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

Препринты ИПМ • Препринт № 10 за 2007 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

В. Н. Коваленко

Комплексное программное обеспечение грида вычислительного типа

Рекомендуемая форма библиографической ссылки: Коваленко В. Н. Комплексное программное обеспечение грида вычислительного типа // Препринты ИПМ им. М.В.Келдыша. 2007. № 10. 39 с.
https://library.keldysh.ru/preprint.asp?id=2007-10

ОРДЕНА ЛЕНИНА
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
ИМЕНИ М.В.КЕЛДЫША
РОССИЙСКОЙ АКАДЕМИИ НАУК

В.Н.КОВАЛЕНКО

Комплексное программное обеспечение грида вычислительного типа

Москва
2007 г.

УДК 519.68

В.Н.Коваленко. **Комплексное программное обеспечение грида вычислительного типа.**

Появление развитого программного обеспечения, реализующего подход грид, способствовало возникновению вычислительных инфраструктур, которые объединяют пространственно распределенные (в мировом масштабе) компьютерные установки. Ресурсы таких инфраструктур используются совместно большими коллективами специалистов различных областей науки.

Работа посвящена анализу наиболее полной формы современного программного обеспечения грида – программных платформ. Рассмотрены: формы пользовательской деятельности с распределенными компьютерными, файловыми и информационными ресурсами; программные методы интеграции ресурсов; технологии поддержки функционирования грида и виртуальных организаций.

Результаты выполненного исследования показывают, что имеющаяся программная база грида может служить основой для массовых распределённых приложений, способных комплексно обслуживать профессиональную деятельность в сферах производства, бизнеса и управления.

Ключевые слова: грид, gLite, виртуальная организация, виртуализация ресурсов, глобальная файловая система, распределенная база данных.

V.N.Kovalenko. **Complex Software of Computational Grid.**

The development of advanced software implementing the Grid approach facilitated creation of the computational infrastructures that integrate spatially distributed (on a global scale) computer installations. Resources of such infrastructures are shared by large collectives of specialists in different scientific areas.

The paper presents the analysis of the grid platforms - the most complete form of contemporary grid software. The main issues discussed are: forms of computing with distributed computer, file and information resources; program methods of resource integration; support technologies for grid operation and virtual organizations.

Results of the research reveal, that the existing grid program platforms can serve as a basis for the distributed applications of mass character, capable of comprehensive maintenance of professional activities in various spheres of business, production and management.

Key words and phrases: grid, gLite, virtual organization, resource virtualization, global file system, distributed data base.

Работа выполнена при поддержке Российского фонда фундаментальных исследований (проекты 05-01-00626, 06-07-89111).

ОГЛАВЛЕНИЕ

1. ВВЕДЕНИЕ

В 2003-2004 годах в мире были созданы и начали активно использоваться несколько крупных компьютерных систем нового типа - **грид**: EGEE (www.eu-egee.org), NorduGrid (www.nordugrid.org), Open Science Grid (www.opensciencegrid.org), TeraGrid (www.teragrid.org). Интерес к ним вызывают два обстоятельства. Во-первых, в короткое время в них удалось аккумулировать компьютерные ресурсы сверхбольшого объема (десятки тысяч процессорных единиц), которые применяются для массового счета вычислительно сложных приложений из различных областей, преимущественно науки. Во-вторых, при создании гридов применен принципиально новый подход, составляющий содержание концепции вычислительного грида: они образованы из компьютеров, расположенных в разных местах, то есть грид представляет собой пространственно распределенную систему компьютеров или **инфраструктуру**.

Говоря о гриде как об инфраструктуре, мы подчеркиваем, что это не просто множество компьютеров, каждый из которых потенциально доступен благодаря наличию сети. Подход грида ставит задачу достижения более глубокого уровня интеграции “сырой” ресурсной базы, а **технологии интеграции** составляют основной предмет исследований в этой области. Эти технологии должны обеспечить возможность работы не с отдельными ресурсами, а с их общим полем (в рамках естественных физических ограничений) независимо от их расположения, конкретных адресов, используя средства, скрывающие наличие сети.

Данная работа посвящена гриду вычислительного типа, который ориентирован на обработку заданий с большим временем счета и большим объемом обрабатываемых данных. Достигнутый в действующих инфраструктурах уровень интеграции компьютерных ресурсов таков, что позволяет говорить о гриде как о единой операционной среде для работы с расчетными заданиями. Набор операций вычислительного грида, реализованных в пользовательском и программном интерфейсе, позволяет дистанционно управлять обработкой заданий (запустить, снять, получить результаты). При этом никакого прямого контакта с теми компьютерами, на которых происходит выполнение задания, не требуется: выбор компьютеров осуществляется автоматически, на них создается среда выполнения, включая доставку необходимых файлов, по окончании результаты доставляются в точку запуска (на рабочее место пользователя).

Интеграция компьютеров – не единственный предмет даже для вычислительного грида: оказывается продуктивной и общая постановка проблемы интеграции ресурсов различных типов. Ее общность определяется широкой трактовкой понятия ресурса. Во-первых, можно говорить об интеграции ресурсов, которые составляют аппаратную базу грида. И это не только собственно компьютеры, но также хранилища данных (дисковые

массивы, библиотеки со сменными носителями) и источники данных – датчики, научные инструменты. Во-вторых, можно рассматривать также задачу интеграции объектов компьютинга: файлов, структурированных информационных массивов баз данных и даже программных объектов. Интеграция разных ресурсов требует специальных решений, но, поскольку интеграционные технологии основаны на общей концепции, в них есть место и для общих методов, к которым общепризнанно относится архитектурный подход [1], и к которым мы относим технологии поддержки функционирования грида (п. 8).

При многообразии разработанных на сегодня технологий и программных средств актуальны вопросы о месте подхода грид в информационной сфере, квалификации его возможностей и области применения. Для ответа на эти вопросы необходимым представляется рассмотрение:

- способов, которыми различные виды компьютинга осуществляются в гриде;
- имеющихся средств компьютинга;
- технологий поддержки компьютинга.

Продолжая исследования, начатые в [2], [3], данная работа анализирует с этой точки зрения уровень, достигнутый в наиболее полной с функциональной точки зрения форме современного программного обеспечения грида – программных платформах.

2. НАЗНАЧЕНИЕ И СОСТАВ ПЛАТФОРМЫ GLITE

Основную системообразующую роль при трансформации множества распределенных ресурсов в грид играет программное обеспечение (ПГО – программное обеспечение грида). Его развитие начиналось от базовых средств, поддерживающих дистанционный доступ к ресурсам, прошло стадию отдельных систем, их пакетов и привело к созданию **платформ** – взаимосогласованных наборов средств, способных дать комплексное решение задачи обслуживания грид-инфраструктур производственного назначения.

Создание многих современных грид-инфраструктур сопровождалось разработкой собственного ПГО, так что сейчас имеется несколько различающихся платформ: ARC [4], Alien [5], LCG [6], DataGrid (www.datagrid.org), Unicore (www.unicore.org), gLite (www.glite.org). Тем не менее, они имеют много общего, поскольку в той или иной степени основаны на одной базе – системе Globus Toolkit (www.globus.org). Для наших целей из перечисленных выше платформ наибольший интерес представляет gLite, как наиболее крупная и последовательно развиваемая, и ею будет ограничено дальнейшее рассмотрение.

gLite представляет собой ПГО крупнейшего в мире проекта грид EGEE, сменяя в этом качестве комплекс LCG, который использовался в этой инфраструктуре прежде. Он создавался с учетом опыта предшествующих

исследовательских проектов DataGrid, Globus, DataTag [7], GriPhyN [8], iVDGL [9] и вобрал в себя ряд разработанных в этих проектах компонент. gLite предназначен для поддержки трех видов компьютинга в гриде:

- управления вычислительными заданиями,
- управления данными, представленными в форме файлов,
- управления структурированной информацией баз данных.

Функция управления заданиями первична для ПГО вычислительного грида, но уже в базовых средствах аналогичного назначения серьезное внимание уделяется дистанционной работе с файлами, а ПГО управления информацией применяется как средство получения состояния ресурсов грида и также является необходимым. В gLite эти три функции получают качественно новый уровень развития. В аспекте управления заданиями главные новации - автоматический подбор ресурсов для выполнения заданий (**исполнительных ресурсов**), а также поддержка заданий разных типов: составных, многопроцессорных, интерактивных, с контрольными точками. Функция управления данными перестает быть лишь вспомогательной - в gLite предложено решение по распределенному хранению файлов и прозрачному доступу к ним из приложений на основе глобальной файловой системы. Самостоятельную и более широкую роль способны играть и средства информационного обслуживания. Они базируются на абстрактной архитектуре мониторинга грид GMA (Grid Monitoring Architecture) [10] и могут обеспечить оперативную поставку структурированной информации произвольной природы от множества распределенных источников в распределенную базу данных, которая имеет единую схему и позволяет выполнять поисковые запросы в общем информационном пространстве.

Таким образом, по существу gLite полноценно поддерживает три типа ресурсов: компьютерных, хранения данных и информационных (источники и базы данных). Технологии их интеграции и соответствующие функции управления реализуют три основные части платформы: WorkLoad Management System (WLMS), Data Management System и Relational Grid Monitoring Architecture (R-GMA).

3. АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И СЛУЖБЫ ПОДДЕРЖКИ ФУНКЦИОНИРОВАНИЯ ГРИДА

С точки зрения программной архитектуры gLite является распределенным комплексом: его компоненты устанавливаются в различных точках грида и взаимодействуют между собой по глобальной сети. Распределенность поддерживается специальной формой ПГО: базовыми структурными компонентами выступают службы, то есть программные единицы, которые полностью реализуют отдельные функции и имеют описываемые в стандартной форме программные интерфейсы, делающие эти функции доступными дистанционно из других компонентов (обычных программ или

других служб). Современный подход к архитектуре ПГО предполагает унификацию дистанционного взаимодействия на основе концепции Web-служб и соответствующих стандартных протоколов [11]. Необходимо, однако, отметить, что хотя ведутся работы по переводу gLite на эти стандарты, не все составляющие соответствуют им в полной мере.

Помимо трех представленных в предыдущем разделе групп “содержательных” функций управления компьютингом, мы выделяем четвертую группу – поддержки функционирования грида. gLite создавался для эксплуатации в производственном режиме масштабной ресурсной базы, используемой коллективно большим числом пользователей. Соответственно, предъявляются жесткие требования к качеству функционирования, и gLite решает несколько задач этого плана, обеспечивая: 1) безопасность и надежность всех выполняемых операций, 2) мониторинг хода выполнения операций, 3) обслуживание коллективов пользователей, объединенных в **виртуальные организации**.

В условиях распределенности программного обеспечения, а также рабочих мест пользователей и ресурсов выполнение операций в гриде инициируется выдачей запроса к некоторой службе и осуществляется, как правило, последовательностью взаимодействующих служб, каждая из которых выполняет часть требуемой обработки. В состав gLite входят специальные службы, контролирующие распределенную последовательность обработки запросов:

- отслеживая переходы между шагами обработки,
- собирая диагностику, выдаваемую на каждом шаге,
- регистрируя количество использованных ресурсов.

Надежность обработки запроса обеспечивается путем мониторинга сбоев и перезапуска запроса на альтернативных ресурсах.

В вопросах обеспечения безопасности gLite следует принципам GSI (Grid Security Infrastructure) [12], расширяя их в аспекте авторизации:

- средства авторизации на ресурсах допускают настройку на различную локальную политику безопасности,
- определение прав доступа на основе выделения экаунтов производится динамически, что делает реальным обслуживание практически любого числа пользователей.

Проблема обслуживания больших пользовательских коллективов решается в gLite на основе концепции виртуальных организаций, регистрирующих пользователей, их роли и права. Поддержка заключается в организации хранения регистрационных данных и выдачи по запросам пользователей и различных программных компонент.

4. ОРГАНИЗАЦИЯ ГРИД-ИНФРАСТРУКТУРЫ

Создаваемая на базе комплекса gLite грид-инфраструктура содержит две составляющие: ресурсную и программную. Ресурсная составляющая строится в соответствии с подходом, изложенным в [13], и имеет двухуровневую структуру: в грид включаются не отдельные ресурсы, а **ресурсные центры**. Если иметь в виду компьютерные ресурсы, то соответствующие центры в gLite называются Computing Element (CE). Каждый такой ресурсный центр включает множество компьютеров (**исполнительные компьютеры**), предназначенных для выполнения заданий. Эти компьютеры принадлежат одному административному домену и обслуживаются своими владельцами. Кроме того, предполагается, что исполнительные компьютеры кластеризованы: они управляются **локальным менеджером** - системой пакетной обработки, которая имеет собственный интерфейс управления заданиями и обеспечивает их обработку, поддерживая очереди и распределяя задания по ресурсам своего CE.

Включение ресурсного центра - кластера - в грид происходит путем установки в его локальной среде шлюзовой службы gLite, которая имеет грид-интерфейсы для приема запросов по управлению заданиями. Получая запросы, служба реализует их через локальные интерфейсы системы пакетной обработки.

Похоже обстоит дело с ресурсами хранения, только здесь ресурсным центром выступает хранилище данных - Storage Element (SE), которое может включать несколько разных типов реальных устройств хранения, а вместо системы пакетной обработки выступает менеджер ресурсов памяти.

Программная составляющая, в качестве которой выступают компоненты gLite, представлена во-первых, пользовательскими интерфейсами, и во-вторых, службами. Пользовательские интерфейсы устанавливаются на рабочих местах и представляют собой утилиты, вызываемые из командной строки. Службы ресурсных центров размещаются в них, как правило, на выделенных серверах, отличных от исполнительных компьютеров. Примером могут служить упоминавшиеся выше шлюзовые службы, которые устанавливаются на CE и SE каждого ресурсного центра. Помимо того, есть еще **инфраструктурные** службы, входящие в состав различных систем gLite. Такие службы используются всеми клиентскими программами, работающими в гриде, и могут быть установлены в единственном на всю инфраструктуру экземпляре.

Отметим, что хотя архитектура грида рассчитана на гетерогенную ресурсную базу и использование произвольных платформ в качестве серверов, реализация gLite выполнена только для операционной системы Linux (Scientific Linux 3-4).

5. СИСТЕМА УПРАВЛЕНИЯ ЗАДАНИЯМИ

Основная и наиболее развитая часть в составе комплекса gLite – это система управления заданиями (Workload Management System, WLMS). Ее назначение – поддержка выполнения программ на распределенных и организованных в виде грида компьютерах.

Для пользователя грида вычислительная деятельность выглядит таким образом: он запускает задание, которое доставляется на один или, в случае многопроцессорных заданий, несколько компьютеров из общего ресурсного пула грид, задание выполняется, а результаты могут быть получены на рабочее место, с которого осуществлялся запуск. Реализованные в WLMS технологии поддерживает распределенную обработку такого рода, обеспечивая: автоматическое выделение подходящих исполнительных компьютеров (это основное отличие WLMS от базовых средств управления заданиями, в которых требуется явное указание исполнительных ресурсов); перемещение программы, входных и диагностических файлов; создание среды выполнения, в том числе домашней директории, на исполнительном компьютере.

Формы пользовательской деятельности в гриде и на отдельном компьютере отличаются в нескольких отношениях.

1. Программный код задания выполняется на исполнительном компьютере без участия пользователя (в пакетном режиме). Сам код в рядовом случае не требует адаптации к условиям грид. В некоторых случаях может потребоваться его дополнение прологом/эпилогом, которые выполняют подготовительные/завершающие операции, например, доставку обрабатываемых данных.

2. Задание представляется WLMS в виде формализованного описания, составленного на языке JDL (Job Description Language). Способы описания заданий подробно обсуждаются далее.

3. Ресурсы грид используются коллективно множеством пользователей, поэтому задание не обязательно начинает выполняться сразу после запуска: оно может ждать освобождения ресурсов, занятых другими заданиями. Ожидающие ресурсов задания хранятся в очередях (WLMS или ресурсных центров CE).

4. Вопросы безопасности в гриде gLite решаются на основе принципов GSI. Перед тем как начать работать с WLMS (или любой другой системой gLite), пользователь должен сгенерировать временный прокси-сертификат с помощью команд grid-proxy-init или voms-proxy-init.

5.1. Команды пользовательского интерфейса

Пользователь взаимодействует с WLMS со своего рабочего места, на котором установлен интерфейс (User Interface, UI) этой системы, включающий следующий набор команд:

- glite-job-submit <jdl_file>
  запуск задания, описание которого содержится в файле <jdl_file>. После

приема этого запроса WLMS возвращает идентификатор задания <job_Id>, который используется во всех последующих операциях управления заданием.

- glite-job-cancel <job_Id>
  снятие задания в любой момент процесса обработки.
- glite-job-status <job_Id>
  получение состояния задания. Процесс обработки в гриде многоэтапный, команда выдает название этапа обработки, на котором находится задание, и информацию о его состоянии.
- glite-job-output <job_Id>
  получение суммарной диагностики от всех выполненных этапов обработки.

Как видно из этого списка, набор команд достаточно полон: он позволяет запустить задание, отслеживать ход обработки и при необходимости снять его. Возвращаясь к особенностям работы в гриде, обратим внимание на то, что обработка команд производится распределенным образом различными службами WLMS, которые взаимодействуют между собой по сети. В связи с этим время выполнения команд достаточно велико, порядка минуты.

Отметим также, что функции управления заданиями gLite реализованы не только в пользовательском интерфейсе командной строки, но и в прикладном программном интерфейсе (API) для языков C++ и Java, что позволяет использовать вычислительные ресурсы из различных приложений.

5.2. Описание задания

В этом разделе раскрывается смысл, который в гриде придается понятию вычислительного задания. Файл описания задания – JDL-файл задается в качестве параметра команды job-submit и содержит характеристики задания в виде пар: <атрибут> = <значение>. Начнем рассмотрение с примера, приведенного на рис. 1.

```
[Type = "Job";
JobType = "Normal";
Executable = "myexe";
StdInput = "myinput.txt";
StdOutput = "message.txt";
StdError = "error.txt";
InputSandbox = {"/users/pacini/example/myinput.txt",
"/users/pacini/example/myexe"};
OutputSandbox = {"message.txt", "error.txt"};
Requirements = other.GlueCEInfoLRMSType == "PBS";
Rank = other.FreeCPUs;]
```

Рис. 1. Пример файла описания задания.

Среди атрибутов можно выделить три основные группы, которые определяют: тип задания, используемые в задании файлы и способ выбора ресурсов.

Тип задания описывается двумя атрибутами: Type и JobType. WLMS поддерживает работу как с простыми заданиями, так и с составными, атрибут Type имеет, соответственно, значения “Job” или “DAG”. Тип DAG (direct acyclic graph of dependent jobs) отвечает заданию, в котором должны быть выполнены в определенной последовательности ряд простых заданий.

Для простого задания применим второй атрибут - JobType. Оно может быть обыкновенным (“Normal”), а также:

- интерактивным (“Interactive“). Задание такого рода поддерживает связь с точкой запуска;
- параллельным (MPICH), то есть требующим для выполнения нескольких процессоров;
- с контрольными точками (Checkpointable). Такие задания сохраняют промежуточные состояния и при необходимости их можно перезапустить не сначала, а с какого-то из запомненных состояний;
- сериализуемым (Partitionable). Выполняется несколько экземпляров одного обыкновенного задания, которые обрабатывают разные исходные наборы данных.

Перечисленные характеристики могут сочетаться друг с другом, например, JobType= {“Checkpointable”, “MPICH”}.

5.3. Простые задания

Поскольку составные задания базируются на простых, начнем с последних и рассмотрим атрибуты доставки стандартных файлов и выбора ресурсов.

5.3.1. Доставка стандартных файлов

WLMS обеспечивает перемещение файлов между компьютером рабочего места и исполнительным компьютером, но ограничивается, однако, минимально необходимым набором стандартных файлов операционной системы Unix: исполняемым файлом задания, входных данных, диагностических сообщений и файлом сообщений об ошибках. Трем последним файлам соответствуют потоки stdin, stdout и stderr, которые используются в приложениях предопределенным образом.

Входные файлы (исполняемый и файл данных), лежащие в файловой системе рабочего компьютера пользователя UI, передаются сначала на сервер WLMS, а затем при запуске задания на исполнительном компьютере загружаются в созданную для этого задания домашнюю директорию. Загрузка производится скриптом командной оболочки, который строится одной из компонент WLMS - Job Adapter и выступает как пролог задания. Перемещаемые входные файлы должны быть описаны, во-первых, своими

абсолютными именами в атрибуте JDL-файла InputSandbox (см. рис. 1), и во-вторых, относительными именами в атрибутах Executable (Executable = "myexe") и StdInput (StdInput = "myinput.txt").

Выходные файлы диагностики и ошибок порождаются при выполнении задания в домашней директории исполнительного компьютера, и доставляются по его окончании не в точку запуска, а на сервер WLMS, откуда их можно получить командой glite-job-output. Аналогично входным файлам, выходные должны быть описаны в атрибутах OutputSandbox, StdOutput и StdError.

Поскольку перемещение файлов происходит через временный буфер на сервере WLMS, предполагается, что все они будут небольшого размера. Тем самым, рассматриваемые средства доставки файлов направлены на минимально необходимую поддержку удаленного выполнения программ. Что касается прикладных данных, которые обрабатываются или производятся приложением, то для работы с ними в программе должны использоваться средства системы Управления данными, рассматриваемой в разделе 6.

5.3.2. Выбор ресурсов

Одно из самых важных достижений WLMS – технология виртуализации, реализующая автоматическое распределение заданий по исполнительным ресурсам. Распределение производится с точностью до ресурсного центра, то есть определяется не конкретный исполнительный компьютер, а некоторый CE. При этом в определенной степени учитывается состояние и состав его ресурсов (число заданий в очереди, платформа исполнительных компьютеров и т.д.). Эти сведения поставляются диспетчеру заданий WLMS информационной службой gLite.

Выбор CE управляется двумя условиями: требованиями (Requirements) и рангом (Rank), которые задаются пользователем в атрибутах описания задания. Требования определяют подмножество ресурсов, которые подходят для выполнения. Ранг выражает предпочтения на множестве подходящих ресурсов.

Атрибут **Requirements** записывается на языке classAd [14] в виде булевского выражения. В качестве переменных в нем указываются атрибуты, описывающие CE в информационной базе (в записи им предшествует префикс “other.”). Задание может быть распределено на CE, если значение выражения равно true на его атрибутах. Пример:

```
Requirements = other.GlueCEInfoLRMSType == "PBS" &&
other.GlueCEInfoTotalCPUs > 2 &&
Member("IDL1.7",other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

Это выражение задает требование, чтобы подбираемое CE имел в качестве локального менеджера систему PBS, и в нем было не меньше 2 процессорных единиц. В выражении используется также функция classAd – Member, которая возвращает true, если на CE установлен программный тег "IDL1.7".

Атрибут ранжирования ресурсов. Атрибут **Rank** позволяет определить, какие CE являются предпочтительными для выполнения задания. Он задается арифметическим выражением, зависящим от значений информационных атрибутов CE. Задание будет направлено на CE, которое: 1) удовлетворяет требованиям, заданным в атрибуте Requirements и 2) имеет наивысшее значение атрибута Rank. Этот атрибут имеет важное значение, так как им определяется, как долго задание будет находиться в очереди CE, то есть полное время его обработки. Один из возможных способов задания Rank:

Rank = other.GlueCEPolicyMaxRunningJobs – other.GlueCEStateRunningJobs;

Здесь атрибут GlueCEPolicyMaxRunningJobs представляет максимальное число заданий, которые могут одновременно выполняться на CE, GlueCEStateRunningJobs – число выполняющихся заданий.

Другой способ (применяемый, если атрибут Rank опущен):

Rank = -other.GlueCEStateEstimatedResponseTime;

Указанный здесь информационный атрибут представляет собой оценку времени ожидания задания в очереди локального менеджера ресурсов до начала выполнения. Этот способ определения Rank кажется самым предпочтительным, однако оценка времени ожидания может быть дана в WLMS очень приблизительно.

5.3.3. Параллельные задания

WLMS обеспечивает ограниченную поддержку многопроцессорных параллельных заданий, использующих протокол MPI. Для таких заданий в JDL-файле задается атрибут NodeNumber, определяющий число необходимых процессоров. Кроме того к выражению, заданному в атрибуте Requirements, автоматически добавляется (оператор AND):

(other.GlueCEInfoTotalCPUs $>=$ NodeNumber) &&

Member(other.GlueHostApplicationSoftwareRunTimeEnvironment,"MPICH")

Что означает выбор CE, на котором число процессоров больше NodeNumber, и установлена исполнительная среда MPICH.

5.3.4. Интерактивные задания

Несмотря на то что WLMS обрабатывает задания в пакетном режиме, существует возможность прямого контакта с заданием (если оно простое) на этапе его выполнения. Для этого оно определяется как интерактивное JobType=”Interactive”. В этом случае стандартные потоки stdin, stdout и stderr перехватываются на исполнительном компьютере и перенаправляются на компьютер запуска. Связь осуществляется по X-протоколу, так что на компьютере запуска должен быть стартован X-сервер, который открывает окно для ввода и вывода. Окно и процесс (grid_console_shadow), который слушает порт, отведенный для связи, автоматически запускаются при выполнении glite-job-submit. При разрыве соединения или при необходимости соединения с другого компьютера оно может быть возобновлено командой glite-job-attach.

5.3.5. Задания с контрольной точкой

Для специального типа заданий с контрольной точкой JobType="Checkpointable" WLMS поддерживает возможность периодического сохранения состояния, так что при сбое выполнения их можно перезапустить, но не с начала, а с какого-то промежуточного этапа. Это свойство представляется необходимым при долговременном счете.

JDL-файл для заданий с контрольной точкой строится из тех же атрибутов, которые применяются для простых заданий, однако должны быть заданы также дополнительные атрибуты - JobSteps и CurrentStep. Первый перечисляет имена контрольных точек или задает их максимальное количество, а второй определяет, с какой точки должен быть выполнен запуск.

Предполагается, что исполняемая в задании программа должна быть организована специальным образом и использовать API контрольных точек. Вначале она должна получать информацию о номере контрольной точке, с которой стартовала, а в процессе счета должна запоминать свое состояние в файле контрольных точек. Запись состояния содержит: идентификатор текущей точки, набор переменных программы и их значений. Отметим, что пока предлагаемый аппарат контрольных точек ограничен пользовательскими средствами и не содержит средств автоматического перезапуска заданий в случае, например, сбоев оборудования.

5.4. Составные задания

Наряду с простыми, WLMS умеет управлять наборами зависимых между собой простых заданий, оформляемых в виде одного составного. Составные задания применяются для организации параллельно-последовательной обработки данных, состоящей из ряда этапов, часть из которых зависима по входным данным от результатов других этапов. Зависимость этапов проявляется в том, что некоторые из них могут выполняться параллельно, а другие не могут начать выполнение прежде, чем закончатся их предшественники. Этому соответствует модель составного задания – ациклический граф с ориентированными ребрами (DAG - directed acyclic graph), в котором узлы обозначают задания. Ребро графа, направленное из узла A в узел B, означает, что задание B не может выполняться раньше задания A (см. пример на рисунке 2).

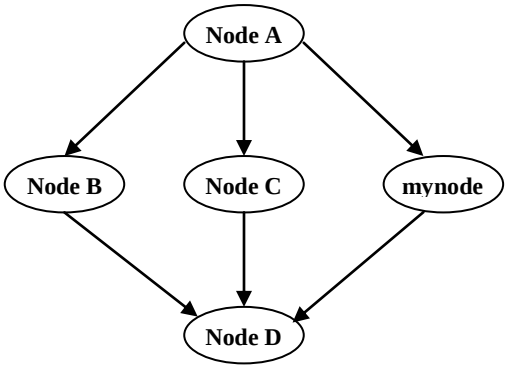

Рис.2. Пример DAG составного задания.

В текстовой форме описание составного задания представляется JDL-файлом специального вида, который включает описание задания в целом, описания его простых составляющих и зависимости. Структура JDL-файла выглядит следующим образом.

```
[
    Type = "dag";
    <Общие атрибуты составного задания>
    nodes = [
                <Имя составляющей> = [description = [атрибуты составляющей]];
                <Имя составляющей> = [description = [атрибуты составляющей]];
                    …
    dependencies = <зависимость узлов>;];
];
```

В общей части JDL-файла используются атрибуты, применяемые для простых заданий, и они наследуются всеми составляющими, если не перекрываются явно в их описаниях. В атрибуте nodes определяются составляющие (атрибут description или file, если описание находится в файле) и их зависимости (dependencies). Базовая форма записи зависимостей – список пар { базовый узел, зависимый узел}, например:

dependencies = { { nodeA, nodeB }, { nodeA, nodeC }, {nodeA, mynode }}

Допускается также сокращенная запись:

{ { node1, node2, node3 }, node4 }, где node4 зависит от трех остальных узлов.

Обработкой составных заданий управляет специальная компонента – DAG Manager (Dagman), который контролирует завершение составляющих и осуществляет запуск новых, если все задания, от которых они зависят, выполнены.

5.4.1. Сериализуемые задания

Хотя формально сериализуемые задания (JobType = "partitionable") отнесены к простым, фактически они являются специальным классом

составных заданий. Их составляющие – подзадания с одним и тем же исполняемым файлом, который требуется выполнить многократно с различными входными данными. В этом случае задания не зависят друг от друга, и имеется компактный способ записи JDL-файла – в нем описывается одна сериализуемая составляющая и две других, выполняющих пре- и пост-обработку.

При запуске сериализуемого задания с помощью команды glite-job-submit генерируется DAG-задание, состоящее из N серийных заданий, а также заданий пре- и пост-обработки. Задание предобработки выполняется первым и от него зависят все серийные заданий. Задание постобработки зависит от всех серийных, в его функцию входит сборка полученных результатов. Пример описания сериализуемого задания приведен на рис. 3 (выделены специфические атрибуты сериализуемых заданий).

Исполняемый файл для серийных заданий готовится по тем же правилам и с помощью того же API, что и задания с контрольной точкой.

```
[
 JobType = "partitionable";
 Executable = "hsum" ;
 JobSteps = {"cms0", "cms1", "cms2", "cms3", "orca"};
 CurrentStep = 0;
 Stdoutput = "std.out" ;
 StdError =  "std.err" ;
 InputSandbox = "/home/cms/prod/hsum";
 OutputSandbox ={"std.out" ,"std.err"} ;
 requirements = Member("GATE-1.0-3",
        other.GlueHostApplicationSoftwareRunTimeEnvironment);
 rank = -other.GlueCEStateEstimatedResponseTime ;
 prejob=[ Executable = "prod_prepa";
          InputSandbox = "/home/cms/prod_prepa";
          rank = other.GlueCEStateFreeCPUs;
          requirements =  other.GlueCEInfoTotalCPUs > 2 ; ];
 postjob=[ JobType = "checkpointable";
           Executable = "aggregator" ;
           Arguments = "5";
           InputSandbox = "/home/cms/prod/aggregator";
           rank = -other.GlueCEStateEstimatedResponseTime;
           requirements =  other.GlueCEStateStatus == "Production";];
]
```

Рис. 3. Пример описания сериализуемого задания.

5.5. Схема обработки заданий

Обработка задания, выполняемая системой WLMS, включает три последовательных этапа.

1. Прием запросов - осуществляется службами Network Server или WMProxy.
2. Определение исполнительных ресурсов – это функция диспетчера заданий (WM);
3. Передача задания в ресурсный центр – производится компонентой CondorC. В случае составных заданий в управлении запуском составляющих участвует Dagman.

В процессе обработки WLMS дистанционно взаимодействует с рядом внешних служб.

1. Обработка задания инициируется командой job-submit. Пользовательский интерфейс производит формальную проверку JDL-файла и возвращает идентификатор задания. После этого посылается дистанционный запрос в одну из двух альтернативных компонент WLMS - Network Server или WMProxy. Network Server - это служба с частным интерфейсом, в то время как WMProxy является службой, реализованной в соответствии со спецификациями Web-служб, в частности для нее есть опубликованное описание интерфейса в формате WSDL. Еще одно отличие заключается в том, что WMProxy поддерживает запросы на запуск составных заданий, в то время как Network Server ограничен только простыми.

2. Центральной компонентой WLMS является диспетчер заданий, который подбирает для него ресурсы. Подбор выполняется механизмом Matchmaking [14], ведущим начало из системы Condor, и заключается в сопоставлении критериев, которые содержатся в описании задания, с характеристиками ресурсов. Для получения сведений о ресурсах диспетчер заданий взаимодействует с информационной системой gLite, база данных которой наполняется и поддерживается в актуальном состоянии информационными сенсорами, расположенными в ресурсных центрах CE. При подборе ресурсов учитывается также политика, регламентирующая права доступа к ним. Для этого диспетчер заданий взаимодействует с сервером виртуальной организации VOMS.

Диспетчер заданий устроен так, что процесс диспетчеризация может быть настроен на работу в двух режимах. В ленивом режиме для поступающего задания сразу же подбирается наиболее подходящий ресурс, и оно направляется на выбранное CE. В жадном режиме задания придерживаются в очереди WLMS до тех пор, пока не произойдет освобождение какого-либо ресурса, и только тогда производится диспетчеризация задания. В этом режиме задание не стоит в очереди ресурсного центра, а сразу начинает выполняться.

3. Передача задания для выполнения на CE осуществляется модулем CondorC. При подготовке передачи JDL-файл преобразуется к формату

CondorC модулем Job Adapter. Кроме того, им строится стартовый скрипт, который будет запускаться на исполнительном компьютере, создавая среду и инициируя выполнение исполняемого файла задания. Задание передается на CE по протоколу службы Gram [13] системы Globus Toolkit. Производится аутентификация, порождается процесс Jobmanager, который запускает задание в локальный менеджер ресурсов CE, используя для этого его пользовательский интерфейс. Задание помещается в локальную очередь CE и распределяется через какое-то время на исполнительный компьютер в соответствии с политикой управления ресурсами, реализуемой планировщиком локального менеджера. Процесс Jobmanager, который порождается для каждого задания, попадающего на CE, работает в течение всего времени обработки задания, собирая сведения о ходе его выполнения.

На всех этапах распределенной обработки задания действуют механизмы, соответствующие технологиям поддержки функционирования грид.

- Ведется наблюдение за запущенным заданием. CondorC периодически опрашивает процесс Jobmanager, выясняя состояние задания, и перезапускает его на другие ресурсы в случае сбоя, не вызванного ошибками самого задания. Dagman осуществляет запуск частей составного задания в соответствии с графом зависимостей.
- Log Monitor (LM) наблюдает за лог-файлами CondorC, перехватывая существенные, то есть изменяющие состояние заданий события.
- Служба протоколирования (LB) собирает информацию от различных компонентов WLMS и сохраняет описания событий, происходящих с заданием. Все компоненты WLMS содержат код, поставляющий информацию в LB. Пользователь может узнать состояние заданий, опрашивая с помощью команд интерфейса службу LB. Кроме того, он может подписаться на получение уведомлений о каких-либо определенных изменениях состояния, например, об окончании задания.

6. СИСТЕМА УПРАВЛЕНИЯ ДАННЫМИ

Второй функциональной составляющей комплекса gLite является Управление данными (Data Management). Созданная в контексте вычислительных приложений, фактически эта система ограничена одной формой данных – файлами, оставляя в стороне, например, структурированные данные информационных приложений. В большой степени система Управления данными независима от WLMS, что позволяет рассматривать ее в качестве средства создания грида распределенного хранения данных. В основе грида этого типа, также как и вычислительного, лежат технологии интеграции и виртуализации ресурсов.

Интеграционные технологии решают задачу образования общего поля постоянной памяти из множества распределенных запоминающих устройств различных типов: допускаются отдельные диски, дисковые массивы,

ленточные устройства массовой памяти (MSS). Запоминающие устройства структурированы в гриде подобно вычислительным: устройства из одного административного домена объединяются в ресурсный центр Storage Element (SE), который управляется локальным менеджером. Множество SE образуют грид хранения данных - доступ к каждому из них осуществляется по унифицированным интерфейсам и протоколам, и память устройств SE может использоваться потенциально любым пользователем грида для размещения файлов.

Технологии виртуализации делают следующий шаг: после того как файл помещается в грид, он становится доступен пользовательским программам, где бы они не выполнялись, примерно так же, как в локальной среде одного компьютера – по имени и посредством стандартных операций. Виртуализация реализуется на основе глобальной файловой системы, общей для всех пользователей грида. Пространство именования файлов иерархически структурировано, то есть помимо собственно файлов могут создаваться директории. Поэтому, хотя файловая система одна на всех пользователей, может быть обеспечена уникальность присваиваемых пользователем имен - так называемых логических имен файлов LFN. В качестве эталона предлагается такой вид имени: lfn://grid/<MyVO>/<MyDirs>/<MyFile>, где <MyVO> - виртуальная организация пользователя, <MyDirs> - его корневая директория.

6.1. Средства работы с файлами

Работа с глобальной файловой системой осуществляется с помощью набора средств ввода/вывода gLite I/O, которые представлены, во-первых, в форме прикладного программного интерфейса (API) и, во-вторых, в виде утилит, вызываемых через интерфейс командной строки (CLI). В составе операций можно выделить три группы: операции с директориями, операции по передаче файлов, операции удаленного доступа к файлам.

Утилиты работы с директориями имеют префикс glite-catalog- и включают:

- Установку прав доступа (chmod);
- Создание, переименование и удаление директории (mkdir, mv, rmdir);
- Выдачу состава директории (ls).

Эти операции имеют прямые аналоги в локальных файловых системах и не требуют особых комментариев (права доступа будут рассмотрены отдельно в п. 6.5).

Операции с файлами реализованы в трех утилитах: glite-put, glite-get, glite-rm, которые, соответственно, помещают файл в грид, копируют из грида в локальный файл, удаляют файл. Формат команд следующий:

glite-put <localfilename> lfn://<lfn>

glite-get lfn://<lfn> <localfilename>

glite-rm lfn://<lfn>

Здесь <localfilename> - это имя локального файла, который помещается в грид под именем <lfn> (в операции put) или в который помещается файл из грида. Семантически помещение файла в грид означает, что производится его копирование на некоторый SE. Сейчас выбор места размещения файла определяется конфигурационным файлом, в котором задается “ближайшее” SE, однако, по-видимому, в дальнейшем будут использоваться более изощренные стратегии.

Перечисленный набор файловых операций и операций с директориями решает проблему доступа к файлам из приложений, которые в условиях грида могут выполняться на различных компьютерах. Соответствующая дисциплина состоит в том, что файлы постоянно хранятся в гриде, а перед началом выполнения приложения копируются (операция get) в локальную файловую систему. Тогда доступ из приложения к файлам может осуществляться обычными средствами, и не требуется адаптации кода приложения к гриду.

Такой способ работы с файлами хорош не всегда. Когда обрабатываемые приложением файлы большие, и обрабатываются они не целиком, а частично, может оказаться предпочтительным вариант непосредственного удаленного доступа к файлу на SE. При такой дисциплине требуется модификация приложения: оно должно быть собрано с клиентской библиотеки gLite I/O. Модификация в большой степени формальная – набор функций этой библиотека включает операции open, close, read, write и lseek, по форме полностью соответствующих стандарту POSIX. С их помощью можно работать с удаленными файлами таким же образом, как с локальными. Известно [15], однако, что в реальности дистанционный доступ отличается от локального и, прежде всего, по временам задержек и по управлению ошибками. Поэтому есть параллельный набор функций glite_* с дополнительными параметрами, способствующими оптимизации и диагностике ошибок.

Таким образом, gLite I/O предоставляет полный набор средств для работы с файлами в пространстве логического именования грида, а также для передачи файлов между гридом и локальной файловой системой. Реализация таких средств для распределенной среды хранения требует специальной поддержки, которая в gLite осуществляется рядом служб: службой каталогов, службами SE и службой передачи файлов (FTS – File Transfer Service).

6.2. Служба каталогов

Файл грид может храниться на любом элементе памяти SE. Поэтому, помимо логического имени LFN он имеет “физическое” имя SURL вида: <sfn | srm>://<SE_hostname>/<path>, где <sfn|srm> - обозначение альтернативных протоколов доступа, SE_hostname – адрес SE, на котором он хранится, а path, идентифицируя файл в локальной среде SE, зависит от организации этого SE (протокола доступа). Помимо того, по ряду технических причин, из которых наиболее важная – обеспечение уникальности логических имен, при помещении файла в грид ему присваивается уникальный идентификатор

GUID. Идентификаторы GUID строятся автоматически по стандарту UUID [16], запомнить которые не представляется возможным (пример: guid:93bd772a-b282-4332-a0c5-c79e99fc2e9c), но при работе с файлами используются не они, а логические имена.

Связь между LFN, GUID и SURL поддерживается службой каталогов - LCG File Catalog (LFC). При помещении файла средствами glite I/O в грид (операция put) производится не только его пересылка на определенный элемент хранения SE, но и регистрация всех трех идентификаторов в каталоге. Каталог также неявно корректируется при выполнении операций переименования и удаления.

Помимо определения месторасположения файлов служба каталогов решает задачу управления пространством именования. Именно в каталоге хранятся структура директорий, состав файлов и права доступа. Поэтому операции с директориями, выполняясь полностью на сервере каталога, имеют префикс glite-catalog-.

6.3. Репликация и служба передачи файлов

Важнейшим механизмом виртуализации доступа к файлам, представленном в gLite, является репликация. Механизм репликации заключается в хранении нескольких экземпляров (реплик) файлов на разных SE, благодаря чему можно уменьшить нагрузку на сеть, “приблизив” в метрике сетевых ресурсов место хранения к месту регулярного использования файла. Репликация поддерживается службой каталогов: в то время как отношение LFN<->GUID взаимооднозначно, для каждого GUID может существовать произвольное количество идентификаторов SURL, адресующих SE, на которых хранятся реплики одного файла. В операциях gLite I/O выбор той или иной реплики осуществляется автоматически, сейчас по принципу статически определяемой “ближайшей”.

Архитектура управления размещением реплик представляет собой трехуровневый стек служб. Первый (верхний) уровень соответствует службе Data Scheduler, роль которой – сбор запросов на перемещение файлов. Предполагается, что запросы поступают либо от администраторов файловой системы, либо от WLMS, выдающей их при распределении вычислительных заданий.

Второй уровень занимают службы FPS (File Placement Service). В инфраструктуре грида они работают на каждом SE, опрашивая Data Scheduler и получая от него те запросы, в которых задано перемещение файлов на данный SE. Полученные запросы помещаются FPS в очередь.

На третьем, нижнем уровне действуют агенты Transfer Agent, управляющие передачей файлов посредством библиотеки File Transfer Library и контролирующие весь процесс, включая обновление каталога реплик.

Отметим, что в этой архитектуре перемещение файлов происходит параллельно со всеми остальными процессами грида и оформляется в виде

заданий. Имеющиеся средства позволяют управлять маршрутами передачи, назначая заданиям различные сетевые каналы.

6.4. Структура элемента памяти

Концепция грида подразумевает, что в него могут включаться устройства хранения разных типов. Для того чтобы с ними всеми можно было единообразно работать, введена абстракция элемента памяти SE, шлюзовые службы которого поддерживает стандартные для грида протоколы доступа к файлам.

Рассмотрим внутреннюю организацию SE. SE включает одно или несколько устройств хранения, образующих общее пространство памяти для файлов грида. Вся совокупность устройств SE управляется локальным менеджером, поддерживающим размещение файлов и операции над ними. Примерами локальных менеджеров являются Disk Pool Manager (DPM) [17] – для множества дисков, dCache [18] – для дисковых массивов, CASTOR [19] – для систем массовой памяти с кеширующими дисками и ленточным хранилищем.

Локальные менеджеры имеют различающиеся внешние интерфейсы и протоколы выполнения файловых операций, так что для доступа из грида непосредственно они не могут использоваться. Грид-интерфейсы SE реализуются тремя службами. Первые две осуществляют транзит запросов (от gLite I/O и FTS), производя отображение протоколов грид во внутренние протоколы локальных менеджеров (рис. 4). Это:

- Служба доступа к файлам с внешним интерфейсом GRID I/O.
- Служба передачи файлов, поддерживающая грид-протокол GridFTP;

Третья служба - Storage Resource Management (SRM) предназначена для управления памятью хранения, выполняя такие функции как резервирование памяти, подготовка данных для передачи по определенному протоколу и т.д.

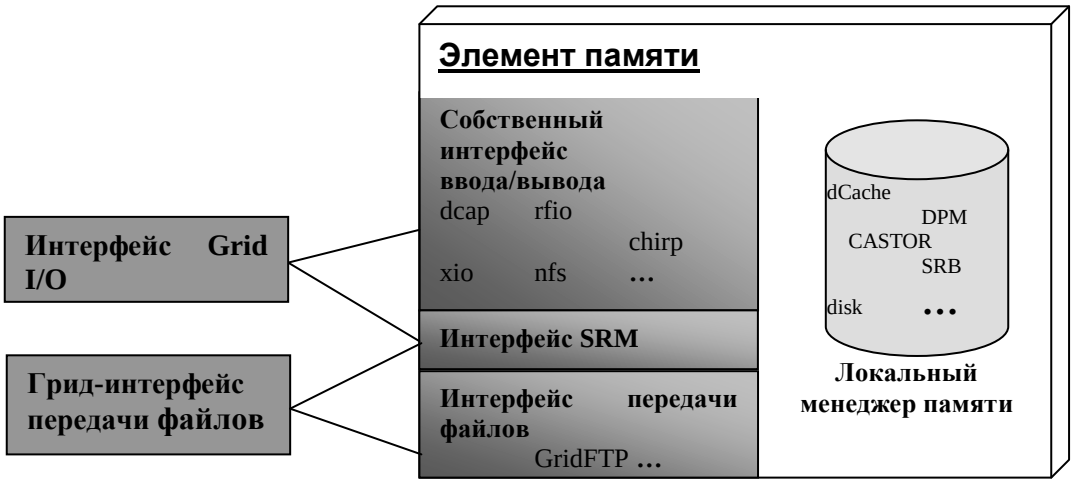

Рис. 4. Грид-интерфейсы SE.

Отметим, что набор поддерживаемых протоколов является расширяемым, и поддержка двух: gLite I/O и GridFTP, является минимальным требованием.

В соответствии с такой организацией схема обработки запроса, к примеру на открытие файла посредством gLite I/O, выглядит так (рис. 5):

- Клиентская библиотека получает в качестве параметра LFN или GUID файла и передает его серверу gLite I/O;
- Производится авторизация операции – посредством службы File Authorization Service и на основе пользовательского сертификата проверяется, имеет ли право пользователь на выполнение операции;
- Путем обращения к каталогу глобальное имя файла разрешается в идентификатор SURL, в котором уже определено конкретное SE;
- Служба gLite I/O обращается к SRM этого SE, получая в ответ локальный идентификатор TURL (Transport URL), соответствующий внутреннему протоколу локального менеджера;
- Производится обращение к интерфейсу POSIX I/O, который запускает внутренний протокол для выполнения операции.

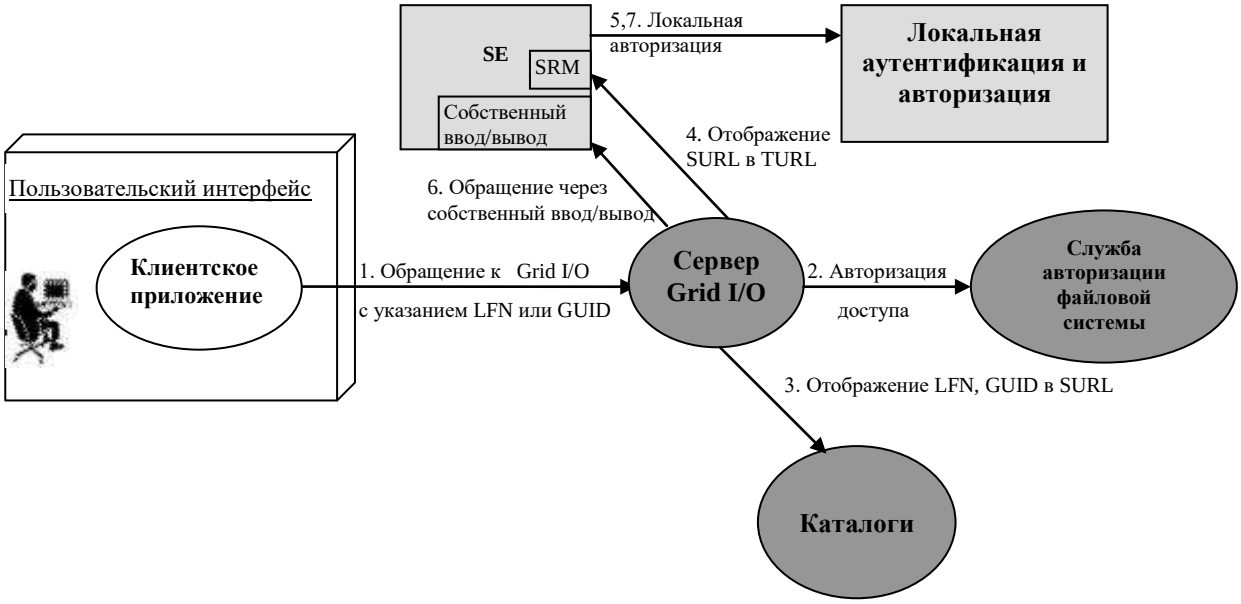

Рис. 5. Схема выполнения файловых операций

6.5. Безопасность файловой системы

Система Управления данными решает задачу защиты файлов на двух уровнях: файловой системы грид, поддерживаемой службой каталогов, и на уровне элементов памяти SE.

Защита в глобальной файловой системе построена по образцу Unix-систем на основе списков контроля доступа (Access Control List – ACL). Каждый файл/директория имеют владельца и атрибуты доступа (owner, group, others), определяющие правомочность выполнения операций для владельца, членов

его группы и всех остальных. Аналогично локальному случаю, владелец и группа идентифицируются числами (uid, gid), которые хранятся в службе каталогов. Во время авторизации пользователя соответствующие ему идентификаторы извлекаются по его глобальному имени DN, содержащемся в сертификате. При создании элементы файловой системы (файл или директория) наследуют атрибуты вышестоящих директорий, которые, впрочем, могут быть изменены операцией chmod. Предполагаемое развитие направлено на введение расширенных ACL, определяющих принадлежность к дополнительной группе на основе определенных в сертификатах VOMS пользовательских ролей.

Безопасность при доступе к грид-файлам (средствами gLite I/O или FTS) обеспечивается тем, что любая операция содержит этап обращения к службе каталогов, который производит разрешение глобальных имен файлов LFN в SURL (либо SURL в TURL). Служба каталога, используя с одной стороны хранящиеся в нем ACL, а с другой стороны – (uid, gid) пользователя, определяет правомочность операции, разрешая или запрещая доступ. В последнем случае она не выдает SURL, что делает продолжение операции невозможным.

Помимо контроля доступа на глобальном уровне решается задача сохранности файлов во внутренней среде SE. Один из возможных способов – обеспечить взаимно-однозначное отображение глобальных идентификаторов (uid, gid) на локальные права доступа. Реализовать это трудно, поскольку, во-первых, локальные механизмы безопасности многообразны, и, во-вторых, проблематично обеспечить синхронность регистрации атрибутов файлов на локальном и глобальном уровне.

Более простой подход основан на том, что владельцем всех файлов грид, хранящихся на SE, назначается специальный административный пользователь (*gstorage*, например), что исключает возможность доступа к ним любых других пользователей, работающих с устройствами SE. Когда же запрос к файлу производится средствами грида, то он всегда поступает вначале на сервер, например, сервер gLite I/O. Как описано выше, сервер производит аутентификацию пользователя по предъявляемому сертификату, проверяет возможность доступа, и, далее, обращается к SE для выполнения операции. Однако, при этом сервер предъявляет не делегированный пользовательский сертификат (как предполагается в первом способе), а собственный, который авторизуется в экаунт администратора памяти (*gstorage)*.

7. ИНФОРМАЦИОННАЯ СИСТЕМА

В комплексе gLite информационная система играет вспомогательную роль, осуществляя сбор, хранение и поиск сведений о деятельности пользователей, состоянии ресурсов для других систем, включая управление заданиями и данными. Фактически параллельно существуют две информационные системы, причем каждая обслуживает свой состав данных. Первая - Globus

Monitoring and Discovery Service (MDS) [20] базируется на OpenLDAP (открытой реализации протокола Lightweight Directory Access Protocol, LDAP) и хорошо подходит для поиска и просмотра информации, но имеет недостатки в аспекте модификации быстро меняющихся данных.

Второй вариант информационной системы построен с помощью системы R-GMA (Relational Grid Monitoring Architecture) [21] в соответствии с архитектурной моделью мониторинга грида GMA (Grid Monitoring Architecture) [10]. Это дает основание рассматривать возможность применения R-GMA для решения более широкого класса задач информационного характера. К числу таких задач мы относим:

1 - хранение и извлечение структурированной информации произвольной природы;
2 - поставка информации от распределенных источников в базу данных в оперативном режиме (мониторинг);
3 - поставка информации от распределенных источников в приложения.

Первая задача совпадает с той, которая решается традиционными системами управления базами данных (СУБД). В этом плане R-GMA является СУБД реляционного типа, следуя всем положениям реляционной модели. В базе данных информация хранится в форме таблиц, которые состоят из множества кортежей (строк). База данных определяется схемой, описывающей структуру каждой таблицы: состав атрибутов и их типы. Работа с информацией ведется с помощью подмножества языка SQL, совместимого со стандартом SQL92. В числе основных операций: вставка строки (Insert), извлечение множества строк по поисковому критерию (Select), определение таблицы (CreateTable).

Особенность R-GMA в том, что это СУБД для распределенных баз данных, в которых различные реляционные таблицы или части одной и той же таблицы могут храниться на разных ресурсах, а наличие общего централизованного хранилища хотя и возможно, но не обязательно. Части БД, физически располагающиеся на одном ресурсе будем называть **фрагментами**. Заметим, что в архитектуре R-GMA отдельный фрагмент локально управляется собственной СУБД (MySQL).

Несмотря на распределенность базы данных, при использовании она выступает как единое целое: имеет общую схему, и поисковые операции выполняются по всему объединенному информационному массиву. Механизм виртуализации, необходимый для обеспечения доступа к информации независимо от места ее расположения, опирается на Реестр - компоненту R-GMA, в которой в дополнение к схеме содержатся описания состава данных отдельных фрагментов, что позволяет отрабатывать запросы по всем релевантным фрагментам. В связи с этим механизмом базу данных, поддерживаемую R-GMA, называют виртуальной.

Вторая решаемая R-GMA задача – мониторинг: помимо вопросов хранения и поиска, традиционных для локальных информационных систем, в ней

решаются вопросы оперативного сбора и доставки информации (указанные выше задачи 2 и 3). R-GMA предлагает средства, с помощью которых:

- может быть создана виртуальная БД и отдельные фрагменты;
- могут быть установлены связи между источниками информации и фрагментами БД, причем один источник может поставлять данные множеству фрагментов и один фрагмент может получать данные от многих источников;
- в виртуальную БД могут быть включены фрагменты, созданные отличными от R-GMA средствами;
- поставка данных от источников может осуществляться не только в БД, но и в приложения.

7.1. Принципиальная схема мониторинга

И задачи поиска, и задачи поставки информации решаются в R-GMA единообразно – в основе лежит схема взаимодействия Поставщиков данных и Потребителей (рис. 6). Поставщик представляет собой компоненту, которая принимает информацию и вводит ее в виртуальную БД. Как будет показано далее, существует несколько вариантов, откуда информация поступает Поставщику, самый простой из них – поставка от источников данных. Поставщик делает поступающие данные доступными Потребителям. Для этого Поставщик объявляет состав данных, которые он способен поставлять, декларируя одну или несколько таблиц (точнее, некоторую выборку столбцов таблицы), определенных в Схеме виртуальной БД.

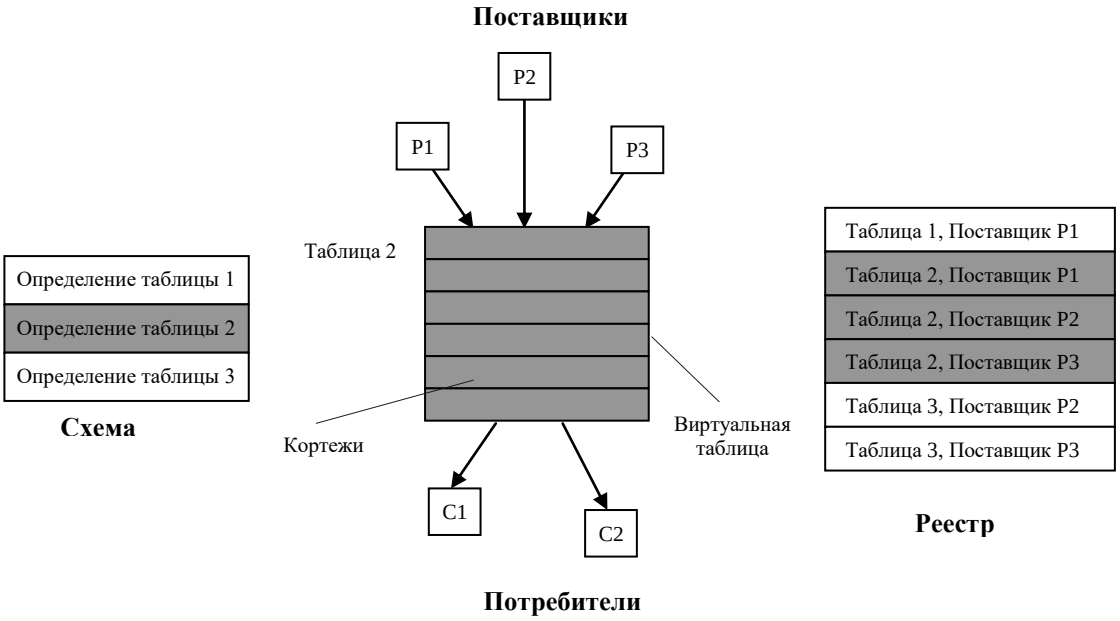

Рис.6. Основные компоненты R-GMA.

Чтобы получать данные, Потребитель не обращается к какому-то конкретному Поставщику, а публикует поисковый SQL-запрос Select Where, получая в ответ удовлетворяющие запросу данные от разных Поставщиков.

Интерпретация запроса, включающая определение списка релевантных Поставщиков и выдачу им частичных поисковых запросов, поддерживается автоматически программными компонентами Потребителей и Реестром.

Способ взаимодействия Поставщиков с Потребителями применим не только для поиска, но таким же образом выполняется и поставка данных. R-GMA является системой мониторинга, в связи с чем она расширяет реляционное представление данных еще в одном аспекте. Поставщики способны хранить историю – кортежи, которые имеют одинаковые ключи, но относятся к разным моментам времени поставки. При вводе к кортежам автоматически добавляется временная метка (time stamp). Предполагается, что кортежи хранятся в течение ограниченного периода времени (retention period), а по его истечении автоматически удаляются. На базе истории возможны несколько режимов возврата результатов поисковых запросов: Latest – выдача удовлетворяющей запросу информации из самых свежих кортежей, History – выдача информации из всех кортежей, поступивших за заданный период времени, Continuous – постоянная передача информации из кортежей, поступающих в течение заданного периода.

Результаты поисковых запросов заносятся во временный буфер (память кортежей) Потребителей, откуда могут быть считаны прикладным кодом с помощью специальной операции (Pop). Прикладным кодом, подключаемым к R-GMA на стороне Потребителя, может быть, например, пользовательский интерфейс, визуализирующий поступающие данные, либо программа, записывающая их на постоянное хранение в БД. Таким образом, описанная схема реализует цепочку доставки источник-Поставщик-Потребитель-приложение, причем связи между Поставщиками и Потребителями имеют избирательный характер и могут устанавливаться динамически. В этом плане R-GMA выступает как средство адресной поставки данных от множества находящихся в разных местах Поставщиков к множеству распределенных Потребителей.

7.2. Типы поставщиков

В развитие рассмотренной схемы взаимодействия Поставщиков и Потребителей модель мониторинга определяет три разновидности Поставщиков – первичный (Primary Producer), вторичный (Secondary Producer) и по запросу (On-Demand Producer). Главным образом они различаются тем, откуда они получают данные.

Первичный поставщик получает данные непосредственно от источников. Под источником понимается прикладной код, который порождает данные и вводит их в виртуальную БД операцией Insert, выдаваемой Поставщику. Весьма важно, что в архитектуре мониторинга отдельный Первичный поставщик обслуживает ограниченный круг “ближайших” источников. Из этого следует, что состав собираемых таким поставщиком кортежей одной таблицы неполон – он ограничен кругом связанных с ним источников, в то время как другие кортежи той же таблицы порождаются “чужими”

источниками. Поэтому Первичные поставщики предназначены не для постоянного, а лишь для временного хранения фрагментов БД.

Функция постоянного хранения возлагается на **Вторичных поставщиков**, с помощью которых может быть организована многоуровневая распределенная структура, в которой в одном фрагменте БД содержатся полные таблицы с данными от других Поставщиков и различные сочетания таблиц. Вторичный Поставщик, несмотря на название, совмещает роли и Потребителя, и Поставщика. Он получает информацию не от источников, а от любых других Поставщиков, выдавая им поисковый запрос, то есть использует технические средства Потребителя. С другой стороны, Вторичный Поставщик способен поставлять свои данные другим Потребителям, в том числе другим Вторичным поставщикам - для этого он декларирует состав своих таблиц.

Вторичный поставщик имеет память кортежей, но она уже предназначена для долговременного хранения и выступает как полноценный фрагмент БД. Более того, информация, поступающая Вторичному поставщику, заносится в этот фрагмент автоматически, какого-либо дополнительного прикладного кода не требуется.

Основное назначение Вторичного поставщика – архивация информации: он создает реальные таблицы из виртуальных, собирая кортежи, которые Источники заносят в виртуальную БД. Вторичный Поставщик может хранить произвольный набор таблиц, изначально публикуемых разными Поставщиками. Это позволяет ему отвечать на поисковые запросы, включающие реляционную операцию Join между хранимыми у него таблицами. Помимо того, осуществляемая им републикация информации дает возможность дублировать данные, повышая надежность хранения и уменьшая нагрузку на других Поставщиков.

Последний тип Поставщика - **По запросу** является средством интеграции внешних по отношению к R-GMA баз данных. В нем в качестве источника данных выступает прикладной код, роль которого – преобразование и передача поисковых запросов от потребителей в БД произвольной организации.

7.3. Управление компонентами R-GMA

R-GMA содержит четыре типа программных компонент, реализующих функции Потребителя, Поставщика, Схемы и Реестра. Простая распределенная база данных может быть создана из них с помощью командного интерфейса, однако она будет рассчитана на ручной ввод данных. Для того, чтобы построить полноценную информационную систему с программной поставкой информации требуется использование прикладного программного интерфейса (API).

С точки зрения программирования существенно, что все программные компоненты R-GMA реализованы в форме служб грид: они имеют стандартизованные интерфейсы, допускающие дистанционное обращение по протоколам Web-служб и включающие операции инициализации, запуска,

остановки и управления временем жизни. При разработке информационной системы службы R-GMA дополняются прикладными программами, которые:

- создают распределенную программную инфраструктуру, запуская компоненты R-GMA, и определяют схему БД;
- вводят данные в Поставщики, реализуя источники;
- осуществляют прием данных в Потребителях.

Распределенная база данных имеет общую схему, которая создается в первую очередь с помощью операций компоненты Схема: createSchema, createTable. Последняя операция выполняется для всех таблиц информационной системы и определяет их структуру.

В состав **Первичного поставщика** входят: прикладной код, служба приема запросов, память кортежей и процессор SQL-запросов. Прикладной код реализует функцию источника данных, а также инициализирует компоненту Первичного поставщика (операция createPrimaryProducer) и декларирует публикуемые им таблицы (один Поставщик может публиковать несколько таблиц). Декларация таблиц производится операцией declareTable, в которой указывается имя таблицы, предикат Поставщика и время хранения кортежей в его памяти. Поставщик и декларируемые им таблицы регистрируются в Реестре.

Таблица может публиковаться Поставщиком не полностью, а лишь частично. Для этого используется предикат в форме конструкции SQL WHERE:

WHERE column = constant AND column = constant AND ...

где column – название столбца, с фиксированным для данного Поставщика значением constant. Например, таким столбцом может быть имя поставщика, в то время как полная таблица будет содержать информацию от множества Поставщиков.

Следующая функция прикладного кода – ввод информации. Она вводится в службу Первичного Поставщика в виде отдельных кортежей операцией SQL INSERT:

INSERT INTO tablename (column, . . .) VALUES (value, . . .)

Кортежи заносятся в память Поставщика, к ним автоматически добавляются временные метки. В памяти кортежи хранятся в течение периода, заданного при декларации таблицы.

Кортежи, находящиеся в памяти Поставщика, становятся доступны для доставки Потребителям, которые подписались на их получение, то есть выдали соответствующий поисковый запрос. Обслуживание реализуется процессором SQL-запросов, который поддерживает все типы запросов - Latest, History и Continuous. Как правило, память Первичного Поставщика используется под временное хранение кортежей и имеет небольшой период хранения, поэтому основное его применение – обслуживание непрерывных запросов, поступающих от вторичных поставщиков.

Вторичный Поставщик создается операцией createSecondaryProducer и предназначен для републикации одной или нескольких таблиц. Републикация задается выполнением операции declareTable, в которой в виде параметра задается запрос на выборку данных “SELECT * FROM tableName WHERE predicate”. Для Вторичного поставщика не требуется дополнительного прикладного кода, обрабатывающего результаты запроса – он самостоятельно помещает поступающие данные в ту же таблицу, имя которой задано в запросе на поиск.

Потребитель – это программная компонента, с помощью которой выполняются поисковые запросы в виртуальной базе данных. Пользовательский код порождает Потребителя (createConsumer), передавая ему запрос SQL SELECT, с указанием режима (History, Latest, Continuous) поставки и интервала, к которому должны относиться выбираемые кортежи. Операция start начинает процесс связывания потребителя с поставщиками: отдельная составляющая Потребителя – медиатор, получая сведения от Реестра обо всех поставщиках, строит план выполнения запроса. План представляет собой набор поставщиков, к которым должен обратиться Потребитель, и частичные запросы к каждому поставщику.

После обращения к поставщикам из списка Потребитель получает результаты через отдельный потоковый сервер, выполняет их слияние и помещает в память – буфер результирующих кортежей. Приложение может их получать, периодически выполняя операцию Pop.

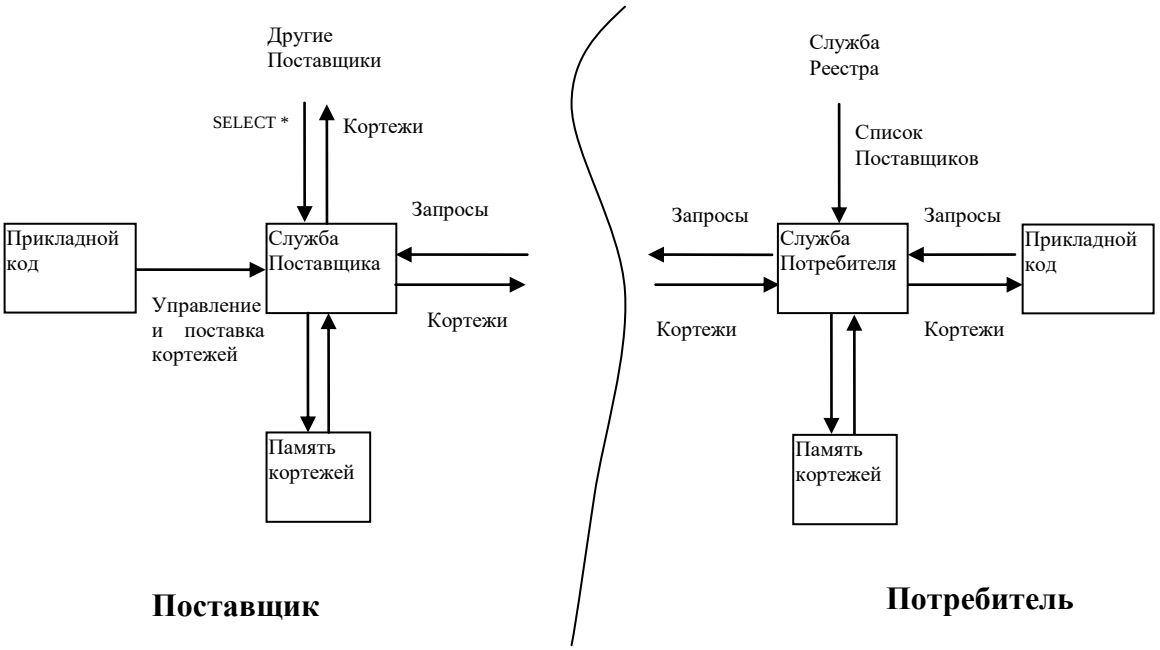

Рис. 7. Схема поставки информации от Поставщиков к Потребителям.

8. СЛУЖБЫ ПОДДЕРЖКИ ФУНКЦИОНИРОВАНИЯ

В условиях распределенной инфраструктуры и децентрализованной обработки запросов особым образом должны решаться вопросы обеспечения безопасности, надежности, мониторинга процессов, обслуживания больших пользовательских коллективов. В gLite сделан существенный шаг в этом направлении, что дает основание выделить отдельную группу технологий поддержки функционирования грида. В то же время следует отметить, что большинство из этих технологий применяется пока лишь в контексте системы WLMS, хотя перспективы их использования в системе управления данными и информационной системе достаточно очевидны.

Рассмотрим ряд служб, которые реализуют поддерживающие технологии.

8.1. Служба протоколирования процесса обработки заданий

Вычислительные задания обрабатываются в гриде распределенно различными компонентами: пользовательским интерфейсом, составляющими WLMS и ресурсного центра CE. В этих условиях возникает проблема информирования пользователей о ходе выполнения запущенных ими заданий. Служба LB (Logging and Bookkeeping service) [22] решает эту проблему:

- предоставляя средства для агрегирования диагностических сообщений, относящихся к каждому отдельному заданию, в виде единого протокола обработки;
- обеспечивая хранение протоколов и предоставление их по запросам;
- посылая уведомления об изменении состояния задания.

Формирование протокола обработки происходит в результате того, что каждая обрабатывающая компонента посредством прикладного интерфейса LB регистрирует существенные события, происходящие с заданием (помещение в очередь, определение исполнительных ресурсов, начало выполнения и т.п.). Каждое событие фиксируется с набором атрибутов, одним из которых является уникальный идентификатор задания.

В конечном счете события собираются на сервере хранения протоколов, но реализуется это в два этапа. На первом этапе событие поступает локальному демону locallogger, который выполняется в непосредственной физической близости от источника события, что позволяет избежать каких-либо сетевых проблем при передаче. Локальный демон записывает событие в файл и возвращает источнику подтверждение об успешности операции.

Далее на втором этапе за доставку события отвечает другой демон interlogger. Получая событие либо прямо от демона locallogger, либо, в случае его падения, с диска, он пересылает его в конечную точку назначения - серверу хранения (bookkeeping server), причем надежность на этом этапе обеспечивается механизмом повторной передачи. Можно заметить, что в целом двухэтапный способ доставки сообщений соответствует схеме,

применяемой в R-GMA: демоны LB играют роль поставщика, а сервер хранения – потребителя.

В инфраструктуре может быть несколько серверов хранения, однако события от всех заданий одного пользователя поступают только на один из них (он определяется статически в конфигурации). Сервер хранения выполняет несколько функций. Во-первых, обрабатывая поступающие “сырые” события, он формирует обобщенное представление о состоянии задания, в котором содержатся такие атрибуты как описание задания (JDL-файл), адрес исполнительного CE, код завершения и т.д. Во-вторых, сервер хранения предоставляет интерфейс Web-служб для получения информации и о состоянии задания (команда glite-job-status), и о всех событиях (glite-job-logging-info).

В-третьих, сервер хранения поддерживает подписку на получение уведомлений об изменении состояния задания. Для подписки клиент регистрируется на сервере хранения, задавая условия получения уведомлений (идентификатор задания, тип события). Уведомления доставляются принимающему клиенту на стороне пользователя асинхронно, тогда, когда происходит событие, указанное в условиях регистрации (например, окончание задания). Этот способ слежения за заданием имеет преимущества, позволяя избежать многократного повторения запросов о состоянии с одним и тем же результатом и снизить, тем самым, нагрузку на сервер.

8.2. Служба учета потребления ресурсов

Входящая в состав gLite служба учета - DataGrid Accounting System (DGAS) представляет собой средство сбора информации об использовании ресурсов пользователями грид. Информационной единицей в этой системе выступает **запись использования** (Usage Records - UR), которая создается для каждого обработанного задания и содержит такие данные:

- идентификатор задания,
- ресурсный центр, в котором оно выполнялось,
- использованные при выполнении ресурсы,
- время: запуска на счет, поступления в очередь локального менеджера, окончания,
- время создания записи.

Сбор этих данных осуществляет демон Gianduia, работающий в среде CE и взаимодействующий с его локальным менеджером. Механизм сбора выглядит следующим образом. Когда задание начинает выполняться на исполнительном компьютере, его стартовый скрипт Job Wrapper создает в специальной директории файл, содержащий имя задания, его идентификатор и адрес сервера, в который должна быть направлена формируемая запись использования. Обнаруживая такой файл, демон Gianduia ищет учетную информацию о задании в лог-файле локального менеджера. Когда задание

заканчивается, вся учетная информация выбирается из этого лог-файла и присоединяется к файлу, созданному вначале.

Собираемые записи использования хранятся в децентрализованной инфраструктуре, образованной множеством серверов, на которых установлена служба Home Location Register (HLR). Организация инфраструктуры хранения имеет в DGAS особенности, вызванные следующим обстоятельством.

Обслуживание запросов на выдачу информации в инфраструктуре из множества серверов существенно упрощается, если информация распределена между ними таким образом, что все данные, относящиеся к одному пользователю сконцентрированы на одном из них. Именно так обстоит дело в DGAS: каждый пользователь приписан к своей “домашней ” службе HLR. Однако, DGAS предназначена для обслуживания не только пользователей, но и провайдеров ресурсов. Если пользователи заинтересованы в получении сведений только о своих заданиях, то провайдерам нужны данные о всех заданиях, которые были обработаны на их ресурсах. При ориентации только на пользовательское обслуживание провайдеры были бы вынуждены опрашивать каждый сервер.

В связи с этим в инфраструктуре хранения поддерживаются два вида служб HLR: для пользователей и для провайдеров. Записи использования, собираемые компонентами DGAS на CE, сначала поставляются в одну из возможных служб HLR, а затем дублируются ею в другую. Поддерживается несколько сценариев поставки:

- Сначала в пользовательский HLR:
  CE =>User HLR =>Resource HLR
- Сначала в HLR провайдеров:
  CE => Resource HLR => User HLR
- Только в HLR провайдеров:
  CE => Resource HLR

Представляет интерес еще одна служба в составе DGAS - служба расценок (Price Authority - PA). Эта служба поддерживает базу данных цен ресурсов и хранит историю изменения цен. Цены в ней могут устанавливаться вручную администратором, но могут также и вычисляться программно по различным алгоритмам, например по алгоритму торгов. В PA реализован механизм встраивания алгоритмов установки цен в форме динамической библиотеки.

Наличие расценок ресурсов в службе PA, с одной стороны, и количества потребляемых заданиями ресурсов в записи HLR, с другой, дают возможность вычислять стоимость заданий. Таким образом, реализованные в DGAS службы позволяют перейти к экономическим методам управления гридом на основе расчетов между пользователями и провайдерами.

8.3. Безопасность и поддержка виртуальных организаций

Вопросы безопасности, значение которых в условиях грида с распределенными автономными ресурсами, большим и меняющимся составом пользователей чрезвычайно велико, решаются в gLite на основе Инфраструктуры публичных ключей PKI [23]. Основные положения подхода заключаются в следующем.

Объекты грида (пользователи и службы) располагают цифровым удостоверяющим документом – сертификатом стандарта X.509, который заверен Сертификационным центром (Certificate Authority). В качестве ключевого атрибута сертификат содержит уникальный идентификатор объекта Distinguished Name (DN). Используемые в гриде протоколы гарантируют надежную взаимную аутентификацию объектов при всех дистанционных взаимодействиях. **Аутентификация** осуществляется путем предъявления сертификатов и не требует участия пользователя.

Механизм делегирования прав открывает возможность выполнения операций программными компонентами от имени пользователя. Такая потребность возникает - например, компонента Job Control должна производить запуск заданий на CE от имени его владельца, но для этого она должна иметь его удостоверение. Надежный способ делегирования состоит в том, что на основе базового сертификата генерируется прокси-сертификат с коротким сроком действия, который и используется для контактов от имени пользователя.

Потребность обслуживания масштабных гридов поставила ряд серьезных проблем в отношении второго аспекта безопасности – **авторизации**. Содержание авторизации заключается в том, что при входе пользователя в систему (в данном случае в ресурсный центр) определяются его полномочия на выполнение определенных действий (выполнение приложения, доступа к файлу или таблице базы данных). В результате авторизации производится отображение полномочий на локальные права (credentials) в конкретной среде безопасности, в Unix-системах - в экаунт пользователя.

Базовый механизм авторизации в гриде, ведущий начало от первых версий Globus Toolkit, основан на gridmap-файле – списке, в котором перечисляются имена DN всех авторизованных на данном ресурсе пользователей грида и соответствующие локальные регистрационные имена. Такой простой способ предполагает, что соглашения между пользователями и провайдерами – где и с какими правами пользователь может получать ресурсы – устанавливаются на персональной основе, что противоречит принципам грида и в реальных условиях невозможно.

В подходе грида за организацию работы пользователей и их взаимотношения с провайдерами отвечают Виртуальные организации (ВО), которые определяют полномочия пользователей и заключают соглашения с провайдерами об их обслуживании. Первой попыткой улучшения базового механизма авторизации стало централизованное ведение списков пользователей грида виртуальной организацией. Список публикуется на

LDAP-сервере, откуда каждый провайдер может его получить и с помощью утилиты mkgridmap сгенерировать gridmap-файл. Такое решение имеет два недостатка. Во-первых, состав пользователей подвержен частым изменениям, и безопасность может быть обеспечена, только если gridmap-файл отражает эти изменения максимально быстро, так что все провайдеры должны регулярно (например, раз в сутки) обновлять список пользователей с сервера. Во-вторых, задача определения прав пользователей в этой схеме по-прежнему возлагается на каждого провайдера.

Для решения перечисленных проблем в gLite предложена реализованная несколькими программными средствами процедура, которая автоматически определяет и проводит в жизнь локальную политику по авторизации пользователей, используя формализованное описание их полномочий.

1. **Формальное описание полномочий.** Аппарат авторизации опирается на формальное описание полномочий пользователя ВО тремя атрибутами [24]:

- группа ВО, к которой принадлежит пользователь,
- его роль внутри группы,
- возможности выполнения операций.

Совокупность этих атрибутов составляет так называемое “Fully Qualified Attribute Names” (FQANs) [25].

2. **Получение авторизационной информации.** Служба Virtual Organization Membership Service (VOMS), представляющая собой грид-интерфейс к реляционной базе данных под управлением MySQL, хранит авторизационные атрибуты пользователей ВО и выдает их по запросам. Административный клиент этой службы позволяет добавлять пользователей, задавать значения атрибутов, создавать группы и т.д. Пользовательский клиент контактирует со службой VOMS, предъявляя сертификат некоторого пользователя, и получает список (если пользователь принадлежит нескольким ВО) его групп, ролей и возможностей.

Использование VOMS позволяет создать схему, в которой авторизация проводится исключительно исходя из авторизационных атрибутов, а доставка этих атрибутов осуществляется путем их включения в состав прокси-сертификата расширенного формата. В этой схеме пользователь использует в качестве клиента VOMS команду voms-proxy-init, которая заменяет использовавшуюся прежде grid-proxy-init. Разница в том, что прокси-сертификат, генерируемый новой командой, содержит авторизационную информацию в виде расширения – атрибутного сертификата [25], совместимого со старым форматом. Таким образом, провайдеры ресурсов освобождаются от необходимости получать и обновлять списки членов ВО – авторизационная информация получается самим пользователем в начале сеанса работы в гриде и передается в составе прокси-сертификата при всех дистанционных контактах.

3. **Выполнение авторизации.** В среде ресурсов на основе представляемой в прокси-сертификате авторизационной информации в соответствии с локальной политикой выносится авторизационное решение. Служба безопасности ресурсного центра (Gatekeeper) начинает с того, что проверяет валидность сертификата, и обращается к службе Local Credential Authorization Service (LCAS) [26]. LCAS представляет собой настраиваемую программную оболочку с разъемами для подключения вставных модулей. Поставляемые в ее составе модули поддерживают проверку специфических для ресурса ограничений: “черный список” пользователей, ограничения по запрашиваемому для обработки заданий времени и т.д. Модули, специально разработанные для VOMS, выделяют авторизационные атрибуты и на их основе выносят решение о возможности авторизации. Заметим, что LCAS может работать как с сертификатами VOMS, так и с сертификатами старой версии, используя в этом варианте gridmap-файл.

Вторая служба - Local Credential Mapping Service (LCMAPS) [26] отображает авторизационную информацию на локальные права доступа, например идентификаторы системы UNIX (uid,gid) таким образом, чтобы получаемые права ограничивали круг возможных действий только допустимыми. LCMAPS, как и LCAS, является модульной средой и способен поддерживать как фиксированное отображение авторизационных атрибутов в предопределенные значения пар (uid,gid), так и динамическое отображение в пул лизинга экаунтов, выделяемых, например, посредством программы Gridmapdir.

4. **Управление экаунтами**. Программа Gridmapdir [27] решает проблему создания и управления экаунтами в Unix-системах путем их динамического выделения из общего пула. Пул экаунтов, создаваемый администратором, не должен быть очень большим: требуемое количество экаунтов ограничивается предполагаемым числом одновременно работающих в ресурсном центре пользователей. Экаунты используются в режиме лизинга: они выделяются при поступлении запроса на обработку задания или передачу файлов и закрепляются за пользователем только на время выполнения задания, затем экаунт возвращается обратно в пул. Для обеспечения взаимооднозначного соответствия между пользователями и экаунтами применяются блокирующие файлы. Поскольку в Unix-системах права доступа к файлам являются производными от экаунтов, способ выделения экаунтов предотвращает возможность влияния друг на друга заданий разных пользователей.

ЗАКЛЮЧЕНИЕ

Сама метафора грида – интеграция глобально распределенных ресурсов – несет большой заряд привлекательности. Однако объективная оценка возможностей подхода должна, по нашему мнению, основываться на том, какие формы компьютинга способны поддерживать технологии, реализующие эту метафору. С этой целью мы рассмотрели технологии, разработанные для

грида вычислительного типа, способы их применения и методы программной реализации. Основные результаты заключаются в следующем.

1. В контексте вычислительного грида разработаны интеграционные технологии для трех видов ресурсов: собственно вычислительных, хранения данных и информационных. Программные средства, развитые для двух последних видов ресурсов, имеют самостоятельное значение, позволяя создавать гриды, ориентированные на такие формы компьютинга как управление распределенными файлами и адресная поставка структурированной информации. Заметим, что эти возможности уже используются: например, в проекте DILIGENT (www.diligentproject.org) платформа gLite применена для создания распределенных цифровых библиотек.

2. Интеграционные технологии, естественно, специфичны для каждого из видов ресурсов. Однако в работе выделен класс грид-технологий поддержки функционирования, которые, наделяя операционную среду грид важнейшими свойствами, являются универсальными. До сих пор большинство из этих технологий были задействованы в программных средствах обработки заданий вычислительного грида, но их применение в равной степени необходимо для всех других форм компьютинга.

3. Характеризуя в целом возможности программного обеспечения вычислительного грида и, в том числе платформы gLite, можно сказать, что они ориентированы на поддержку базовых форм компьютинга, в которых работа ведется с программными объектами (заданиями, файлами, таблицами). В то же время архитектура грида, наличие развитых инструментальных средств позволяют существенно расширить область применения подхода путем создания специализированных распределённых приложений, способных комплексно обслуживать профессиональную деятельность широкого круга пользователей. Представляется, что приведенная детализация грид-технологий обосновывает этот вывод (см. также [3]).

ЛИТЕРАТУРА

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

[2]. Коваленко В.Н., Корягин Д.А. Оценка возможностей программных платформ Грид. Труды международной конференции «Распределенные вычисления и Грид-технологии в науке и образовании» Дубна, 29 июня - 2 июля 2004 г., стр. 128-132.

[3]. В.Н. Коваленко, Е.И. Коваленко, Д.А. Корягин, Э.З. Любимский. Перспективы развития грид: распределенные приложения. Труды 2-й международной конференции «Распределенные вычисления и Грид-технологии в науке и образовании», Дубна, 26 -30 июня 2006 г., стр. 301-308.

[4]. "Advanced Resource Connector middleware for lightweight computational Grids". M.Ellert et al., Future Generation Computer Systems 23 (2007) 219-240.

[5]. P. Buncic, A. J. Peters, P.Saiz. The AliEn system, status and perspectives. Computing in High Energy and Nuclear Physics, 24-28 March 2003, La Jolla, California.
[6]. LCG middleware documentation. http://grid-deployment.web.cern.ch/grid-deployment/cgi-bin/index.cgi?var=documentation
[7]. Research & technological development for a Data TransAtlantic Grid (DataTag) http://datatag.web.cern.ch
[8]. Grid Physics Network (GriPhyN). http://www.griphyn.org
[9]. International Virtual Data Grid Laboratory (iVDGL). http://www.ivdgl.org/
[10]. Tierney, B., Aydt, R., Gunter, D., Smith, W., Taylor, V., Wolski, R., Swany, M. A grid monitoring architecture. Tech. Rep. GWD-PERF-16-2, Global Grid Forum, January 2002. http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-2.pdf
[11]. Web Services Architecture. W3C Working Group Note, February 2004. http://www.w3.org/TR/ws-arch/wsa.pdf.
[12]. Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective. Version 4, 2005.
[13]. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. A Resource Management Architecture for Metacomputing Systems. Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pg. 62-82, 1998.
[14]. R. Raman, M. Livny, and M. Solomon. Matchmaking: An extensible framework for distributed resource management. Cluster Computing, 2(2), 1999.
[15]. Kendall, S.C., Waldo, J., Wollrath, A., and Wyant, G. 1994. A note on distributed computing. Sun Microsystems, Technical Report TR-94-29.
[16]. P. Leach, M. Mealling, R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. July 2005, http://www.ietf.org/rfc/rfc4122.txt
[17]. Disk Pool Manager (DPM). POOL - Persistency Framework. Pool Of persistent Objects for LHC. http://lcgapp.cern.ch/project/persist
[18]. Patrick Fuhrmann. dCache, the Overview. http://www.dcache.org/manuals/dcache-whitepaper-light.pdf
[19]. CASTOR. http://cern.ch/castor/
[20]. MDS 2.2 Features in the Globus Toolkit 2.2 Release, http://www.globus.org/toolkit/mds/#mds gt2
[21]. R-GMA: Relational Grid Monitoring Architecture. http://www.r-gma.org/
[22]. LB Service User's Guide. https://edms.cern.ch/document/571273/
[23]. Инфраструктура публичных ключей. http://www.ietf.org/html.charters/pkixcharter.html
[24]. S. Farrel and R. Housley, An Internet Attribute Certificate Profile for Authorization, RFC3281 (2002)
[25]. A. Frohner, V. Ciaschini, VOMS Credential Format, EDG Draft, 2004
[26]. R.Alfieri, R.Cecchini, V.Ciaschini, L.dell'Angelo, A.Gianoli, F.Spataro, F.Bonnassieux, P.Broadfoot, G.Lowe, L.Cornwall, J.Jensen, D.Kelsey, A.Frohner, D.L.Groep, W.Som de Cerff, M.Steenbakkers, G.Venekamp, D.Kouril, A.McNab,

O.Mulmo, M.Silander, J.Hahkala, K.Lorentey. Managing Dynamic User Communities in a Grid of Autonomous Resources, Computing in High Energy and Nuclear Phisics, La Jolla, California, 24-28 March 2003.
[27]. The gridmapdir patch for Globus: http://www.gridpp.ac.uk/gridmapdir/