**Министерство образования и науки Российской Федерации**

**Федеральное агентство по образованию**

**ГОУ Владимирский государственный университет**

**Кафедра информационных систем и информационного менеджмента**

# РЕШЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧ С ПОМОЩЬЮ SUN N1 GRID ENGINE

ВЛГУ. 230100. 12. 04. 00. ПЗ

студент гр. ИМ-105

_____ Н. В. Морев

”___”_____ 2006 г

2006

## Содержание



Подп. и дата | Инв. № дубл. | Взам. инв. № | Подп. и дата | Инв. № подл.

| Изм. | Лист | № докум. | Подп. | Дата | ВлГУ. 230100. 12. 04. 00. ПЗ | | |
|---|---|---|---|---|---|---|---|
| Разраб. | Морев Н. В. | | | | Решение вычислительных задач с помощью Sun N1 Grid Engine | Лит. | Лист 2 / Листов 28 |
| Пров. | | | | | | | |
| Н. контр. | | | | | | ИМ-105 | |
| Утв. | | | | | | | |

# 1 Введение

Количество компонентов, помещающихся на одной интегральной микросхеме, а соответственно и производительность вычислительных систем непрерывно растет. Причем этот рост имеет экспоненциальный характер. Примерно также с каждым годом увеличиваются и другие характеристики ВС: объем оперативной памяти, вместимость постоянных носителей информации, пропускная способность линий связи. Но несмотря на это, потребность в вычислительных ресурсах не удовлетворяется современными вычислительными средствами. Ученые производят огромное количество экспериментальных данных, которые необходимо хранить и анализировать, предпринимаются попытки моделирования сверхсложных процессов, например свертка белков[1]. Крупные предприятия нуждаются в мощных информационных системах, поддерживающих их бизнес, а также решениях, позволяющих более эффективно использовать имеющиеся вычислительные средства. У любого современного бизнеса возникают задачи интеграции информационных систем для взаимодействия с партнерами или аутсорсинга технологических процессов обработки информации. Все эти задачи невозможно эффективно решать без использования распределенных систем.

На сегодняшний день существует два устоявшихся подхода к созданию высокопроизводительных систем: суперкомпьютеры и кластеры. Кластеры — более выгодная альтернатива традиционным суперкомпьютерам. Всегда проще наращивать какие-либо показатели соединяя множество простых компонентов, чем используя один сложный комплекс. Однако, оба эти подхода имеют ряд недостатков, критичных для их массового распространения. Недостатки эти связаны с такими применениями распределенных систем, в которых свойство «распределенности» гораздо важнее производительности: интеграция независимых информационных систем, объединение значительно разнесенных в пространстве вычислительных ресурсов, создание систем, в разы превосходящих по количеству узлов самые мощные кластеры, и т. д.

Грид-вычисления — это следующий шаг в развитии распределенных вычислений. Основное отличие гридов от кластеров заключается в том, что гриды могут быть гетерогенными, т. е. ячейки грида могут функционировать на разном оборудовании, под разными операционными системами и быть значительно распределенными в пространстве, соединяясь посредством глобальных сетей, в то время как обычные кластеры должны быть гомогенными, работать под одной и той же ОС и подчиняться определенным правилам по совместимости оборудования. Кроме того настоящий грид может быть распределен по обычным пользовательским рабочим компьютерам, в то время как понятие

[1] http://folding.stanford.edu

кластера чаще всего связано со специально выделенными для них вычислительными центрами.

Кроме того определение грида можно дать на основе характеристик, которыми они обладают или должны обладать, исходя из идей теоретиков этого направления распределенных вычислений:

- Огромные масштабы системы. Гриды обычно имеют дело с количеством ресурсов от нескольких единиц до нескольких миллионов.
- Географическая распределенность.
- Гетерогенность. Грид состоит из аппаратных и программных ресурсов, которыми могут быть, как данные, программные компоненты или программы, так и датчики, научные инструменты, дисплеи, компьютеры, суперкомпьютеры, сети.
- Распределенное использование ресурсов. Ресурсы грида, принадлежащие одним организациям могут предоставляться в пользование другим организациям или пользователям.
- Множество различных административных ограничений. Каждая огранизация может устанавливать свои правила использования ресурсов.
- Координация ресурсов. Ресурсы грида необходимо координировать, чтобы система была единым целым.
- Прозрачный доступ. С пользовательской точки зрения грид должен выглядеть, как один большой виртуальный компьютер.
- Надежность. Грид должен обеспечивать необходимое качество сервиса.
- Совместимость. Все взаимодействия в гриде должны быть основаны на стандартных и открытых протоколах.
- Ресурсы должны адаптироваться, к изменяющейся среде грида так, чтобы обеспечить надежность вычислений и наибольшую эффективность.

# 2 Технологии построения грид-систем

Грид — это устойчивая инфраструктура, поддерживающая вычислительно-требовательные и оперирующие большими объемами данных совместные задачи, особенно в случае, когда задачи охватывают несколько организаций. Грид-вычисления способствуют образованию «виртуальных организаций» для совместного использования распределенных вычислительных ресурсов.

Учитывая значительную распределенность узлов грид-системы, для решения на гриде лучше всего подходят задачи, удовлетворяющие следующим требованиям:

- Задачу возможно разделить на параллельные подзадачи, которые совсем не зависят или мало зависят друг от друга по данным.
- Соотношение количества данных к объему вычислений должно быть достаточно мало, т.к. данные передаются по Интернет-каналам, имеющим ограничения по пропускной способности и цене.
- В некоторых случаях, например, в волонтерских вычислениях невозможно быть уверенным на все 100% в корректности данных, полученных от узлов, поэтому в приложении, выполняющем анализ данных это необходимо предусмотреть, например введя избыточность.

## 2.1 Настольные грид-вычисления

Наиболее успешные на сегодняшний день грид-проекты по количеству пользователей и освещению в СМИ, такие как *SETI@Home* (анализ данных, поступающих с радиотелескопа в Аресибо, Пуэрто-Рико), *distributed.net* (взлом стойких криптографических алгоритмов), стали основой для продвижения идеи грид-систем в массовую аудиторию. Общее название для такого типа проектов — настольные грид-вычисления (desktop grid computing) или волонтерские вычисления (volunteer computing). Общим для всех подобных проектов является то, что их целевой аудиторией и вычислительным ядром являются обычные пользователи, подключенные к Интернету постоянно или время от времени, а задачи выполняются, когда компьютер простаивает без вычислительной нагрузки. Хотя в принципе эти задачи могут выполняться и на более серьезное технике, типа серверов и даже кластеров.

Изначально под каждый такой проект с нуля писали серверное и клиентское программное обеспечение, не основываясь на каких-либо уже существующих разработках в этой области. Это затрудняло создание новых проектов для ученых и переход с одного проекта на другой для пользователей, которым приходилось каждый раз скачивать

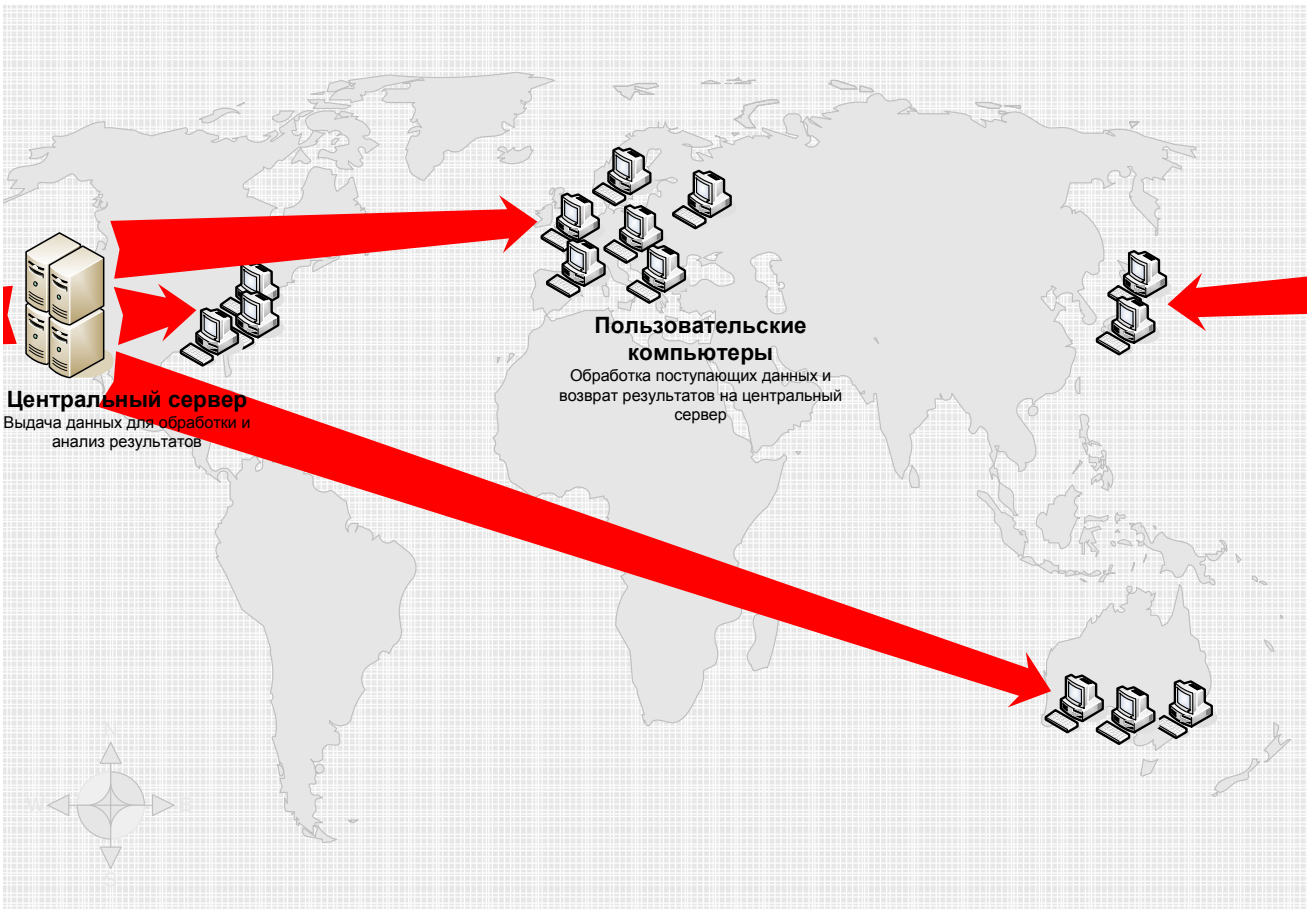



Рис. 1: Настольные грид-вычисления

новое ПО для новой задачи. Это приводит к тому, что иногда клиенты для старых проектов продолжают работать на некотором количестве машин даже после того, как проект прекратил свое существование.

После того, как проект *SETI@Home* доказал результативность волонтерских вычислений для решения вычислительно-емких научных задач, в калифорнийском университете Беркли, родоначальнике этого проекта, была разработана общая инфраструктура для таких проектов — *BOINC*.

*BOINC* (Berkeley Open Infrastructure for Network Computing) — это программная платформа для распределенных вычислений, использующих волонтерские вычислительные ресурсы. Она может быть использована для самых разных независимых друг от друга проектов, каждый из которых использует свои собственные сервера. Пользователи могут участвовать в нескольких проектах и управлять распределением своих ресурсов между ними. Когда определенный проект испытывает технические трудности или прекратил выдачу данных для обработки, вычислительные ресурсы пользователя автоматически перебрасываются на другой проект.

Среди других характеристик *BOINC* можно отметить:

- Позволяет использовать существующие приложения с минимальными изменениями. Новые версии приложений устанавливаются автоматически без участия пользователя.
- Продуманная система безопасности, использование цифровой подписи.
- Поддержка распределенности и отказоустойчивости на стороне сервера.
- Открытый исходный код под лицензией LGPL.
- Поддержка работы с большими объемами данных.

По данным на сентябрь 2006 года, в *BOINC* участвуют несколько десятков проектов и 475.000 активных компьютеров общей средней вычислительной мощностью около 615 TFLOPS, что в несколько раз превосходит самый мощный суперкомпьютер из списка *Top 500*.

Схема взаимодействия программ в типичном проекте, использующем инфраструктуру *BOINC* показана на рис. 3. Серверная часть состоит из специфичной для проекта части, которая выдает задания пользовательским приложениям и производит анализ результатов, и комплекса серверов *BOINC*, которые включают:

- Один или несколько серверов планировщиков, осуществляющих взаимодействие с пользовательскими приложениями.

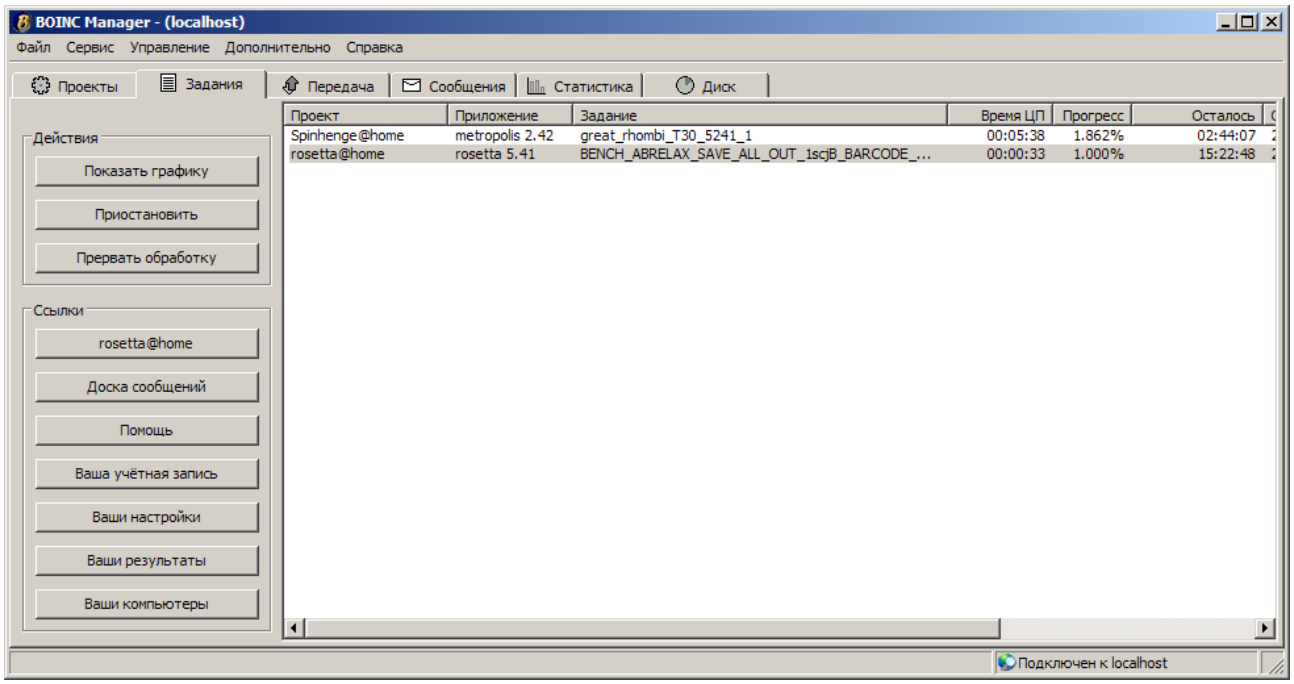



Рис. 2: Окно менеджера заданий *BOINC*

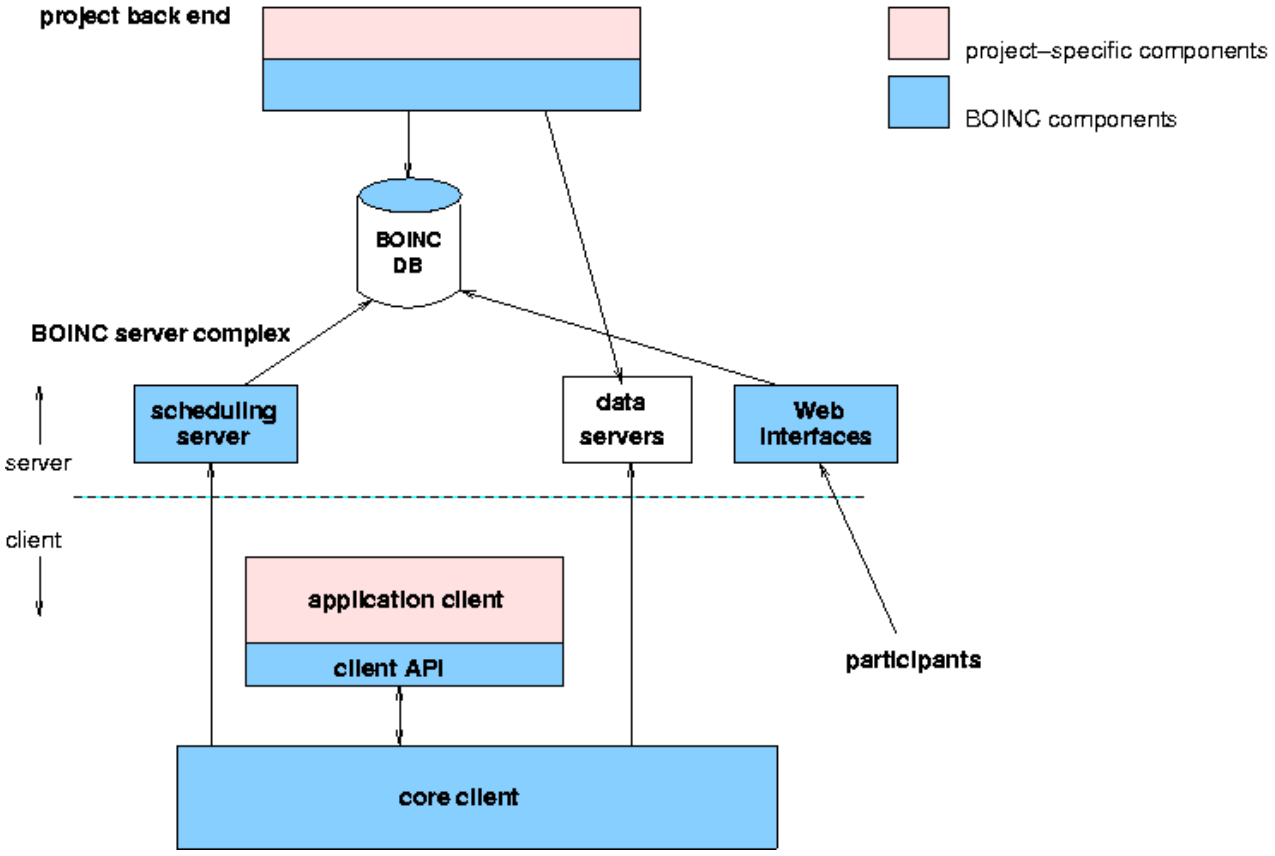



Рис. 3: Схема взаимодействия программ в проекте под управлением *BOINC*

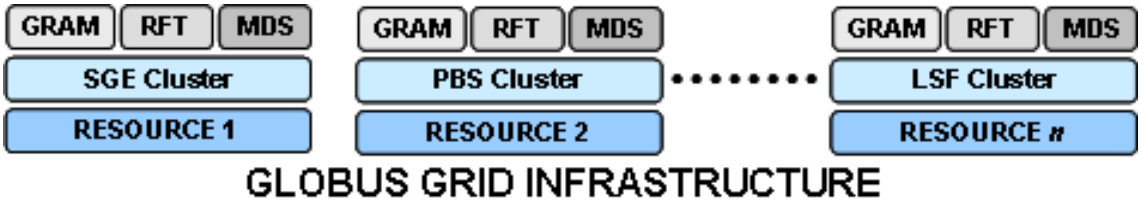



Рис. 4: Модель инфраструктуры грида с использованием Globus Toolkit

- Реляционная база данных, хранящая информацию о работе, результатах и участниках.
- Утилиты, которые осуществляют взаимодействие специфичной части проекта с комплексом серверов *BOINC*.
- Веб-интерфейс для участников и разработчиков.
- Сервера данных, распространяющие входные файлы и собирающие результаты.

### 2.2 Программное обеспечение промежуточного уровня

Другой вид грид-систем, часто использующийся на практике в различных научных проектах, предполагает объединение уже существующих в учреждениях мощных вычислительных кластеров в единую вычислительную систему, превосходящую по своим параметрам и возможностям каждый из кластеров по отдельности. Примеры таких проектов: Enabling Grids for E-sciencE (EGEE), TeraGrid. На каждом из узлов такой системы может быть установлено различное ПО, обеспечивающее распараллеливание. Наиболее популярны: среды мультипрограммирования PVM, MPI, системы распределения задач PBS, Condor, Sun Grid Engine (SGE). При этом стоит задача унифицировать доступ к такой гетерогенной среде (рис. 4).

Связывание этого множества разнородных систем в единую среду достигается с помощью различных открытых стандартов, не последнюю роль среди которых играют стандарты организации Open Grid Forum[2] (OGF), образовавшейся в 2006 году слиянием Global Grid Forum (GGF) и Enterprise Grid Alliance. Это организация разрабатывает спецификацию открытой архитектуры грид-сервисов (OGSA), которая определяет основные архитектурные принципы, на которых должны строиться грид-системы. Здесь сервис определяется, как сетевая сущность, предоставляющая какой-либо ресурс: вычислительные ресурсы, хранилища данных, сетевые средства, программы и базы данных.

Доступная на данный момент реализация OGSA — это Globus Toolkit[3] (GT). Globus Toolkit разрабатывается с конца 90-х годов для создания средств поддержки разработ-

[2] http://www.ogf.org/
[3] http://www.globus.org/toolkit/

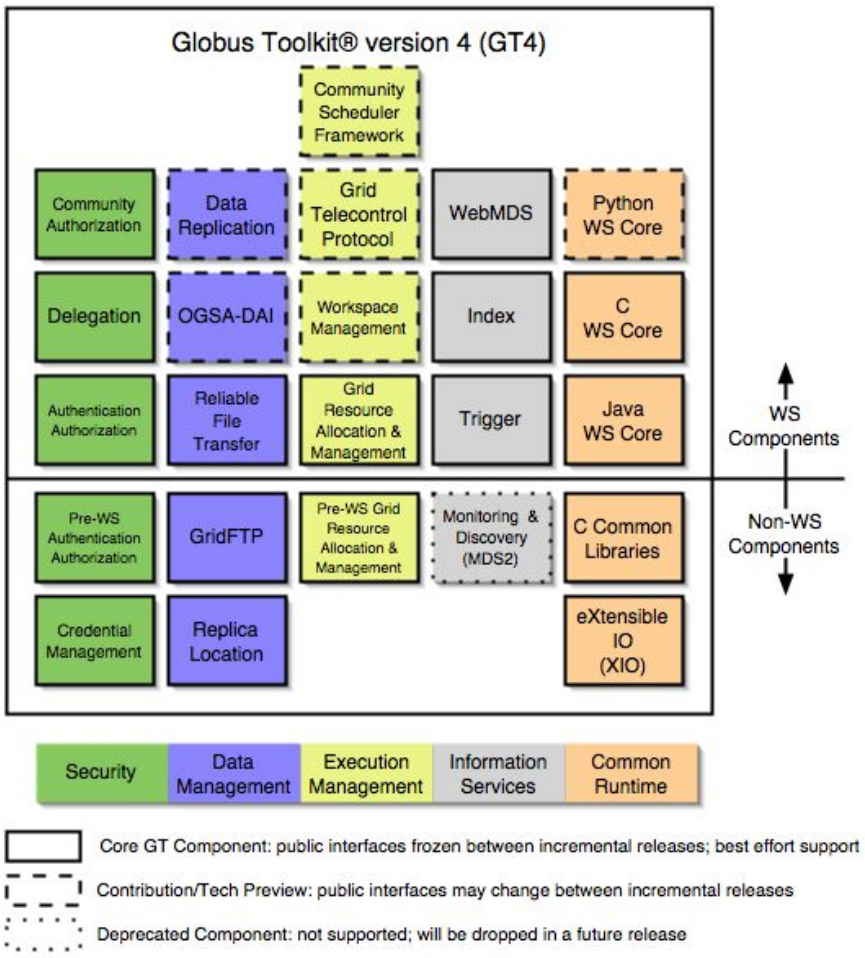



Рис. 5: Компоненты Globus Toolkit 4

ки сервисно-ориентированных распределенных вычислительных приложений и инфраструктур. Компоненты ядра GT связаны с базовыми потребностями грида: обеспечение безопасности, доступ к ресурсам и управление ими, перемещение данных, адресация ресурсов и т.д. (рис. 5). Дополнительные компоненты GT строятся на основе базовых и предоставляют различные полезные функции уровня приложений. GT используется в той или иной мере в большинстве научных грид-проектов.

### 2.3 Промышленные грид-системы

Основная особенность развития грид технологий в коммерческом секторе заключается в том, что до сих пор в общественном сознании не сложилось точного и всеобъемлющего понимания этого понятия. Компании разработчики решений в области информационных технологий пользуются этим и пытаются «тянуть одеяло на себя», раскручивая свое собственное определение грида, соответствующее тем разработкам, которые данная компания предлагает. Тем самым вносится еще бóльшая неразбериха. В этой работе технология грид рассматривается только в аспекте распределенных вычислений.

Надежды на технологию грид в отделах ИТ предприятий в основном связывают со следующими возможностями:

- Консолидация отдельных серверов для решения общих задач, равномерное распределение нагрузки на отдельные вычислительные ресурсы предприятия, загрузка недоиспользованых вычислительных ресурсов. Все это позволит минимизировать задержки, связанные с обработкой информации.
- Использование простаивающих без нагрузки компьютеров на рабочих местах для решения задач, стоящих перед предприятием.
- Перевод ИТ инфраструктуры от дорогих высокопроизводительных суперкомпьютеров к комплексу из дешевых, легко заменяемых серверов. Кроме минимизации издержек такой подход также позволяет ввести избыточность и дублирование вычислительных ресурсов, что снижает или совсем устраняет негативные последствия сбоев. Кроме того это положительно отражается на масштабируемости инфраструктуры — мы легко можем добавить несколько серверов к остальным, если требуется бóльшая вычислительная мощность.

Нетрудно заметить, что реализовать данные цели на практике проще всего, когда вычисления производятся в виде пакетного выполнения задач, т. е. когда задача некоторое достаточно продолжительное время выполняется с заданными входными данными и выдает результат. Это значит, что грид не поможет, например, ускорить работу интерактивных приложений. Кроме того, технология грид не выполняет разбивку крупной

Рис. 6: Кластер в CERN собранный из обычных дешевых компьютеров

задачи на мелкие подзадачи для выполнения на разных узлах, это должно быть сделано заранее.

Хотя с ходу может быть сложно представить себе бизнес-задачи, которые могут удовлетворять всем этим требованиям, но такие задачи все-таки существуют: рендеринг трехмерных моделей, обработка очень больших изображений, моделирование в машиностроении, финансах, расчет рисков, логистика и так далее.

Для параллельного выполнения подобных задач применяют программные средства, называемые «планировщиками задач» (job scheduler) или «системами организации очередей» (batch-queueing system). Иногда их также называют системами управления распределенными ресурсами (DRMS). Планировщик управляет совокупностью задач, которые были запущены на выполнение, и распределяет их в соответствии с приоритетом, оценочным временем выполнения, требуемыми ресурсами, зависимостями между задачами. Планировщик автоматически посылает задачи узлам на выполнение и выдает текущее состояние выполняемых задач.

Существует множество реализаций систем планирования задач, как коммерческих, так и с открытым исходным кодом. Среди наиболее известных можно выделить: Portable Batch System[4] (PBS), Condor, Globus Toolkit, Sun Grid Engine.

Sun Grid Engine (SGE) — это кроссплатформенный планировщик пакетных задач с открытым исходным кодом, поддерживаемый компанией Sun Microsystems. Sun также выпускает коммерческий вариант этой системы — Sun N1 Grid Engine. Эта система включает следующие возможности:

- Множество передовых алгоритмов планирования и мощные правила для описания выделения ресурсов под задачи.

[4] http://www.openpbs.org/

- Кластерные очереди.
- Отказоустойчивость задач и планировщика.
- Создание контрольных точек выполнения задач.
- Массивы задач.
- Поддержка DRMAA.
- Резервирование ресурсов.
- Отчеты о состоянии системы, в том числе в формате XML.
- Учет использованных ресурсов.

# 3 Программный интерфейс DRMAA для выполнения задач на грид-системе

Distributed Resource Management Application API (DRMAA) — Программный интерфейс для приложений управления распределенными ресурсами. Он предоставляет разработчикам приложений программную модель, которая позволяет разрабатывать распределенные приложения тесно связанные с нижележащей системой управления распределенными ресурсами (Distributed Resource Management System, DRMS). Для внедренцев таких распределенных приложений, DRMAA сохраняет гибкость и возможность выбора архитектуры системы.

Спецификация DRMAA призвана унифицировать интерфейсы систем DRMS, чтобы достичь переносимости между ними. Она разрабатывается специальной рабочей группой входящей в состав OGF. Все документы, в том числе рабочие версии, относящиеся к DRMAA можно скачать на сайте [3].

Благодаря тому, что в разработке стандарта DRMAA участвовали представители самых разных коммерческих и исследовательких организаций, он быстро был принят сообществом. В настоящее время существует несколько реализаций DRMS, поддерживающих этот API, из которых наиболее полной и стабильной является Sun Grid Engine. В спецификации API описывается астрактно на языке описания интерфейсов IDL, что позволило на уровне языков программирования реализовать поддежку DRMAA для C/C++, Java, Perl, Python, Ruby.

Спецификация DRMAA обеспечивает независимость грид-приложения от используемой DRMS. Для этого в нее введено понятие категории задания. При отсылке задачи на грид программист задает категорию, которая на конкретной системе отображается на совокупность настроек, которые могут включать указание требований приложения, приоритета выполнения и других специфических для DRMS параметров.

Спецификация DRMAA включает следующие процедуры:

- Инициализация и завершение грид-приложения.
- Задание шаблона задачи, включающее имя выполняемой команды, начальное состояние задачи, параметры среды выполнения, категорию задачи, потоки стандартного ввода/вывода и другие параметры.
- Процедуры отправки на грид отдельных задач и групповых задач.
- Мониторинг и контроль выполнения задач.

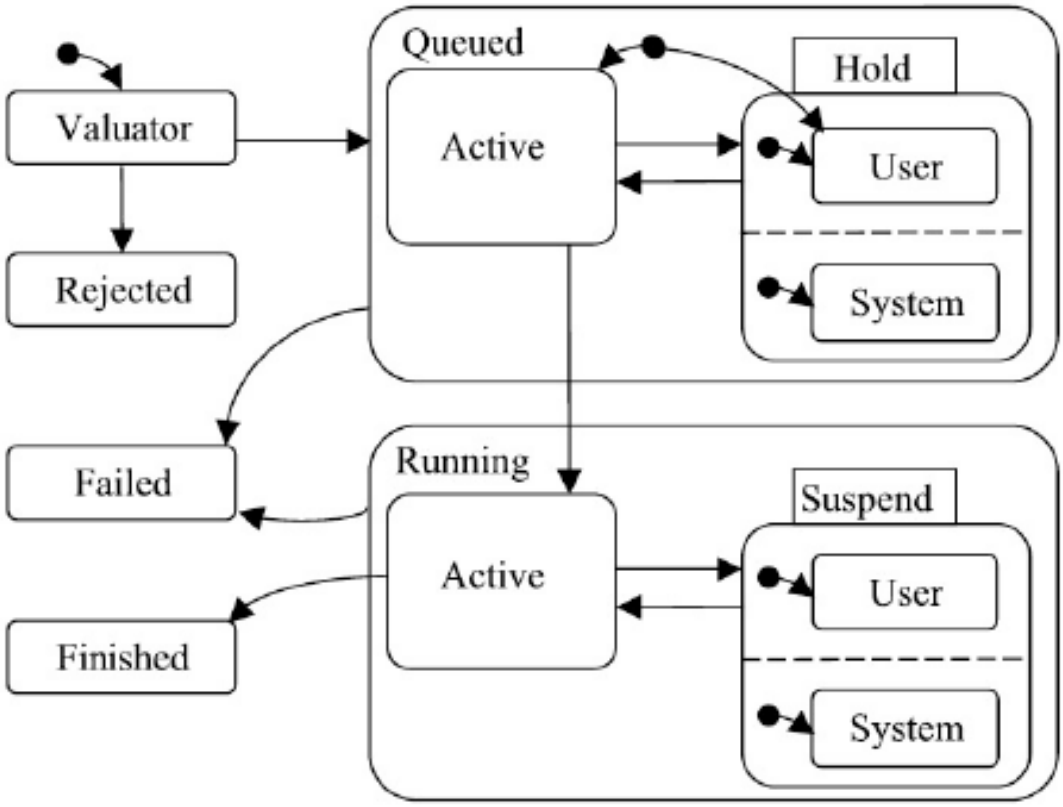



Рис. 7: Диаграмма состояний задачи в гриде

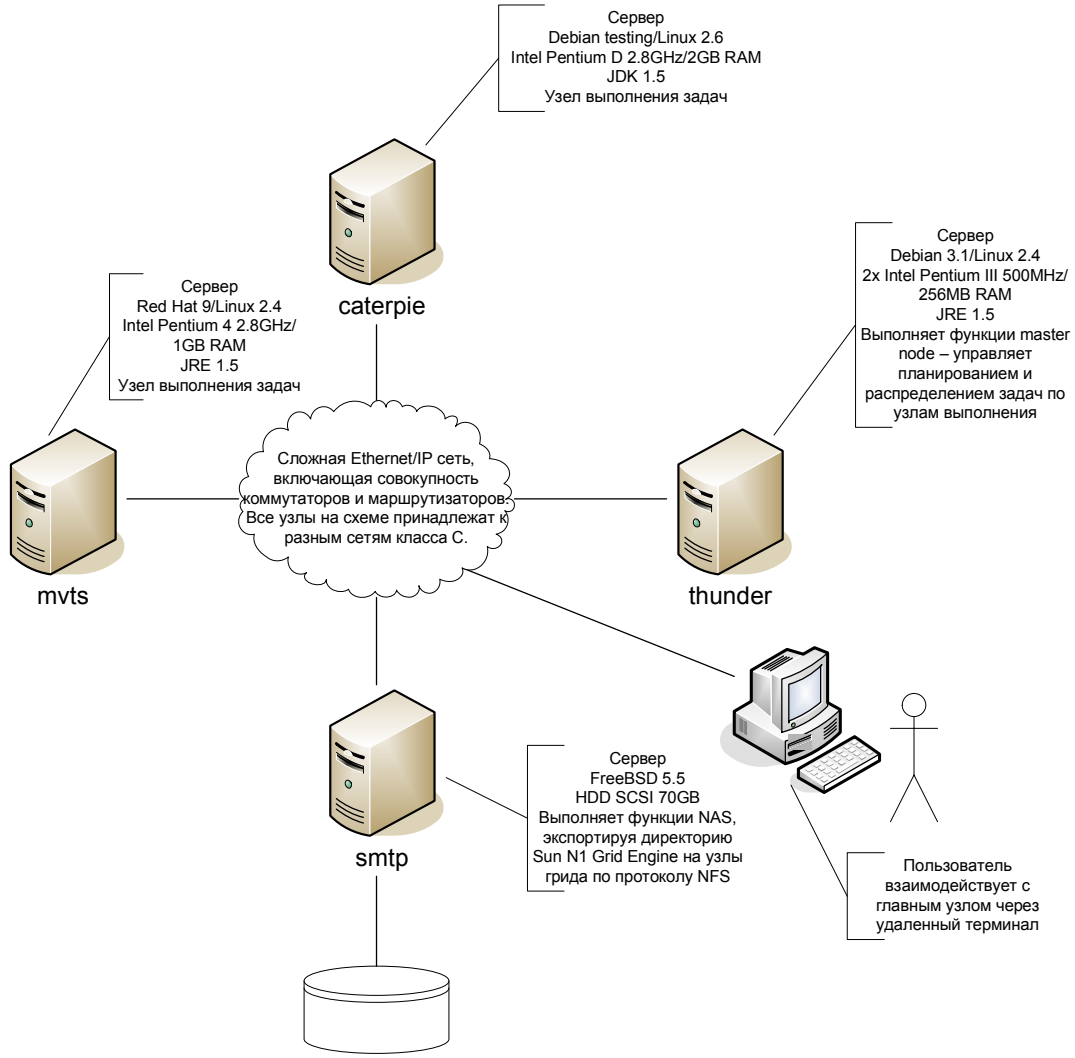



Рис. 8: Структура грид-системы

# 4 Структура грид-системы

Система Sun N1 Grid Engine версии 6.0u8 была развернута на 3 отдельных UNIX-серверах. На наименее мощном из них был установлен master-узел, осуществляющий прием задач, планирование их выполнения и отсылку на узлы исполнения. На остальных серверах установлены узлы выполнения задач. Еще один сервер был использован в качестве системы Network Attached Storage (NAS). С него директория, в которую была установлена система, экспортировалась на остальные узлы. Пользователь взаимодействует с грид-системой, заходя через удаленный терминал на master-узел. Структурная схема грида представлена на рис. 8.

## 4.1 Процедура развертывания системы

1. Экспорт директории по протоколу NFS.

Установить сервер NFS. Создать папку /usr/local/sge, распаковать дистрибутив SGE в нее. В файл `/etc/exports` добавить строки:

```
/usr/local/sge -maproot=root thunder
/usr/local/sge -maproot=root caterpie
/usr/local/sge -maproot=root mvts
```

Ввести команду `kill -s HUP 'cat /var/run/mountd.pid'` для перезагрузки файла `exports`.

2. Подмонтировать папку на остальных серверах.

   Добавить в файл `/etc/fstab` строку:

   ```
   smtp:/usr/local/sge /usr/local/n1ge/      nfs     rw,noauto,tcp 0 2
   ```

   Ввести команду `mount smtp:/usr/local/sge`, чтобы директория подмонтировалась к локальной файловой системе.

3. Добавить на всех узлах в файл `/etc/services` строки:

   ```
   sge_qmaster     536/tcp
   sge_execd       537/tcp
   ```

4. Установить master-узел.

   Перейти в папку `/usr/local/n1ge` и выполнить скрипт `./install_qmaster`. Запустится мастер установки, в котором надо выбрать все предложенные по-умолчанию параметры.

5. Установить узлы выполнения.

   На master-узле добавить узлы выполнения в список узлов, которым доступны администраторские команды, выполнив команду `qconf -ah <имя узла>`.

   Перейти в папку `/usr/local/n1ge` и выполнить скрипт `./install_execd`. Запустится мастер установки, в котором надо выбрать все предложенные по-умолчанию параметры.

6. Проверить правильность установки.

   На master-узле ввести команду `qhost`. Если все установлено правильно, она должна вывести список установленных узлов выполнения и их основные параметры:

```
HOSTNAME                ARCH         NCPU  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS
----------------------------------------------------------------------------------
global                  -               -     -       -       -       -       -
caterpie                lx24-x86        1  1.55    2.0G  878.8M    3.7G   52.0K
mvts                    lx24-x86        1  0.06 1006.7M  101.8M    1.9G    3.3M
```

# 5 Разработка алгоритма

В качестве предметной области был выбран поиск простых чисел. Алгоритмы, связанные с обработкой простых чисел, имеют очень широкое применение в информационных технологиях всвязи с тем, что их свойства являются основой алгоритмов криптографии с открытым ключом. Кроме того простые числа используются в хэш-таблицах и генерации псевдослучайных чисел.

Используемый алгоритм имеет неполиномиальную сложность и очень хорошо распараллеливается, что делает его идеальным кандидатом для реализации на грид-системе. Схема алгоритма представлена на рис. 9.

Основное приложение служит для разделения исходной задачи на подзадачи и запуска подзадач на гриде. Выделение подзадач производится путем разбиения области входных значений на блоки примерно одинакового размера. В процессе выполения подзадач приложение показывает их статус. По окончанию выводятся результаты — простые числа, найденные в заданном отрезке, а также использованное процессорное время. Для взаимодействия с системой DRMS используется программный интерфейс DRMAA, в частности его Java-реализация. Схема алгоритма работы основного приложения представлена на рис. 10.

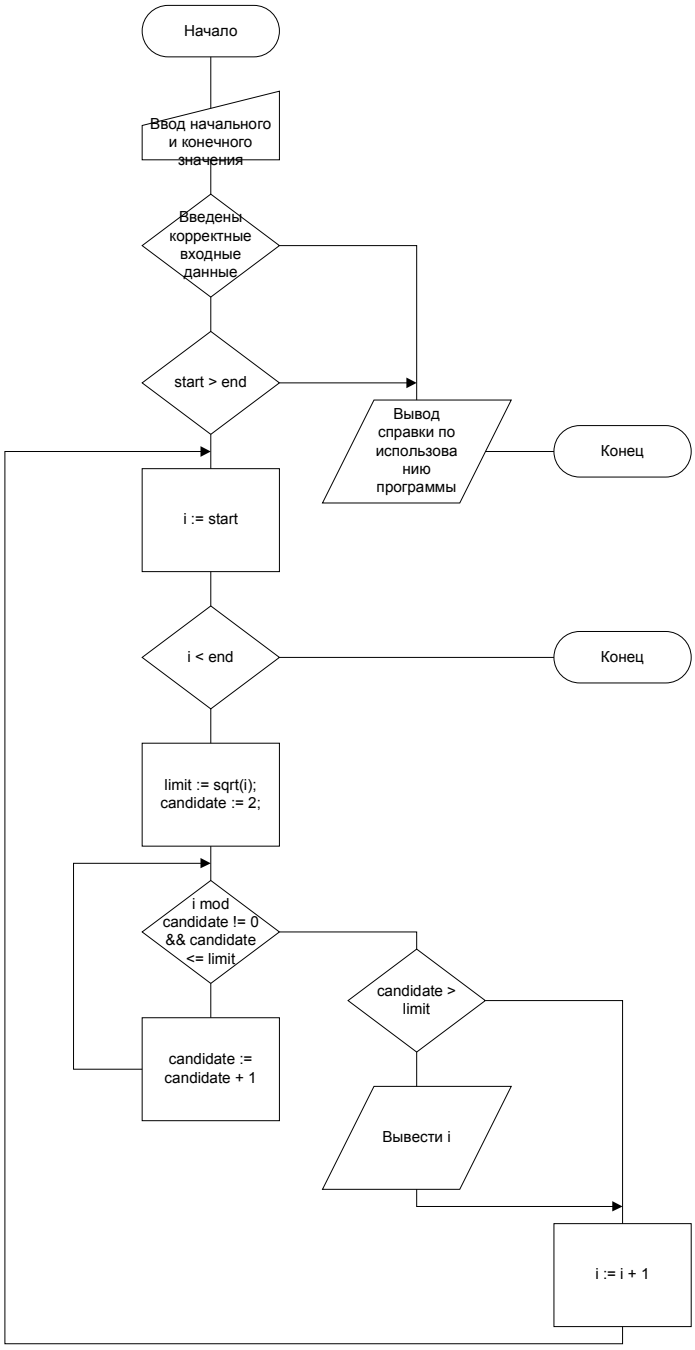
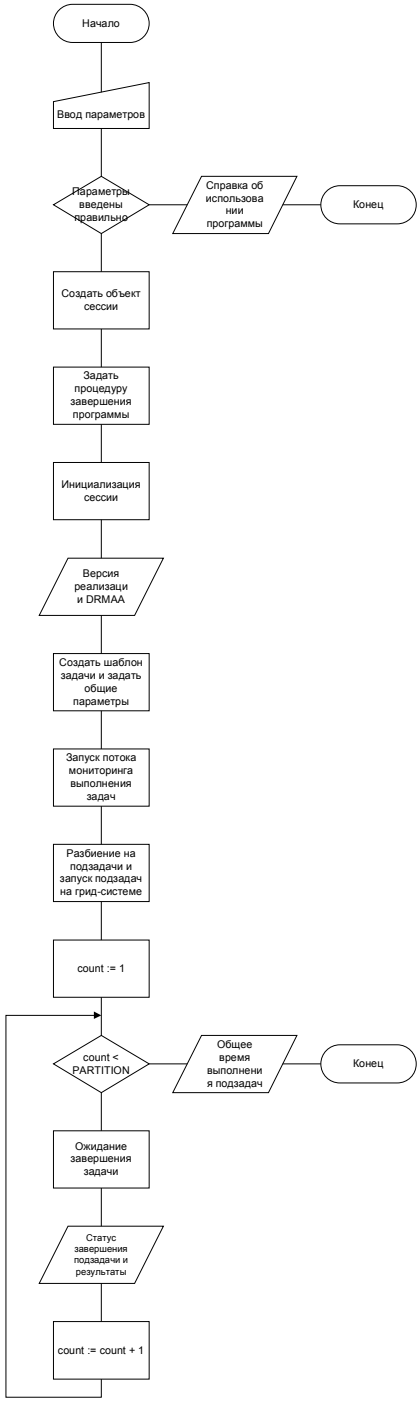



Рис. 9: Алгоритм вычисления простых чисел



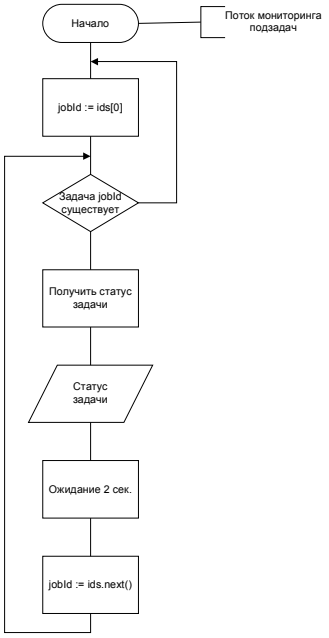




Рис. 10: Алгоритм основного приложения

# 6 Разработка программы

Для реализации грид-приложения был выбран язык программирования Java, т. к. Java — это переносимая платформа, что особенно важно в условиях гетерогенности узлов грида, язык Java поддерживается компанией Sun, производителем используемой в работе DRMS системы и для Java существует полная и описанная в спецификации реализация программного интерфейса DRMAA.

Приложение состоит из двух модулей: программа, реализующая алгоритм выбранной предметной области, и программа, посылающая задачи на грид и собирающая результаты.

Листинг 1: Реализация алгоритма поиска простых чисел в заданном промежутке

```
package ru.pp.kolia.primer;

public class Primer {

   public static void main (String[] args) {
      long start = 0L;
      long end = 0L;

      if (args.length != 2) {
         printUsage ();
         System.exit (1);
      }

      try {
         start = Long.parseLong (args[0]);
      }
      catch (NumberFormatException e) {
         System.err.println (Long.toString (start) + " is not a number");
         printUsage ();
         System.exit (1);
      }

      try {
         end = Long.parseLong (args[1]);
      }
      catch (NumberFormatException e) {
         System.err.println (Long.toString (end) + " is not a number");
         printUsage ();
         System.exit (1);
      }

      if (end < start) {
         System.err.println ("End of range may not be less than beginning of range.");
         printUsage ();
         System.exit (1);
```

```
      }

      calculate (start, end);
   }

   public static void calculate (long start, long end) {
      for (long i = start; i < end; i++) {
         long limit = (long)Math.sqrt (i);
         long candidate = 2;

         while (((i % candidate) != 0) && (candidate <= limit)) {
            candidate++;
         }

         if (candidate > limit) {
            System.out.println (Long.toString (i));
         }
      }
   }

   private static void printUsage () {
      System.out.println ("Usage: java –jar primer.jar <start> <end>");
   }
}
```

Программа, посылающая задачи на грид разбита на несколько классов: основная программа, процедура мониторинга выполняющихся задач и процедура, выполняющаяся при ненормальном завершении основной программы.

Листинг 2: Программа взаимодействия с DRMS

```
/*
 * Main.java
 *
 * Created onДекабрь 13 г 2006 ., 21:13
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package ru.pp.kolia.gridapp;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;
import org.ggf.drmaa.DrmaaException;
import org.ggf.drmaa.JobInfo;
import org.ggf.drmaa.JobTemplate;
```

```
import org.ggf.drmaa.Session;
import org.ggf.drmaa.SessionFactory;

/**
 *
 * @author kolia
 */
public class Main {
    private final static String SGE_ROOT = "/usr/local/n1ge";
    private final static String WORKING_DIR = "/default/gridapp";
    private final static String JOB_NAME = "primer";
    private final static int PARTITION = 2;

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Usage: java -jar gridapp.jar <start> <end>");
            System.exit(0);
        }

        SessionFactory factory = SessionFactory.getFactory();
        final Session session = factory.getSession();

        Runtime.getRuntime().addShutdownHook(new ShutdownHook(session));

        try {
            session.init("");
            System.out.println("Version: " + session.getDrmaaImplementation());

            JobTemplate jt = session.createJobTemplate();
            jt.setJobName(JOB_NAME);
            jt.setRemoteCommand(SGE_ROOT + "/gridapp/primer.sh");
            jt.setArgs(args);
            jt.setWorkingDirectory(SGE_ROOT + WORKING_DIR);
            jt.setJobCategory("primer");

            long start = Long.parseLong(args[0]);
            long end = Long.parseLong(args[1]);
            long range = (end - start) / PARTITION;

            List<String> jobIds = Collections.synchronizedList(new ArrayList(8));
            double cpuTime = 0.0;

            Thread monitor = new JobStatusMonitor(session, jobIds);
            monitor.setDaemon(true);
            monitor.start();

            int count = 0;
            for (count = 0; count < PARTITION-1; count++) {
                jt.setArgs(new String[] {
```

```
                Long.toString(start + count*range),
                Long.toString(start + (count+1)*range - 1)
            });
            jobIds.add(session.runJob(jt));
        }

        jt.setArgs(new String[] {
            Long.toString(start + count*range),
            Long.toString(end)
        });
        jobIds.add(session.runJob(jt));

        session.deleteJobTemplate(jt);

        for (count = 0; count < PARTITION; count++) {
            String jobId = jobIds.get(count);
            JobInfo info = session.wait(jobId, Session.TIMEOUT_WAIT_FOREVER);

            if (!info.hasExited()) {
                System.err.println("Job " + jobId + " terminated abnormally");
                System.exit(1);
            }
            printExitStatus(info);

            Map usage = info.getResourceUsage();
            cpuTime += Double.parseDouble((String)usage.get("cpu"));
            printJobOutput(jobId);
        }

        System.out.println("CPU time: " + cpuTime);

    } catch (DrmaaException e) {
        e.printStackTrace();
        System.exit(1);
    }
}

private static void printExitStatus(JobInfo info) {
    if (info.hasExited()) {
        System.out.println("Job exited with code " + info.getExitStatus());
    } else if (info.hasSignaled()) {
        System.out.println("Job exited on signal " + info.getTerminatingSignal());

        if (info.hasCoreDump()) {
            System.out.println("Core dumped");
        }
    } else if (info.wasAborted()) {
        System.out.println("Job never ran");
    } else {
        System.out.println("Exit status is unknown");
    }
}
```

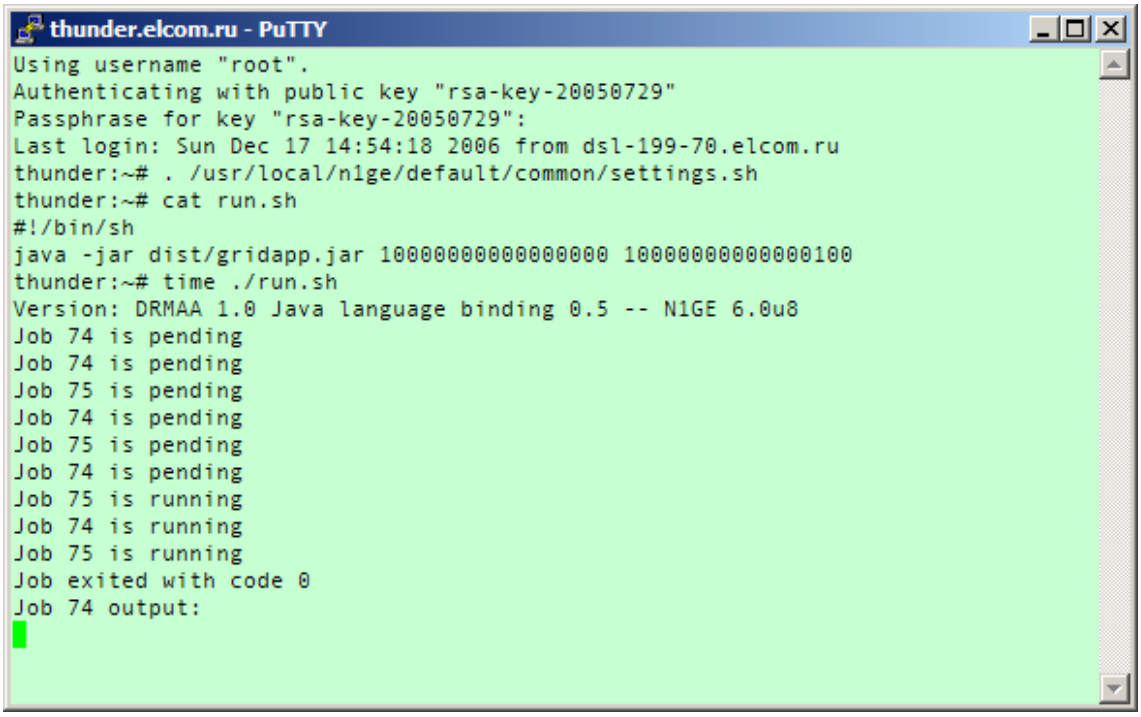



Рис. 11: Программа в процессе выполнения

```
    private static void printJobOutput(String jobId) {
        try {
            BufferedReader reader = new BufferedReader(new FileReader(SGE_ROOT + WORKING_DIR + "/" +
                JOB_NAME + ".o" + jobId));
            System.out.println("Job " + jobId + " output:");
            while (reader.ready()) {
                System.out.println(reader.readLine());
            }
        } catch (FileNotFoundException e) {
            System.out.println("Job output file not found");
            System.exit(1);
        } catch (IOException e) {
            System.out.println("Error reading job output file");
            System.exit(1);
        }
    }

}
```

## 6.1 Испытание разработанной программы

Снимок экрана в процессе выполнения программы показан на рис. 11.

Листинг 3: Полный вывод программы

```
Version: DRMAA 1.0 Java language binding 0.5 -- N1GE 6.0u8
Job 74 is pending
Job 74 is pending
Job 75 is pending
```

```
Job 74 is pending
Job 75 is pending
Job 74 is pending
Job 75 is running
Job 74 is running
Job 75 is running
Job exited with code 0
Job 74 output:
Job 75 is running
Job 75 is running
Job 75 is running
Job 75 is running
Job 75 is running
Job 75 is running
Job 75 is running
Job 75 is running
Job exited with code 0
Job 75 output:
10000000000000061
10000000000000069
10000000000000079
10000000000000099
CPU time: 35.04

real    0m55.838s
user    0m1.362s
sys     0m0.269s
```

# 7 Заключение

В данной работе было дано общее понятие о грид-вычислениях, рассмотрены основные принципы, применяющиеся при построения гридов, рассмотрен процесс установки системы DRMS Sun N1 Grid Engine, применяющейся при создании грид-систем, а также создано приложение, использующее ресурсы грид-системы для выполнения прикладной задачи.

Гриды — это относительно новая и стремительно развивающаяся технология. Сейчас она активно применяется в научной среде. Бизнес тоже присматривается к возможностям, которые она предоставляет. В развитии грид-технологий проявляют заинтересованность крупнейшие производители программного обеспечения (Microsoft, Sun, IBM, SAP и др.). Ими разрабатываются первые пробные внедрения грид-технологий в существующие программные продукты. Тесная связь грид-технологий с сервисно-ориентированными архитектурами является еще одним фактором, способствующим широкому интересу к ним. Сложно сказать, станут ли когда-нибудь грид-технологии такими же распространенными в повседневной жизни, как интернет и WWW, но безусловно они займут свое место в Enterprise-приложениях, ориентированных на поддержку бизнеса крупных предприятий.

## Список литературы


[1] docs.sun.com: N1 Grid Engine 6 Collection (http://docs.sun.com/app/docs/coll/1017.3)

[2] Sun Microsystems (http://www.sun.com)

[3] DRMAA-WG Documents (http://forge.ggf.org/sf/docman/do/listDocuments/projects.drmaa-wg/docman.root)