ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 49 за 2010 г.

Березовский П.С.

Реализация системы диспетчеризации заданий SARD в одноуровневом гриде

Рекомендуемая форма библиографической ссылки: Березовский П.С. Реализация системы диспетчеризации заданий SARD в одноуровневом гриде // Препринты ИПМ им. М.В.Келдыша. 2010. № 49. 32 с. URL: http://library.keldysh.ru/preprint.asp?id=2010-49

О р д е н а Л е н и н а
Институт прикладной математики
имени М.В.Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

П.С. Березовский

Реализация системы диспетчеризации заданий SARD в одноуровневом гриде

Москва 2010

УДК 519.68

П.С.Березовский. **Реализация системы диспетчеризации заданий SARD в одноуровневом гриде.**

Статья посвящена описанию реализации системы диспетчеризации для одноуровневого грида с неотчуждаемыми компьютерами, в основу которой положены теоретические результаты, изложенные автором в предыдущих работах.

В статье приводится состав и функции системы SARD (StandAlone Resource Dispatcher), состоящей из трёх основных компонентов: диспетчера, агента на исполнительном компьютере и пользовательского интерфейса. Рассматриваются детали реализации перечисленных компонентов этой системы.

Ключевые слова: грид, распределённые вычисления, некластеризованные ресурсы, разделяемые компьютеры, управление ресурсами, веб-службы.

P.S. Berezovskiy**. Development of the dispatching system SARD for grid with standalone resources.**

The article is devoted to description of the dispatching system for grid with stand-alone resources. The system is implemented according to author’s theoretical results, stated in previous publications.

Proposed SARD (StandAlone Resource Dispatcher) system consists of three main components: dispatcher, agent on the execution computer and user interface. Details of each SARD component implementation are described in the article.

Key words: grid, distributed computing, non-clustered resources, shared computers, resource management, web-services.

Содержание

Введение

На сегодняшний день применение технологий одноуровневого грида, т.е. грида, состоящего из некластеризованных (отдельных) компьютеров, выходит за рамки исследовательских проектов [1], [2] и публичного компьютинга [3]. Программные системы, объединяющие такие ресурсы, применяются для решения реальных практически важных прикладных и научных задач. 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) [4] и объединении исполнительных компьютеров в одноранговые сети; третий подход характеризуется системами с централизованным управлением; четвёртый подход представлен частными разработками корпоративных систем. В работе [6] рассмотрены наиболее развитые системы, реализующие каждый из представленных подходов, и сделан вывод о том, что ни одна из них не решает всех задач, существенных для создания полноценных грид-инфрастуктур.

Так, платформа BOINC [3], предназначенная для поддержки проектов, не предусмотрена для запуска приложений, т.к. 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.

Слабыми местами систем, реализующих подход с централизованным управлением (например, X-Com [5]), является планирование, которое не позволяет обеспечить выполнение задания в заранее указанный пользователем срок. Необходимость предварительной подготовки задания в X-Com также сужает область применения этой системы. Кроме того, в этих разработках, а также в системах CCOF [1] и OurGrid [2], представляющих P2P-подход, не используются ставшие фактическими стандартами протоколы грида. Это приводит к тому, что такого рода разработки не являются интероперабельными с другими грид-системами. То же самое можно сказать практически обо всех корпоративных решениях.

Исходя из анализа существующих систем, в [6] предложены требования к программному обеспечению грида с некластеризованными компьютерами, выполнение которых позволит учитывать специфику таких ресурсов и организовывать на основе разрабатываемого программного обеспечения полноценные грид-инфраструктуры. В настоящей работе рассматривается программная реализация системы SARD (StandAlone Resource Dispatcher) для одноуровневого грида с неотчуждаемыми, т.е. используемыми их владельцами, компьютерами. Реализация выполнена в соответствии с изложенными в [6] архитектурой системы диспетчеризации заданий и

требованиями к программному обеспечению грида, а также соответствует теоретическим результатам, полученным автором в предыдущих работах [7] и [11].

Краткое описание функций системы SARD представлено в разделе 1. Остальная часть работы организована следующим образом. В разделе 2 рассматривается использование стороннего программного обеспечения, используемого в SARD в качестве базового. Третий, четвёртый и пятый разделы посвящены деталям реализации компонентов системы: диспетчеру, агенту и пользовательскому интерфейсу соответственно. Подробно описан процесс обработки задания, реализация механизмов взаимодействия диспетчера и агента, сценарии использования системы с помощью пользовательского интерфейса, а также другие аспекты реализации. В шестом разделе даётся описание реальных прикладных задач, решённых с помощью разработанной системы. В разделе 7 перечисляются основные направления будущих работ по развитию SARD.

1. Состав и функции системы SARD

Для объединения некластеризованных ресурсов и включения их в состав грид-инфраструктуры в [6] предлагается архитектура программного обеспечения грида в виде системы диспетчеризации, которая состоит из трёх компонентов: диспетчера, агента и пользовательского интерфейса.

В соответствии с предложениями работы [7] исполнительные компьютеры ресурсной инфраструктуры недоступны пользователям непосредственно, а все внешние интерфейсы доступа к ним сосредоточены в диспетчере. Диспетчер реализует интерфейсы доступа к сегменту грида с некластеризованными ресурсами, выполняя распределение заданий между зарегистрированными в нём компьютерами.

Структурно диспетчер SARD представляется набором грид-служб. Входящие в состав диспетчера службы реализуют функции по взаимодействию с агентами, управлению заданиями, планированию и передаче данных.

Диспетчер устанавливается на выделенный сервер в сегменте грида, и к нему подключается множество пространственно распределённых исполнительных компьютеров. Для обеспечения интероперабельности с существующими грид-системами диспетчер имеет стандартный интерфейс запуска заданий (аналогичный GRAM [8]).

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.

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.

2. Стороннее программное обеспечение

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

2.1. Использование системы Condor

В реализации SARD для управления заданиями на исполнительных компьютерах используется система Condor [9], которая 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. 1.

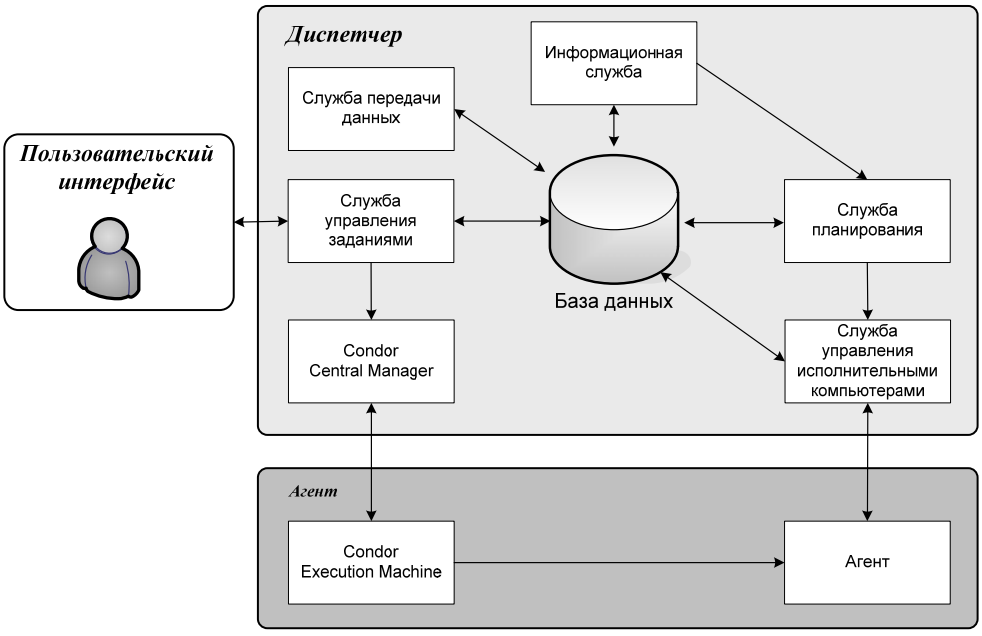

Рис. 1. Архитектура с интеграцией компонентов системы Condor

2.2. Использование базовых служб Globus Toolkit 4

Программный инструментарий Globus Toolkit 4 (GT4) [10] 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.

3. Диспетчер одноуровневого грида

Диспетчер одноуровневого грида представляется набором веб-служб, которые выполняются в контейнере Globus Toolkit 4. В настоящем разделе рассмотрены детали реализации служб, входящих в состав диспетчера системы SARD, их функций, а также механизмов взаимодействия между собой и другими компонентами системы.

3.1. 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.

В соответствии с перечисленными функциями выделены следующие типы сообщений взаимодействия агента с диспетчером через указанную службу:

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

Рассмотрим обработку каждого их указанных типов сообщений.

3.1.1. 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.

3.1.2. 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.

3.1.3. 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.

3.1.4. 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.

3.1.5. 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.

3.1.6. 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 определенные 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.

3.2. Служба управления заданиями

Во время обработки задание может находиться в одном из 12 состояний (Рис. 2). Для сохранения наглядности на диаграмме не обозначены переходы между состояниями в случае перепланирования (подробнее об этой ситуации будет рассказано ниже).

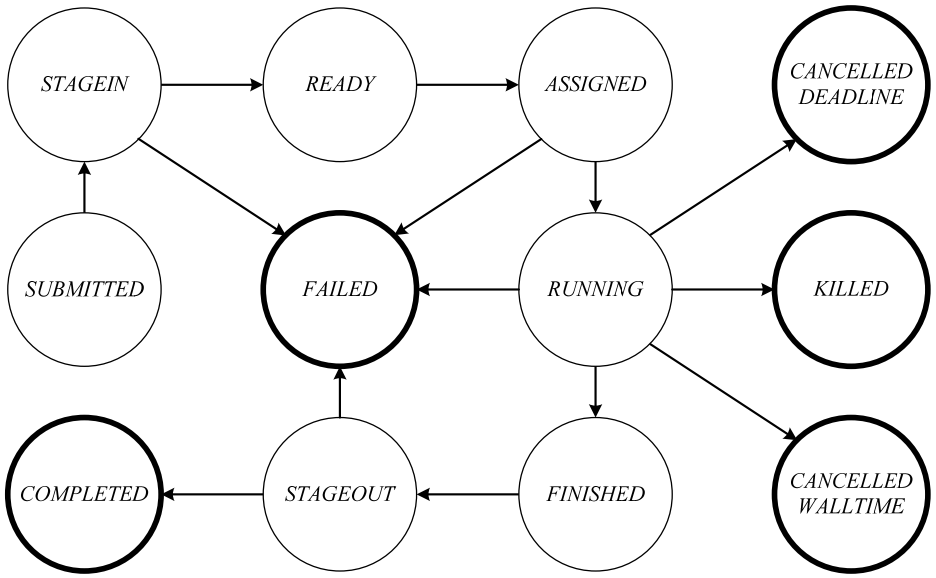

Рис. 2. Диаграмма состояний задания в системе диспетчеризации (жирной линией обозначены конечные состояния)

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.

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 [11] 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*. Этот статус означает начало запуска задания и доставки 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 отметить, 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 и предельного срока исполнения 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.

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, сохраненная 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.

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 — внешнюю доставку, а затем система Condor — внутреннюю доставку, т.е. передачу файлов между диспетчером и исполнительным компьютером. Файлы задания доставляются на машину диспетчера SARD ввиду 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 [12], 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. Важно отметить, что 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 штатным поведением. Несмотря на это, пользователь информируется о таких ситуациях и получает список недоставленных результирующих файлов.

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 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.

4.1. 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.

4.1.1. 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.

4.1.2. 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.

4.1.3. 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.

4.2. Реализация функций агента

Рассмотрим более подробно особенности реализации агентской части системы диспетчеризации.

4.2.1. Хранение конфигурационной информации

Конфигурационная информация хранится в файле *agent.config*, который считывается при запуске агента. Наиболее важным и обязательным параметром, расположенном 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.

4.2.2. 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.

4.2.3. Тест производительности центрального процессора

Компонент агента, отвечающий за получение информации о производительности процессора путём выполнения теста производительности, реализован на основе известного теста LINPACK [13]. 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.

4.2.4. 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 [14], 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.

4.2.5. 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 [15]. Со стороны веб-служб поддержка SOAP реализована в инструментарии Globus Toolkit 4, для поддержки этого протокола на стороне агента применяется кроссплатформенная библиотека gSOAP [16].

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.

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* [17]. 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 1.

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 [18]). 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
```

6. Практические применения системы SARD

Наиболее эффективно некластеризованные 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. В настоящем разделе 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.

6.1. Задача пространственного распределения энергии ионизирующего излучения

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.

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 [19].

6.1.1. Актуальность применения технологий грида

В рассматриваемом классе задач для обеспечения необходимой точности обычно требуется 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 необходимо соответствующим образом модифицировать программный код для поддержки параллельных вычислений.

6.1.2. Расчётная задача

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

В ходе эксперимента алюминиевая труба облучается источником гамма излучения S на основе изотопа Se^{75}, широко применяющегося в дефектоскопии. На внутренней и внешней поверхностях трубы расположены точки-детекторы, в которых рассчитываются характеристики электронных потоков.

6.1.3. Программа 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, содержащий информацию о поглощённой энергии в каждой исследуемой пространственной ячейке. Полученный файл может быть использован для визуализации поглощённой объектом энергии.

6.1.4. Результаты расчётов

Проведение расчётов на инфраструктуре с некластеризованными компьютерами, состоящей из десяти машин, 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.

6.2. 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» («магнитное обжатие лайнеров»), предназначенная для исследования работы всех ступеней модуля установки «Байкал» [20] 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.

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.

Одним из ключевых вопросов, стоящих перед создателями установки, является определение такого набора входных параметров устройства, при котором генерируемый на выходе импульс имеет оптимальные характеристики. На макете МК была проведена серия экспериментальных запусков [21], 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.

6.2.1. 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, необходимо провести ряд экспериментов, варьируя значение исследуемых. Это сопряжено с большими временными затратами, т.к. для получения информации о динамике изменения свойств системы необходимо проводить серию из 20–30 запусков. При этом однократный расчёт 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.

6.2.2. Расчётная задача

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.

С помощью разработанной системы диспетчеризации проведено две серии оптимизационных расчётов.

В первой серии варьировался момент замыкания цепи лайнера *TA* в диапазоне от 50 до 100 микросекунд (процесс ускорения лайнера занимает 120–130 микросекунд). Проведённые расчёты подтвердили, что оптимальным является значение *TA*=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).

Результаты расчётов показали, что 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, т.е. плотность вещества на краях лайнера в четыре раза больше, чем плотность в центре пластины.

6.2.3. Программа расчёта

Построенные математические модели лайнера реализованы в виде двух приложений и набора файлов с данными. Первое приложение — основная программа расчёта, второе — вспомогательная программа Gridder2D [22] 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.

6.2.4. Полученные результаты

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

- подтверждено оптимальное значение момента замыкания цепи, рассчитанное 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.

7. Направление развития системы

К настоящему времени создано и используются на практике довольно большое количество систем распределённого компьютинга и, в частности,

систем, поддерживающих обработку заданий на неотчуждаемых компьютерах. Такое многообразие естественно и объясняется различными условиями, для которых эти системы разработаны [6]. В то же время в них имеются практически одинаковые компоненты, которые реализуют аналогичные функции. При таком положении дел продуктивным представляется подход, когда при создании новых систем реализация базируется на опробованном практикой программном обеспечении.

Основной целью использования системы Condor в SARD был компонент управления заданиями на исполнительном компьютере. Хотя Condor свободно распространяется, описания интерфейсов этого компонента, несмотря на предпринятые усилия, оказались недоступны, что привело к необходимости включения также и пользовательского диспетчера – машины запуска. Машина запуска стала узким местом настоящей реализации SARD, которое не влияет на работоспособность системы, однако снижает её эффективность. Проблема здесь в том, что для каждого запущенного в систему Condor задания, на машине диспетчера стартует отдельный теневой процесс, который занимает её ресурсы до момента завершения задания на исполнительном компьютере. Это накладывает ограничение на количество одновременно выполняющихся заданий.

Другим недостатком текущей реализации можно считать застревание заданий в очереди системы Condor «по неизвестным причинам». Данная проблема хорошо известна применительно к системе Condor, однако на текущий момент причина её возникновения не детерминирована и регулярных способов по её решению для произвольного задания не предложено. В системе SARD предусмотрено возникновение такого рода ошибок и использованы механизмы их корректной обработки.

В ближайших планах развития системы стоит отказ от использования Condor и реализация собственного механизма разделения ресурсов.

Заключение

Несмотря на широкое применение технологий одноуровневого грида для решения вычислительных задач, существующие программные средства обладают рядом недостатков. В первую очередь это вызвано их узкой специализацией либо на конкретных типах заданий, либо на режиме использования выделяемых в грид ресурсов. Кроме того, большинство систем, как правило, несовместимо друг с другом.

Настоящая публикация является развитием работы [6], в которой изложены результаты анализа существующих систем для построения одноуровневого грида и предложены архитектурные решения для построения системы диспетчеризации, включая требования к программному обеспечению такого грида.

Описанная в данной работе программная реализация SARD поддерживает полный цикл обработки заданий и использует в качестве базовых средств возможности инструментария Globus Toolkit 4 и системы Condor. Реализация компонентов системы выполнена с учётом требований, предъявляемых к программному обеспечению грида, и предоставляет стандартные средства запуска заданий.

Разработанная система внедрена в Институте прикладной математики им. М.В. Келдыша РАН и используется для решения актуальных научных и производственных проблем. Результаты проведённых расчётов и вычислительных экспериментов демонстрируют практическую важность системы SARD при решении различного класса вычислительных задач.

Литература

[1]. 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.

[2]. 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.

[3]. Проект BOINC — http://boinc.berkeley.edu

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

[5]. 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.

[6]. 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.

[7]. 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.

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

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

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

[11]. 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.

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

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

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

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

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

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

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

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

[20]. Э.А. Азизов, С. Г. Алиханов, Е.П. Велихов, М.П. Галанин, В.А. Глухих, Е.В. Грабовский, А.Н. Грибов, Г.И. Долгачев, А.М. Житлухин, Ю.Г. Калинин, А.С. Кингсеп, А.И. Кормилицын, В.П. Ковалев, М.К. Крылов, В.Г. Кучинский, В.А. Левашов, А.П. Лотоцкий, С.Л. Недосеев, О.П. Печерский, В.Д. Письменный, Ю.П. Попов, Г.П. Рыкованов, В.П. Смирнов, Ю.А. Халимуллин, В.И. Четвертков. Проект «Байкал». Отработка схемы генерации электрического импульса // Вопросы атомной науки и техники, серия Термоядерный синтез. 2001. №. 3. С. 3–17.

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

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

Приложение 1. Формат описания задания

```
<?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/photons.exe</sourceUrl>
      <destinationUrl>photons.exe</destinationUrl>
</transfer>
<transfer>
      <sourceUrl>gsiftp://sard.keldysh.ru/settings.txt</sourceUrl>
      <destinationUrl>settings.txt</destinationUrl>
</transfer>
<transfer>
      <sourceUrl>gsiftp://sard.keldysh.ru/al_cv_f_r.tab</sourceUrl>
      <destinationUrl>al_cv_f_r.tab</destinationUrl>
</transfer>
</fileStageIn>
<fileStageOut>
<transfer>
      <sourceUrl>stdout.txt</sourceUrl>
      <destinationUrl>gsiftp://sard.keldysh.ru/out</destinationUrl>
</transfer>
<transfer>
      <sourceUrl>stderr.txt</sourceUrl>
      <destinationUrl>gsiftp://sard.keldysh.ru/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.