Российская академия наук
Институт прикладной математики имени М. В. Келдыша

На правах рукописи

Березовский Павел Сергеевич

УПРАВЛЕНИЕ ЗАДАНИЯМИ В ГРИДЕ С НЕКЛАСТЕРИЗОВАННЫМИ РЕСУРСАМИ

Специальность 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

ДИССЕРТАЦИЯ

на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
кандидат физико-математических наук,
заведующий сектором ИПМ им. М.В. Келдыша РАН
Коваленко В.Н.

Москва — 2011

Содержание

Введение

Грид с некластеризованными ресурсами

В настоящее время трудно представить себе получение научных результатов без использования вычислительной техники. В одних случаях для этого достаточно обычных компьютеров, в других же — необходимо произвести большое количество сложных расчётов, что требует большого количества вычислительных ресурсов. Для этих целей создаются вычислительные комплексы, которые могут быть географически распределены, либо функционировать в рамках одной организации.

В связи с этим приобрела популярность концепция распределённой вычислительной инфраструктуры под названием *грид*. В настоящей работе будет рассматриваться грид, состоящий из *неотчуждаемых некластеризованных* ресурсов (отдельных компьютеров, используемых их владельцами) и ориентированный на обработку последовательных заданий, которым для выполнения требуется один процессор, а также сериализуемых заданий, то есть набора последовательных заданий, которые решают одну задачу, но не взаимодействуют между собой в процессе выполнения.

Следует отметить, что наибольшее распространение получил метод построения грид-инфраструктур из кластерных узлов, в которые объединяется множество компьютеров, принадлежащих одному административному домену — *«двухуровневый»* способ организации [8]. При такой организации кластеры ресурсов находятся под управлением локального менеджера, который осуществляет непосредственное распределение заданий и их запуск на вычислительных установках. Двухуровневая организация ресурсов поддерживается наиболее

распространенным программным обеспечением грида, например, инструментально-базовой системой Globus Toolkit [9] и основанным на ней комплексом gLite [10]. Существенно, что способ управления ресурсами, реализованный в gLite и других комплексах, основанных на Globus Toolkit, фактически предполагает, что компьютеры должны полностью выделяться в грид и не могут использоваться владельцами — сотрудниками тех организаций, в которых они установлены. Такие ресурсы будем называть *отчуждаемыми*.

Помимо кластеров в глобальной сети имеются некластеризованные вычислительные ресурсы — рабочие станции пользователей, серверы и пр., обладающие большой суммарной производительностью, в то время как их средняя загрузка достаточно мала. Наличие большого количества таких компьютеров представляет интерес для пользователей грида, которые могли бы решать на них свои задачи, но далеко не всегда владельцы компьютеров имеют возможность организовывать и поддерживать кластерную инфраструктуру. Кроме того, использование этих компьютеров составляет проблему в силу двух обстоятельств: 1) владельцы выполняют на них собственные программы и 2) включают/выключают компьютеры в непредсказуемые моменты.

Сейчас в основном такие ресурсы используются для решения сверхкрупных по требуемому времени счёта задач [11], [12], когда владельцы ресурсов на короткий период времени объединяют имеющиеся мощности в рамках отдельных проектов.

Ограниченные возможности системного программного обеспечения, применяемого в проектах, не позволяют квалифицировать его как средство поддержки грида. Однако популярность этой формы распределённых вычислений демонстрирует, что способ организации грида на ресурсной базе из отдельных некластеризованных компьютеров представляет интерес с практической точки зрения. В работе [8] предложены основные принципы построения грид-инфраструктур из таких ресурсов и введён термин

«одноуровневый грид» (или грид с вертикальной интеграцией). В настоящей работе ставится вопрос об архитектуре и свойствах программного обеспечения одноуровневого грида.

Главное отличие двухуровневого и одноуровневого гридов заключается в режиме использования ресурсов, от которого существенно зависят способы управления инфраструктурой в целом и отдельными исполнительными компьютерами. В первом случае сформированная кластерная инфраструктура обычно отчуждается от владельца. В случае одноуровневого грида пространственно распределённые исполнительные компьютеры целиком принадлежат своим владельцам, а задания грида поступают на счёт только в том случае, если они не мешают выполнению локальных заданий. При этом владелец должен сохранять способность полностью контролировать свой компьютер, определяя условия предоставления его ресурсов для обработки задач грида. В предлагаемых решениях эффективное разделение ресурсов между владельцем и гридом рассматривается как основное условие применимости одноуровневого грида на практике. В то же время сама по себе одноуровневая организация не накладывает ограничений на режим использования компьютера и может применяться для отчуждаемого режима — в котором компьютер выделяется в грид целиком. Однако в любом случае предполагается, что компьютеры включаются в грид без кластеризации.

Актуальность темы

Одноуровневая архитектура имеет два преимущества. О первом речь шла выше: позволяя задействовать для задач грида холостые циклы процессоров, она предоставляет возможность создавать гриды на основе существующей ресурсной базы, в том числе из персональных компьютеров.

Второе преимущество заключается в её направленности на упрощение создания грид-инфраструктур. Для поддержки функционирования

одноуровневого грида необходим управляющий центр — диспетчер, который существует в единственном экземпляре на всё множество пространственно распределённых компьютеров, интегрируемых в грид. Диспетчер нужен и для двухуровневого грида, но, помимо этого, в каждом ресурсном узле должна быть установлена система управления кластером и службы доступа из грида. В одноуровневом гриде от владельцев компьютеров-ресурсов требуется лишь установка компактного и просто конфигурируемого программного обеспечения, после чего компьютер может использоваться для обработки задач грида. В обоих случаях при создании и поддержке функционирования грида необходим профессиональный подход, но в одноуровневой архитектуре не требуется наличия специального персонала, который бы занимался обслуживанием исполнительных ресурсов в связи с их включением в грид.

Рассматривая одноуровневый грид как широко доступное средство дистанционного использования вычислительных ресурсов, можно указать несколько сценариев его применения.

1. Создание распределённых инфраструктур высокой пропускной способности. Одноуровневый грид сохраняет возможность создания широкомасштабных грид-инфраструктур, которые в современной практике строятся на основе локальной кластеризации ресурсов. Необходимо, однако, отметить ограничения на класс задач, которые связаны с неотчуждаемостью ресурсов: становится невозможной обработка параллельных (многопроцессорных) приложений ввиду того, что процессы, находящиеся на разных компьютерах, будут выполняться дискретно — в периоды малой активности их владельцев — и несогласованно друг с другом. По той же причине полное время обработки обычного задания на исполнительном компьютере не совпадает с требуемым чистым процессорным временем. В связи с этим одна из наиболее важных проблем, решаемых программными средствами — обеспечение завершения задания в заданное время.

В таких условиях наиболее привлекательным выглядит применение масштабных одноуровневых гридов для выполнения серийных расчётов в виде набора независимых заданий, которые могут обрабатываться параллельно на разных ресурсах, не обмениваясь данными. В этом смысле их можно рассматривать как части одного слабо связанного параллельного задания.

2. Объединение ресурсов в рамках временных коллективов. Грид чаще всего ассоциируется с рекордными по необходимым вычислительным ресурсам задачами. Простота одноуровневой организации может сделать грид обыденным средством для решения класса задач, для которых требуется кратное (а не на порядки) увеличение мощности по сравнению с мощностью современных персональных компьютеров. Для этого необходимо, чтобы один центр управления гридом был способен поддерживать деятельность множества небольших коллективов, в которые объединяются лица, предоставляющие ресурсы, и лица, их использующие. Такие коллективы могут образовываться динамически, и их можно рассматривать как аналоги крупных виртуальных организаций современных гридов. Диспетчер одноуровневого грида должен обеспечивать автономность подобных «виртуальных организаций» в рамках объединённой ресурсной инфраструктуры многих таких организаций.

3. Персональный грид. Вычислительные средства, которыми располагает отдельный пользователь, как правило, ограничены единственным компьютером. Накопление ресурсного парка создаёт предпосылки для преодоления этого «барьера одного компьютера», но необходима адекватная техническая и технологическая поддержка вычислительной деятельности в более сложной среде, состоящей из нескольких компьютеров. В качестве средства, обеспечивающего такую поддержку, может выступать одноуровневый грид. Владелец нескольких компьютеров может подключить их к гриду и образовать персональную виртуальную организацию, ограничивая доступ к своим ресурсам всем,

кроме себя самого. В результате он получает общую точку доступа ко всей совокупности компьютеров — через управляющий центр грида, который обеспечивает эффективную балансировку их загрузки. Аналогичный эффект можно получить и от кластеризации компьютеров, но подход грида исключает проблемы обслуживания кластера.

4. Предоставление дистанционного доступа к приложениям. Понятие ресурса в гриде является очень широким и не ограничивается только системными ресурсами компьютеров (процессор, память, дисковое пространство). Ресурсом также может являться, например, устройство, подключенное к сети, а также любое приложение, которое по каким-либо причинам не может быть установлено у всех, кто хочет обрабатывать с его помощью свои данные. Путём подключения к гриду владелец компьютера, на котором установлено такое приложение, может предоставить к нему доступ и определить круг лиц, которые могут им пользоваться. Альтернативой может служить оформление приложения в виде грид-службы, но этот вариант более сложен.

Приведённые сценарии использования одноуровневой организации ресурсов свидетельствуют о наличии ситуаций, в которых применение такой модели даёт дополнительные возможности при решении вычислительных задач.

Цель и задачи работы

Целью диссертационной работы является разработка методов управления заданиями в гриде с некластеризованными ресурсами (компьютерами), в том числе такими, которые используются совместно с их владельцами, то есть не отчуждаются целиком в грид. Для достижения поставленной цели в работе решаются следующие основные задачи.

1. Исследование существующих способов диспетчеризации заданий в распределённой вычислительной среде, состоящей из некластеризованных компьютеров.

2. Разработка архитектуры системы управления инфраструктурой из некластеризованных компьютеров. Архитектура должна быть согласована с принципами и стандартами грида, а также содержать функции, необходимые для условий неотчуждаемых компьютеров.

3. Разработка метода планирования распределения заданий, направленного на минимизацию нарушений заданного срока выполнения заданий, и оценка качества планирования на основе разработанного метода.

4. Программная реализация системы диспетчеризации для рассматриваемой формы грида.

Научная новизна

В диссертации разработан новый метод управления заданиями, направленный на минимизацию нарушения завершения заданий в заданный срок и позволяющий повысить степень загрузки компьютеров для грида с некластеризованными неотчуждаемыми ресурсами. Предлагаемый подход к распределению заданий основывается на учёте статистической информации о загрузке компьютеров. Эта информация используется для предсказания остающейся свободной доли вычислительных ресурсов в будущем.

В рамках этого подхода разработан оригинальный алгоритм планирования, использующий при принятии решения информацию об эффективной производительности компьютеров — средней величине процессорной мощности компьютера, которая остаётся невостребованной при работе владельца и может быть использована для выполнения заданий грида. С помощью моделирования показано преимущество разработанного алгоритма по сравнению с обычно применяемыми и сформулированы

предложения по эффективному использованию пула неотчуждаемых компьютеров.

Предложена архитектура системы диспетчеризации заданий, выполненная в соответствии с требованиями, предъявляемыми к программному обеспечению одноуровневого грида. В частности, в предложенной архитектуре учитывается свойство неотчуждаемости ресурсов, для чего предусмотрены средства, обеспечивающие на исполнительном компьютере в первую очередь выполнение процессов владельца, автономность ресурсов компьютера, а также динамичность среды.

Разработаны механизмы управления заданием на некластеризованных компьютерах: запуска, получения информации о загрузке компьютера (ресурсы, потреблённые заданиями грида, и остающаяся свободной мощность компьютера), ограничения количества потребляемых системных ресурсов.

Практическая значимость

На основе архитектурных решений, концепций и методов, предложенных в диссертационной работе, была реализована система диспетчеризации заданий для грида с некластеризованными ресурсами. Разработанная система позволяет объединять в грид-инфраструктуры обычные компьютеры для решения практически важных прикладных научных и производственных задач, используя эти компьютеры в режиме разделения с их владельцами.

Практическая значимость работы подтверждается использованием разработанной системы для задач проектирования физических устройств, а также проведения вычислительных экспериментов при разработке программного обеспечения для моделирования физических процессов.

Апробация работы

Основные результаты работы докладывались и обсуждались на следующих конференциях и семинарах:

1. 1-я международная конференция «Распределённые вычисления и грид-технологии в науке и образовании». Доклад «Политика предоставления ресурсов в грид», Дубна, 29 июня–2 июля 2004 г.
2. Семинар МГУ им. М.В. Ломоносова «Проблемы современных информационно-вычислительных систем» под руководством д.ф.-м.н. В.А. Васенина. Доклад «Способы планирования в гриде и их реализация в грид-диспетчере», Москва, 12 апреля 2005 г.
3. Семинар группы разработчиков программного обеспечения для грид-инфраструктуры EGEE ARDA под руководством M. Lamanna. Доклад «KIAM in GT4 Evaluation Activity and Grid Research», CERN, Женева, 12 октября 2005 г.
4. 13-я Всероссийская научно-методическая конференция «Телематика-2006». Доклад «Создание прототипа центра базовых грид-сервисов нового поколения для интенсивных операций с распределёнными данными в федеральном масштабе», Санкт-Петербург, 5–8 июня 2006 г.
5. 2-я международная конференция «Распределённые вычисления и грид-технологии в науке и образовании». Доклад «Способ построения грид из некластеризованных ресурсов», Дубна, 26–30 июня 2006 г.
6. Научная конференция «Ломоносовские чтения», факультет ВМиК МГУ им. М.В. Ломоносова. Доклад «Способ построения грида из некластеризованных ресурсов», Москва, 16–24 апреля 2008 г.
7. 3-я международная конференция «Распределённые вычисления и грид-технологии в науке и образовании». Доклад «Механизмы управления разделяемыми компьютерами в гриде», Дубна, 29 июня–4 июля 2008 г.

8. 4-я международная конференция «Распределённые вычисления и грид-технологии в науке и образовании». Доклад «Применение грида с некластеризованными ресурсами для задач проектирования физических устройств», Дубна, 28 июня–3 июля 2010 г.

По материалам диссертации опубликовано 7 печатных работ [1], [2], [3], [4], [5], [6] и [7], в том числе, одна [4] — в журнале, рекомендованном ВАК для публикации основных результатов докторских и кандидатских диссертаций по вычислительной технике и информатике.

Структура и объём работы

Работа состоит из введения, пяти глав, заключения, списка литературы и двух приложений. Общий объём диссертации — 128 страниц. Список литературы содержит 60 наименований. В работе содержится 20 рисунков и одна таблица.

Краткое содержание работы

Первая глава является обзорной и содержит анализ существующих решений для организации вычислительных инфраструктур из некластеризованных компьютеров. При рассмотрении наиболее развитых средств выделены четыре подхода: первый основан на создании проектов, к которым подключаются исполнительные компьютеры; второй подход состоит в применении P2P-технологий и объединении исполнительных компьютеров в одноранговые сети; третий подход характеризуется системами с централизованным управлением; четвёртый подход представлен частными разработками корпоративных систем. В заключительной части главы формулируются требования, которым должно удовлетворять программное обеспечение одноуровневого грида.

Во второй главе предлагается архитектура системы диспетчеризации заданий для грида с некластеризованными ресурсами. Состав, структура и функции компонентов системы отвечают сформулированным в первой главе требованиям. Архитектура состоит из трёх компонентов: диспетчера, агента и пользовательского интерфейса. Диспетчер реализуется набором веб-служб и устанавливается на выделенный компьютер в сегменте грида. Ресурсы исполнительных компьютеров грид-инфраструктуры не доступны пользователям непосредственно, а все внешние интерфейсы доступа к ним сосредоточены в диспетчере. Основная функция агента, устанавливаемого на исполнительном компьютере, — это управление заданием на стадии выполнения. Интерфейс пользователя предоставляет ему возможность управлять своими заданиями стандартным для грида способом, а также получать информацию о состоянии задания.

В третьей главе рассматриваются способы планирования заданий в гриде с некластеризованными ресурсами. Глава состоит из трёх частей. В первой части даётся классификация условий планирования в зависимости от имеющейся у планировщика информации, на основе которой он принимает решение о направлении задания на тот или иной исполнительный компьютер. Во второй части сравниваются алгоритмы планирования по различным условиям, и делается вывод о применимости их для решения поставленной задачи. В третьей части главы приводится описание предложенного автором метода планирования, направленного на минимизацию нарушения выполнения заданий в заданный срок. Метод основан на использовании статистической информации о загрузке ресурсов компьютера.

Четвёртая глава посвящена программной реализации системы диспетчеризации заданий для грида с некластеризованными ресурсами, выполненной в соответствии с предложенной во второй главе архитектурой. Глава содержит подробное описание способа реализации функций системы, определённых во второй главе. Особое внимание уделено вопросу интеграции разрабатываемых компонентов со сторонним программным

обеспечением, используемым в системе на базовом уровне. Рассмотрены различные сценарии работы с системой через пользовательский интерфейс.

В пятой главе содержится описание прикладных задач, решённых с помощью разработанной системы в Институте прикладной математики им. М.В. Келдыша РАН. Первая задача состоит в измерении пространственного распределения энергии ионизирующего излучения в многокомпонентных объектах. Использование вычислительной инфраструктуры из некластеризованных компьютеров позволило в несколько раз сократить время расчёта задачи. Вторая задача заключается в расчёте модели движения лайнера в магнитном компрессоре. При осуществлении вычислительного эксперимента в серии расчётов были получены результаты, позволившие уточнить численную модель и оптимизировать значения параметров лайнера. Примеры применений позволяют сделать вывод о наличии широкого класса задач вычислительного эксперимента и технических расчётов, на проведение которых затрачивается существенно меньшее время благодаря использованию грид-инфраструктур вычислительного типа.

В заключении перечисляются основные результаты работы.

Глава 1

Существующие решения для управления некластеризованными компьютерами

На сегодняшний день практика объединения компьютеров, находящихся в персональном владении, достаточно распространена. Задача объединения решается как в географически распределённой среде, так и в рамках одного предприятия.

При организации подобных инфраструктур можно выделить четыре подхода: первый основан на создании проектов, к которым подключаются исполнительные компьютеры; второй подход состоит в применении P2P-технологий (Peer-to-Peer) [13] и объединении исполнительных компьютеров в одноранговые сети; третий подход характеризуется системами с централизованным управлением; четвёртый подход представлен частными разработками корпоративных систем.

Следует отметить, что это разделение является условным и призвано показать многообразие используемых технологий, а также широту применения подобных систем. Так, рассматриваемые далее системы могут пересекаться по различным аспектам реализации или архитектурным решениям.

1.1. Проекты @Home и платформа BOINC

Одним из первых примеров применения одноуровневой схемы организации пространственно распредёленных компьютеров стал проект SETI@Home Калифорнийского университета, направленный на решение

задачи поиска внеземного разума. Участникам проекта предлагается скачать с веб-сайта университета небольшую программу, работающую вместо хранителя экрана в те периоды, когда компьютер не используется его владельцем. Эта программа получает данные радиотелескопических наблюдений с сервера, анализирует их и отправляет результат обратно. Радиотелескоп генерирует большое количество информации, но она может быть разбита на небольшие порции, которые могут быть обработаны независимо с помощью одного приложения. Приложение меняется довольно редко, поэтому закачивается на исполнительный компьютер однократно и обновляется по мере необходимости.

Позднее появилось множество проектов из серии @Home, предназначенных для решения других прикладных задач, обладающих похожими свойствами, однако SETI@Home по-прежнему является одним из самых популярных. По состоянию на июль 2007 года в нём принимали участие 658 тысяч пользователей, а суммарная мощность компьютеров вычислительной инфраструктуры составляла 262 Tflops. В результате разработка университета вылилась в создание программной платформы BOINC [14], упрощающей создание проектов для новых задач (рис. 1).

Распределённые вычисления в системе BOINC основаны на *проектах*. Один сервер BOINC может поддерживать несколько независимых проектов, каждый из которых создаётся для выполнения конкретного приложения и обладает собственной программной и аппаратной инфраструктурой.

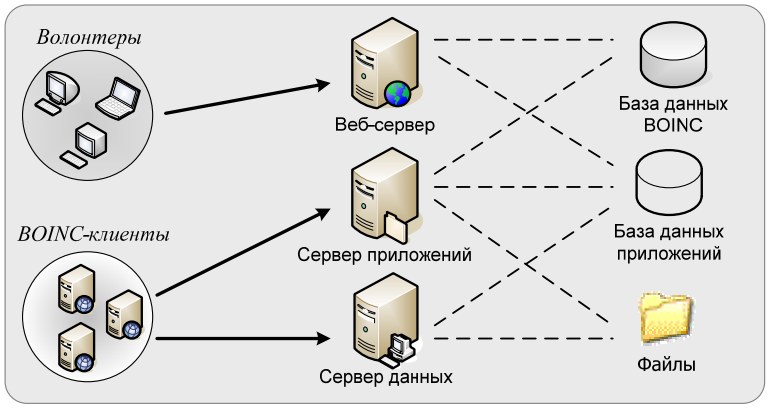


Рис. 1. Вычислительная инфраструктура под управлением системы BOINC

Предполагая, что имеется компьютер, удовлетворяющий необходимым минимальным системным требованиям, создаётся сервер BOINC. Процесс создания сервера и развёртывания проекта состоит в последовательном выполнении следующих шагов.

1. Установка необходимого программного обеспечения. BOINC опирается на общераспространённое базовое программное обеспечение, которое должно быть установлено и сконфигурировано перед началом работы. Так, для хранения различной информации (пользователи, исполнительные компьютеры, поддерживаемые платформы, блоки входных данных, приложения и пр.) используется СУБД MySQL [15], а доступ пользователей к информации проекта (регистрация, настройка учётной записи, мониторинг и пр.) осуществляется через веб-интерфейс. Дополнительно для настройки BOINC-сервера необходимо установить Python (для запуска конфигурационных скриптов), PHP (для генерации html-страниц веб-интерфейса) и libcurl (для передачи данных).

2. Установка программного обеспечения (ПО) BOINC-сервера и создание проекта. Для установки сервера выполняется процесс сборки из исходных кодов. Затем запускается специальный скрипт для создания проекта и определяется его имя. При этом для нового проекта создаётся структура директорий, а также база данных, в которой будет храниться вся информация, относящаяся к этому проекту.

3. Установка и настройка веб-сервера. BOINC использует веб-сервер Apache, который выполняет три основные функции. Во-первых, он предоставляет пользователям доступ к информации о проекте через веб-интерфейс. Во-вторых, в процессе функционирования через веб-сервер осуществляется взаимодействие BOINC-сервера с исполнительными компьютерами для сбора информации об их характеристиках и состоянии. И, в-третьих, веб-сервер обеспечивает возможность передачи данных для загрузки входных порций на исполнительные компьютеры, а также для отправки результатов вычислений обратно на сервер.

4. Конфигурирование проекта. Во время создания проекта в его директории формируется конфигурационный файл, содержащий типичные настройки, подходящие для большинства проектов, но значения некоторых параметров перед запуском проекта всё же необходимо отредактировать. В частности, необходимо указать имя пользователя и пароль для доступа к базе данных проекта. Далее необходимо осуществить настройку демона, который генерирует порции входных данных, и планировщика, распределяющего эти порции по исполнительным компьютерам. В настройках планировщика указывается, каким образом порции распределяются по компьютерам (на каждого пользователя по одной порции или на каждый компьютер по одной порции), частота отправки порций на компьютеры и т.д.

5. Создание и подготовка запускаемого приложения. Ключевым моментом подготовки проекта является создание приложения, которое необходимо написать в соответствии с требованиями системы BOINC. Это приложение будет запущено на всех исполнительных компьютерах. Для выполнения требований необходимо использовать соответствующие функции API BOINC. В частности, в начале и в конце выполнения программа должна вызывать функции инициализации (boinc_init) и завершения (boinc_finish). Во время выполнения приложения должны вызываться функции, сообщающие пользователю о прогрессе выполнения (в процентах) задания на его компьютере, функции по работе с графикой и пр. В том случае, когда приложение использует входные файлы или генерирует выходные файлы, их имена должны быть преобразованы с помощью специальной функции, входящей в состав API. Таким образом, программа работает с логическими именами файлов, которые преобразуются в физические только во время выполнения. Эта мера предостерегает от конфликта имён во время выполнения приложения, так как имена файлов при каждом запуске, как правило, одинаковые. Написанное по таким правилам приложение может быть скомпилировано для различных платформ (Windows, Mac, Linux и др.), и каждый экземпляр приложения будет

автоматически распределяться на исполнительный компьютер с соответствующей архитектурой.

Создавая свой собственный BOINC-сервер для управления инфраструктурой проекта, организация публикует информацию о нём, чтобы привлечь добровольцев для решения одной прикладной задачи. Эта информация, в частности, содержит описание проекта и адрес в сети, откуда можно скачать клиентское приложение, который необходимо указать в настройках клиентского приложения.

Для участия в проекте владелец компьютера регистрируется на сайте этого проекта и получает персональный ключ, по которому он будет идентифицироваться при последующих подключениях. После подключения сервер определяет архитектуру исполнительного компьютера и предоставляет соответствующий экземпляр приложения, а также порцию входных данных. Далее осуществляется доставка приложения и входных данных на компьютер-клиент, и приложение начинает выполняться в те моменты, когда владелец снижает свою активность. После завершения обработки полученной порции данных клиент отправляет результат серверу и получает очередную порцию. Как только все порции данных будут обработаны, полученные от клиентов частичные результаты объединяются на сервере, и выполнение проекта прекращается.

1.2. P2P-подход. Системы Cluster Computing On the Fly и OurGrid

1.2.1. Cluster Computing On the Fly

Примером системы, позволяющей использовать вычислительные ресурсы отдельных компьютеров, является разработка Орегонского университета Cluster Computing On the Fly (CCOF) [16]. Архитектура системы соответствует подходу P2P и построена по аналогии с файлообменными сетями, только вместо файлов в качестве ресурса

выступает исполнительный компьютер. В состав компонентов программной архитектуры этой системы входят планировщик пользователя, работающий на его компьютере, и планировщик исполнительного компьютера. Система решает задачу управления распределённой инфраструктурой, которую образуют исполнительные компьютеры, путём координации планирования их планировщиками.

Как и в любой P2P-сети, здесь все компьютеры равноправны, нет выделенного центрального сервера и клиентов. Каждый компьютер является связующим звеном между своими соседями (рис. 2) и хранит у себя информацию о ресурсах соседей. Даже когда владелец компьютера начинает интенсивно использовать его ресурсы, и никакие внешние задания не поступают, компьютер продолжает принимать и пересылать сообщения, которые проходят через него. Компьютер предоставляет всем остальным участникам шаблон, описывающий занятость ресурсов в определённые промежутки времени.

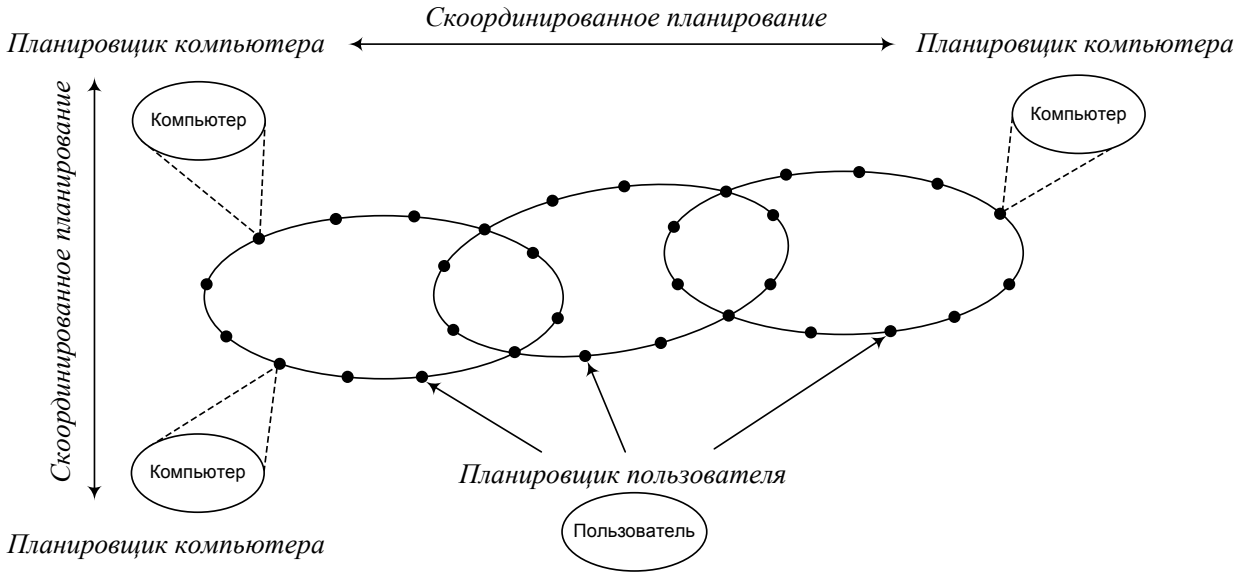


Рис. 2. Архитектура системы CCOF

Поиск свободных ресурсов осуществляется на основе представленных шаблонов. Имеется несколько алгоритмов поиска исполнительных компьютеров, суть которых состоит в том, что, получив от пользователя

ресурсный запрос, планировщик компьютера сравнивает его со своим шаблоном и определяет, может ли он запустить это задание. Если запрос удовлетворяет шаблону, и компьютер свободен, пользователь передаёт ему задание.

Во время выполнения задания планировщик пользователя периодически получает сигналы от исполнительного компьютера о том, что компьютер «жив», и выполнение задания продолжается. В том случае, если на исполнительном компьютере произошёл сбой, сигналы прекращаются, а планировщик пользователя перезапускает своё задание на другом исполнительном компьютере.

По сути, распределение заданий осуществляется планировщиком пользователя, который расположен на его компьютере. Его запросы принимаются планировщиком исполнительного компьютера — кандидата для приёма задания. На основе локальных политик по некоторым критериям (производительность, ранг, степень доверия и т.д.) планировщик исполнительного компьютера выбирает те задачи из предлагаемых пользовательским планировщиком, которые он готов запустить у себя. Таким образом, система CCOF реализует децентрализованную схему планирования, которая обладает всеми преимуществами и недостатками, свойственными этому подходу.

1.2.2. Система OurGrid

В системе OurGrid [17] сделана попытка разработать универсальный подход, пригодный для создания распределённых инфраструктур, построенных как из кластеров, так и из отдельных компьютеров. Кроме того, OurGrid сочетает централизованное распределение ресурсов с децентрализованным на основе P2P-сетей.

Ресурсная инфраструктура поделена на части, каждая из которых называется сайтом и состоит из компьютеров, относящихся к одному административному домену. Эти компьютеры могут быть как

персональными, так и входить в состав кластера, находящегося под управлением таких менеджеров ресурсов, как PBS и LSF.

В системе реализовано три типа компонентов: диспетчер сайта, компонент исполнительной машины и пользовательский интерфейс (рис. 3). Для управления сайтом выделяется отдельный компьютер, на котором устанавливается программный компонент OurGrid peer — диспетчер сайта. Компонент SWAN (Sandboxing Without A Name) располагается на всех исполнительных компьютерах каждого сайта, которые занимаются непосредственной обработкой заданий. Пользователи взаимодействует со всей системой через персонального брокера MyGrid, которого они устанавливают на свои машины.

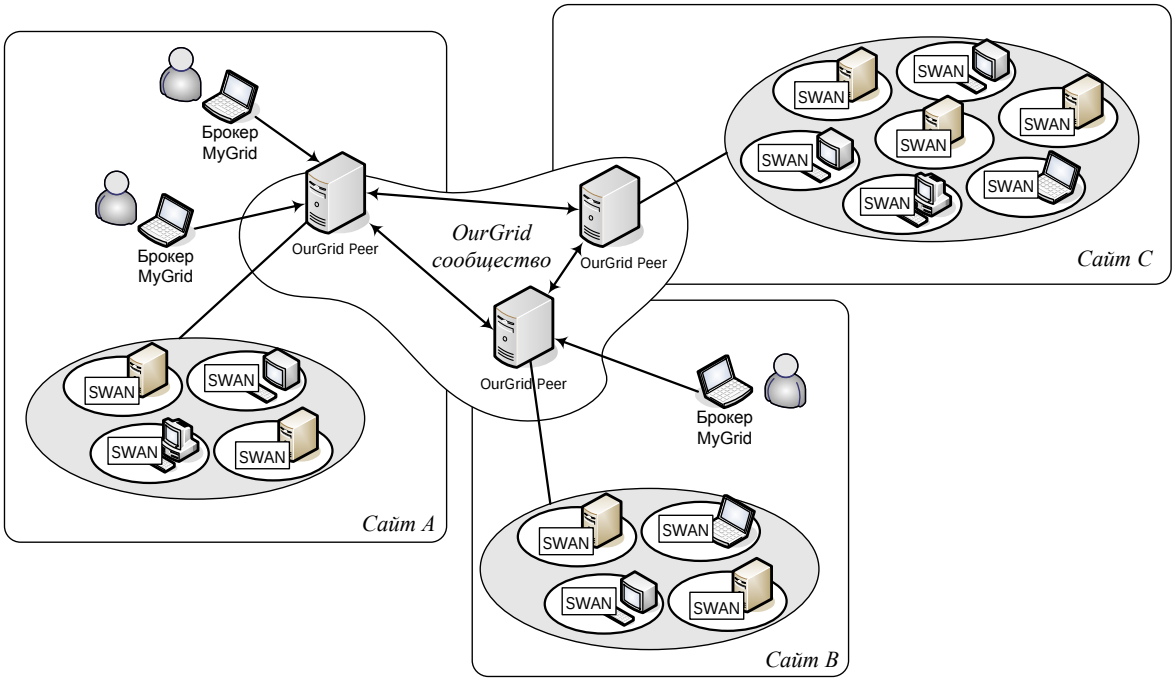


Рис. 3. Архитектура системы OurGrid

Брокер MyGrid осуществляет распределение заданий по ресурсам и предоставляет набор абстракций, скрывающих от пользователя гетерогенность вычислительной среды. Пользовательское задание передаётся брокеру, который запрашивает необходимые ресурсы у всех сайтов, на которых пользователь имеет учётную запись (account). Каждый сайт

находится под управлением своего диспетчера (OurGrid peer), который по запросу брокера сообщает, какие ресурсы доступны пользователю. Эти ресурсы могут быть локальными (входить в состав сайта, который контролируется самим диспетчером) или внешними, то есть находящимися под управлением диспетчеров других сайтов. Компьютеры, на которых установлен компонент OurGrid peer, образуют P2P-сообщество (или одноранговую сеть). Информацию о внешних ресурсах диспетчер получает, взаимодействуя с другими диспетчерами сайтов в P2P-сети способом, аналогичным применяемому в системе CCOF.

Реализованный в брокере способ планирования ориентирован на сериализуемые задания. Получив информацию о ресурсах, брокер приступает к распределению частей сериализуемых заданий, используя в качестве целевой функции планирования критерий минимизации времени выполнения задания в целом. Для того, чтобы гарантировать завершение задания, брокер использует механизм миграции, учитывая возможность отключения исполнительных компьютеров.

Служба безопасности SWAN реализована с помощью монитора виртуальных машин Xen [18], который изолирует код пользовательского приложения внутри оболочки (sandbox). Выполняясь внутри этой оболочки, задание не только не имеет доступа к данным реального компьютера, но также не может использовать сеть. Таким образом, в системе поддерживается обработка заданий, не требующих обмена данными во время выполнения.

Хотя изначально система OurGrid разрабатывалась для построения гридов, состоящих из кластеров, с её помощью можно организовать вычисления в одноуровневом гриде. Для этого на каждый исполнительный компьютер необходимо помимо компонента SWAN установить диспетчер OurGrid peer.

1.3. Системы с централизованным управлением

1.3.1. Система X-Com

Система метакомпьютинга X-Com [19], [20] предназначена для организации распределённых вычислительных сред, в которых могли бы решаться различные прикладные задачи. Разработчиками системы предоставляется инструментарий, с помощью которого пользователь может запускать свою программу распределённо на некотором количестве вычислительных узлов. Система позволяет контролировать ход выполнения программы, балансировать нагрузку между вычислительными узлами и визуализировать ход выполнения. При подготовке программы, которая может быть параллельной, в системе X-Com пользователю требуется определённым образом переработать её, в частности, выделить в программе серверную и клиентскую части. Ключевыми свойствами системы X-Com являются, во-первых, способность её работы в глобальной сети, что позволяет потенциально подключать большое количество компьютеров, а во-вторых, возможность использования гетерогенных ресурсов — от кластеров до обычных рабочих станций, работающих под управлением различных программно-аппаратных платформ.

1.3.1.1. Архитектура и функции системы X-Com

Система X-Com имеет трёхуровневую архитектуру, состоящую из пользовательского и серверного уровней, а также уровня исполнительных компьютеров.

На пользовательском уровне находятся процессы, обеспечивающие взаимодействие пользователя с системой. Эти процессы позволяют пользователю поставить задание в очередь на выполнение и отслеживать его состояние.

Уровень исполнительных компьютеров представлен компонентом *исполнитель X-Com*, который получает очередную порцию входных данных

от сервера соответствующего задания, запускает вычислительную часть приложения пользователя и возвращает результаты счёта на сервер.

Процессами серверного уровня являются «сервер задания», «управляющий сервер» и «менеджер исполнителей». Эта группа процессов отвечает за поддержку выполнения заданий пользователя.

- *Сервер задания* обеспечивает выполнение одного конкретного приложения пользователя — он взаимодействует с серверной частью приложения, генерирующей порции входных данных и осуществляющей финальную обработку результатов, реализует логику выдачи порций исполнителям, а также взаимодействует с исполнителями через вспомогательные службы (файловую службу, а также службы запроса, результата и статистики).
- *Управляющий сервер* обеспечивает управление множеством заданий — реализует логику очередей заданий. Для каждого задания управляющий сервер порождает свой сервер задания. Также этот процесс передаёт команды пользователя серверу задания, которые затем направляются соответствующим клиентам.
- При первом запуске каждого исполнителя *менеджер исполнителей* принимает их запросы и перенаправляет запросы исполнительных компьютеров на те серверы заданий, требования которых соответствуют характеристикам компьютера.

Взаимодействие между компьютерами и сервером в системе X-Com происходит через сеть Интернет, при этом используется только стандартный протокол HTTP, что позволяет подключать к системе практически любые вычислительные мощности, имеющие доступ в Интернет. Система не требует настройки для работы через прокси-сервер, firewall и другие системы защиты, а данные, передаваемые системой, аналогичны обычному трафику Интернет.

1.3.1.2. Масштабируемость X-Com

Для организации эффективного взаимодействия и подключения большего количества компьютеров вычислительная инфраструктура под управлением системы X-Com может быть организована в виде произвольного иерархического дерева. Листьями такого дерева всегда будут исполнительные компьютеры, а в корне дерева останется центральный сервер X-Com. Промежуточные серверы с точки зрения центрального сервера выглядят как обычные компьютеры, а с точки зрения нижележащих исполнительных компьютеров — как центральный сервер.

1.3.1.3. Средства запуска заданий и планирование

Система X-Com во многом схожа с платформой BOINC, однако имеются отличия, устраняющие некоторые ограничения этой платформы. В частности, в системе X-Com на одной программно-аппаратной инфраструктуре может выполняться множество приложений, в то время как при использовании BOINC требуется для каждого нового приложения (оформляемого в виде проекта) создавать свою собственную инфраструктуру. Тем не менее во время запуска очередного задания управляющий сервер порождает для него отдельный сервер задания, для которого должен быть составлен файл настройки.

Каждая прикладная программа должна быть особым образом адаптирована и представлена в виде двух частей — серверной и клиентской. Серверная часть прикладной программы осуществляет подготовку порций входных данных, которые будут обрабатываться клиентской частью программы на вычислительных узлах. Допускается использование заранее подготовленных порций, представленных в виде отдельных файлов. В этом случае не требуется написание серверной части прикладной программы, однако необходимо соответствующим образом описать порции входных данных в конфигурационном файле и позаботиться об их доставке на сервер.

Запуск заданий осуществляет подсистема управления заданиями, принимающая задания пользователей, из которых она формирует очередь и последовательно передаёт на выполнение на исполнительные компьютеры одно или несколько заданий, а также позволяет отслеживать состояние запущенных заданий.

Если задание может выполняться на компьютерах с различной программно-аппаратной платформой, для каждой из них на сервере должен существовать соответствующий файл с клиентской частью приложения.

Для распределения заданий по компьютерам в системе X-Com реализован метод, суть которого состоит в том, что на компьютер, запрашивающий задание, попадает случайное (например, первое по порядку в очереди) подходящее по аппаратным требованиям задание. Время выполнения задания на компьютере, а также предельный срок выполнения не учитываются.

При подготовке задания пользователь формирует файл описания задания, в котором указываются следующие данные:

- имя задания;
- идентификатор;
- параметры командной строки для запуска клиентской части приложения;
- требования к ресурсам, на которых задание может выполняться;
- адреса серверов, откуда следует скачивать необходимые для задания файлы.

Клиентская часть X-Com собирает и отправляет серверу данные о ресурсах вычислительного узла. Эти данные включают в себя следующую информацию:

- тип операционной системы;
- аппаратную архитектуру процессора;
- производительность процессора;

- объём оперативной памяти.

Чтобы определить подходит ли узел для выполнения задания, в системе X-Com осуществляется сравнение описания задания с описанием узла.

1.3.1.4. Доставка файлов

В системе X-Com необходимые для выполнения задания данные делятся на два типа: непосредственно файлы задания (исполняемые файлы и файлы данных) и порции входных данных для счёта.

Файлы первого типа могут располагаться на любом доступном системе http-сервере, откуда клиентская часть X-Com скачает их на вычислительный узел перед началом выполнения задания. Файлы с порциями данных для счёта должны находиться на сервере до запуска задания на выполнение. Они могут быть доставлены туда либо вручную заблаговременно и специальным образом описаны, либо сгенерированы по определённым правилам серверной частью приложения.

Для дальнейшей обработки задания используется общепринятый механизм работы с сериализуемыми приложениями. После того как все необходимые файлы будут загружены на вычислительный узел, клиент обращается за порцией данных для счёта и запускает процесс обработки порции. Как только порция данных будет обработана, клиент отсылает результат на сервер и получает очередную порцию. Этот процесс продолжается до тех пор, пока все порции не будут обработаны.

1.3.1.5. Безопасность

В системе X-Com предлагаются средства для обеспечения защиты от следующих типов угроз:

- модификация сетевого трафика, приводящая к нарушению конфиденциальности и достоверности передаваемых данных;

- подмена одной из сторон передачи данных, позволяющая злоумышленнику использовать возможности системы для атаки узлов инфраструктуры;
- внедрение ложного клиента, позволяющее использовать модифицированный клиент X-Com с возможностью получения доступа к порциям данных, предназначенных для обработки и отправки недостоверных результатов.

Для решения описанных выше проблем используются механизмы шифрования трафика, а также идентификация серверов и клиентов на основе их сертификатов в соответствии со стандартом X.509 [21].

1.3.2. Система XtremWeb

1.3.2.1. Архитектура системы XtremWeb

Система XtremWeb позволяет построить вычислительную инфраструктуру, в которой каждый компьютер в зависимости от установленного на него компонента системы может выполнять одну из трёх ролей — координатора, исполнителя или клиента. В системе может быть один координатор и произвольное количество исполнителей и клиентов.

Координатор. В системе XtremWeb координатор отвечает за управление заданиями — распределяет задания и осуществляет хранение результатов вычислений, а также выполняет административные функции. Учитывая то, что система ориентирована на использование неотчуждаемых ресурсов с непредсказуемыми моментами отключения от инфраструктуры, взаимодействие между исполнителями и координатором основано на pull-модели. При использовании такой модели инициация действия исходит от исполнителя, который получает необходимую информацию в ответ на свои запросы. При этом решается проблема взаимодействия с компьютерами, находящимися внутри локальных сетей или защищёнными сетевыми экранами (firewall), так как обычно они настроены на ограничение входящего трафика, но не ограничивают исходящий. Единственной машиной в

инфраструктуре XtremWeb, на которой необходимо открыть порт входящим соединениям, является координатор. Однако в данном случае используется порт веб-приложений (80), который обычно открыт.

В системе XtremWeb поддерживается обработка как скомпилированных под определённую архитектуру приложений, так и java-приложений. Планирование реализовано по принципу очереди (FIFO — first in, first out). При запросе одним из исполнителей следующего задания выбирается первое из очереди, если конфигурация компьютера удовлетворяет требованию задания к ресурсам. В качестве требований выступают тип процессора и версия операционной системы. Если пользователь хочет запустить java-приложение, то необходимо также указать требование по наличию на исполнительном компьютере Java.

После запуска каждого задания координатор ожидает от исполнителя результата выполнения. В случае ошибки задание будет перепланировано и заново запущено на другом компьютере.

Клиенты. Клиент предоставляет возможность пользователям системы запускать задания и получать результаты выполнения. С помощью этого компонента пользователь предоставляет координатору все необходимые для выполнения задания файлы и указывает параметры для запуска.

Исполнители. Исполнитель — это компонент системы, который устанавливается на исполнительные компьютеры, осуществляющие расчёт пользовательских заданий. Если исполнитель свободен, он запрашивает для обработки очередное задание и загружает всё необходимое для его выполнения (исполнительные файлы, файлы с данными, аргументы запуска). После того как файлы будут загружены, осуществляется запуск задания. Задание выполняется до тех пор, пока оно не завершится, либо не прекратится из-за сбоя (в том числе локальной политики использования ресурсов компьютера).

Во время выполнения задания исполнитель периодически посылает координатору сигналы о том, что процесс обработки продолжается в

штатном режиме. Если в какие-то моменты времени невозможно установить связь с координатором, то процесс выполнения задания не останавливается, а соответствующее сообщение будет отправлено, когда это станет возможным. В случае возникновения ошибок (выключение компьютера, несоответствие политике использования ресурсов или сетевые проблемы) прерванное задание перезапускается с начала, так как механизм контрольных точек в системе XtremWeb не предусмотрен. Если задание по какой-либо причине было перепланировано, то исполнитель получит соответствующий сигнал и остановит процесс выполнения.

После завершения выполнения задания исполнитель отправляет координатору результаты вычислений и запрашивает новое задание.

1.3.2.2. Масштабируемость системы XtremWeb

По данным разработчиков система способна управлять вычислительной инфраструктурой из сотни тысяч компьютеров. Возможности такого масштабирования могут достигаться применением следующих решений:

1. Пропускная способность сервера может быть увеличена за счёт использования пула серверов и мета-сервера. При такой организации каждый из серверов пула работает в режиме планировщика, а мета-сервер осуществляет диспетчеризацию между ними и доставку заданий. В случае выхода из строя одного из серверов мета-сервер перенаправляет запросы исполнителей на доступные серверы.
2. Может быть использовано стороннее программное обеспечение для балансировки загрузки отдельных компонентов системы.
3. Для сбора и обработки результатов вычислений могут быть использованы выделенные компьютеры, что позволит снизить загрузку серверов.

Однако принятие решения о реализации тех или иных способов оптимизации лежит на администраторах конкретной инфраструктуры под

управлением системы XtremWeb. В настоящее время готовый сценарий или конкретные рекомендации разработчиками не предлагаются.

Помимо использования пула серверов и мета-сервера ещё одним из вариантов масштабирования можно считать возможность выполнения заданий, запущенных пользователями XtremWeb, на компьютерах под управлением системы Condor. Этому вопросу посвящена работа [22], однако такое решение не автоматизировано, так как требует предварительной подготовки, и не прозрачно в смысле запуска произвольного приложения. В частности, для подключения ресурсов из пула системы Condor необходимо предварительно запустить на каждом из его компьютеров исполнителя системы XtremWeb, оформив в виде задания, причём запуск таких заданий не должен отразиться на запуске остальных заданий пользователей системы Condor.

1.3.3. Система Condor

Одной из широко используемых разработок в области распределённого компьютинга, осуществляющих распределение заданий на некластеризованные ресурсы, является система Condor [24], которая была создана в университете штата Висконсин, США. Система разрабатывалась для объединения компьютеров университета и используется для решения сложных задач, требующих большого количества вычислительных ресурсов. В отличие от традиционных систем пакетной обработки, которые управляют отчуждаемыми ресурсами, Condor позволяет распределять задания как на отчуждаемые, так и не неотчуждаемые компьютеры, используя их вычислительные мощности в моменты простоя.

Объединяя множество ресурсов в один пул, система Condor (рис. 4) предоставляет средства распределения заданий на исполнительные компьютеры, запуска заданий, а также управления заданиями и ресурсами.

Распределение ресурсов основано на использовании языка ClassAd (Classified Advertisement) [25], с помощью которого описываются и ресурсы

компьютеров, и требования заданий к этим ресурсам. Важной особенностью ClassAd является расширяемость множества используемых в нём атрибутов. На основе этого языка реализован механизм (matchmaking) [26], осуществляющий поиск исполнительных компьютеров для каждого задания из очереди путём сопоставления информации о компьютерах и ресурсных запросов заданий.

Компоненты системы Condor на исполнительных компьютерах осуществляют разделение ресурсов между локальными заданиями и заданиями внешних пользователей, соблюдая неотчуждаемость компьютера. В случае возрастания активности владельца, Condor-задание может быть приостановлено и затем возобновлено с того места, на котором оно остановилось.

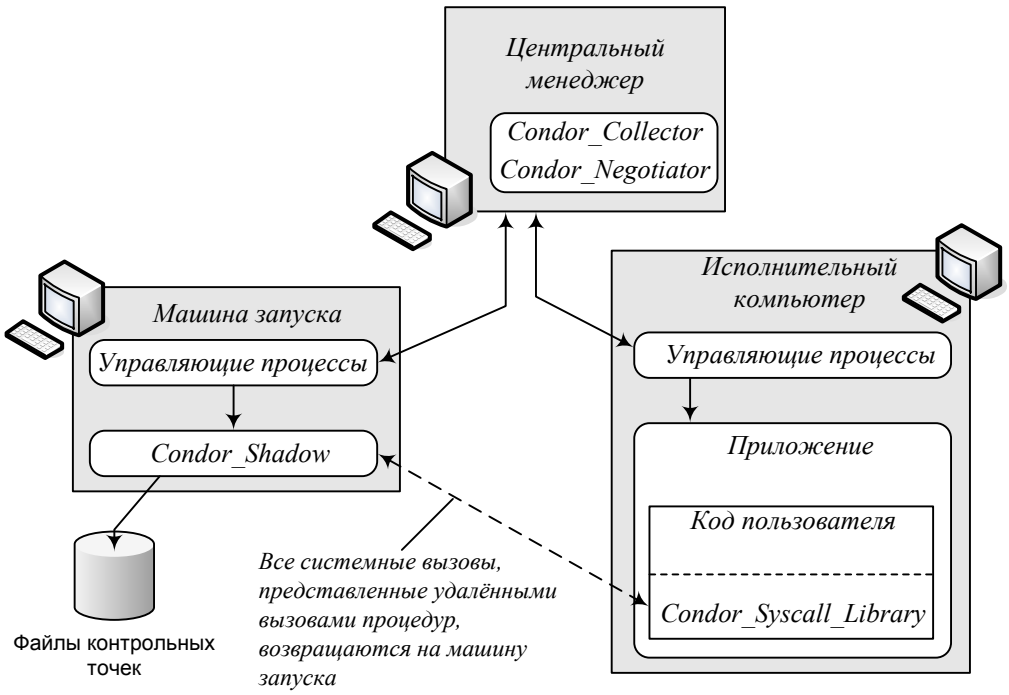


Рис. 4. Архитектура системы Condor

Для того, чтобы запустить задание в системе Condor, пользователю необходимо создать для него файл описания, содержащий требования к исполнительным компьютерам, а также подготовить все необходимые входные файлы, включая исполняемый файл. Все файлы должны находиться

на машине запуска в директории задания. На основе этой информации по команде с машины запуска создается описание задания на языке ClassAd, которое используется для поиска ресурсов, затем задание попадает в очередь.

В системе Condor поддерживается распределённая очередь заданий, обработка которой осуществляется на двух уровнях. На первом уровне каждая машина запуска поддерживает очередь заданий, которые были через неё запущены. Эти задания обрабатываются в соответствии с приоритетами пользователей, кроме того, пользователь может установить порядок выполнения своих заданий. На втором уровне центральный менеджер собирает полную информацию об очередях запускающих машин и формирует глобальную очередь. Получая информацию о состоянии ресурсов пула и их характеристиках, он осуществляет поиск исполнительных компьютеров для каждого задания из этой очереди в порядке их приоритетов.

После того как для задания найдены ресурсы, оно запускается на исполнительном компьютере, а машина запуска осуществляет слежение за этим заданием. Средства управления заданиями системы Condor позволяют пользователю в любой момент времени снять своё задание с выполнения, а также узнать его статус.

После завершения выполнения задания результат автоматически передаётся на машину запуска с помощью средств доставки файлов. Имеется два способа доставки. Первый из них предполагает использование общей файловой системы, которая должна объединять машину запуска и все исполнительные компьютеры. При отсутствии доступа к общей файловой системе, вторым способом является использование собственного механизма системы Condor для передачи файлов.

В системе Condor поддерживаются механизмы миграции заданий, создания контрольных точек, а также удалённого вызова процедур. С помощью механизма удалённого вызова процедур система Condor обеспечивает доступ к среде машины запуска, несмотря на то, что выполнение задания осуществляется на удалённом компьютере.

Пользователям не приходится заботиться об обеспечении доступа к файлам с данными, поскольку задание может работать с ними так, как если бы оно выполнялось на той же машине, откуда было запущено. При этом файлы остаются на машине запуска и не передаются на исполнительный компьютер.

В случае возрастания активности владельца исполнительного компьютера механизм миграции совместно с механизмом создания контрольных точек позволяют продолжить обработку задания на другом компьютере с того места, на котором она приостановилась.

Эти механизмы позволяют эффективно выполнять задания пользователей на ресурсах пула, которые не отчуждаются от своих владельцев. Однако для их использования необходимы исходные коды приложения, которые должны быть скомпилированы с библиотеками системы Condor.

Механизмы безопасности и поддержания целостности данных отвечают требованиям, предъявляемым к грид-системам, обеспечивают надежную передачу информации между компонентами системы, а также гарантируют защиту данных пользователя, запустившего задание, и владельца ресурса. Одним из возможных способов аутентификации в системе является GSI аутентификация на основе публичных ключей с использованием сертификатов.

1.4. Корпоративные системы: система Entropia

Отдельный класс систем распределённого компьютинга составляют корпоративные решения, нацеленные на повышение эффективности работы машинного парка предприятия. Подобные разработки используют компьютеры организации для решения ресурсоёмких задач (например, анализ финансовых рисков) в те часы, когда они слабо загружены, либо простаивают. Так, в системе Entropia [23] предложен новый подход к

объединению персональных компьютеров для обработки заданий, в том числе сериализуемых.

Инфраструктура Entropia состоит из сервера и множества клиентов, на которых установлена клиентская часть системы. Компонент сервера делит задание на множество подзадач и запускает их на клиентских машинах. После завершения вычислений результаты выполнения подзадач собираются вместе и возвращаются пользователю.

Программная инфраструктура системы Entropia (рис. 5) состоит из трех уровней. Компоненты уровней управления исполнительными компьютерами и распределения ресурсов устанавливаются, соответственно, на клиентских машинах и на сервере. Компоненты уровня управления заданиями устанавливаются только на сервере. Совместно с системой Entropia также могут функционировать и другие системы управления заданиями.

Система поддерживает выполнение любого Win32-приложения без каких-либо изменений, путём изоляции его внутри виртуальной машины.

Рис. 5. Архитектура системы Entropia

Программное обеспечение системы Entropia опирается на частные протоколы взаимодействия и базовую программную среду Windows.

1.5. Возможность применения существующих систем для построения одноуровневого грида

Каждая из рассмотренных систем внесла вклад в развитие методов распределённого компьютинга, но в то же время ни одна из них не решает всех задач, существенных для создания одноуровневого грида.

1. Платформа BOINC предназначена для поддержки проектов, а не для запуска приложений. Основное отличие состоит в том, что проект поддерживает выполнение одного или нескольких приложений, разработанных организацией, которая создаёт BOINC-сервер. Все проекты независимы, имеют свои серверы, базы данных и выполняют разные приложения. Таким образом, для каждого нового приложения необходимо создавать свою программно-аппаратную инфраструктуру. Для такого рода систем характерно отсутствие средств запуска и управления произвольными пользовательскими приложениями, оформляемыми в виде заданий.

2. Несмотря на большое количество достоинств и применение системы в распределённой среде при расчётах реальных прикладных задач, система X-Com обладает рядом особенностей, которые не позволяют использовать её в качестве готового решения поставленной задачи — построения одноуровневого грида. Необходимость предварительной подготовки задания, встроенный и неизменяемый механизм планирования, а также отсутствие средств интеграции с существующими системами вычислительного грида делает невозможным использование отдельных компонентов системы независимо от других.

3. Система XtremWeb является одной из наиболее развитых в своём классе, в ней реализован ряд необходимых механизмов. Среди основных механизмов можно выделить следующие: определение активности владельца компьютера, контроль потребления ресурсов заданием грида и определение политики использования ресурсов, а также оригинальные методы разделения ресурсов компьютера между локальными процессами и заданиями грида.

Однако слабым местом системы является планирование, которое не позволяет учитывать требование пользователя о выполнении задания в указанный срок. К тому же в системе XtremWeb отсутствует возможность расширения ресурсного запроса, что делает невозможным качественное улучшение планирования, хотя такая возможность предусмотрена путём конфигурирования планировщика.

4. Entropia представляет собой корпоративное решение на частных протоколах, отличных от применяемых в гриде и ориентированных на локальную среду предприятия. В связи с этим ресурсы, находящиеся под управлением системы Entropia, невозможно интегрировать в существующие грид-инфраструктуры. Кроме того, система строго ориентирована на обработку приложений под ОС Windows.

5. Системы OurGrid и CCOF объединяют ресурсы по типу P2P-сетей и поддерживают выделение ресурсов для произвольных заданий. Обе системы реализуют механизм миграции, кроме того, OurGrid обеспечивает защиту исполнительной машины. В то же время в них не используются ставшие фактическими стандартами протоколы грида. Это приводит к тому, что такого рода разработки не являются интероперабельными с другими грид-системами.

6. Наиболее близка к рассматриваемой в диссертационной работе задаче система Condor, в которой поддерживается использование отдельных исполнительных компьютеров в неотчуждаемом режиме. Однако эта система получила распространение прежде всего как локальный менеджер ресурсов в одном административном домене, аналогичный, например PBS или LSF. Более того, именно в этой роли она широко применяется в гридах, построенных с помощью комплекса Globus Toolkit, в котором служба управления заданиями GRAM [27] имеет встроенный интерфейс с системой Condor в качестве одного из вариантов.

Некоторые механизмы Condor, такие как удалённый вызов процедур и общая файловая система, очевидно локальны, но для них имеются

альтернативы, сближающие Condor со стандартами грида. По-видимому, это и даёт основание разработчикам позиционировать её как систему, способную работать в глобальной сети. Тем не менее при большом числе локальных установок, проекты, в которых Condor управляет глобально распределёнными некластеризованными ресурсами, неизвестны, и такое её применение остаётся проблематичным.

Анализ архитектуры системы Condor выявляет ряд её недостатков с точки зрения одноуровневого грида. В частности, в ней используются собственные интерфейсы взаимодействия компонентов, не основанные на принятых в гриде стандартах. Узким местом системы является её пользовательский интерфейс. В Condor машина запуска, через которую пользователи отправляют свои задания на обработку, также исполняет роль монитора той части заданий, которые через неё запускаются. Для каждого выполняющегося задания на машине запуска стартует отдельный процесс, который завершается только после окончания выполнения задания. Это накладывает дополнительные требования на аппаратные ресурсы машины запуска и делает систему в целом слабо масштабируемой.

Также следует отметить, что механизм распределения ресурсов не в полной мере учитывает специфику неотчуждаемых ресурсов, не обеспечивая, например, завершение обработки заданий в приемлемое время. Для этого необходима дополнительная информация, описывающая состояние процесса вычислений на исполнительном компьютере, а также наличие поставщика, предоставляющего эту информацию. Соответствующим образом должен быть модифицирован сам алгоритм распределения заданий.

Тем не менее Condor предоставляет широкий набор средств, с помощью которых можно решить некоторые из обозначенных проблем. Так, гибкий механизм описания ресурсов и заданий ClassAd позволяет расширить множество атрибутов, описывающих выполнение задания, и учитывать их при поиске подходящих ресурсов.

Следует отметить, что с точки зрения архитектуры грида общим недостатком рассмотренных выше систем является отсутствие информационной службы, публикующей информацию о ресурсах. Наличие такой службы необходимо для включения сегмента из некластеризованных ресурсов в грид-инфраструктуру более высокого уровня.

1.6. Требования к программному обеспечению одноуровневого грида

Выше были рассмотрены существующие подходы и решения для объединения некластеризованных ресурсов в вычислительные инфраструктуры. Показано, что ни одна из разрабатываемых систем не решает в полной мере поставленных в диссертационной работе задач. Исходя из анализа рассмотренных систем, в настоящем разделе будут предложены требования к программному обеспечению грида с некластеризованными компьютерами, выполнение которых позволит учитывать специфику рассматриваемых ресурсов при построении одноуровневого грида.

1.6.1. Общие требования к программному обеспечению одноуровневого грида

Среди основных условий, которым должно удовлетворять программное обеспечение грида (ПОГ), объединяющее некластеризованные ресурсы, выделим следующие.

Запуск и управление заданиями на некластеризованных ресурсах. ПОГ должно реализовывать главную функцию вычислительного грида: обеспечивать выполнение заданий на множестве ресурсов. Программные средства должны принимать задания, описания которых представлены в общепринятой форме (например, RSL [28] или JSDL [29]). Заметим, что используемые языки описания допускают введение дополнительных параметров, связанных со спецификой одноуровневого грида. Одним из

таких параметров может быть ограничение на общее время обработки задания.

Должно поддерживаться управление заданиями — удаление, получение информации о состоянии. Программные интерфейсы запуска и управления должны определяться в соответствии с архитектурой грида. Тем самым, обеспечивается доступ к гриду из любых приложений, которым необходимы дополнительные ресурсы для выполнения прикладной обработки данных. Пользовательский интерфейс управления заданиями является примером такого приложения.

Ресурсы грида используются коллективно и разделяются между различными заданиями. При выполнении задания система должна выделять ему определённое количество исполнительных ресурсов, доставлять входные файлы и результаты, а также контролировать выполнение прикладного кода.

Стандартизованность. ПОГ должно удовлетворять общепринятым стандартам грида. Это обеспечит возможность функционирования соответствующих грид-инфраструктур как автономно, так и в составе объемлющего грида с любыми формами организации ресурсов. В качестве таких стандартов рассматривается архитектура OGSA [30], стандарты веб-служб WSDL [31], SOAP и др.

Поддержка виртуальных организаций. ПОГ должно поддерживать деятельность большого числа динамических объединений пользователей (виртуальных организаций), обеспечивая разграничение доступа к ресурсам, принадлежащих разным виртуальным организациям.

Интероперабельность. ПОГ должно в стандартной форме предоставлять информацию о ресурсах поддерживаемой инфраструктуры, на основе чего может организовываться управление иерархически вложенными сегментами грида.

В дальнейшем изложении в качестве основного варианта одноуровневого грида будет рассматриваться грид с неотчуждаемыми ресурсами. Это предполагает, что они, с одной стороны, используются для

обработки заданий грида, а с другой — что на них выполняются процессы, которые запускают локальные пользователи компьютера (будем называть их процессами владельца компьютера или просто локальными процессами). Заметим, однако, что большинство требований справедливы и для другой важной формы ресурсной организации — когда компьютеры не кластеризуются, но выделяются в грид полностью.

1.6.2. Поддержка неотчуждаемых ресурсов

Специфические особенности неотчуждаемых ресурсов определяют дополнительные требования, которым должна удовлетворять разрабатываемая система.

Соблюдение приоритетов. Задания владельца ресурса должны обладать более высоким приоритетом по сравнению с заданиями грида, то есть выполнение задания грида не должно мешать выполнению локального.

Автономность ресурсов. Ресурсы должны сохранять автономность, то есть владелец должен иметь возможность устанавливать политику их использования и лимитировать объём ресурсов, получаемых заданиями грида. Особое внимание следует уделить безопасности как исходных кодов и файлов с данными задания грида, так и безопасности исполнительного компьютера в целом.

Динамичность среды. Должна учитываться динамичность грида, то есть непрогнозируемые выключения и включения отдельных ресурсов, а также обеспечиваться возможность автоматического, не требующего административного вмешательства, включения компьютеров в состав ресурсной базы грида.

В том случае, если владелец компьютера выделяет под нужды грида ограниченное количество ресурсов, дополнительные ограничения накладываются на способ реализации ПОГ. Так, составляющая ПОГ, которая будет устанавливаться на исполнительный компьютер, должна быть компактной, расходовать минимум системных ресурсов, а её установка и

администрирование не должны требовать от владельца дополнительных знаний и навыков. Требование минимизации программного обеспечения обусловлено тем, что ПОГ увеличивает накладные расходы ресурсов компьютера, тем самым уменьшая долю тех ресурсов, которые может получить задание грида. При этом стремление к минимизации делает затруднительным применение распространённых средств грида, поскольку все они подразумевают установку серверных компонентов с соответствующими накладными расходами на их сопровождение.

На основе сформулированных требований в работе будут предложены решения по архитектуре ПОГ с некластеризованными ресурсами и методам управления ими, а также рассмотрена реализация системы диспетчеризации заданий в гриде с такого рода ресурсами.

Глава 2

Архитектура системы диспетчеризации заданий в гриде с некластеризованными ресурсами

В настоящей главе описывается архитектура системы диспетчеризации заданий в гриде с некластеризованными ресурсами, разработанная в соответствии с представленными ранее требованиями к программному обеспечению грида. Рассматриваются компоненты системы диспетчеризации, их функции и механизмы взаимодействия между собой.

2.1. Состав компонентов системы диспетчеризации

Для объединения некластеризованных ресурсов и включения их в состав грид-инфраструктуры предлагается архитектура ПОГ в виде системы диспетчеризации, которая состоит из трёх компонентов: диспетчера, агента и пользовательского интерфейса (рис. 6).

В соответствии с предложениями работы [6] исполнительные компьютеры ресурсной инфраструктуры не доступны пользователям непосредственно, поэтому внешний интерфейс доступа к ним сосредоточен в диспетчере. Диспетчер устанавливается на выделенный сервер в сегменте грида, и к нему подключается множество исполнительных компьютеров, в том числе пространственно распределённых. Для обеспечения интероперабельности с существующими грид-системами диспетчер имеет стандартный интерфейс запуска заданий (аналогичный GRAM), а реализация

информационной службы позволяет публиковать данные о доступных ресурсах сегмента в информационной системе объемлющей инфраструктуры.

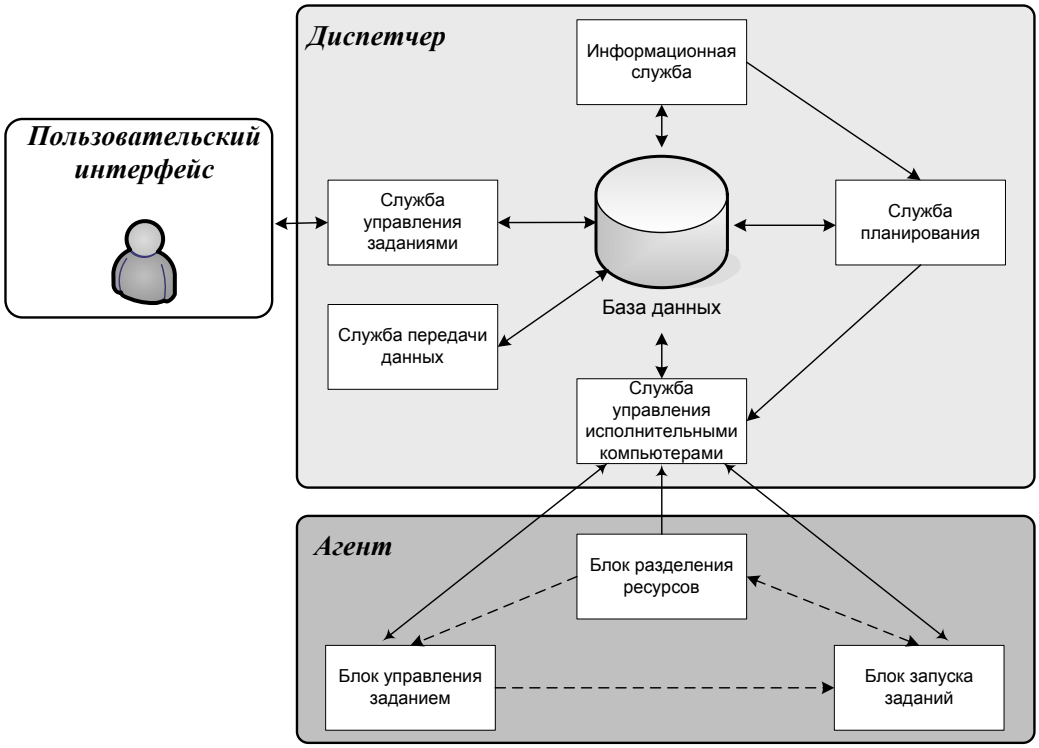


Рис. 6. Архитектура системы диспетчеризации

Основная функция агента, устанавливаемого на исполнительном компьютере, — это управление заданием на стадии выполнения.

Пользовательский интерфейс предоставляет возможность пользователям управлять заданиями стандартным для грида способом, а также получать информацию о состоянии задания.

2.2. Диспетчер одноуровневого грида

Диспетчер одноуровневого грида реализует интерфейсы доступа к сегменту грида с некластеризованными ресурсами, выполняя распределение заданий между зарегистрированными в нём компьютерами.

Диспетчер представляется набором грид-служб, в котором реализованы необходимые базовые компоненты ПОГ, а также средства взаимодействия с агентами. Как известно, грид-службы опираются на стандарты веб-служб, которые, во-первых, широко применяются при построении распределённых

систем, а во-вторых, используются при разработке грид-приложений. Кроме того, реализация подсистем в виде служб позволяет получить модульную систему. В состав диспетчера входят следующие службы:

- служба взаимодействия с агентами;
- служба управления заданиями;
- служба планирования (планировщик);
- информационная служба;
- служба передачи данных.

Для хранения необходимой информации (о пользователях, заданиях, исполнительных компьютерах, некоторой служебной информации) используется реляционная база данных.

Далее рассмотрим подробнее базу данных, а также состав и функции компонентов диспетчера.

2.2.1. База данных диспетчера

Важную роль в системе диспетчеризации играет информационная база, которая хранит информацию о заданиях, исполнительных компьютерах и пользователях. Кроме того, в базе содержится информация о потреблённых заданиями ресурсах грида, назначениях заданий на исполнительные компьютеры и связанных с заданиями файлах, а также некоторая служебная информация.

Информацию о задании можно разделить на регистрационную и информацию о текущем состоянии. Регистрационная информация поставляется службой управления заданиями и содержит следующее: приоритет задания, описание задания (включая ресурсный запрос и информацию о необходимых входных данных), а также информацию о пользователе, который запустил задание. Задание проходит несколько стадий обработки, и его состояние меняется. После того как заданию выделены ресурсы, в базу заносится информация о том, на каком компьютере оно будет запущено. В процессе выполнения задания информация о нём поставляется

службой взаимодействия с агентами и содержит: состояние задания (новое, выполняется, завершено, сброшено и т.д.), а также количество полученных заданием ресурсов (процессорное время, дисковое пространство).

В системе могут быть реализованы различные механизмы ограничения потребления ресурсов компьютера. В простейшем случае такая проверка может осуществляться только средствами диспетчера на основании ресурсного запроса задания. Если по каким-то причинам такой способ является неудовлетворительным, в дополнение агент может самостоятельно контролировать потребление ресурсов исполнительного компьютера. Для поддержки такой функциональности в момент регистрации нового компьютера агент должен предоставить конфигурационный файл с описанием ресурсов (производительность процессора, дисковое пространство, память), в котором будет указан максимальный объём ресурсов, который владелец компьютера отдаёт под нужды грида. Кроме того, он может указать предпочтительный интервал времени использования своего компьютера (например, это могут быть ночные часы, когда компьютер простаивает).

2.2.2. Служба взаимодействия с агентами

Служба взаимодействия с агентами поддерживает связь между диспетчером и исполнительными компьютерами. Эта служба выполняет следующие функции:

- предоставляет интерфейс автоматической регистрации новых исполнительных компьютеров: при регистрации агент обращается к этой службе и передаёт файл с описанием ресурсов компьютера;
- разбирает файл с описанием ресурсов компьютера и заносит полученную информацию в базу данных;
- принимает от агента сведения о состоянии задания и количестве потреблённых им ресурсов.

Взаимодействие агента с диспетчером происходит по внутренним протоколам, которые могут отличаться от протоколов грида. Безопасность обеспечивается взаимным предъявлением сертификатов и наличием доверительных соглашений.

Служба взаимодействия с агентами заносит в базу данных информацию о ресурсах всех компьютеров, подключённых к гриду, а также их статус (занят или свободен). В момент регистрации нового компьютера предоставляется конфигурационный файл с описанием ресурсов (архитектура и производительность процессора, операционная система, дисковое пространство, память). Информация о характеристиках компьютера не может меняться в течение одного интервала активности, поэтому получать её целесообразно только при регистрации и в последующие моменты подключения компьютера к гриду.

В контексте использования разделяемых компьютеров необходимо определить понятие *доступности* исполнительного компьютера. В диссертационной работе рассматривается три типа доступности исполнительного компьютера. Первый тип доступности характеризует способность исполнительного компьютера взаимодействовать с диспетчером в данный момент времени. Иными словами, компьютер доступен по первому типу, если он включён, и на нём запущено агентское программное обеспечение.

Второй тип доступности означает возможность запуска на этом компьютере в данный момент времени задания грида. То есть компьютер должен быть доступен по первому типу, а локальная политика должна позволять запуск конкретного задания.

Третий тип доступности исполнительного компьютера также связан с конкретным заданием грида и характеризуется объёмом вычислительных ресурсов, который может получить это задание при выполнении на данном компьютере.

Помимо информации о текущей доступности компьютеров для более точного планирования может требоваться прогноз состояния и объёма ресурсов в будущем, для чего необходимо накапливать информацию о доступности ресурсов в предыдущие периоды времени. Накопление такой информации осуществляется во время периодических отчётов агентов, в которых указывается, в какие периоды времени компьютер был готов принять задание грида, и каким объёмом ресурсов он располагал.

2.2.3. Служба управления заданиями

Служба управления заданиями принимает задания (осуществляет разбор файла описания задания и добавляет информацию о нём в информационную базу) и выполняет другие команды управления по запросам пользователей. В результате приёма задания пользователю возвращается идентификатор, используя который он может управлять своим заданием, посылая соответствующие команды службе, а также получать информацию о состоянии задания и количестве потреблённых заданием ресурсов. Механизмы авторизации позволяют управлять заданием только его владельцу, то есть тому зарегистрированному пользователю, который ввёл задание в систему.

2.2.4. Служба планирования

Служба планирования (планировщик) отвечает за распределение заданий по исполнительным компьютерам. В системе используется приоритетное планирование с очередью заданий, цель которого состоит в том, чтобы запустить максимальное количество заданий на имеющихся ресурсах (компьютерах) и обеспечить окончание заданий в отведённое время.

Все необходимые для планирования данные хранятся в информационной базе. Первичная информация (статическая и динамическая информация о ресурсах, а также информация о заданиях) поступает в базу от

служб управления ресурсами и заданиями, после чего обрабатывается информационной службой.

Для каждого задания из очереди планировщик сначала ищет подходящий исполнительный компьютер и инициирует запуск задания. После запуска он осуществляет мониторинг выполнения для обеспечения своевременного завершения задания. Рассматриваемый компьютер считается подходящим, если он удовлетворяет ресурсным требованиям задания (архитектура, операционная система, дисковое пространство, оперативная память и пр.), ожидаемое (по статистической информации) получение ресурсов обеспечивает выполнение в срок, а пользователь обладает правами запуска на этом компьютере.

Специфика неотчуждаемых ресурсов накладывает ограничения на способ распределения заданий. Сейчас в гриде применяются два способа: диспетчер сам запускает задание (push) или задание запрашивается у него исполнительным компьютером (pull). Первый способ широко применяется для грида из кластеризованных ресурсов. Такой подход предполагает наличие на исполнительном компьютере службы приёма заданий, поэтому в рассматриваемом варианте некластеризованных ресурсов предпочтительным представляется второй способ (pull), в котором агент, работающий на исполнительном компьютере, сообщает диспетчеру о своей готовности принять задание.

Для реализации планирования в условиях неотчуждаемости ресурсов необходимо прогнозировать их доступность на каждом из компьютеров грида. Такой прогноз основывается на данных о том, сколько процессорного времени компьютеры отдают заданиям грида. Эта информация собирается на исполнительных компьютерах и периодически передаётся в информационную базу. Так как величина выделяемого времени носит случайный характер, необходимо накапливать статистику по различным периодам, учитывая, например, что в ночное время и выходные дни вычислительные ресурсы обычно простаивают или слабо загружены.

При планировании выполняется сопоставление данных о предоставляемом времени с характеристиками задания: количеством нормированного процессорного времени, необходимого заданию по оценке пользователя (*walltime*), и временем, до которого задание должно быть выполнено (*deadline*).

2.2.5. Информационная служба

Информационная служба (ИС) осуществляет двухэтапную обработку непосредственно поставляемой в базу первичной информации о ресурсах.

Исполнительный компьютер может быть занят владельцем и недоступен для выполнения заданий грида. Предсказать, когда это произойдет, невозможно, и, распределяя задания, нельзя ориентироваться на показатели доступных для грида ресурсов в какой-то определённый момент времени. Поэтому на первом этапе первичная информация о каждом компьютере, поставляемая агентами, перерабатывается информационной службой в статистическую, которая отражает количество предоставляемых им ресурсов (процессорного времени) заданиям грида в различные интервалы некоторого периода времени, например суток или недели. Для каждого такого интервала собирается статистика, на основе которой определяется оценка вероятности того, что грид может получить определённое количество ресурсов на данном компьютере. Статистика является основой для планирования распределения заданий.

Второй этап обработки направлен на обобщенное представление суммарного количества ресурсов, которыми располагает сегмент грида. Компьютеры, имеющие схожие характеристики (архитектуру, операционную систему, быстродействие процессора, объём памяти), объединяются в группы. На основе статистической информации в каждой группе вычисляется агрегированное количество ресурсов, доступных для грида в те или иные интервалы времени.

Агрегированная по группам информация служит средством представления сегмента грида в объемлющих грид-инфраструктурах и может публиковаться во внешних информационных системах. Это позволяет остальным участникам грида (пользователям и брокерам ресурсов) видеть информацию о ресурсах сегмента и автоматически обрабатывать её. Для публикации информации в объемлющей инфраструктуре необходимо использовать стандартные и общепринятые протоколы. Обновление агрегированной информации происходит в зависимости от интенсивности изменения ситуации, а также по запросу внешних систем на получение информации о ресурсах сегмента.

2.2.6. Служба передачи данных

Служба передачи и хранения данных осуществляет доставку файлов задания на исполнительные компьютеры с внешних хранилищ, а также доставляет результирующие файлы с исполнительных компьютеров во внешние хранилища. В соответствии со стандартным протоколом управления заданиями GRAM система диспетчеризации поддерживает доставку на машину диспетчера стандартных файлов: исполняемого, входных данных и диагностики. Предполагается, что прикладные файлы располагаются на серверах хранения и считываются диспетчером по протоколам грида управления данными (GridFTP [48]).

2.3. Агент на исполнительном компьютере

Агентское программное обеспечение устанавливается на исполнительных компьютерах, владельцы которых хотят предоставить свои ресурсы в грид.

Ресурсы исполнительного компьютера делятся между процессами владельца и заданиями грида. Если агентское программное обеспечение реализует средства ограничения использования ресурсов, при регистрации исполнительного компьютера его владелец заполняет конфигурационный

файл, в котором указывает, какое количество ресурсов каждого типа (процессор, дисковое пространство, оперативная память и т.д.) он готов выделять в грид-инфраструктуру. Независимо от того, предоставляет ли владелец такое описание, задание не может потребить больше ресурсов, чем было заявлено в его ресурсном запросе, несмотря на то, что свободные ресурсы на исполнительном компьютере фактически ещё могут оставаться. Как только задание выходит за границы потребления ресурсов, указанных в запросе (например, файлы задания занимают больше дискового пространства, или задание потребляет больше процессорного времени, чем было заявлено), его выполнение прекращается.

Агент выполняет запуск задания и управление им на исполнительном компьютере, осуществляя разделение ресурсов между процессами владельца и заданием грида, а также обеспечивая передачу данных и безопасность.

Безопасность рассматривается в трёх аспектах:

- защита владельца, то есть обеспечение полной функциональности компьютера путём контроля уровня потребления ресурсов внешним приложением;
- защита конфигурации компьютера, находящихся на нём программ и данных;
- защита грид-приложения, включая программу, данные и результаты.

Агент на исполнительном компьютере состоит из трёх блоков. Блок запуска заданий информирует диспетчер о том, что ресурсы исполнительного компьютера освободились, и запрашивает новое задание. По этому запросу планировщик выбирает из очереди подходящее задание и сообщает его идентификатор агенту. Затем блок запуска заданий осуществляет доставку файлов задания на исполнительный компьютер, формирует исполнительную среду и запускает задание. После того как задание запущено, управление передаётся блоку разделения ресурсов.

В том случае, когда подходящего задания для освободившегося ресурса нет, агент периодически обращается за ним через определённые интервалы времени.

Блок разделения ресурсов обеспечивает изоляцию программы и данных пользователя грида от данных владельца компьютера, размещая файлы задания в защищённой директории и выполняя задание от имени непривилегированного пользователя. Также осуществляет контроль за потреблёнными заданием ресурсами. Периодически данные об использовании ресурсов заносятся в информационную базу. Кроме того, блок разделения ресурсов осуществляет мониторинг активности владельца исполнительного компьютера. В ситуации, когда выполняющееся задание превысило выделенный ему лимит ресурсов, блок разделения ресурсов инициирует принудительное завершение задания. Блок управления заданием посредством взаимодействия с операционной системой и блоком разделения ресурсов управляет заданием во время его выполнения. Если задание не завершено, а владелец исполнительного компьютера активизировал свою деятельность, блок управления заданием приостанавливает выполнение задания. После того как активность владельца снизилась, выполнение задания возобновляется.

Таким образом, на исполнительном компьютере задание обрабатывается по следующей схеме:

1. Ресурсы на исполнительном компьютере, занятые заданием грида, освобождаются, и агент запрашивает у диспетчера новое задание.

- Блок запуска заданий информирует службу взаимодействия с агентами о том, что исполнительный компьютер готов принять новое задание. Эта служба обращается к службе планирования, которая выдаёт идентификатор подходящего задания с наивысшим приоритетом, а также меняет статус исполнительного компьютера в базе данных.

- Затем блок запуска заданий создаёт на исполнительном компьютере структуру директорий и по полученному на шаге 1 идентификатору доставляет файлы задания через службу передачи и хранения данных.
- Блок запуска заданий формирует исполнительную среду, запускает задание и передаёт идентификатор системного процесса, соответствующего глобальному заданию, блоку разделения ресурсов. Кроме того, блок запуска заданий через службу взаимодействия с агентами меняет статус задания (выполняется).

2. Агент обеспечивает выполнение задания на исполнительном компьютере с соблюдением неотчуждаемости ресурсов.

- Блок разделения ресурсов контролирует количество потреблённых заданием ресурсов. Если задание начинает потреблять больше ресурсов, чем было заявлено в запросе или увеличивается загрузка компьютера за счёт локальных процессов владельца, то посылается запрос блоку управления заданием о снятии задания или приостановлении вычислений. Также блок управления заданием прекращает выполнение задания по запросу пользователя. При этом в информационной базе меняется статус задания.
- Во время выполнения задания блок разделения ресурсов отправляет диспетчеру информацию о количестве потреблённых ресурсов, а блок управления заданием посылает информацию о статусе задания, если статус меняется. Эта информация также заносится в базу данных.

3. После окончания вычислений агент завершает процесс выполнения задания.

- После завершения задания блок запуска отправляет выходные файлы через службу передачи и хранения данных. Затем все данные, связанные с заданием, удаляются с исполнительного компьютера.

- Служба передачи и хранения данных информирует службу управления заданиями о завершении задания и сообщает его идентификатор.
- По идентификатору задания служба управления заданиями доставляет выходные данные пользователю и меняет статус задания в информационной базе (завершено).
- Блок запуска заданий запрашивает статус исполнительного компьютера у блока разделения ресурсов. Если ресурсы свободны, блок запуска заданий информирует диспетчер о том, что исполнительный компьютер готов принять новое задание, и цикл повторяется.

2.4. Пользовательский интерфейс

Пользовательский интерфейс представляет собой клиентское приложение, созданное с помощью API служб диспетчера. Данное приложение принимает от пользователя файл с описанием задания на языке RSL и предоставляет возможность запуска задания в фоновом режиме, получения информации о задании и управления заданием. Во время выполнения задания, осуществляемого в пакетном режиме, пользователь может узнать, в каком состоянии находится задание, количество потреблённых заданием ресурсов, а также остановить выполнение задания. Таким образом, пользовательский интерфейс реализует три функции:

- ввод нового задания;
- отмена введённого задания;
- получение статуса и информации о введённом задании.

Глава 3

Планирование заданий в гриде с некластеризованными ресурсами

Проблема организации грид-инфрастуктур из отдельных пространственно распределённых, но не кластеризованных компьютеров остаётся актуальной и вызывает практический интерес. Об этом свидетельствуют не только теоретические работы в этой области, но и реализации систем управления такими инфраструктурами. Основная причина интереса состоит в том, что подключённые к сети рабочие станции, серверы приложений, персонально используемые компьютеры многочисленны и мало загружены, а их суммарная производительность весьма велика. Задача состоит в том, чтобы использовать это процессорное время для обработки внешних заданий, поступающих по сети (заданий пользователей грида). Однако чтобы использовать весь имеющийся потенциал требуется применение новых программных методов управления такого рода ресурсами, так как компьютеры не выделяются в грид полностью, а делятся между заданиями грида и процессами владельцев.

Можно указать две главные проблемы, которые необходимо решить. Во-первых, должно быть обеспечено разделение ресурсов компьютера (процессора, памяти, дискового пространства и т.д.) между основными процессами и заданиями грида, причём безусловный приоритет должен отдаваться первым. Во-вторых, в зависимости от деятельности владельца компьютера, его загрузка меняется во времени, так что доля остающихся свободными и достающихся заданиям грида ресурсов отличается от их

общего количества. Из-за этого возникает проблема качественного планирования, учитывающего требование завершения заданий в *заданный предельный срок*. Для обеспечения гарантии выполнения задания в заданный срок наиболее известным способом является режим использования разделяемых ресурсов совместно с отчуждаемыми (выделенными) компьютерами, обеспечивающими гарантированный резерв мощности. В том случае, когда задание не успевает выполниться ни на одном из разделяемых компьютеров, выполнение в срок обеспечивается запуском его на одном из выделенных компьютеров [32]. Таким образом, важной задачей является повышение эффективности использования разделяемых компьютеров и минимизации нарушения выполнения заданий в заданный предельный срок.

Первой проблеме посвящён ряд исследований (например, [33], [34]), в которых даны оценки ресурсного потенциала неотчуждаемых компьютеров, количества остающихся свободными ресурсов, влияния внешних программ на интерактивную работу пользователей. Предложен метод *заимствования циклов* (Fine-Grained Cycle Steeling) [35], [36], позволяющий безболезненно использовать свыше 90% свободных циклов процессора.

Настоящая глава посвящена второй проблеме. В разделе 3.1 рассматриваются существующие методы планирования заданий для распределённых вычислительных сред из некластеризованных компьютеров, однако, как будет показано, ни один из них не соответствует в полной мере поставленной в настоящей работе задаче и не позволяет учесть требование пользователя о выполнении задания в заданный срок. В разделе 3.3 предлагается оригинальный метод планирования заданий для одноуровневого грида из неотчуждаемых ресурсов, основная идея которого состоит в использовании статистических данных о загруженности компьютеров и учёта этих данных в алгоритме планирования при распределении заданий. Наличие такой информации позволяет существенно повысить качество планирования, сохраняя автономность вычислительных ресурсов.

3.1. Существующие способы планирования в одноуровневом гриде

Способам планирования в распределённых средах посвящён ряд исследований. В работе [37] предлагается обзор существующих методов планирования для одноуровневого грида. Исходя из анализа этой работы, способы распределения заданий по исполнительным компьютерам могут оцениваться по трём аспектам. Во-первых, возможности планирования существенно зависят от полноты информации о заданиях и исполнительных компьютерах. Во-вторых, планирование может преследовать различные цели, такие как увеличение доли успешно выполненных заданий или справедливое распределение ресурсов, в зависимости от которых используются различные стратегии. И, в-третьих, планирование может осуществляться в различных режимах — пакетном, или в режиме онлайн. В пакетном режиме распределяется некоторая совокупность заданий с учётом требований каждого из них. В этом случае преследуется цель эффективного выполнения всей совокупности. В режиме онлайн для каждого задания выбирается наиболее подходящий компьютер, независимо от того, какие компьютеры нужны остальным заданиям.

3.1.1. Способы планирования в зависимости от полноты информации о заданиях и компьютерах

Получение полной информации о заданиях и исполнительных компьютерах практически невозможно. Это вызвано частым отключением и подключением исполнительных компьютеров, недостоверностью оценок времени выполнения заданий, а также активностью владельцев ресурсов. Поэтому можно рассчитывать лишь на наличие достаточно общей информации о компьютерах (вычислительная мощность, средняя загрузка, текущая доступность и т.д.) и заданиях (ресурсные требования к исполнительному компьютеру, время выполнения, предельный срок исполнения и т.п.). На основе имеющейся информации планировщик может

произвести ранжирование исполнительных компьютеров, вычислив для каждого из них т.н. *функцию полезности* (utility function), показывающую в какой степени данный компьютер подходит для выполнения задания.

Несмотря на то, что вопросам сбора информации о состоянии компьютеров посвящено достаточно много работ [38], [39], её получение в гриде является сложной задачей. Кроме того, особую ценность для осуществления планирования представляет не простой мониторинг ресурсов, а предсказание их состояния в некотором будущем. По этой причине наиболее распространены алгоритмы планирования, не требующие для принятия решения информации о заданиях и исполнительных компьютерах — случайные алгоритмы.

3.1.2. Способы планирования в зависимости от поставленных целей

К работе планировщика заданий могут предъявляться различные требования, например, такие как минимизация отказов в обработке заданий, повышение отказоустойчивости в случае возникновения различного рода сбоев или справедливое распределение имеющихся вычислительных ресурсов между всеми заданиями.

Отказоустойчивые алгоритмы. Высокая динамичность грида не позволяет в достаточной степени предсказать поведение владельцев исполнительных компьютеров, пользователей грида, и тем более возникновение различного рода ошибок, как оборудования, так и программного обеспечения. Поэтому для увеличения доли успешно выполненных заданий необходимо использовать устойчивые к возникновению ошибок алгоритмы планирования, в которых способами борьбы с ошибками являются механизмы *репликации* и *контрольных точек*.

Репликация заключается в одновременном выполнении нескольких копий (реплик) одного задания на разных исполнительных компьютерах. В некоторых системах репликация используется для обеспечения

достоверности расчётов путём сравнения результатов выполнения одного задания на нескольких различных компьютерах.

Наличие механизма контрольных точек позволяет во время выполнения задания сохранять промежуточные результаты и значения параметров окружения. Затем, в случае возникновения ошибки приложения или выхода из строя исполнительного компьютера эта информация позволяет продолжить выполнение задания на другом компьютере с момента сохранения последней контрольной точки.

Оба механизма обладают как преимуществами, так и недостатками. Репликация не требует дополнительных накладных расходов, которые необходимы в случае реализации механизма контрольных точек, так как для его поддержания, как правило, выделяется отдельный сервер, хранящий контрольные точки. Кроме того, процесс создания контрольной точки, передачи необходимой информации на новый исполнительный компьютер и последующего восстановления требует определённого времени. С другой стороны, существенным недостатком алгоритмов с использованием репликации является многократное увеличение потерянных (потраченных впустую) вычислительных ресурсов за исключением небольшой части, которая использовалась для обеспечения достоверности результата.

Алгоритмы со справедливым распределением ресурсов. В локальных средах в качестве целей планирования часто используются критерии скорейшего выполнения заданий пользователей и эффективной загрузки имеющихся ресурсов. Однако в такой многозадачной и многопользовательской среде, какой является грид, важно также обеспечить гибкую систему распределения вычислительных ресурсов между заданиями, так как одновременно на один и тот же ресурс могут претендовать несколько заданий [1]. Основная сложность этой задачи состоит в *справедливом* распределении имеющихся ресурсов между пользователями грида. Под справедливым распределением понимается такое разделение вычислительных ресурсов, при котором каждый пользователь

гарантированно получает за определенный период времени ограниченное количество ресурсов, соответствующее проводимой в виртуальной организации политике.

Конкретная политика может реализовываться различными механизмами, например, с помощью квот ресурсов или на основе экономических принципов. В механизме квот общее количество ресурсов представляется как единое целое, части которого в пропорциональных долях (квотах) распределяются между пользователями.

Во втором механизме (экономические принципы) могут использоваться экономические модели, отражающие различные способы ценообразования (торги, аукцион, биржа, договор и т.д.). Здесь вводится понятие бюджета, то есть ограничения денежных средств, которыми пользователь может расплачиваться за ресурсы. Смысл этого подхода в том, что на исполнительном компьютере запускается то задание, за выполнение которого предлагается наибольшая цена, складывающаяся, например, в ходе торгов. В то же время экономические отношения могут пониматься и в упрощенном виде, когда процесс ценообразования не является динамическим [42].

3.1.3. Пакетный и онлайн режимы планирования

Результат планирования зависит не только от наличия информации о компьютерах и заданиях. Независимо от имеющейся информации роль играет и то, в какие моменты осуществляется цикл планирования — одновременно распределяется некоторая совокупность заданий, находящихся в очереди, или каждое очередное задание распределяется независимо от других. В связи с этим планирование может осуществляться в двух режимах — *пакетном* или *онлайн*, соответственно. В режиме онлайн для каждого конкретного задания ищется наиболее подходящий компьютер, независимо от того, какие компьютеры нужны остальным заданиям. Использование пакетного режима позволяет выполнить множество (пакет) имеющихся

заданий в целом за минимальное время. При слабом потоке заданий и/или коротком цикле планирования предпочтителен режим онлайн распределения, при котором задание сразу отправляется на выполнение, не дожидаясь поступления других заданий. Если поток заданий интенсивный и/или цикл планирования длится достаточно долго (во время цикла планирования успевает поступить некоторое количество новых заданий, из-за чего очередь постоянно растёт) наиболее подходящим представляется пакетный способ планирования, позволяющий загружать ресурсы более эффективно и тем самым выполнить задания за меньшее время. В конкретных алгоритмах планирования эти режимы могут совместно использоваться на различных стадиях планирования, когда некоторое количество заданий обрабатывается в пакетном режиме, а затем вновь поступающие задания распределяются в режиме онлайн.

3.2. Сравнение различных алгоритмов планирования

В зависимости от свойств того или иного алгоритма планирования, он может быть классифицирован по каждому из трёх предложенных выше аспектов. Далее будут рассмотрены отличительные особенности алгоритмов в зависимости от того или иного аспекта.

3.2.1. Случайные алгоритмы планирования

Как отмечалось ранее, наиболее распространёнными на данный момент в рассматриваемой области являются случайные алгоритмы планирования. В первую очередь это объясняется трудоёмкостью, а в некоторых случаях невозможностью, получения в полном объёме информации о ресурсах и заданиях, а также простотой реализации. В рамках предложенной выше классификации такие алгоритмы относятся к типу алгоритмов, которые не требуют для принятия решения информации о состоянии исполнительных компьютеров или характеристиках заданий, могут обладать различными средствами для повышения отказоустойчивости, а распределение заданий

осуществляется, как правило, в режиме онлайн. Среди алгоритмов данного типа рассмотрим более подробно базовый случайный алгоритм с репликацией, алгоритм с репликацией и автоматическим перезапуском, а также устойчивый к ошибкам алгоритм с репликацией.

3.2.1.1. Стандартный алгоритм с репликацией

Распределение заданий в базовом алгоритме с репликацией (WorkQueue with Replication — WQR) осуществляется на случайным образом выбранные свободные компьютеры. Если после запуска всех заданий остаются свободные компьютеры или во время выполнения какие-то компьютеры освобождаются, планировщик запускает на них реплики до тех пор, пока не будет достигнуто заранее установленное максимальное количество реплик. Как только одна из реплик завершается, остальные снимаются с выполнения. Как показано в [43], за счёт использования большего количества вычислительных ресурсов WQR обладает такими же характеристиками по среднему времени выполнения пакета заданий, что и алгоритмы, использующие для принятия решения информацию об инфраструктуре.

3.2.1.2. Алгоритм с репликацией и автоматическим перезапуском

Принимая во внимание высокую динамичность среды, возможны ситуации, когда в случае сбоя при выполнении задание не может быть успешно выполнено на выбранном компьютере. Чтобы избежать такой ситуации, может быть использована модификация алгоритма, использующая во время работы механизм перезапуска реплик (WorkQueue with Replication and automatic Restart — WQR-R). В случае ошибки при выполнении одной из реплик, она будет перезапущена на освободившемся компьютере, если для всех остальных заданий существует хотя бы одна реплика.

3.2.1.3. Устойчивый к ошибкам алгоритм с репликацией

Несмотря на то, что алгоритм с репликацией WQR-R обеспечивает успешное выполнение заданий, достигается это ценой многократного

увеличения ресурсов, большая часть вычислительной мощности которых тратится впустую. Решить эту проблему помогает механизм контрольных точек, позволяющий продолжить выполнение приложения с момента создания последней контрольной точки. В соответствующей модификации WQR-FT (WorkQueue with Replication Fault Tolerant) предполагается, что восстановление реплики возможно на компьютере с той же операционной системой, на которой была создана контрольная точка. Если же к моменту восстановления контрольной точки свободного компьютера с необходимой операционной системой нет, то выполнение реплики начинается с начала на одном из свободных компьютеров.

3.2.1.4. Выводы

Случайный алгоритм не требует для планирования информации о характеристиках компьютеров или параметрах заданий, поэтому он может использоваться в системах, где такая информация недоступна, либо её получение сопряжено с трудностями. Рассмотренные модификации алгоритма позволяют увеличить его эффективность (увеличить долю выполненных заданий и сократить общее время их обработки) за счёт использования репликации и механизма контрольных точек. Однако, несмотря на увеличение числа выполненных заданий и уменьшение времени завершения, существенным недостатком этих модификаций является чрезмерное (в разы) использование процессорных ресурсов. Это связано с тем, что для каждого задания запускается несколько реплик, результат выполнения которых не представляет ценности для общего счёта.

В следующем разделе будет рассмотрен класс устойчивых к ошибкам алгоритмов, использующих для принятия решения информацию о состоянии исполнительных компьютеров и характеристики заданий. Такие алгоритмы эффективнее случайных, и позволяют более рационально использовать ресурсы исполнительных компьютеров.

3.2.2. Алгоритмы, использующие информацию о ресурсах и заданиях

Как отмечалось ранее, алгоритмы планирования, не учитывающие при принятии решения информацию о заданиях и вычислительных компьютерах (случайные алгоритмы), обладают рядом недостатков. В качестве основного из них можно отметить нерациональное использование вычислительных ресурсов. В настоящем разделе будут рассмотрены методы распределения заданий, в которых учитывается информация об исполнительных компьютерах и заданиях. По предложенной выше классификации они могут быть отнесены к алгоритмам, которые используют для принятия решения информацию об исполнительных компьютерах и заданиях, работают в режиме онлайн, кроме того, в них могут использоваться различные средства повышения отказоустойчивости.

3.2.2.1. Способы выбора очередного задания

В зависимости от имеющейся информации планировщик может принимать решение о распределении заданий, основываясь на характеристиках компьютеров, параметрах самих заданий, а также используя оба эти типа информации. В дальнейшем рассмотрении алгоритмов планирования предполагается, что для исполнительного компьютера могут быть получены данные о производительности и доступности (то есть моментах его отключения), а о задании известно, какое количество времени оно выполняется на эталонном компьютере, то есть на компьютере с производительностью P=1.

Во время работы планировщику необходимо принять два основных решения — какое задание распределять следующим, и на какой исполнительный компьютер его направить. В предыдущем разделе был представлен случайный алгоритм WQR-FT, который, не обладая информацией о компьютерах и заданиях, случайным образом определял очередное задание и таким же образом выбирал для задания исполнительный

компьютер. В дальнейшем рассмотрении выделяется два возможных варианта для выбора очередного задания (самое короткое — Short, или самое длинное — Long) и четыре возможных варианта полноты информации о компьютерах (нет информации — NoInfo, информация о производительности — CpuInfo, информация о доступности — AvailInfo, наличие полной информации — FullInfo). Таким образом, в зависимости от имеющейся информации получается восемь возможных алгоритмов планирования.

Имея информацию о времени выполнения задания, можно более рационально загружать вычислительные ресурсы в зависимости от того, какой эффект должен быть достигнут. Алгоритм Short позволяет выполнить как можно больше коротких заданий, чтобы затем на освободившихся компьютерах можно было запустить максимальное количество реплик длинных заданий, увеличивая тем самым вероятность успешного выполнения всех заданий. В свою очередь, алгоритм Long направлен на то, чтобы сначала были завершены длинные задания, так как они наиболее требовательны к производительности компьютеров.

3.2.2.2 Способы определения подходящего компьютера

В зависимости от имеющейся информации о компьютерах, выбор осуществляется следующим образом. Если о компьютере ничего не известно (NoInfo), то выбирается любой из числа свободных. При наличии информации о производительности (CpuInfo) выбирается компьютер с максимальной производительностью. Это позволяет как можно быстрее выполнить задание и не допускать захвата слабых компьютеров (когда длинное задание попадает на слабый компьютер).

В том случае, когда планировщик может предсказать момент времени, в который компьютер будет недоступен (AvailInfo), выбирается самый надёжный — с самым поздним моментом отключения. При таком сценарии уменьшается вероятность возникновения ошибки во время выполнения задания, вследствие чего нет необходимости перезапускать задание на

другом компьютере, также в этом случае может быть создана наиболее поздняя контрольная точка. Если же планировщик обладает информацией как о производительности, так и доступности компьютеров (FullInfo), он может определить, сколько по времени задание будет выполняться на каждом компьютере. В таком случае выбирается компьютер с максимальной производительностью, на котором задание успеет выполниться до момента отключения. Если ни на одном из свободных компьютеров задание не успевает выполниться до отключения, то выбирается наиболее мощный из них (как в CpuInfo).

3.2.3. Алгоритмы с пакетным распределением заданий

В разделах 3.2.1 и 3.2.2 были рассмотрены алгоритмы, распределяющие задания в режиме онлайн. Несмотря на то, что такие алгоритмы имеют важное преимущество — задания распределяются на исполнительные компьютеры как можно быстрее, а также выбирается наиболее мощный компьютер, такой способ не всегда походит для обработки интенсивного потока заданий и поддержки большого количества исполнительных компьютеров. В этом случае за время поиска наиболее подходящего компьютера может поступить некоторое количество новых заданий. Использование пакетного режима распределения позволяет рассматривать очередь заданий как единое целое и более эффективно использовать имеющиеся ресурсы для завершения всех заданий за минимальное время.

В случае пакетного распределения заданий планировщику необходимо рассмотреть все возможные варианты назначений заданий на различные ресурсы и выбрать наиболее подходящее назначение. Для оценки оптимальности вводится т.н. *функция полезности* (*UF* — utility function), выражающая в численном виде преимущество назначения задания на тот или иной компьютер. Функция полезности может быть выражена через самые различные характеристики, такие как цена задания, время выполнения

задания, эффективная загрузка используемых компьютеров и т.п., а также через комбинации этих характеристик. В большинстве случаев функция полезности зависит от того, какой информацией о заданиях и компьютерах обладает планировщик.

Обозначим величиной UF_{ij} полезность выполнения задания i на компьютере j. Тогда построение решения заключается в поиске оптимального распределения заданий по исполнительным компьютерам в соответствии с полезностью каждого назначения и состоит в максимизации суммы $\sum_{i,j} UF_{ij} * X_{ij}$, где $X_{ij} = 1$ в том случае, если задание i выполняется на компьютере j, и $X_{ij} = 0$ в противном случае.

3.2.4. Выводы

В разделе 3.2 были рассмотрены различные сценарии работы планировщиков при распределении заданий в зависимости от условий и целей алгоритмов планирования. Отмечено, что при отсутствии информации об исполнительных компьютерах механизмы репликации и контрольных точек позволяют улучшить характеристики случайного алгоритма в отношении устойчивости к ошибкам и времени обработки множества заданий.

Несмотря на то, что получение полной информации о компьютерах и заданиях в распределённой среде является сложной задачей (необходимо определить состав собираемой информации, разработать механизмы её сбора и т.д.), некоторую информацию, такую, как производительность компьютера и длину задания, получить просто. Кроме того, центральный сервер диспетчеризации может фиксировать моменты отключения и включения компьютеров, которые используются для оценки их надёжности. Учёт такой информации позволяет улучшить качество планирования как по общему времени выполнения заданий, так и по количеству потерянного процессорного времени. Наконец, планирование сразу большого количества

заданий из очереди позволяет сократить время выполнения множества заданий, однако использование пакетного распределения в случае неинтенсивного потока заданий не оправдано.

3.3. Метод планирования, направленный на завершение заданий в заданный срок

Рассмотренные в предыдущем разделе алгоритмы планирования используются в распределённых системах, однако применить их для решения поставленной в работе задачи не представляется возможным, так как ни в одном из предложенных способов не учитывается свойство *неотчуждаемости* компьютеров. Планировщику остаётся принимать решение о размещении задания, основываясь лишь на характеристиках оборудования компьютеров и мгновенных показателях их загрузки. Учитывая условия одноуровневого грида, применение механизма миграции может приводить к постоянному перезапуску заданий на различных компьютерах, в результате чего большинство заданий так и не смогут выполниться. Ориентированность на наличие механизма контрольных точек не позволяет применять алгоритмы, использующие это средство, для запуска произвольных заданий (в том числе не поддерживающих контрольные точки).

В связи с этим разработка нового метода планирования заданий в гриде, направленного на обеспечение завершения заданий в заданный срок, является актуальным направлением деятельности, результаты которой позволят качественно улучшить распределение заданий. В настоящем разделе будет рассмотрен оригинальный метод, направленный на достижение поставленной цели.

3.3.1. Задача планирования в гриде с неотчуждаемыми ресурсами

В связи с режимом использования компьютеров, когда их ресурсы разделяются между заданиями владельца и заданиями грида, возникает ситуация, при которой доля таких ресурсов, получаемая внешним заданием, меняется в зависимости от активности владельца компьютера. С точки зрения системы диспетчеризации такие ресурсы являются *недетерминированными*, так как невозможно точно предсказать время окончания задания, выполняющегося на неотчуждаемом компьютере. Это порождает проблему планирования процесса обработки, которое должно обеспечивать завершение выполнения каждого задания в заданный предельный срок. Для решения этой проблемы необходимо применять специальные меры, направленные на минимизацию нарушений задаваемого пользователем грида предельного времени выполнения задания.

Для планирования в гриде с неотчуждаемыми компьютерами наиболее важными являются характеристики, определяющие загрузку ресурсов и позволяющие её прогнозировать. Разработаны модели [40], [41], [44], которые предсказывают состояние ресурсов в пределах коротких отрезков времени (1–15 секунд). Эти модели служат основой для разработки механизмов разделения ресурсов компьютера.

В гриде обрабатываются задания с длительным временем выполнения (порядка нескольких часов), и для планирования их распределения между компьютерами более полезным выглядит использование статистических (усреднённых по большим интервалам времени) характеристик компьютеров. В контексте работ по созданию системы управления заданиями в гриде с неотчуждаемыми некластеризованными компьютерами проведено исследование возможностей улучшения стандартных методов планирования путём построения прогноза загрузки ресурсов на основе статистических характеристик.

В качестве основного показателя, который характеризует функционирование компьютера с точки зрения его работы на грид, введём понятие *эффективной производительности*. Эффективная производительность компьютера на некотором интервале времени T определяется как средняя величина процессорной мощности, которая остаётся неиспользованной при работе владельца компьютера и может быть получена внешним заданием. Она задаётся в условных единицах по отношению к производительности эталонного компьютера, принимаемой за единицу. Если задание (рассматриваются счётные задания) на эталонном компьютере выполняется за время T, то на компьютере с эффективной производительностью H оно выполнится за время T/H.

В системе диспетчеризации сбор данных, по которым вычисляется эффективная производительность, производится отдельно для каждого компьютера. Для этого в составе агента, управляющего обработкой внешних заданий на компьютере, реализуется программа, которая, получает от операционной системы значения текущей загрузки процессора [3] и периодически посылает их в информационную базу планировщика.

По собранным данным составляется прогноз эффективной производительности. Предлагаемый способ получения прогноза основан на предположении, что деятельность владельца компьютера в основном регулярна: мало изменяется день ото дня, но в течение суток могут быть перерывы в работе, а в конце рабочего дня компьютер, как правило, отключается. В связи с этим прогноз эффективной производительности представляется набором значений H, относящихся к временным интервалам, на которые разбиваются сутки. Для вычисления H на одном интервале производится усреднение полученных от компьютера значений текущей загрузки процессора. Кроме того, производится усреднение методом скользящего окна значений H, относящихся к нескольким предыдущим суткам.

Следует отметить, что существуют нерегулярные факторы, например, отпускные дни, в которые компьютер может быть выключен в течение длительного времени. Учесть их трудно, и следует исходить из того, что прогноз не является точным.

3.3.2. Использование прогноза эффективной производительности

В получивших известность и описанных в первой главе системах управления неотчуждаемыми компьютерами не учитывается, что их производительность не совпадает с эффективной производительностью. В соответствии с ведущейся на компьютерах основной деятельностью эффективная производительность меняется во времени. Вследствие этого возникают ситуации, когда задание, распределённое на некоторый компьютер, не успевает завершиться в заданный пользователем предельный срок (deadline). Прогноз эффективной производительности позволяет дать оценку времени завершения задания и использовать её как дополнительный критерий при выборе компьютера. Как будет показано далее, таким способом можно уменьшить число отказов — превышений предельного срока — даже при неточном предсказании.

Сформулируем условия, в которых решается задача планирования. В грид поступает поток заданий $\{J_i,\ i=1,2,\ldots\}$. Алгоритм планирования выполняет распределение заданий по компьютерам $\{C_k,\ k=1,\ldots,Nc\}$ в режиме on-line, то есть каждое задание распределяется независимо от других. Исходными данными планирования являются:

- характеристики заданий: длина (время выполнения на эталонном компьютере) W_i и предельный срок выполнения D_i;
- характеристики компьютеров: эффективная производительность H_k.

Обычно для неотчуждаемых компьютеров используются алгоритмы планирования, которые опираются лишь на характеристики заданий: рассмотренный ранее алгоритм WQ и его модификации, а также такие

алгоритмы как First Come-First Serve (FCFS) и List Scheduling. Однако любой из этих алгоритмов может быть модифицирован для учёта эффективной производительности компьютеров. В следующих разделах проводится сравнение качества планирования алгоритмов FCFS и его модификации ECP-FCFS (Effective Computer Power-FCFS). Критерием выступает число отказов по превышению предельного срока.

Алгоритм FCFS.

1. Если очередь не пуста, и имеются свободные компьютеры, то выбирается случайный компьютер и на него распределяется первое задание из очереди.

2. В противном случае ожидаются события поступления задания или освобождение компьютера.

Алгоритм ECP-FCFS.

В этом алгоритме на шаге 1 меняется способ выбора исполнительного компьютера:

1.1. Выбирается первое задание из очереди.

1.2. Задание распределяется только на тот компьютер, где оно может окончиться в срок. Если таких компьютеров нет, делается попытка распределить следующее задание из очереди.

3.3.3. Оценка качества планирования

Сравнение алгоритмов FCFS и ECP-FCFS выполнено путём моделирования их работы на синтетических входных данных, которые описывают поток заданий и характеристики компьютеров ресурсного пула.

Компьютеры в количестве Nc=100 имеют эффективные производительности, генерируемые случайным образом по равномерному распределению из диапазона [0.1, 1.0]. Длины заданий и времена их поступления также задаются равномерным распределением. Диапазон длин заданий — [150, 750], задания поступают на временном отрезке [0, TS], где TS — минимально возможное время обработки потока, вычисляемое как

отношение суммарной длины всех заданий к суммарной производительности компьютеров. Предельный срок выполнения заданий выбирается пропорционально их длине $D_i=\alpha_i*W_i$. Коэффициент α_i фактически определяет нижнюю границу эффективной производительности, при которой задание может выполниться в срок. α_i равномерно распределены в диапазоне [1.1, 5.0].

В результате моделирования обработки пакета из 1000 заданий получено, что количество отказов алгоритма FCFS (201, то есть 20.1% от числа заданий) превосходит количество отказов алгоритма ECP-FCFS (30, то есть 3%). Улучшение, которое даёт учёт эффективной производительности, — значимое, но не слишком большое. Объясняется это тем, что компьютеры со слабой производительностью ($H<0.5$), на которых в основном и происходят отказы, вносят небольшой вклад в суммарную производительность грида. Если число таких компьютеров увеличивается, и соответственно растёт их вклад, преимущество ECP-FCFS становится весьма существенным. Это иллюстрируется экспериментом, в котором к компьютерному пулу, описанному выше, добавляются дополнительные слабые ($H=0.2$) компьютеры. График зависимости количества отказов от числа добавленных компьютеров представлен на рис. 7. Хотя суммарная производительность растёт, в алгоритме FCFS число отказов линейно увеличивается, в то время как в ECP-FCFS оно остаётся постоянным.

В реальности прогнозируемая производительность будет отличаться от получаемой заданием во время выполнения. В следующем эксперименте изучалось влияние точности прогноза на число отказов. Относительное отклонение реально получаемой производительности Hr от прогнозируемой H задаётся величиной R, $Hr=H*(1+R)$. Значения R генерируются по нормальному распределению со средним значением 0 и стандартным отклонением σ. Величина σ является параметром исследования: при $\sigma=0$ время выполнения заданий совпадает с прогнозируемым, при больших σ оно может значительно отклоняться от прогноза.

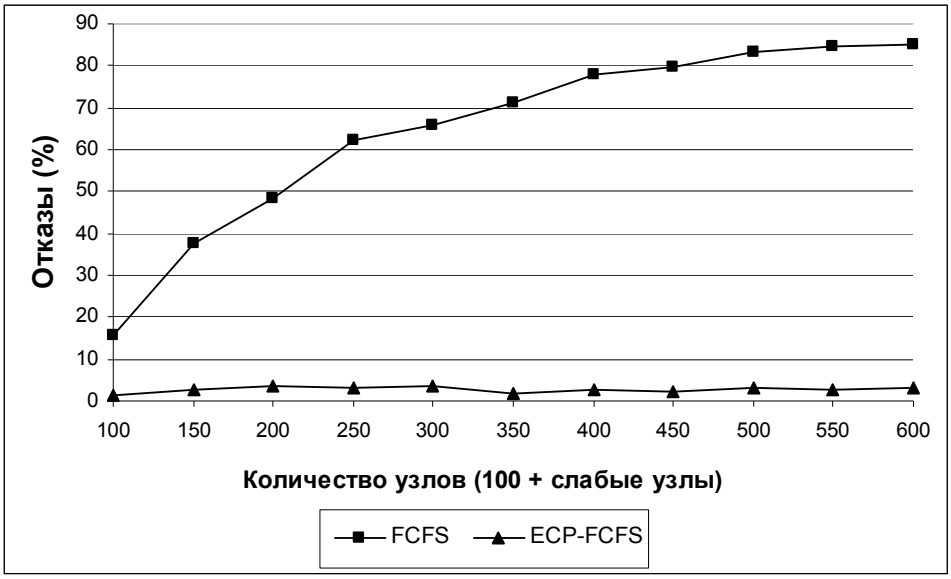


Рис. 7. Зависимость числа отказов от количества добавленных слабых компьютеров

Рисунок 8 соответствует условиям, когда производительности компьютеров и коэффициенты α_i (определяющие предельные сроки заданий) распределены равномерно. Как видно из графика, использование прогноза в алгоритме ECP-FCFS уменьшает число отказов в исследованном диапазоне $0 \le \sigma < 1$. При добавлении в пул слабых компьютеров (рис. 9) эффект становится ещё более заметным.

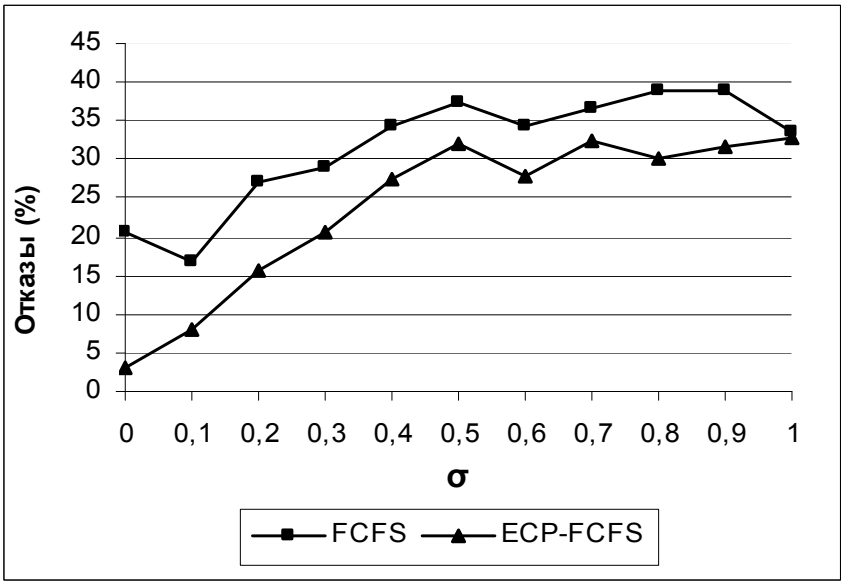


Рис. 8. Зависимость числа отказов от параметра σ

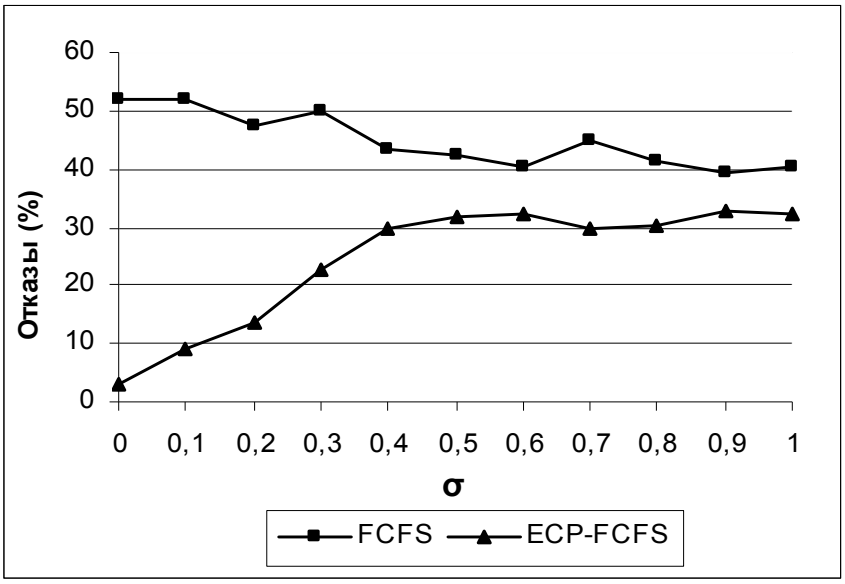


Рис. 9. Зависимость числа отказов от параметра σ (добавлено 100 слабых компьютеров)

3.3.4. Эффективное использование пула неотчуждаемых компьютеров

Помимо уменьшения числа отказов учёт эффективной производительности компьютеров даёт дополнительный эффект: при точном прогнозе (σ=0) невозможность обработки задания в срок определяется в период его нахождения в очереди, так что задание даже не распределяется на компьютер. Когда $\sigma \neq 0$ возможны два вида отказов: по превышению времени ожидания в очереди и по превышению времени выполнения (рис. 10).

Наличие прогноза позволяет, во-первых, заблаговременно информировать владельцев заданий о невозможности их обработки в срок и, во-вторых, не тратить ресурсы впустую. Для оценки того, насколько продуктивно задействуются мощности ресурсного пула, будем использовать степень полезной загрузки ресурсов U теми заданиями, которые завершаются в срок: $U = \dfrac{\sum_i (FT_i - ST_i) * H_{i_k}}{MS * HT}$

Суммирование в числителе отношения ведётся по всем заданиям, которые обработаны в срок, ST_i и FT_i — время старта и завершения задания i, H_{i_k} — эффективная производительность компьютера во время выполнения

задания. В выражении *MS*HT*, стоящем в знаменателе, *MS* обозначает время счёта пакета заданий, $HT = \sum_k H_k$, где H_k — эффективная производительность компьютера за время *MS*, и суммирование ведётся по всем компьютерам ресурсного пула.

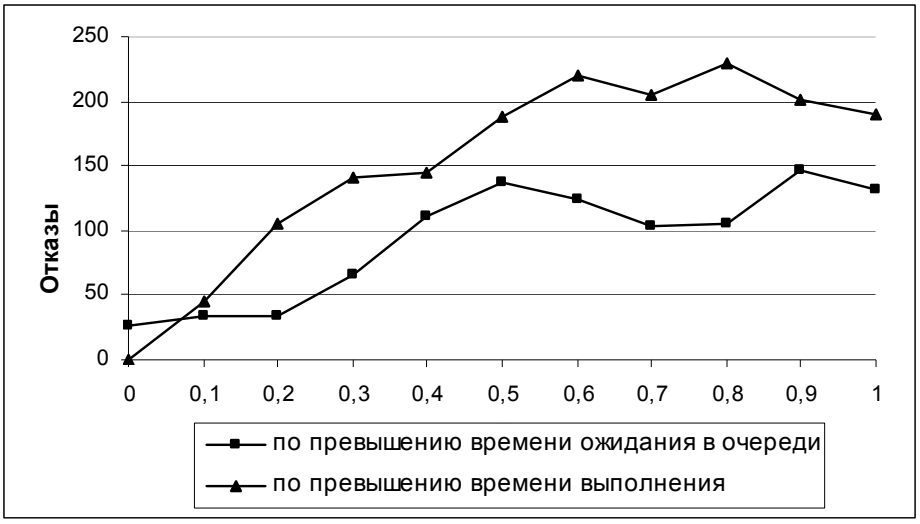


Рис. 10. Соотношение числа отказов по превышению времени ожидания в очереди и по превышению времени выполнения

Для исходного пула (с равномерным распределением производительностей) полезная загрузка составляет 40%. Когда к нему добавляются слабые компьютеры, суммарная мощность возрастает, но доля полезной загрузки уменьшается до 15% (рис. 11).

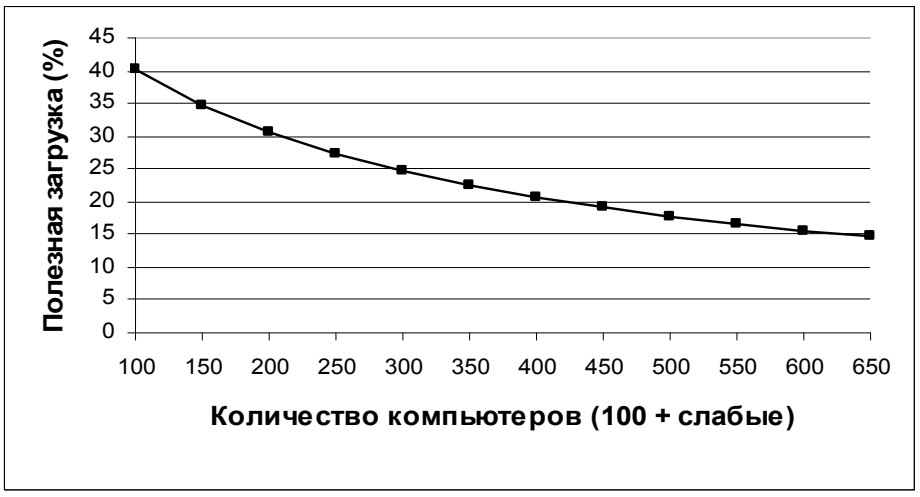


Рис. 11. Полезная загрузка пула при добавлении слабых компьютеров

Причина того, что задания получают отказ в обслуживании, в то время как в системе имеются незанятые компьютеры, причём в любом количестве, состоит в том, что эти компьютеры имеют недостаточно высокую производительность для выполнения заданий в срок. Более полная загрузка ресурсов может быть достигнута с помощью известного способа запуска заданий в виде пакетов подзаданий (Bag-of-Tasks) [46]. Согласно этому способу задание J разбивается на множество подзаданий $\{SJ_1,\ldots, SJ_M\}$, каждое из которых выполняет часть вычислений. Наибольший эффект такое разбиение даёт, если подзадания являются независимыми друг от друга и могут выполняться параллельно. Тогда можно обработать всю совокупность подзаданий в срок, определённый для задания J, но при меньших требованиях к производительности компьютеров.

Время обработки задания J длины W на компьютере с производительностью H составляет $T = \frac{W}{H} + Q + DV$, здесь $\frac{W}{H}$ — время выполнения, Q — время ожидания в очереди, DV — время доставки входных и выходных файлов. Тогда минимальная производительность компьютера, обеспечивающая выполнение задания за время DL, оставшееся до наступления предельного срока, равна $H_{\min} = \frac{W}{DL - DV}$.

Если задание J разбито на M подзаданий SJ_i, которые отличаются только входными данными, то время обработки каждого $T_i = \frac{W_i}{H_i} + Q_i + DV_i$. Всё множество подзаданий будет обработано за тот же предельный срок DL, что и исходное задание, если $\frac{W_i}{H_i} + Q_i + DV_i \le DL$ для всех i=1,..,M. Предполагая, что длительность выполнения всех подзаданий и время доставки файлов примерно одинаковые и равны соответственно $W_i = \frac{W}{M}$, $DV_i = \frac{DV}{M}$, можно дать следующую оценку минимальной производительности

компьютеров: $H_{\min_i} = \frac{W}{M * (DL - DV)}$. При условии, что время доставки файлов каждого подзадания много меньше его выполнения, требования к производительности компьютеров снижаются в *M* раз.

Эффективность декомпозиции и параллельного выполнения заданий иллюстрирует таблица 1. В первой строке приведены данные по обработке исходного пакета заданий, во второй — пакета, который получается из исходного путём разбиения заданий на подзадания одинаковой длины. Количество подзаданий выбирается таким образом, чтобы каждое могло выполниться на компьютерах с минимальной производительностью (в данном случае 0.2). Приводятся стандартные показатели планирования — время счёта пакета заданий и степень загрузки ресурсов, а также количество отказов. Как видно из этих данных, пакет с разбиением на подзадания обрабатывается с существенно меньшим числом отказов за меньшее время (оно близко к минимальному). Эффект объясняется более высокой степенью загрузки ресурсов. Выигрыш будет ещё более ощутим для пула, который содержит большое количество слабых компьютеров.

Таблица 1.

Пакет	**Минимальное время счёта пакета**	**Время счёта пакета**	**Загрузка ресурсов**	**Число отказов**
Задания	7160	7971	43.6	556
Разбиение на подзадания	7160	7637	91.2	48

Результаты моделирования позволяют сделать вывод, что учёт эффективной производительности является необходимым аспектом планирования в гриде с неотчуждаемыми компьютерами, обеспечивающим предсказуемость обработки заданий.

Глава 4

Программная реализация системы диспетчеризации

Настоящая глава посвящена описанию реализации системы диспетчеризации для одноуровневого грида с неотчуждаемыми компьютерами, в основу которой положены теоретические результаты, изложенные во второй и третьей главах. В главе приводится архитектура системы SARD (StandAlone Resource Dispatcher), включающая в себя три основных компонента: диспетчер, агент на исполнительном компьютере и пользовательский интерфейс. Далее в главе рассматривается реализация компонентов предлагаемой системы.

4.1. Состав и функции системы SARD

В соответствии с предложенной во второй главе архитектурой системы диспетчеризации реализованы основные компоненты диспетчера, агент и пользовательский интерфейс. В составе диспетчера разработаны службы управления заданиями, управления исполнительными компьютерами и планирования.

При разработке SARD использован ряд компонентов распространённых систем распределённого компьютинга с учётом того, что они реализуют необходимую функциональность и выполнены на приемлемом для рассматриваемой постановки задачи уровне. С помощью этих компонентов в системе реализована функциональность управления заданиями на исполнительном компьютере, передачи файлов на

исполнительный компьютер, а также решён вопрос интероперабельности SARD с существующими грид-системами.

4.2. Стороннее программное обеспечение базового уровня

В настоящем разделе рассмотрим, какие функции SARD реализованы с помощью стороннего программного обеспечения, а также решения по использованию этого программного обеспечения в разработанной системе.

4.2.1. Использование системы Condor

В реализации SARD для управления заданиями на исполнительных компьютерах используется система Condor, анализ которой был произведён в разделе 1.3.3, а оценка применимости этой системы для решения поставленной в диссертационной работе задачи была дана в разделе 1.5.

В рамках реализации SARD система Condor обеспечивает выполнение функций по обработке задания на исполнительном компьютере: распределение локальных ресурсов компьютера, доставку всех необходимых файлов с машины диспетчера на исполнительный компьютер, а также доставку результата выполнения заданий с исполнительного компьютера на машину диспетчера. Для того, чтобы использовать указанные возможности системы Condor, на машине диспетчера устанавливаются компоненты центрального менеджера и машины запуска, а на исполнительных компьютерах — компонент исполнительной машины Condor.

Архитектура системы диспетчеризации с компонентами системы Condor представлена на рис. 12.

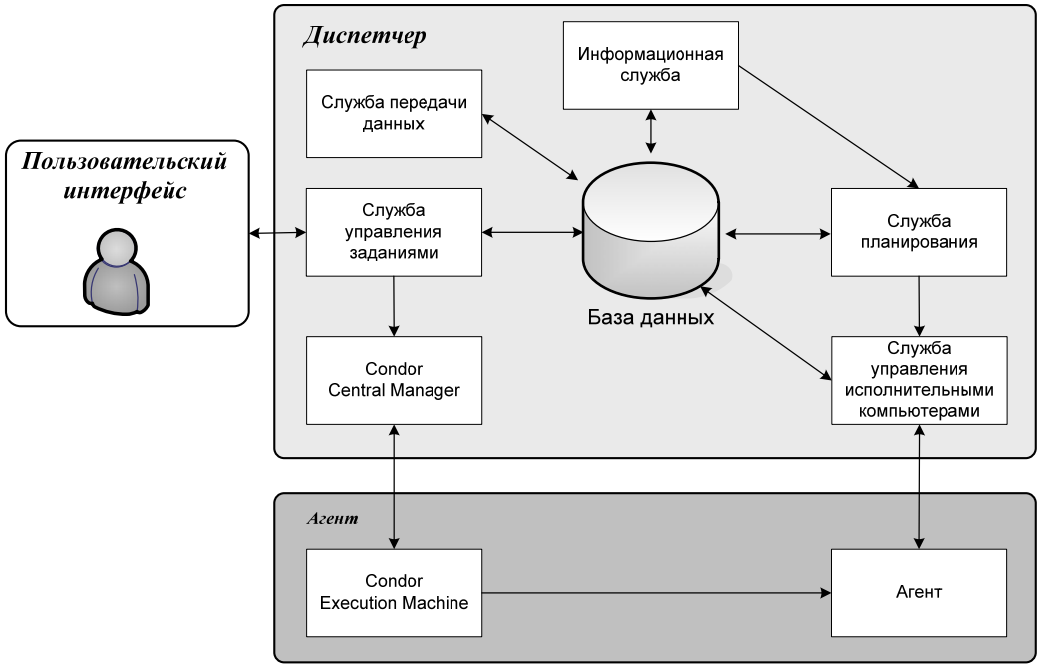


Рис. 12. Архитектура с интеграцией компонентов системы Condor

4.2.2. Использование базовых служб Globus Toolkit 4

Программный инструментарий Globus Toolkit 4 был выбран по нескольким причинам. Во-первых, использование инструментария позволяет построить систему, интероперабельную с существующими разработками, которые также используют GT4. Во-вторых, инструментарий даёт возможность воспользоваться готовыми и отлаженными службами базового уровня, обеспечивающими, в частности, безопасность. И, в-третьих, для функционирования служб диспетчера системы SARD был использован контейнер GT4.

Далее в работе будут рассмотрены все три архитектурных компонента системы диспетчеризации и их функции.

4.3. Диспетчер одноуровневого грида

Как отмечалось ранее, диспетчер одноуровневого грида представляется набором веб-служб, которые выполняются в контейнере Globus Toolkit 4. Настоящий раздел посвящён рассмотрению деталей реализации служб,

входящих в состав диспетчера системы SARD, их функций, а также механизмов взаимодействия между собой и другими компонентами системы.

Для хранения информации о пользователях, заданиях, исполнительных компьютерах, а также необходимой служебной информации используется база данных под управлением СУБД PostgreSQL [47]. Структура базы данных диспетчера приведена в Приложении 1.

4.3.1. Служба взаимодействия с агентами

В разделе 2.2.2 для службы взаимодействия с агентами, расположенными на исполнительных компьютерах, были перечислены функции, в соответствии с которыми выделены следующие типы сообщений взаимодействия агента с диспетчером через указанную службу:

- регистрация;
- снятие с регистрации;
- подключение;
- периодический отчёт;
- отключение.

Рассмотрим обработку каждого из указанных типов сообщений.

Регистрация исполнительного компьютера. Вновь подключаемый к инфраструктуре исполнительный компьютер должен быть зарегистрирован в системе. Для этого агент посылает диспетчеру соответствующее сообщение, содержащее информацию о производительности компьютера. После получения такого сообщения диспетчер заносит в базу данных полученную информацию и возвращает агенту код результата и идентификатор, присвоенный новому компьютеру. Если код результата свидетельствует об успешной регистрации компьютера, агент записывает идентификатор в файл и использует его при последующем взаимодействии со службой.

Снятие компьютера с регистрации. Если владелец решает исключить свой компьютер из инфраструктуры, он выполняет команду снятия исполнительного компьютера с регистрации, по которой агент

отправляет диспетчеру соответствующий запрос, содержащий идентификатор компьютера. Во время обработки этого запроса диспетчер удаляет запись о компьютере из базы данных и возвращает агенту код результата.

Подключение исполнительного компьютера. Получение данного сообщения информирует диспетчер о старте агента на зарегистрированном исполнительном компьютере. Перед отправкой этого сообщения, как и во время регистрации, агент собирает информацию о характеристиках компьютера и отправляет диспетчеру. Получив такое сообщение, диспетчер обновляет информацию о компьютере в базе данных и считает его доступным. В ответ служба сообщает агенту интервал времени, через который он должен отправить очередной отчёт.

Периодический отчёт. Получив от диспетчера интервал времени для посылки очередного отчёта, агент начинает сбор информации о функционировании компьютера. В состав отчёта входит идентификатор компьютера, описываемый интервал времени и информация о доле свободного процессорного времени за указанный интервал (среднее значение). Если на компьютере выполняются задания грида, в отчёт также включается его состояние, время выполнения на компьютере и количество потреблённого заданием процессорного времени. Задание грида может находиться в одном из четырёх состояний: запущено, выполняется, завершилось и завершилось аварийно. В ответ агенту возвращается интервал до следующего отчёта.

Отключение исполнительного компьютера. При штатном завершении агента он посылает диспетчеру сообщение об отключении исполнительного компьютера с указанием его идентификатора. Данное сообщение также содержит отчёт о функционировании компьютера за интервал с момента отправки последнего отчёта. Получив такое сообщение, диспетчер считает соответствующий компьютер недоступным, а выполняющиеся на нём задания грида (в случае наличия таковых) аварийно

завершившимися. В случае успешной обработки данного сообщения агент получает ответ, не содержащий информации.

Исключительные ситуации. Во время работы системы могут возникать различные исключительные ситуации, в этом случае протокол сетевого взаимодействия различных компонентов должен обеспечивать приведение системы в консистентное состояние. При взаимодействии агента с диспетчером могут возникать следующие ситуации.

Диспетчер недоступен при запуске агента. В такой ситуации агент будет повторять попытки отправки сообщения через определённые промежутки времени в надежде на возобновление работы диспетчера.

Сбой в работе агента. В случае возникновения ошибок в работе агента он перестаёт посылать периодические отчёты. Если через указанный в последнем отчёте интервал времени диспетчер не получит от агента очередной отчёт, компьютер будет считаться недоступным.

Сетевые проблемы между диспетчером и агентом. В отличие от предыдущей ситуации работа агента не прекращается, однако диспетчер перестаёт получать отчёты от агента. В такой ситуации диспетчер будет считать исполнительный компьютер недоступным. Не получая ответы на отправленные отчёты, агент также будет считать себя недоступным и будет предпринимать попытки подключения к диспетчеру.

Перезагрузка диспетчера. В такой ситуации диспетчер считает исполнительный компьютер недоступным, в то время как сам агент считает себя доступным. Данная проблема разрешается во время очередного отчёта. Получив такой отчёт от недоступного компьютера, диспетчер возвратит агенту соответствующий код ошибки. После этого агент будет считать себя еще не подключившимся и направит запрос на подключение.

Проблема с базой данных. В случае проблем с базой данных диспетчер может потерять информацию о зарегистрированном компьютере. В этом случае при обращении к диспетчеру агент получит соответствующий код ошибки и автоматически отправит запрос на регистрацию.

4.3.2. Служба управления заданиями

Во время обработки задание может находиться в одном из 12 состояний (рис. 13). Для наглядности на диаграмме не обозначены переходы между состояниями в случае перепланирования (подробнее об этой ситуации будет рассказано ниже).

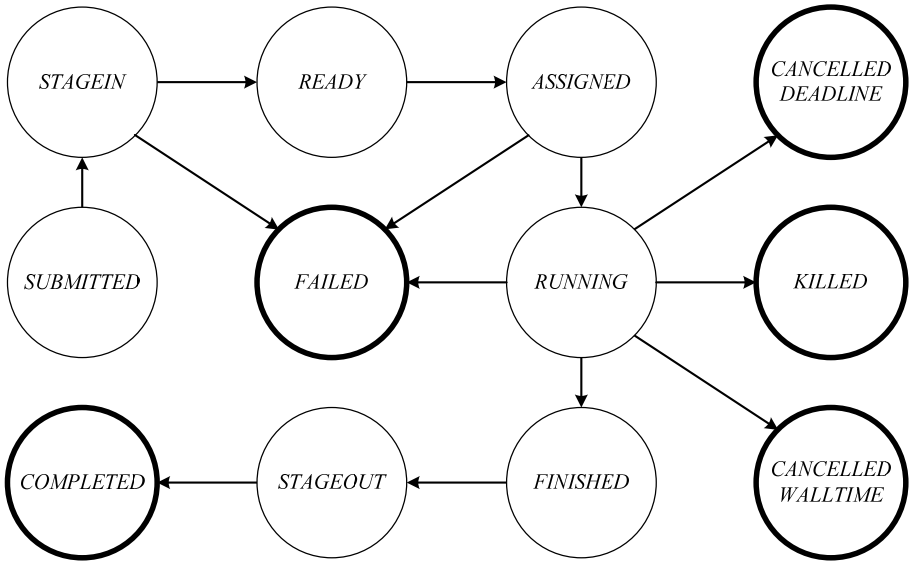


Рис. 13. Диаграмма состояний задания в системе диспетчеризации (жирной линией обозначены конечные состояния)

Управление заданием в системе представлено тремя функциями: ввести новое задание в систему, получить статус задания и отменить незавершённое задание. Рассмотрим более подробно каждый из трёх сценариев.

4.3.2.1. Процесс обработки задания

При вводе нового задания пользователь подготавливает его описание в формате XML и передаёт службе через пользовательский интерфейс. Если описание корректно, из него извлекается необходимая информация, которая помещается в соответствующие таблицы базы данных диспетчера. При этом задание получает статус *SUBMITTED*, а пользователю возвращается уникальный идентификатор задания. В противном случае информация о

задании не попадает в систему, а пользователю возвращается сообщение об ошибке.

После успешного ввода задания в систему начинается процесс доставки необходимых входных данных с внешних хранилищ на компьютер диспетчера, о чём свидетельствует состояние *STAGEIN*. Данный процесс осуществляется службой передачи файлов и выделен явным образом потому, что доставка файлов может занимать продолжительное время, а отдельный статус помогает понять, что именно происходит с заданием и не начинать планирование этого задания пока не будет доставлен весь набор необходимых файлов. После завершения процесса доставки служба передачи файлов уведомляет об этом службу управления заданиями, которая меняет статус задания на *READY* и информирует планировщик о том, что задание готово к обработке и распределению на исполнительный компьютер. Таким образом, задание попадает в очередь планировщика только после того, как доставка данных для него будет полностью завершена. Если в процессе доставки возникают ошибки, и не все файлы могут быть доставлены на машину диспетчера, задание переходит в состояние *FAILED*, и его дальнейшая обработка в системе прекращается.

В случае корректной доставки всех необходимых заданию файлов планировщик осуществляет поиск наиболее подходящего компьютера для его выполнения. Когда планировщик принимает решение о запуске задания на одном из доступных исполнительных компьютеров, им вызывается соответствующий метод службы управления заданиями, которая в свою очередь заносит необходимую информацию в таблицу назначений и меняет статус задания на *ASSIGNED*, означающее начало запуска задания, включая доставку необходимых файлов на исполнительный компьютер. После того как на компьютер было назначено задание, он считается занятым и не учитывается в дальнейшем планировании.

Заметим, что наличие такого состояния особенно важно в условиях использования системы Condor (или любой другой сторонней системы), так

как диспетчер фактически запускает задание в эту систему, и процесс запуска может затянуться по неизвестным для диспетчера причинам. В частности, задания могут «зависать» в очереди Condor, поэтому такие события должны быть корректно обработаны диспетчером. При обнаружении случаев зависания задание снимается с выполнения и возвращается в очередь на перепланирование.

В соответствии с полученными назначениями осуществляется запуск заданий на выполнение средствами системы Condor. Запуск задания непосредственно на исполнительный компьютер осуществляется с помощью программы *condor_submit*, входящей в поставку программного обеспечения Condor. На вход эта программа получает описание задания в специальном формате, которое генерируется на основе данных из таблицы заданий и таблицы передачи данных. В случае успешного запуска на выходе *condor_submit* возвращает внутренний идентификатор задания в системе Condor, который сохраняется в таблице назначений и используется впоследствии для сопоставления с уникальным идентификатором задания в рамках системы диспетчеризации. Следует заметить, что по умолчанию во время запуска Condor самостоятельно осуществляет процесс поиска подходящего компьютера, поэтому для запуска задания на тот компьютер, который был определён планировщиком системы диспетчеризации, необходимо явно указать это в описании задания. Такая возможность в Condor предусмотрена и реализуется добавлением в описание задания специального требования *Requirements = Machine==<hostname>*, где *<hostname>* — имя исполнительного компьютера, назначенного для выполнения задания планировщиком.

После запуска задания с помощью *condor_submit* задание попадает в очередь системы Condor, но, несмотря на то, что компьютер готов принять новое задание, по разным причинам оно может простоять некоторое время в очереди. Учитывая такое поведение системы Condor, служба взаимодействия с агентами сообщает о непосредственном старте задания на исполнительном

компьютере службе управления и продолжает на протяжении всего времени выполнения задания регулярно передавать полученную от агента информацию. При этом задание переходит в состояние *RUNNING*.

В случае возникновения различного рода ошибок во время старта или во время выполнения задания на компьютере, оно снимается и получает статус *FAILED*. Если же во время выполнения задания исполнительный компьютер по какой-либо причине отключается от системы диспетчеризации, задание должно быть перезапущено на другом компьютере. В этом случае задание возвращается в состояние *READY* и будет повторно обработано планировщиком.

После получения сообщения от службы взаимодействия с агентами о завершении выполнения задания на исполнительном компьютере, оно помещается в список заданий, ожидающих окончания обработки системой Condor. После фактического завершения выполнения задания может пройти некоторое время, пока Condor будет считать этот компьютер свободным, поэтому служба управления заданиями периодически запрашивает информацию о статусе заданий Condor с помощью *condor_status*. Как только системой Condor подтверждается завершение задания, оно переходит в состояние *FINISHED*. Исполнительный компьютер освобождается, а задание переходит в состояние *STAGEOUT*, во время которого осуществляется доставка результирующих файлов с машины диспетчера на внешние хранилища, если это было указано пользователем в описании задания. Конечное состояние (все конечные состояния обозначены на диаграмме жирной линией) *COMPLETED* означает успешную доставку результирующих файлов и окончательное завершение обработки задания.

Другими конечными состояниями являются *FAILED*, в которое задание переходит в случае возникновения различного рода ошибок во время планирования, выполнения на компьютере или доставки файлов, а также состояния *KILLED*, *CANCELLED_WALLTIME* и *CANCELLED_DEADLINE*. Последние два состояния означают принудительную отмену диспетчером

выполнения задания на исполнительном компьютере по причине превышения заявленного максимального времени выполнения (*maxWallTime*) и предельного срока исполнения (*deadline*) соответственно. В случае превышения этих параметров планировщик снимает задание с выполнения, которое получает статус *KILLED*. Такой же статус присваивается заданию в случае отмены его пользователем.

4.3.2.2. Получение статуса и отмена задания

Ранее отмечалось, что функции получения статуса и отмены введённого в систему задания доступны только его владельцу.

В результате запроса пользователя о статусе задания, ему возвращается следующая информация, сохранённая в таблице заданий:

- статус задания;
- время ввода задания;
- время запуска (если задание уже было запущено);
- астрономическое и процессорное время выполнения задания (если задание уже было запущено);
- сообщение о последней ошибке (если такая была).

Если задание было введено в систему и ещё не было завершено каким-либо образом — успешно, из-за ошибки, вследствие отмены пользователем или системой и т.д., то запрос на отмену задания прекращает его выполнение.

Для отмены запущенного задания используется программа *condor_rm*, входящая в пакет поставки системы Condor. На вход эта программа получает внутренний (для Condor) идентификатор задания, выполнение которого следует прекратить. Внутренний идентификатор извлекается из базы данных по соответствующему идентификатору задания, полученному от пользователя (или планировщика). Задание при этом завершается принудительно без осуществления доставки результирующих файлов.

4.3.3. Служба передачи и хранения данных

Учитывая, что в текущей реализации системы доставку всех необходимых заданию файлов с машины диспетчера на исполнительный компьютер и доставку результата с исполнительного компьютера обратно на машину диспетчера осуществляет система Condor, передача файлов происходит в два этапа. Сначала осуществляется доставка файлов задания и результатов вычислений между внешними хранилищами и машиной диспетчера — *внешняя доставка*, а затем система Condor производит *внутреннюю доставку*, то есть передачу файлов между диспетчером и исполнительным компьютером. Файлы задания доставляются на машину диспетчера SARD в виду особенностей системы Condor, которая требует наличия на машине запуска всех необходимых заданию файлов перед его запуском с помощью программы *condor_submit*.

Для обеспечения внешней доставки разработана соответствующая служба — служба передачи и хранения данных, в которой для передачи файлов используется протокол GridFTP, являющийся де-факто стандартным средством передачи файлов в гриде. Рассматриваемая служба осуществляет временное хранение всех файлов до завершения выполнения задания. Если задание не может быть завершено на отведённом компьютере (задание получает недостаточно процессорного времени, или компьютер выключили), необходим весь этот набор файлов для перезапуска задания на другом компьютере. Таким образом, службой передачи и хранения данных реализуются следующие функции:

- доставка стандартных файлов задания и связанных с ним прикладных файлов с внешних серверов хранения на машину диспетчера;
- временное хранение всех необходимых заданию файлов до его завершения;
- доставка результата по адресу, который указан в описании задания.

В описании задания содержатся блоки, определяющие какие файлы и куда необходимо доставить. При этом для доставки входных файлов адрес источника — это абсолютный URL файла, а адрес назначения — это относительный путь в директории задания. Для доставки результирующих файлов в качестве источника используется относительный путь к файлам в директории задания на машине диспетчера, а доставляются они по абсолютному URL на внешний сервер хранения. При этом независимо от того, как располагались входные файлы на внешних серверах или как располагались результирующие файлы в директории задания, указание абсолютных URL делает возможным во время доставки файлов создавать требуемую иерархию файлов, а также переименовывать файлы.

В случае возникновения ошибок во время передачи данных система может поступать двумя способами. Если ошибка возникает во время доставки входных файлов, то после нескольких попыток (неудачных) задание завершается со статусом *FAILED*. Это обусловлено предположением, что для корректного выполнения задания ему необходимы все файлы, обозначенные явным образом пользователем в описании задания. Если же во время доставки результирующих файлов система не может обнаружить того или иного файла, отражённого в описании задания, то ошибка не генерируется, и осуществляется доставка остальных файлов. Такое поведение системы продиктовано предположением о том, что в зависимости от алгоритма работы приложения оно может генерировать или нет те или иные файлы, что является её штатным поведением. Несмотря на это, пользователь информируется о таких ситуациях и получает список не доставленных результирующих файлов.

4.4. Агент на исполнительном компьютере

В комплект агентского программного обеспечения входят собственно агент, компонент системы Condor, устанавливаемый на исполнительной

машине, а также специальная утилита — лидер, которая сообщает агенту информацию о фактическом запуске задания. Средства системы Condor позволяют вместо запуска задания осуществлять запуск произвольного приложения, указанного в конфигурационном файле исполнительной машины. Эта возможность используется в SARD для того, чтобы агент получил информацию о задании грида. Когда компонент системы Condor получает новое задание, он запускает программу-лидер, которая в свою очередь осуществляет запуск задания. При этом лидер сообщает агенту идентификатор запустившегося процесса и идентификатор задания в базе данных диспетчера. Программное обеспечение Condor на исполнительном компьютере разделено на несколько *демонов*. Список всех демонов находится в конфигурационном файле и может быть расширен. Во время запуска системы Condor стартует главный демон системы — *master*. В свою очередь этот демон запускает остальные демоны и следит за тем, чтобы они всегда были запущены. В том случае, если какой-то из демонов умирает, master-демон его автоматически перезапускает. Данный механизм был использован для обеспечения одновременного старта и завершения Condor и агента. Для этого агент был оформлен в виде демона системы Condor и добавлен в список демонов в конфигурационном файле.

4.4.1. Архитектура агента

Как отмечалось ранее, функции по управлению заданием на исполнительном компьютере, а также доставка входных и результирующих файлов между машиной диспетчера и исполнительным компьютером в SARD возложены на компоненты системы Condor. В дополнение к этому, в агенте реализованы функции по регистрации (и снятию с регистрации) исполнительных компьютеров в системе диспетчеризации SARD, сбору информации о потреблении ресурсов компьютера, а также отправке периодических отчётов с собранной информацией, сообщениями о

функционировании самого агента и событиями, связанными с изменением статуса задания грида.

Агент имеет модульную архитектуру, состоящую из трёх слоев: ядра, слоя взаимодействия с системой и оболочки агента. Данный способ организации кода позволяет эффективно разделить кроссплатформенные и системно-зависимые компоненты агента, облегчив тем самым создание версий агента для различных платформ, а также модернизацию агента. Также архитектура агента позволяет в перспективе произвести замену заимствованных компонентов на компоненты собственной реализации.

4.4.1.1. Ядро агента

Основной частью агента является кроссплатформенное ядро. Задача ядра состоит в обработке следующих событий:

- инициализация агента (Init);
- регистрация компьютера (Register);
- снятие компьютера с регистрации (Unregister);
- подключение агента (Connect);
- отключение агента (Disconnect);
- обновление состояния компьютера (Refresh);
- запуск нового задания (NewJob).

Метод инициализации вызывается при запуске агента. Результатом выполнения этого метода является инициализация и приведение в рабочее состояние компонентов агента.

События регистрации, снятия с регистрации, подключения и отключения инициируют отправку соответствующих запросов диспетчеру (если это возможно в текущем состоянии агента), и при получении ответов, свидетельствующих об успешном выполнении требуемых операций, агент изменяет своё состояние (зарегистрирован/не зарегистрирован, подключён/отключён).

Оболочка агента во время его работы периодически инициирует событие обновления состояния. В ходе обработки этого события происходит сбор необходимых данных о доле свободного процессорного времени и состоянии выполняющихся заданий. В случае необходимости осуществляется отправка диспетчеру периодических отчётов о состоянии исполнительного компьютера.

Событие запуска нового задания грида позволяет агенту начать слежение за ним.

4.4.1.2. Слой взаимодействия с системой

Во время работы кроссплатформенное ядро агента обращается к коду, реализующему системно-зависимые функции, такие, как сбор характеристик компьютера или слежение за выполнением задания. Этот код вынесен в слой взаимодействия с системой, при этом определены независимые от платформы интерфейсы компонентов. Таким образом, код ядра зависит только от этих интерфейсов, а не от системно-зависимого кода реализации компонентов. Фактически экземпляры объектов создаются системно-зависимым кодом оболочки и передаются ядру при инициализации агента. Такое решение позволяет не менять код ядра при создании модификаций агента для различных платформ.

Компоненты слоя взаимодействия с системой выполняют следующие функции:

- хранение информации о конфигурации агента, как статической (адрес диспетчера), так и меняющейся (идентификатор исполнительного компьютера, получаемый при регистрации);
- сбор информации о характеристиках исполнительного компьютера, инициируемый ядром при запуске агента;
- выполнение теста производительности для определения остальных характеристик компьютера, в результате работы которого

определяется производительность процессора в миллионах операций с плавающей точкой в секунду (Mflops);

- мониторинг загруженности центрального процессора и получение информации о средней доле свободного процессорного времени, которые передаются диспетчеру в периодических отчётах.
- слежение за выполняющимся заданием грида, позволяющее получать информацию об объёме потреблённого заданием процессорного времени, а также обнаруживать факт завершения задания;

Отдельный компонент слоя взаимодействия с системой осуществляет инкапсуляцию кода, отвечающего за посылку сообщений диспетчеру и получение ответов от него.

4.4.1.3. Оболочка агента

Оболочка представляет собой системно-зависимый компонент агента, генерирующий события, которые затем обрабатываются ядром. Помимо этого, оболочка отвечает за создание экземпляров объектов конкретных системно-зависимых компонентов и их передачу в ядро.

Вызывая методы ядра, оболочка инициирует различные типы описанных ранее событий. Регистрация и снятие с регистрации компьютера происходят при запуске агента с указанием соответствующих параметров командной строки. При обычном запуске агента на зарегистрированном исполнительном компьютере происходит попытка подключения к диспетчеру. В случае неудачной попытки (например, из-за недоступности диспетчера), оболочка инициирует повторные попытки через определённые интервалы времени. Периодически во время выполнения происходит обновление состояния. Получив от средств запуска задания сообщение о запуске нового задания, оболочка передаёт его ядру.

При получении от операционной системы запроса на завершение программы происходит передача диспетчеру запроса на отключение агента.

4.4.2. Реализация функций агента

Рассмотрим более подробно особенности реализации агентской части системы диспетчеризации.

4.4.2.1. Хранение конфигурационной информации

Конфигурационная информация хранится в файле *agent.config*, который считывается при запуске агента. Наиболее важным и обязательным параметром, расположенным в этом файле, является адрес службы диспетчера для управления агентами.

Как отмечалось ранее, после успешной регистрации исполнительного компьютера агент записывает присвоенный этому компьютеру идентификатор в файл. При запуске агент пытается считать информацию из этого файла. При удачном считывании идентификатора исполнительный компьютер считается зарегистрированным. Если данный файл отсутствует, компьютер считается незарегистрированным. Нормальная работа агента в этом случае невозможна.

Для регистрации компьютера в системе необходим запуск агента со специальным параметром командной строки «*–register*». Для снятия исполнительного компьютера с регистрации агент должен быть запущен с параметром «*–unregister*», при этом произойдет передача диспетчеру запроса на снятие с регистрации. При удачном снятии с регистрации файл, содержащий идентификатор компьютера, удаляется.

4.4.2.2. Сбор характеристик компьютера

Для сбора информации о характеристиках компьютера используются стандартные функции операционной системы. В реализации агента для платформы Windows для сбора характеристик используются функции GetSystemInfo, GetComputerName, GetVersionEx и GlobalMemoryStatus. Результаты выполнения этих функций преобразуются во внутренний формат хранения и передаются ядру агента для отправки диспетчеру.

4.4.2.3. Тест производительности центрального процессора

Компонент агента, отвечающий за получение информации о производительности процессора путём выполнения теста производительности, реализован на основе известного теста LINPACK [49]. Этот тест состоит в решении методом Гаусса системы линейных алгебраических уравнений с матрицей заданного размера, заполненной случайно сгенерированными коэффициентами. Для заданного размера матрицы определяется число операций с плавающей точкой, необходимых для решения системы. Полученное число делится на процессорное время, затраченное на выполнение теста, что даёт производительность процессора в миллионах операций с плавающей точкой в секунду (Mflops). Возможная погрешность, связанная с неполным использованием процессора при выполнении теста, при данном способе измерения учтена, так как при выполнении теста измеряется затраченное процессорное время, а не общее время выполнения теста.

4.4.2.4. Слежение за выполняющимися заданиями

В контексте разделения ресурсов исполнительного компьютера основной функцией агента является сбор информации о выполняющихся заданиях грида. При решении данной задачи важной проблемой является отслеживание дочерних процессов, порождаемых во время выполнения основного процесса задания. Кроме того, важной является и возможность ограничения количества потребляемых заданием ресурсов.

Для решения указанных задач на исполнительных компьютерах под управлением ОС семейства Windows был выбран API Job Object [50], предоставляющий средства для отслеживания выполнения группы процессов. Соответствующий объект ядра — задание (job) — был введён в ОС Windows, начиная с версии Windows 2000. При использовании этого средства необходимые процессы задания группируются и помещаются в «песочницу», за которой операционная система следит самостоятельно.

Чтобы избежать путаницы в терминологии, под «заданием» будем понимать задание грида, а задание в терминах операционной системы Windows будем называть объектом-заданием.

В случае порождения дочернего процесса каким-либо процессом из объекта-задания, он автоматически добавляется в песочницу. Таким образом, слежение осуществляется за всеми порождающимися процессами без опаски потери таковых даже в том случае, если родительский процесс завершается раньше дочернего процесса.

Использование объектов-заданий позволяет при его создании накладывать различные ограничения на совокупность входящих в него процессов. В первую очередь это касается объёма выделяемой оперативной памяти и потребляемого процессорного времени. Кроме того, появляется возможность защиты процессов и данных владельца компьютера путём ограничения доступа к защищённым ресурсам (файлам, подразделам реестра и т.п.) для всех процессов объекта-задания. Также преимуществом данного способа отслеживания процессов перед другими (когда отслеживаемые процессы независимы, и для контроля над ними периодически осуществляется перестроение дерева процессов) является сравнительно небольшое число системных вызовов, необходимое для считывания информации о потреблённых процессами системных ресурсах.

При получении запроса на запуск задания грида агент создаёт экземпляр объекта-задания и приостановленный экземпляр процесса задания. Затем созданный экземпляр процесса задания добавляется к объекту-заданию, и его выполнение возобновляется. Включение процесса в приостановленном состоянии предотвращает нарушение ограничений, наложенных на объект-задание.

Во время выполнения задания грида ядро агента периодически обращается к компоненту слежения за процессами, обновляя информацию об объекте-задании (общее потреблённое процессорное время, а также текущее число процессов в объекте-задании). По этой информации вычисляется

затраченное процессорное время (перерасчёт на секунды), общее время работы задания и статус задания. Задание грида считается завершившимся, если в объекте-задании не осталось ни одного выполняющегося процесса.

4.4.2.5. Взаимодействие с диспетчером

Как упоминалось ранее, взаимодействие агента с диспетчером осуществляется с помощью передачи сообщений соответствующей веб-службе диспетчера и получения от неё ответов. Формат передаваемых сообщений определяется протоколом SOAP [51]. Со стороны веб-служб поддержка SOAP реализована в инструментарий Globus Toolkit 4, для поддержки этого протокола на стороне агента применяется кроссплатформенная библиотека gSOAP [52].

Для интеграции с диспетчером используется описание веб-службы диспетчера на языке WSDL. Данное описание подаётся на вход программе кодогенерации, входящей в состав библиотеки gSOAP. Результатом работы этой программы является код на языке C++, реализующий взаимодействие со службой.

4.5. Пользовательский интерфейс

Взаимодействие пользователя с клиентским приложением происходит через интерфейс командной строки. Заметим, что при реализации пользовательского приложения сохранены набор операций и их названия, соответствующие клиентскому приложению службы GRAM из состава инструментария Globus Toolkit 4 — утилите *globusrun-ws* [53]. Определение операндов каждой операции также осуществляется с помощью параметров командной строки, аналогичных *globusrun-ws*.

Для того, чтобы осуществить ввод нового задания, пользователю необходимо подготовить описание задания на языке RSL. Используемый в разработанной системе формат основан на формате описания заданий, поддерживаемый службой GRAM, и дополнительно содержит некоторые

параметры, необходимые для обеспечения планирования в одноуровневом гриде. Формат файла описания задания приведён в Приложении 2.

Ввод задания осуществляется выполнением команды:

```
$ sard_run –submit –f <job_description_file> [–c <proxy_cert>]
<service>
```

В данном случае приложение выполняет операцию ввода задания (*–submit*), обращаясь к службе управления заданиями по её URL <service>, и получает на вход файл описания задания <job_description_file>. Необязательный параметр «*–c*» позволяет указать директорию размещения прокси-сертификата пользователя, отличную от директории по умолчанию.

После осуществления синтаксической проверки описания задания клиентское приложение производит делегирование полномочий пользователя системе диспетчеризации с помощью службы делегирования из состава Globus Toolkit 4 (Delegation Service [54]). Делегирование полномочий является стандартным механизмом, применяющимся для того, чтобы диспетчер мог впоследствии осуществить доставку входных и результирующих файлов с правами пользователя. Если описание составлено корректно, осуществляется регистрация задания в системе, после чего пользователю возвращается уникальный идентификатор, который он затем может использовать для управления заданием. Заметим, что для проверки описания без фактического ввода задания можно воспользоваться параметром «*–validate*», что позволяет облегчить процесс поиска ошибок в описании. Для этого необходимо запустить приложение следующим образом:

```
$ sard_run –validate –f < job_description_file>
```

Для управления заданием используются операции *–kill* и *–status*, позволяющие соответственно отменить выполнение задания или получить информацию о состоянии задания. Отмена задания осуществляется запуском

пользовательского приложения с указанием уникального идентификатора (параметр «*–i*») задания:

```
$ sard_run –kill –i <job_id> [–c <proxy_cert>] <service>
```

После успешной отмены задания оно переходит в конечное состояние *KILLED*.

Операция получения статуса задания (*–status*) позволяет пользователю узнать состояние задания, а также получить некоторую дополнительную информацию, например, время ввода задания, время запуска задания на исполнительном компьютере, астрономическое и процессорное время выполнения задания, а также сообщение о последней ошибке, если такая имела место. Получить перечисленную выше информацию можно с помощью команды:

```
$ sard_run –status –i <job_id> [–c <proxy_cert>] <service>
```

Вывод команды имеет следующий вид:

```
$ sard_run https://sard:8443/wsrf/services/SubmitService –status
–i 260
Getting status for job 260 ...
Status: RUNNING
Submitted: Thu Mar 25 15:57:20 MSK 2010
Started: Thu Mar 25 15:57:39 MSK 2010
Wall: 76
Cpu: 76
Log is empty
```

Глава 5

Практические применения системы SARD

В настоящей главе описывается применение разработанной системы для решения практически важных вычислительных задач.

Наиболее эффективно рассматриваемые ресурсы используются при расчёте сериализуемых приложений. Класс таких приложений довольно широк и включает в себя множество задач, решаемых на основе общепринятых вычислительных методов, таких как моделирование методом Монте-Карло, или поиск оптимального значения параметров в серии вычислительных экспериментов. Далее в главе рассматриваются две задачи подобного типа, по которым проведены расчёты в управляемой системой SARD вычислительной инфраструктуре Института прикладной математики им. М.В. Келдыша РАН.

Первая задача состоит в оценке влияния поражающих факторов проникающего излучения на человека и функционирование аппаратуры путём вычисления энергии, поглощенной в материалах объектов. Вторая задача заключается в расчёте модели электромагнитного ускорения и торможения лайнера в магнитном компрессоре, а также определении некоторых параметров ленты лайнера.

5.1. Задача пространственного распределения энергии ионизирующего излучения

Разработанная система диспетчеризации заданий SARD для одноуровневого грида была апробирована в Институте прикладной

математики им. М.В. Келдыша при решении задачи расчёта пространственного распределения энергии ионизирующего излучения, поглощённой в многокомпонентных объектах, облучаемых потоком гамма квантов.

Во многих практических задачах, связанных с функционированием аппаратуры и оборудования в полях ионизирующих излучений, важной является проблема защиты этих объектов от нагрева и других поражающих факторов проникающего излучения.

В большинстве случаев оценка влияния излучения сводится к вычислению энергии, поглощенной в материалах объектов («энерговыделение»). Соответствующие вопросы актуальны и для термостойкости компонентов ядерных реакторов, и для космических аппаратов.

Особое место в данной проблематике занимает задача оценки воздействия проникающих излучений на человека. В этой задаче важным является не только оценка суммарной энергии излучения, поглощенной в тканях (поглощенной дозы излучения), но и распределение дозы излучения в жизненно важных органах.

Модель взаимодействия излучения с материалами объектов строится на основе общепринятых представлений о механизмах поглощения и рассеяния гамма квантов в веществе. Основными типами взаимодействия в исследуемом энергетическом диапазоне (кванты до 1 МэВ) являются: комптоновское и когерентное рассеяние и фотопоглощение излучения. Поглощение энергии в веществе происходит в результате фотопоглощения и комптоновского рассеяния гамма квантов.

Для решения указанных задач построены эффективные статистические методы моделирования переноса гамма излучения в сложных многокомпонентных объектах на основе весовых модификаций метода Монте-Карло [55]. Использование алгоритмов статистического моделирования на основе экономичных модификаций метода Монте-Карло

[56], [57] является одним из наиболее эффективных для решения сложных граничных задач моделирования взаимодействия и трансформации ионизирующего излучения в многокомпонентных объектах сложной внутренней структуры.

5.1.1. Актуальность применения технологий грида

В рассматриваемом классе задач для обеспечения заданной точности обычно необходимо моделирование более миллиарда фотонных историй. Моделирование такого количества историй, при использовании современных компьютеров, требует больших временных затрат порядка 5–10 часов. В то же время, так как фотонные истории моделируются независимо, моделирование всех историй в рамках единственного запущенного приложения и моделирование всех историй частями в рамках многих запущенных приложений с последующим суммированием результатов приведут к идентичным результатам (в рамках статистической погрешности). В этой ситуации разделение «тяжёлой» задачи на несколько «лёгких» подзадач и их параллельное выполнение на множестве компьютеров позволяет кратно уменьшить время расчёта. Подобное ускорение может быть достигнуто и при использовании суперкомпьютеров, однако в этом случае понадобится соответствующим образом модифицировать программный код для поддержки параллельных вычислений.

5.1.2. Проведение расчётов на некластеризованных компьютерах

5.1.2.1. Описание расчётной задачи

С помощью разработанной системы диспетчеризации решена задача по расчёту распределения плотности потока инжектируемых электронов с внешней и внутренней поверхностей трубы.

На рис. 14 изображена схема модельного эксперимента, в котором алюминиевая труба радиуса R со стенками толщиной R/10 облучается источником гамма излучения S на основе изотопа Se^{75}, который широко

применяется в дефектоскопии. Расстояние от ближайшей к источнику точки на поверхности и угол раствора источника подобраны так, чтобы образующая конуса излучения, лежащая в плоскости zOy, была касательной к внешней поверхности трубы (рис. 14). Расположение совокупности точек T_k на внутренней и внешней поверхностях трубы, в которых рассчитываются характеристики электронных потоков, ясно из рис. 15. На этом рисунке изображена половина трубы для отрицательных значений координаты *x*, точки T_k отмечены белым цветом.

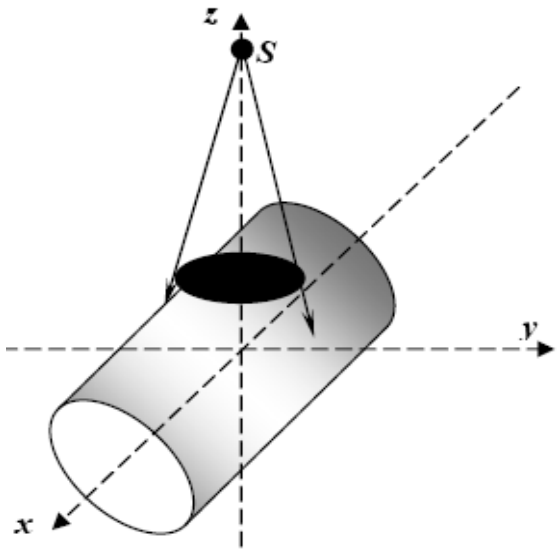


Рис. 14. Схема модельного эксперимента

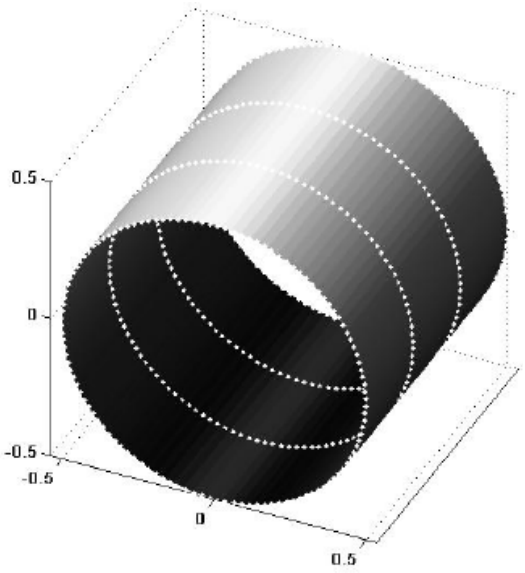


Рис. 15. Расположение «центров» детекторов (белые точки)

5.1.2.2. Описание программы расчёта

Построенные статистические методы моделирования переноса гамма излучения реализованы в виде отдельного приложения, которое может быть запущено практически на любом современном персональном компьютере под управлением операционной системы семейства Unix или MS Windows. Размер исполняемого файла не превышает 60 КБ вместе с файлами конфигурации. Входные данные зависят от каждой конкретной задачи, в нашем случае суммарный объём таких данных составил около 30 МБ.

Конфигурационный файл приложения состоит из набора блоков, в каждом из которых указываются имена файлов, содержащих описание материала, а также параметры источника или объекта.

Описание материала содержит таблицы некогерентного и когерентного углов рассеяния, а также коэффициентов поглощения. В отдельном входном файле содержится информация о спектре излучения источника.

Для каждого объекта, являющегося детектором, создаётся выходной файл, в котором хранится линеаризованный массив $X_{\text{resDet, resSrc, resH, resR, resA}}$. Каждый элемент этого массива задаётся пятью индексами *m, n, i, j, k*. Индексы *i, j, k* — «пространственные», они определяют пространственную ячейку, в которой поглотилась энергия (пространственная дискретизация для каждого объекта-детектора задаётся во входных данных "space res=%resH% %resR% %resA%"). Индексы *m, n* — «энергетические»: первый указывает на то, что энергия поглощалась порциями, относящимися к *m*-той ячейке (detector energies=%resDet% %detMinNrj% %detMaxNrj%) , а второй — то, что энергия поглощалась от фотона, с начальной энергией относящейся к *n*-той ячейке (source energies=%resSrc% %srcMinNrj% %srcMaxNrj%).

Полученный файл может быть использован для визуализации поглощённой объектом энергии.

5.1.3. Результаты расчётов

Проведение расчётов на инфраструктуре с некластеризованными компьютерами с помощью разработанной системы диспетчеризации позволило сократить время выполнения расчётной задачи в шесть раз. Благодаря этому на этапе проведения экспериментов удалось осуществить большее количество запусков, а расчёт реальных данных выполнить с большей точностью.

5.2. Расчёт модели движения лайнера в магнитном компрессоре

Построение многокомпонентных физических установок сопряжено с множеством трудностей, одной из которых является определение характеристик компонентов установки, таких как геометрические размеры, свойства материала деталей, электрические характеристики, компоновка узлов и т.д. Часто компоненты системы разрабатываются одновременно и независимо друг от друга, что не даёт возможности на ранних стадиях проверять в действии работу всей системы в целом и оценивать её показатели. Поэтому основным средством получения необходимой информации является математическое моделирование работы системы и проведение серии экспериментов по определению оптимальных параметров её компонентов.

Примером такого рода систем является установка «МОЛ» (Магнитный Обостритель Лайнер), предназначенная для исследования работы всех ступеней модуля установки «Байкал» [58] и генерации электрического импульса мегаджоульного уровня. Для этой установки в ГНЦ РФ ТРИНИТИ разработан макет усилительного каскада мощности — магнитный компрессор (МК), работа которого основана на сжатии магнитного потока лайнером, ускоренным электродинамическими силами до скорости 1 км/с. На рис. 16 и рис. 17 приведены элементы устройства МК и его схема соответственно.

Интегральные характеристики, полученные с использованием модели плоской (и абсолютно жёсткой) пластины лайнера в достаточной степени подтверждаются результатами практических экспериментов. Но одного этого соответствия недостаточно для оптимизации режимов работы электромагнитного компрессора, предназначенного для генерации коротких импульсов тока. Режим компрессии магнитного поля существенно зависит от геометрической формы зазора между пластинами в момент сжатия

магнитного поля. Для короткого генерируемого импульса отдача кинетической энергии тонкого лайнера должна проводиться одновременно по всей его плоскости, поэтому важно знать динамику деформирования пластины в процессе её ускорения, особенно на стадии электромагнитного торможения лайнера.

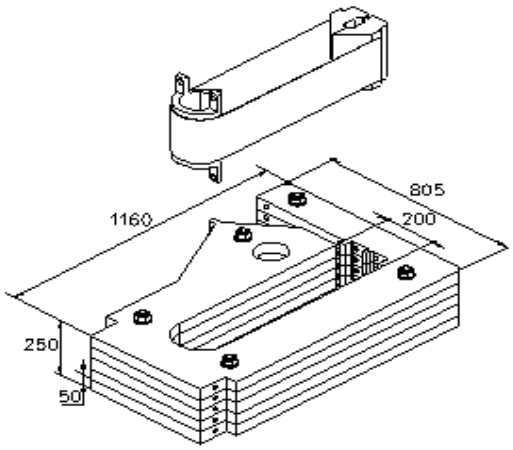


Рис. 16. Ускоряемая плоская лента с натяжным устройством

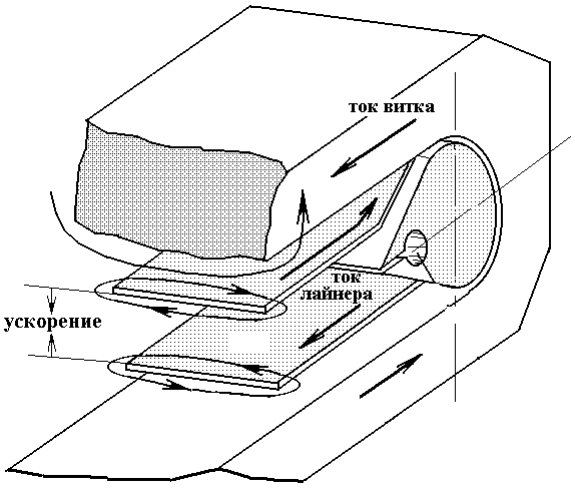


Рис. 17. Поперечное сечение генератора в собранном виде

Одним из ключевых вопросов, стоящих перед создателями установки, является определение такого набора входных параметров устройства, при котором генерируемый на выходе импульс имеет оптимальные характеристики. Для решения этой задачи в ИПМ им. М.В. Келдыша РАН были построены двумерные математические и численные модели, соответствующие поперечному и продольному сечениям магнитного компрессора, а также разработано программное обеспечение для моделирования движения лайнера.

В ГНЦ РФ ТРИНИТИ на макете МК была проведена серия экспериментальных запусков [59], однако из-за того, что процесс ускорения лайнера занимает короткий промежуток времени (порядка сотни микросекунд), а в результате торможения большая часть ленты уничтожается, имеющийся объём экспериментальных данных достаточно ограничен. Таким образом, математическое моделирование и

вычислительный эксперимент являются практически единственными инструментами для получения более подробной информации о движении лайнера в магнитном компрессоре.

5.2.1. Актуальность применения технологии грида

Для получения численных характеристик поведения лайнера с помощью математического моделирования разработан специальный программный комплекс. Входящие в состав комплекса программы позволяют получить характеристики движения ленты лайнера (поле перемещений и скоростей, распределение напряжений и деформаций и т.д.) в зависимости от её формы, материала, момента замыкания цепи и т.д. Чтобы решить поставленную задачу оптимизации входных параметров устройства для получения более мощного выходного импульса, необходимо провести ряд экспериментов, варьируя значение исследуемых параметров для каждой из математических моделей лайнера. Это сопряжено с большими временными затратами, так как для получения информации о динамике изменения свойств системы необходимо проводить серию из 20–30 запусков. При этом однократный расчёт (выполнение программы при определённом наборе параметров) занимает от четырёх до восьми часов на современном персональном компьютере средней производительности.

Следует отметить, что задача расчёта движения ленты лайнера для одного набора значений параметров является сильно связной в частности по причине перестроения сетки на каждом шаге обработки. В численных алгоритмах используются неявные (по времени) схемы, что затрудняет распараллеливание данных алгоритмов. В связи с этим ускорение может быть достигнуто путём параллельного запуска программы моделирования с различными значениями исследуемого параметра на нескольких компьютерах. Кроме того, в зависимости от получаемых результатов в ходе вычислительного эксперимента разработчиками численных моделей и программного комплекса могут вноситься соответствующие изменения с

целью учёта дополнительных свойств магнитного компрессора и поведения программы моделирования.

5.2.2. Проведение расчётов на некластеризованных компьютерах

5.2.2.1. Описание расчётной задачи

Для дискретизации уравнений созданной математической модели применён метод конечных элементов с элементами первого порядка, для чего предварительно проводится триангуляция расчётной области. С целью оптимизации количества узлов в расчётной области осуществляется сгущение сетки вблизи лайнера. Под воздействием магнитного поля пластины лайнера двигаются навстречу друг другу, поэтому сетка в диэлектрической подобласти перестраивается на каждом шаге по времени. На рис. 18 приведена сетка в части области для расчета с упругопластическим лайнером в один из моментов времени.

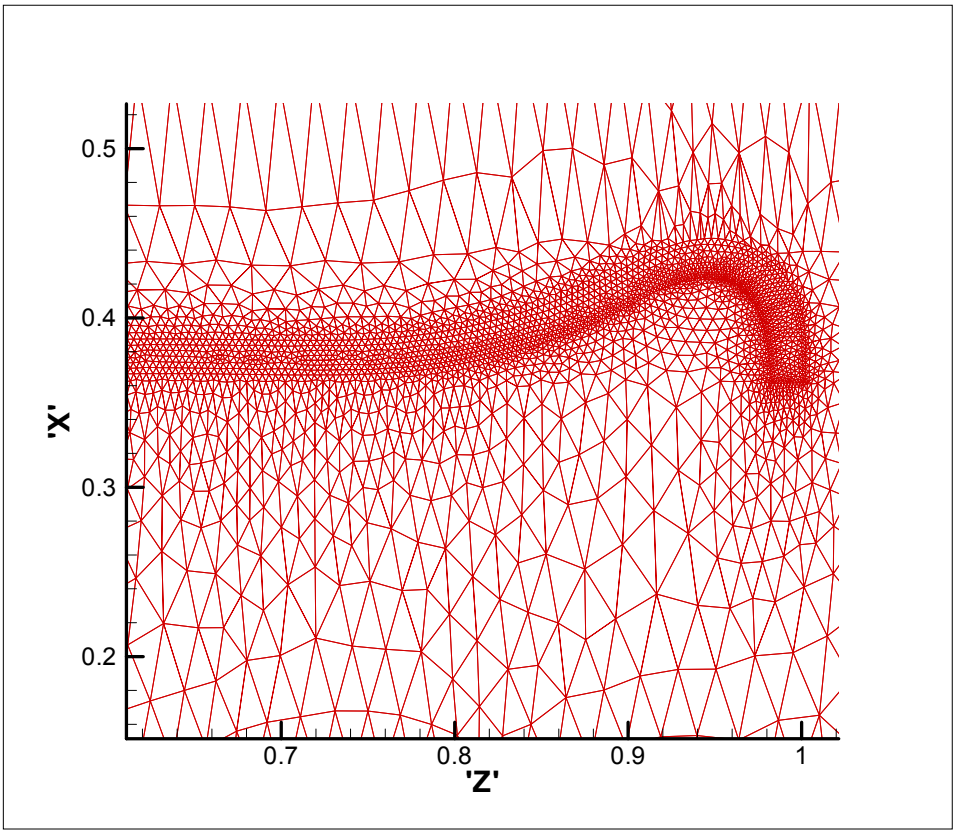
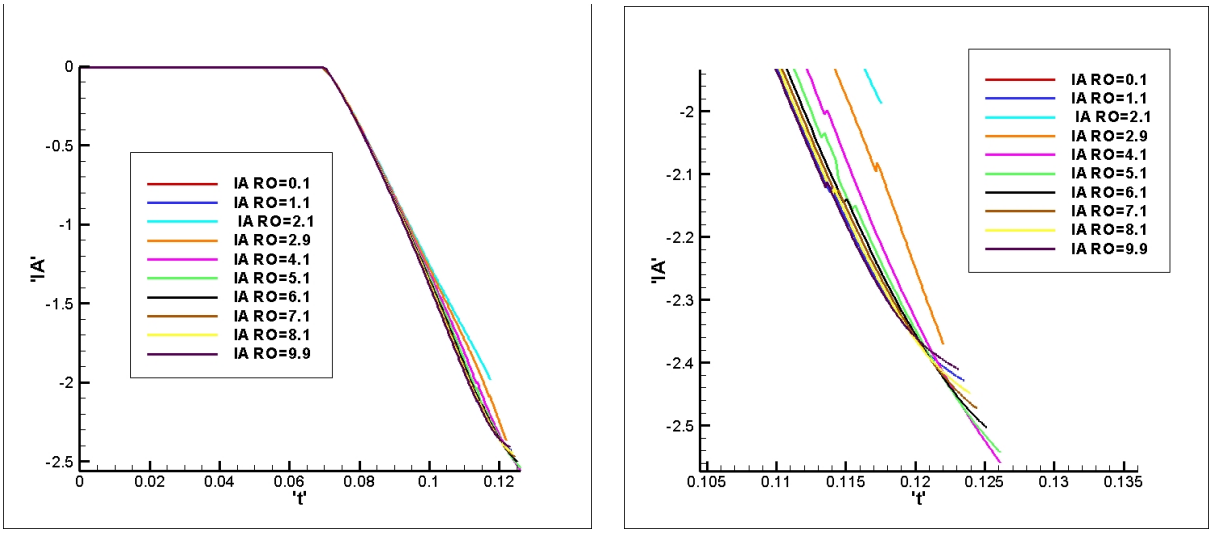


Рис. 18. Сетка для пластического лайнера

С помощью разработанной системы диспетчеризации проведено две серии оптимизационных расчётов [7].

В первой серии варьировался момент замыкания цепи лайнера t_A в диапазоне от 50 до 100 микросекунд (процесс ускорения лайнера занимает 120–130 микросекунд). Проведённые расчёты подтвердили, что оптимальным является значение t_A=70, которое и используется в экспериментальных запусках (это значение было изначально рассчитано исходя из энергетических характеристик устройства).

Во второй серии варьировалось распределение плотности вещества в пластине лайнера: в начальной постановке задачи пластина имела однородную плотность, в дальнейших расчётах принималось, что плотность меняется по линейному закону (в центре пластины коэффициент пропорциональности равен 1, на краях пластины он равен *RO*). Значение *RO* в расчётах менялось от 0.1 до 10 (при этом общая масса лайнера во всех расчётах одинакова). На рис. 19.А показаны графики зависимости от времени выходного импульса (полного тока в цепи лайнера) для различных значений *RO* (на рис. 19.Б приведён увеличенный фрагмент).



А Б

Рис. 19. Графики выходного импульса для различного распределения плотности в лайнере

Как видно из представленных графиков, выбор распределения плотности вещества в пластине оказывает существенное влияние на

выходной импульс (при этом динамика деформирования пластины для различных расчётов сильно отличается). В выбранном диапазоне оптимальным оказалось значение RO=4, то есть плотность вещества на краях лайнера в четыре раза больше, чем плотность в центре пластины. В дальнейшем планируется использовать полученную информацию для определения оптимального начального профиля пластины (плотность вещества будет постоянной, но за счёт изогнутой формы край пластины будет тяжелее, чем центр).

5.2.2.2. Описание программы расчёта

Построенные математические модели лайнера реализованы в виде двух приложений и набора файлов с данными. Первое приложение — основная программа расчёта, второе — вспомогательная программа Gridder2D [60] для перестроения сетки, вызываемая на каждом шаге работы основной программы. Приложения могут выполняться практически на любом современном компьютере под управлением операционной системы семейства Windows. Размер исполняемого файла составляет около 500 КБ, а вместе с файлами конфигурации и входными данными размер пакета поставки не превышает 3 МБ.

В качестве входных данных основная программа получает набор файлов, в которых заданы геометрические размеры области, значения физических характеристик используемых материалов, исходные распределения электромагнитных полей в расчётной области, распределения скорости и температуры в пластине лайнера, а также значения других используемых величин.

В процессе работы программы на каждом шаге по времени после решения систем уравнений, соответствующих электромагнитной и кинематической частям задачи, в текстовые файлы записываются новые рассчитанные значения электромагнитных, скоростных и прочих характеристик, а также формируются файлы для анимации движения

лайнера. После этого для перестроения сетки вызывается программа Gridder2D, и основная программа начинает расчёт следующего шага по времени.

В результате работы программы также создаются файлы для построения графиков зависимости от времени интегральных параметров системы, в том числе график выходного импульса.

5.2.3. Полученные результаты

В ходе проведения серий экспериментов с помощью системы диспетчеризации были получены следующие результаты:

- было подтверждено оптимальное значение момента замыкания цепи, рассчитанного исходя из энергетических характеристик устройства;
- выявлена зависимость выходного импульса от распределения плотности вещества в пластине лайнера;
- определено оптимальное значение распределения плотности вещества в пластине лайнера в заданном диапазоне;
- получена информация о деформировании ленты лайнера, которая в дальнейшем будет использована для определения оптимального начального профиля пластины.

Заключение

В рамках диссертационной работы получены следующие результаты.

1. Разработан новый подход, позволяющий создавать вычислительные инфраструктуры из компьютеров, которые используются в режиме разделения с их владельцами. Предложены сценарии применения таких инфраструктур для массовой обработки вычислительных заданий.
2. Предложена архитектура системы управления инфраструктурой из некластеризованных компьютеров. Архитектура согласована с принципами и стандартами грида, в ней введены функции, необходимые для условий неотчуждаемых компьютеров. Предложенная архитектура допускает использование некластеризованных ресурсов в составе объемлющих грид-инфраструктур с любой формой организации ресурсов.
3. Разработан новый метод планирования распределения заданий, направленный на обеспечение завершения заданий в заданный срок. Путём моделирования показано, что использование прогноза эффективной производительности исполнительных компьютеров существенно повышает качество планирования и позволяет увеличить степень полезной загрузки пула неотчуждаемых компьютеров.
4. На основе предложенных решений выполнена программная реализация системы диспетчеризации, создана вычислительная инфраструктура, на которой проведены расчёты прикладных задач.

Литература

[1]. П.С. Березовский, В.Н. Коваленко. Политика предоставления ресурсов в грид // Распределённые вычисления и Грид-технологии в науке и образовании: Труды международной конференции. Дубна: ОИЯИ, 2004. С. 36–41.
http://gridclub.ru/library/publication.2004-12-27.9733553755

[2]. Березовский П.С., Коваленко В.Н. Способ построения грид из некластеризованных ресурсов // Распределённые вычисления и Грид-технологии в науке и образовании: Труды 2-й международной конференции. Дубна: ОИЯИ, 2006. С. 216–226.
http://gridclub.ru/library/publication.2006-11-30.8170860171

[3]. П.С. Березовский, В.Н. Коваленко. Состав и функции системы диспетчеризации заданий в гриде с некластеризованными ресурсами // Препринт № 67. Москва: ИПМ им. М.В.Келдыша РАН, 2007. 29 с.
http://gridclub.ru/library/publication.2008-10-20.3546711477

[4]. П.С. Березовский, В.Н. Коваленко. Планирование в гриде с разделяемыми ресурсами на основе статистических данных // Программные продукты и системы. 2009. № 1(85). С. 3–6.

[5]. П.С. Березовский. Реализация системы диспетчеризации заданий SARD в одноуровневом гриде // Препринт № 49. Москва: ИПМ им. М.В.Келдыша РАН, 2010. 32 с.

[6]. П.С. Березовский, В.Н. Емельянов, В.Н. Коваленко, Э.С. Луховицкая Механизмы управления разделяемыми компьютерами в гриде // Распределённые вычисления и Грид-технологии в науке и образовании:

Труды 3-й международной конференции. Дубна: ОИЯИ, 2008. С. 303–306.

[7]. П.С. Березовский, А.С. Родин. Проведение оптимизационных расчётов для магнитного компрессора с использованием грида персональных компьютеров, управляемых системой SARD // Препринт № 7. Москва: ИПМ им. М.В.Келдыша РАН, 2011. 31 с.

[8]. Коваленко В.Н., Корягин Д.А. Организация ресурсов в грид // Препринт № 63. Москва: ИПМ им. М.В.Келдыша РАН, 2004. 25 с.

[9]. Инструментарий Globus Toolkit — http://www.globus.org

[10]. Проект gLite — http://glite.web.cern.ch

[11]. D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer. SETI@home: An Experiment in Public-Resource Computing. Communications of the ACM, Vol. 45 No. 11, November 2002, pp. 56–61.

[12]. Проект SZTAKI Desktop Grid — http://szdg.lpds.sztaki.hu/szdg

[13]. Peer-to-Peer — http://www.openp2p.com

[14]. D. P. Anderson: BOINC: A System for Public-Resource Computing and Storage. 5th IEEE/ACM International Workshop on Grid Computing, November 2004, pp. 1–7.

[15]. СУБД MySQL — http://www.mysql.com

[16]. D. Zhou, V. Lo: Cluster Computing on the Fly: resource discovery in a cycle sharing peer-to-peer system. Fourth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'04), 2004, pp. 66–73.

[17]. W. Cirne, F. Brasileiro, N. Andrade, R. Santos, A. Andrade: Labs of the World, Unite!!! Journal of Grid Computing, Vol.4, No. 3, September 2006, pp. 225–246.

[18]. Монитор виртуальных машин Xen — http://www.xensource.com

[19]. С.И. Соболев. Управление заданиями в Виртуальном метакомпьютерном центре на основе технологий X-Com. // Распределённые вычисления и Грид-технологии в науке и образовании:

Труды 2-й международной конференции. Дубна: ОИЯИ, 2006. С. 401–404.

[20]. Воеводин Вл.В., Жолудев Ю.А., Соболев С.И., Стефанов К.С. Эволюция системы метакомпьютинга X-Com // Вестник Нижегородского государственного университета им. Н.И. Лобачевского. №4. 2009. С. 157–164.

[21]. R. Housley, W. Polk, W. Ford, D. Solo. Internet X.509 Public Key Infrastructure. Certificate and Certificate Revocation List (CRL) Profile.

[22]. O. Lodygensky, G. Fedak, F. Cappello, V. Neri, M. Livny, D. Thain. XtremWeb & Condor sharing resources between Internet connected Condor pools. CCGrid, pp.382, Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03), 2003.

[23]. Система Entropia — http://www.entropia.com

[24]. Система Condor — http://www.cs.wisc.edu/condor

[25]. M. Solomon. The ClassAd Language Reference Manual, Version 2.4, 2004.

[26]. R. Raman, M. Livny, M. Solomon. Matchmaking: An extensible framework for distributed resource management, Springer Netherlands, 2004.

[27]. Служба управления заданиями GRAM http://www.globus.org/grid_software/computation/gram.php

[28]. Resource Specification Language. http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/gram_job_description.html

[29]. A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, A.S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) specification, version 1.0. http://www.ogf.org/documents/GFD.56.pdf

[30]. I. Foster, C. Kesselman, J. Nick, S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. http://www.globus.org/reseach/papers/ogsa.pdf

[31]. E. Christensen, F. Curbera, G. Meredith, S. Weerawarana. Web Services Description Language (WSDL) 1.1.

http://www.w3.org/TR/wsdl

[32]. C. Kenyon, G. Cheliotis. Creating Services with Hard Guarantees from Cycle-Harvesting Systems. Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03). pp.224. 2003.

[33]. D.P. Anderson, G. Fedak. The Computational and Storage Potential of Volunteer Computing. IEEE/ACM International Symposium on Cluster Computing and the Grid, Singapore, May 16–19, 2006.

[34]. P. Dinda, G. Memik, R. Dick, B. Lin, A. Mallik, A. Gupta, S. Rossoff. The User In Experimental Computer Systems Research, Proceedings of the Workshop on Experimental Computer Science (ExpCS 2007), June 2007.

[35]. K.D. Ryu, J.K. Hollingsworth. Resource Policing to Support Fine-Grain Cycle Stealing in Networks of Workstations. IEEE Transactions On Parallel And Distributed Systems, Vol. 15, No. 9, September 2004.

[36]. L. Eggert, J.D. Touch. Idletime Scheduling with Preemption Intervals. Proc. SOSP 005. http://www.isi.edu/touch/pubs/sosp2005.pdf

[37]. M. Canonico. Scheduling Algorithms for Bag-of-Tasks Applications on Fault-Prone Desktop Grids. Ph.D. dissertation, University of Turin, 2006.

[38]. Xiaojuan Ren, Seyong Lee, Rudolf Eigenmann, Saurabh Bagchi. Resource Availability Prediction in Fine-Grained Cycle Sharing Systems.

[39]. Dinda, P.A. The statistical properties of host load. Lecture Notes in Computer Science, Volume 1511, 1998, pp. 319–334.
http://reports-archive.adm.cs.cmu.edu/anon/1998/CMU-CS-98-143.pdf

[40]. P. Dinda and D. O'Hallaron. Host load prediction using linear models. In The 8th IEEE Symposium on High Performance Distributed Computing, 1999.

[41]. Jin Liang, Klara Nahrstedt and Yuanyuan Zhou, "Adaptive Multi-Resource Prediction in a Distributed Resource Sharing Environment", the 4th IEEE/ACM Symposium on Cluster Computing and the Grid (CCGrid'04), April 2004.
http://cairo.cs.uiuc.edu/~jinliang/pub/ccgrid04-jinliang.pdf

[42]. DataGrid. DataGrid Accounting System. http://www.infn.it/workload-grid/docs/DataGrid-01-TED-0126-1_0.pdf

[43]. D.P. da Silva, W. Cirne, and F.V. Brasileiro. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. In Proc. of EuroPar 2003, volume 2790 of Lecture Notes in Computer Science, 2003.

[44]. J. Brevik, D. Nurmi, R. Wolski. Automatic Methods for Predicting Machine Availability in Desktop Grid and Peer-to-peer Systems, CCGrid 2004 GP2PC Workshop, 2004, Chicago, IL. http://pompone.cs.ucsb.edu/~nurmi/mypapers/ccgrid04.pdf

[45]. J. Brevik, D. Nurmi, and R. Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. Technical Report 37, Department of Computer Science, University of California, Santa Barbara, 2003.

[46]. M. Mutka. Considering deadline constraints when allocating the shared capacity of private workstations. Int. Journal in Computer Simulation, vol. 4, no.1 pp. 4163, 1994.

[47]. СУБД PostgreSQL — www.postgresql.org

[48]. GridFTP — http://www.globus.org/grid_software/data/gridftp.php

[49]. LINPACK — http://www.netlib.org/linpack/

[50]. J. Richter. Make Your Windows 2000 Processes Play Nice Together With Job Kernel Objects. Microsoft systems journal, 1999.

[51]. SOAP — http://www.w3.org/TR/soap/

[52]. gSOAP — http://www.cs.fsu.edu/~engelen/soap.html

[53]. Утилита globusrun-ws — http://www.globus.org/toolkit/docs/4.0/execution/wsgram/rn01re01.html

[54]. Delegation Service http://www.globus.org/toolkit/docs/latest-stable/security/delegation

[55]. Михайлов Г.А. Весовые методы Монте-Карло. Новосибирск: Изд-во СО РАН, 2000.

[56]. G-R. Jaenisch, C. Bellon, U. Samadurau, M. Zhukovskiy, S. Podoliako. Monte Carlo radiographic model with CAD-based geometry description. Insight, Vol. 48, No 10, October 2006, pp. 618–623.

[57]. М.Е. Жуковский, С.В. Подоляко, М.В. Скачков, Г.-Р. Йениш. О моделировании экспериментов с проникающим излучением. Математическое моделирование, Т19, №5, 2007, с. 72–80.

[58]. Э.А. Азизов, С. Г. Алиханов, Е.П. Велихов, М.П. Галанин, В.А. Глухих, Е.В. Грабовский и др. Проект «Байкал». Отработка схемы генерации электрического импульса // Вопросы атомной науки и техники, серия Термоядерный синтез. 2001. №. 3. сс. 3–17.

[59]. М.П. Галанин, А.П. Лотоцкий. Моделирование разгона и торможения лайнера в устройствах обострения мощности // Радиотехника и электроника. 2005. Т. 50. №2. сс. 256–264.

[60]. Щеглов И.А. Программа для триангуляции сложных двумерных областей Gridder2D // Препринт №60. Москва: ИПМ им. М.В. Келдыша РАН, 2008. 32 с.

Приложение 1. Схема базы данных диспетчера

На рис. 20 представлена схема базы данных диспетчера системы SARD.

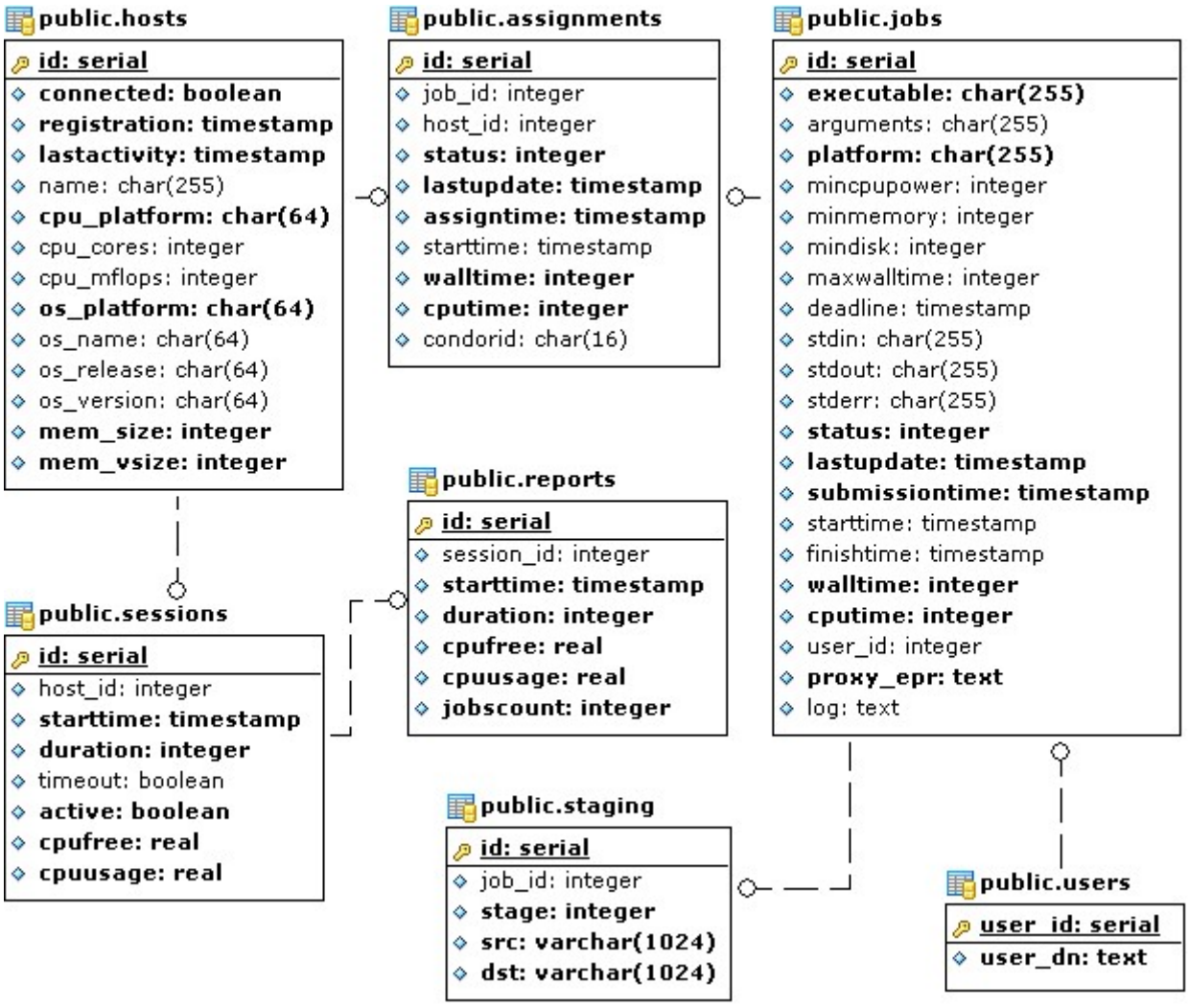


Рис. 20. Схема базы данных диспетчера системы SARD

Приложение 2. Формат описания задания

Листинг 2.1. Описание задания с ограничениями по характеристикам исполнительного компьютера, времени выполнения и предельного срока исполнения.

```
<?xml version="1.0"?>
<job>

<executable>photons.exe</executable>
<argument>-batch</argument>
<architecture>x86</architecture>
<operatingSystem>Windows</operatingSystem>
<minCpuPower>200</minCpuPower>
<minMemory>64</minMemory>
<minDisk>150</minDisk>

<deadline>2010-04-05T16:00:00</deadline>
<maxWallTime>360</maxWallTime>

<stdin>stdin.txt</stdin>
<stdout>stdout.txt</stdout>
<stderr>stderr.txt</stderr>

<fileStageIn>
<transfer>
      <sourceUrl>gsiftp://sard.keldysh.ru/home/pav/photons.exe</sourceUrl>
      <destinationUrl>photons.exe</destinationUrl>
</transfer>
<transfer>
      <sourceUrl>gsiftp://sard.keldysh.ru/home/pav/settings.txt</sourceUrl>
      <destinationUrl>settings.txt</destinationUrl>
</transfer>
<transfer>
      <sourceUrl>gsiftp://sard.keldysh.ru/home/pav/al_cv_f_r.tab</sourceUrl>
      <destinationUrl>al_cv_f_r.tab</destinationUrl>
</transfer>
</fileStageIn>

<fileStageOut>
<transfer>
<sourceUrl>result.txt</sourceUrl>
<destinationUrl>gsiftp://sard.keldysh.ru/home/pav/result.txt</destinationUrl>
</transfer>
<transfer>
      <sourceUrl>stdout.txt</sourceUrl>
      <destinationUrl>gsiftp://sard.keldysh.ru/home/pav/out</destinationUrl>
</transfer>
<transfer>
      <sourceUrl>stderr.txt</sourceUrl>
      <destinationUrl>gsiftp://sard.keldysh.ru/home/pav/err</destinationUrl>
</transfer>
</fileStageOut>

</job>
```

Ниже приводится описание параметров, указываемых в файле описания задания.

Базовые параметры:

executable — обязательный параметр, определяющий название исполняемого файла программы.

argument — параметр командной строки, который будет передан вычислительной программе. В каждом элементе *argument*, может быть указан только один аргумент. Порядок аргументов должен быть таким, как и при запуске на локальном компьютере (если это важно).

stdin, *stdout*, *stderr* — названия файлов, отождествляемых со стандартными потоками. Пользователь может указать файл, который будет использован в качестве источника стандартного ввода, а также файлы, в которые будут перенаправлены стандартный вывод и стандартный поток ошибок.

fileStageIn — внутри данного элемента содержится список файлов, которые необходимо доставить на исполнительный компьютер перед запуском задания. Описание каждого файла находится в элементе *transfer*. Отметим, что все файлы, которые необходимы для запуска задания, должны быть описаны явным образом, в том числе сам исполняемый файл и файл, используемый в качестве источника стандартного ввода (если используется перенаправление).

fileStageOut — внутри данного элемента содержится список файлов, которые необходимо доставить с исполнительного компьютера после завершения задания. Формат описания каждого файла аналогичен *fileStageIn*. При этом файлы, в которые были перенаправлены потоки вывода, также должны быть описаны явно (если таковые были указаны).

transfer — содержит описание одного файла доставки входных или результирующих файлов. Состоит из элементов *sourceUrl* и *destinationUrl*.

sourceUrl и *destinationUrl* — URL источника и назначения для передачи файла. Следует отметить, что в блоке *fileStageIn* в качестве

sourceUrl должен быть указан абсолютный URL, а в качестве *destinationUrl* — относительный; а в блоке *fileStageOut* наоборот: в качестве *sourceUrl* — относительный, а в качестве *destinationUrl* — абсолютный.

Дополнительные параметры:

architecture, *operatingSystem* — параметры, определяющие требования к архитектуре процессора и операционной системе исполнительного компьютера.

minCpuPower — данный параметр ограничивает выбор исполнительных компьютеров по минимальной вычислительной мощности в мегафлопсах.

minDisk, *minMemory* — минимальный объём свободного дискового пространства и оперативной памяти на исполнительном компьютере в мегабайтах, который должен быть доступен заданию грида. В текущей версии системы эти параметры поддерживаются службой управления заданиями и пользовательским интерфейсом, однако не используются при планировании.

maxWallTime — максимальное астрономическое время выполнения в минутах. Задание не может занимать исполнительный компьютер больше этого времени. В случае превышения значения этого параметра задание будет принудительно снято с выполнения.

deadline — крайний срок, к которому должно быть выполнено задание. Указывается как дата и время в формате YYYY-MM-DDThh:mm:ss.