ERR;
  end;

if d_err2 = 0 then
  begin
    jh_best:=0; {Сброс косвенного индекса наилучшего по загрузке}
                  {ресурсов физического компьютера}
    lmax_best:=0; {Сброс оптимального вектора и значения}
                  {целевой функции для лучшего физического компьютера}
    for j:=1 to n do x_best^[j]:=0;

    {Repeat for each available physical computer}
    {Внутренний цикл перебора каждого из оставшихся физических}
    {компьютеров, среди которых выбирается лучший по показателю}
    {загрузки ресурсов при оптимальном распределении на него}
    {подмножества оставшихся логических серверов}
    k:=1;
    repeat
      {Generate constraints}
      for i:=1 to m do {Предварительное обнуление}
        begin {ограничений задачи условной}
          b^[i]:=0; {булевой оптимизации}
          for j:=1 to n do a^[n*(i-1)+j-1]:=0;
        end;

      {Generate main resource-constraints}
      low_flag:=0; {Сброс флага явной нехватки ресурсов}
      for i:=1 to nc do
        begin {Формирование первой группы ограничений}
          for j:=1 to n do
            a^[n*(i - 1) + j - 1]:=
              sst^[base_ns*(i - 1) + rss^[j] - 1];
            {Матрица (левая часть) = требования}
            {оставшихся логических серверов}
            b^[i]:= hst^[base_nh*(i - 1) + rhc^[k] - 1] - bst^[i];
            {Вектор (правая часть) = ресурсы текущего физ. компьютера}
            {за вычетом требований базовой ОС, если после вычета}
            {один из ресурсов окажется < 0 то это явная нехватка}
            if b^[i] < 0 then low_flag:=1; {Крайне важное условие!}
          end;
        end;
      k:=k+1;
    until low_flag=1;
  end;
end;

```



```

end;

if (nx > 0) then      {Если число ограничений больше нуля, то}
begin               {Формирование второй группы ограничений}
  for i:=1 to nx do
  begin
    for j:=1 to n do
      a^[n*(i - 1 + nc) + j - 1]:=
      ext^[base_ns*(vex^[i] - 1) + rss^[j] - 1];
      {Матрица (левая часть) = невырожденные исключения}
      b^[i + nc]:= 1;
      {Вектор (правая часть) - всегда все координаты = 1}
    end;
  end;
  {В результате после первых NC ограничений по ресурсам}
  {располагаются NX дополнительных ограничений-исключений}

  {Special checking for the breaking action by the user!}
  {Проверка флага прерывания процесса решения извне}
  CheckUserAbort (p_stop_flag, d_err3, p_return_err);
  {Если флаг окажется установленным, то в силу логики}
  {алгоритма процедура выйдет из всех циклов, освободит}
  {занятую память и вернет управление вызвавшей ее сущности}

  {Writing the physical computers progress to the Task State}
  if (d_err3 = 0) then
  begin {Запись текущего состояния задачи распределения}
    try {Запись текущего числа обработанных физ. компьютеров}
      p_task_state^.CurrentH:= k;
    except {в случае нарушения доступа}
      On EAccessViolation do
      begin
        p_return_err:= S_TASKSTATE_WR_ERR;
        d_err3:=1;
      end; {Генерация ошибки}
    end;
  end;
end;

{Если нет явной нехватки ресурсов то переходим к}
{генерации целевой функции для булевой задачи}
if ((low_flag = 0) and (d_err3 = 0)) then
begin
  {Generate Criteria Function}
  {Вычисляем общее число критериальных ресурсов,}
  {по которым необходимо оптимизировать загрузку}
  temp:= 0;
  for i:=1 to nc do temp:= temp + cm^[i];

  for j:=1 to n do
  begin
    c^[j]:=0;
    {Для каждого логического сервера вычисляем его}
    {удельный вклад в интегральную загрузку по всем}
    {критериальным (для которых соответствующая}
    {координата вектора маски критериев = 1) ресурсам}

    for i:=1 to nc do if ((cm^[i]=1) and (b^[i]>0)) then
      c^[j]:=c^[j] + ((a^[n*(i-1)+j-1] / b^[i])) * 100;
      {Мы рассматриваем только ограничения по ресурсам,}
      {исключения здесь никакой роли не играют}

    {Коэффициент также нормируется по числу критериев}
    if (temp > 0) then c^[j]:= c^[j] / temp;
  end;
end;

```

```

{В результате такого подхода, каждый критериальный}
{ресурс и каждый лог. сервер будет иметь значение}
{При этом наивысшее возможно значение целевой функции}
{равно 100%, при котором все критериальные ресурсы}
{загружены на 100%}
end;

{Solve the optimization task with the boolean variables}
{Попытка создания и инициализации экземпляра задачи}
try { условной булевой оптимизации}
  BoptExt:= nil;
  BoptExt:= TBoolOptimumTaskExt.Init (m, n,
    max_radius, max_starts,
    a, b, c, x_cur, @blmax_cur, @bttotal_it,
    bopt_flag, bopt_error);

  {Если задача успешно создана и инициализирована}
  {то запускаем ее решение, передав ей также указатель}
  {на флаг принудительного останова и указатель на}
  {переменную состояния задачи.}
  if ((BoptExt <> nil) and (bopt_flag = 0)) then
    BoptExt.Run (bopt_flag, bopt_error,
      p_stop_flag, p_task_state);

  if (BoptExt <> nil) then
    begin {В случае если задача не была создана}
      if (bopt_flag <> 0) then
        begin {или возникла ошибка на этапе}
          d_err3:= 1; {инициализации или решения}
          p_return_err:= bopt_error;
        end; {генерируем ошибку, включив в текст}
      end; {ошибки и текст ошибки булевой задачи}
      if (BoptExt <> nil) then BoptExt.Done;
      BoptExt:= nil;
    except
      On EOutOfMemory do
        begin {В случае нехватки памяти при создании}
          d_err3:= 1; {задачи генерируем ошибку}
          p_return_err:=
            S_BOPTTEXT_MEM_ERR;
        end;
      On EAccessViolation do
        begin {В случае нарушения доступа при создании}
          d_err3:= 1; {задачи генерируем ошибку}
          p_return_err:=
            S_BOPTTEXT_ACCVIOL_ERR;
        end;
    end;

  {Если задача без ошибок решена, то сравниваем}
  {оптимальное значение целевой функции с максимальным}
  {достигнутым значением, полученным при решении задачи}
  {при других физических компьютерах и если текущее значение}
  {лучше, то оно становится новым максимумом и сохраняем}
  {его косвенный индекс k, абсолютный индекс физического}
  {компьютера - это rhc^[jh_best] - jh_best-й элемент}
  {вектора RHC, именно там истинный индекс физ. компьютера}
  {Также сохраняем полученное оптимальное распределение}
  {логических серверов в векторе x_best - в этом векторе}
  {"1" в тех элементах, индексы которых соответствуют}
  {косвенным индексам распределенные лог. серверов,}
  {абсолютные индексы можно получить из вектора RSS}

  if (d_err3 = 0) then

```

```

begin
  if blmax_cur > lmax_best then
    begin
      jh_best:=k;
      lmax_best:= blmax_cur;
      for j:=1 to n do x_best[j]:=x_cur[j];
    end;
  end; {End for "if d_err3 = 0" }

end; {End for "if low_flag = 0" }

k:= k + 1; {Next available physical computer}
{Переход к следующему физическому компьютеру из оставшихся}
until ((k > nh) OR (d_err3 <> 0));

{После перебора всех доступных физических компьютеров, проверяем}
{было ли найдено хотя бы какое-нибудь распределение на одном}
{из оставшихся физических компьютеров. Если ни на один из}
{физических компьютеров не удалось распределить ни один из}
{логических, то jh_best окажется равным нулю, т.к. решения}
{задач всегда будет возвращать нулевое значение целевой функции}
{из-за отсутствия допустимого непустого распределения, а это}
{значит далее задачу будет бессмысленно решать}

if ((jh_best <> 0) and (d_err3 = 0)) then
begin
  {В матрицу распределения для выбранного лучшего физического}
  {компьютера помечаем все логические серверы, которые на него}
  {оптимально распределились. Здесь мы уже обязаны работать с}
  {абсолютными индексами серверов, поэтому для этого}
  {используем векторы индексов доступных (оставшихся)}
  {на данный момент физ. компьютеров и логических серверов}
  for js:=1 to ns do if x_best[js]=1 then
    dt^[base_ns*(rhc^[jh_best]-1)+rss^[js]-1]:=1;

  {Удаление индекса наилучшего физического компьютера}
  {из вектора индексов оставшихся физ. компьютеров}
  temp:=0;
  for jh:=1 to nh do {переписываем элементы из вектора}
    begin {оставшихся физ. компьютеров}
      if (jh <> jh_best) then
        begin {во временный вектор, но без элемента}
          {соответствующего выбранному лучшему компьютеру}
          temp:= temp + 1;
          rhc_temp^[temp]:= rhc^[jh];
        end;
      end;
    {копируем элементы из временного вектора}
    {в вектор индексов оставшихся физ. компьютеров}
    for jh:=1 to temp do rhc^[jh]:= rhc_temp^[jh];
    {число физ. компьютеров уменьшается на 1}
    nh:= nh - 1;

  {Удаление индексов лог. серверов, распределенных}
  {на выбранный наилучший физический компьютер}
  {из вектора индексов оставшихся лог. серверов}
  temp:=0;
  for js:=1 to ns do {переписываем элементы из вектора}
    begin {оставшихся лог. серверов}
      if x_best[js]=0 then
        begin {во временный вектор, но без элементов}
          {соответствующих лог. серверам, распределенных}
          {на выбранный наилучший физический компьютер}
          temp:= temp + 1;

```

```

        rss_temp^[temp] := rss^[js];
    end;
end;
{копируем элементы из временного вектора}
{в вектор индексов оставшихся лог. серверов}
for js:=1 to temp do rss^[js] := rss_temp^[js];
{число лог. серверов = число оставшихся серверов}
{после удаления индексов лог. серверов, распределенных}
{на выбранный наилучший физический компьютер}
ns := temp;

{Если число исключений не равно 0, то в матрице исключений
среди исключений, использовавшихся ранее как невырожденные
ищем вновь появившиеся вырожденные, поскольку множество
индексов оставшихся логических серверов изменилось}
{Удаляем из вектора индексов невырожденных исключений
индексы вновь появившихся вырожденных исключений}
if (nx > 0) then
begin
    temp:=0;
    for i:=1 to nx do
    begin {подсчитываем число "1" для каждого}
        intsum:=0; {бывшего невырожденным исключения}
        for js:=1 to ns do
        {но теперь уже с учетом нового множества}
        {индексов оставшихся логических серверов}
        intsum:= intsum +
        ext^[base_ns*(vex^[i]-1) + rss^[js]-1];
        {Если число "1" >= 2, то исключение - невырожденное}
        if (intsum >=2) then
        begin {сохраняем его индекс}
            temp:= temp + 1; {во временном векторе}
            vex_temp^[temp] := vex^[i];
        end;
    end;
    {копируем элементы из временного вектора}
    {в вектор индексов невырожденных исключений}
    for i:=1 to temp do vex^[i] := vex_temp^[i];
    {Число исключений = число невырожденных исключений}
    nx:= temp;
end;
end;
end; {End for "if d_err2 = 0" }

{Запись состояния задачи}
if (d_err2 = 0) then
begin
    try {Запись числа оставшихся физ. компьютеров и лог. серверов}
        p_task_state^.RemainNH:= nh; {Число оставшихся физ. комп.}
        p_task_state^.RemainNS:= ns; {Число оставшихся лог. серв.}
    except
        On EAccessViolation do
        begin {Генерируем ошибку при нарушении доступа}
            p_return_err:= S_TASKSTATE_WR_ERR;
            d_err2:=1;
        end;
    end;
end;
end;

{Освобождаем память временных данных для задачи булевой оптимизации}
if a <> nil then FreeMem(a,m*n*sizeof(a^[1]));
if b <> nil then FreeMem(b,m*sizeof(b^[1]));
if c <> nil then FreeMem(c,n*sizeof(c^[1]));
if x_cur <> nil then FreeMem(x_cur,n*sizeof(x_cur^[1]));

```

```

if x_best <> nil then FreeMem(x_best, n*sizeof(x_best^[1]));
a:= nil; b:= nil; c:= nil;
x_cur:= nil; x_best:= nil;

{Если число физ. компьютеров и лог. серверов}
{не равны нулю, если не возникло ошибок и если было найдено}
{хотя бы одно непустое распределение для одного из}
{оставшихся физ. компьютеров, то продолжаем цикл}
until ((ns = 0) OR (nh = 0) OR (jh_best = 0)
      OR (d_err2 <> 0) OR (d_err3 <> 0));
end; {End for "if d_err1 = 0" }

if ((d_err1 = 0) and (d_err2 = 0) and (d_err3 = 0)) then
begin
  try
    p_task_state^.Stage:= 'Processing results...';
  except {в случае нарушения доступа}
    On EAccessViolation do
    begin
      p_return_err:= S_TASKSTATE_WR_ERR;
      d_err4:=1;
      end; {Генерация ошибки}
    end;

  jh:=1;          {Для каждого физического компьютера}
  while ((jh <= base_nh) and (d_err4 = 0)) do
  begin
    i:= 1;
    while ((i <= base_nc) and (d_err4 = 0)) do
    begin
      {Для текущего физ. компьютера, рассчитываем}
      realsum:=0;      {загрузку всех типов его ресурсов по известному}
      js:= 1;          {распределению на него лог. серверов}
      while ((js <= base_ns) and (d_err4 = 0)) do
      begin
        {Для всех распределенных на него лог. серверов суммируем}
        {их требования по каждому типу ресурсов в отдельности}
        if (dt^[base_ns*(jh-1)+js-1] = 1) then
          realsum:=realsum + sst^[(i-1)*base_ns+(js-1)];
        {Check the user abort flag}
        {Проверка флага прерывания выполнения извне}
        CheckUserAbort (p_stop_flag, d_err4, p_return_err);

        js:= js + 1; {переход к следующему лог. серверу}
      end;
      {по каждому типу ресурсов делим сумму требований на}
      {ресурс физ. компьютера за вычетом требований базовой ОС}
      if (realsum > 0) and ((hst^[(i-1)*base_nh+(jh-1)]-bst^[i]) > 0)
      then realsum:= 100 * (realsum /
        (hst^[(i-1)*base_nh+(jh-1)]-bst^[i]));
      rl^[base_nc*(jh-1)+i-1]:= realsum;

      i:= i + 1; {переход к следующему типу ресурсов}
    end;
    jh:= jh + 1; {переход к следующему физ. компьютеру}
  end;
end;

{В независимости от ошибок информация}
{о состоянии задачи должна быть обнулена}
try {Обнуление состояния задачи распределения при ее завершении}
  p_task_state^.CurrentH:= 0; {Число обработанных физ.компьютеров}
  p_task_state^.RemainNH:= 0; {Число оставшихся физ.компьютеров}
  p_task_state^.RemainNS:= 0; {Число оставшихся лог.серверов}
  p_task_state^.TotalNH:= 0; {Общее число физ.компьютеров}

```

```

    p_task_state^.TotalNS:= 0;      {Общее число лог.серверов}
    p_task_state^.Stage:= ''; {Стадия решения задачи}
except
    On EAccessViolation do {Если возникло нарушение доступа}
    begin
        {то генерация ошибки}
        p_return_err:= S_TASKSTATE_WR_ERR;
        d_err4:= 1;
    end;
end;

{Подготовка выходного флага и текста ошибки}
if ((d_err1 <> 0) or (d_err2 <> 0) or (d_err3 <> 0) or (d_err4 <> 0))
then p_return_flag:= 1 else p_return_flag:= 0;
p_return_err:= DISTEXT_ERR_HEAD + p_return_err;
end;

{Деструктор - уничтожает экземпляр класса}
destructor TDistributionTaskExt.Done;
begin
    {Освобождение памяти индексных векторов и временных векторов}
    if ((base_nx > 0) and (vex <> nil))
    then FreeMem(vex, base_nx * sizeof(vex^[1]));
    if ((base_nx > 0) and (vex_temp <> nil))
    then FreeMem(vex_temp, base_nx * sizeof(vex_temp^[1]));
    vex:= nil; vex_temp:= nil;
    {Вызов унаследованного деструктора}
    Inherited Done;
end;
end.

```

Исходный код дополнительного модуля (окно информации об авторе)

```

interface
{Используемые модули}
uses AUTHORITY, RSC32, Windows, SysUtils, Classes,
    Graphics, Forms, Buttons, ExtCtrls, StdCtrls, Controls;

type
  TAboutBox = class(TForm)           {Класс формы для окна}
  {Поля класса}
    Panell: TPanel;                 {Панель внутри окна}
    ProgramIcon: TImage;             {Статическое изображение - логотип}
    ProductName: TLabel;
    ProductVersion: TLabel;          {Статический текст - версия программы}
    DesignedBy: TLabel;              {Статический текст - строка об авторстве}
    Authority: TLabel;               {Статический текст - автор программы}
    OKButton: TButton;               {Кнопка для закрытия окна}

  {Методы класса}
    {Обработчик события создания формы для окна}
    procedure FormCreate(Sender: TObject);
  end;

var
  AboutBox: TAboutBox;               {экземпляр класса формы для окна}

implementation

{$R *.dfm}

{Обработчик события создания формы для окна}
{Входные параметры:
  Sender: компонента, с которой связано событие}

procedure TAboutBox.FormCreate(Sender: TObject);
var
  MyName: TAuthority;                {Здесь производится создание экземпляра}

begin
  {специального класса и вызов его методов}
  MyName:= nil;                      {для получения информации об истинном авторе.}
  MyName:=TAuthority.Create;          {Класс и его методы находятся в модуле}
  if (MyName <> nil) then              {AUTHORITY, для которого автор}
    Authority.Caption:= MyName.GetAuthority; {оставляет за собой право}
    if (MyName <> nil) then MyName.Done;    {не публиковать его код}
    ProductVersion.Caption:='Version: 1.8.16.0';
  end;

end.

```

## **Приложение 6. Акт о внедрении**