Р10-2011-55

В. В. Галактионов

GridCom, GRID COMMANDER: ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ДЛЯ РАБОТЫ С ЗАДАЧАМИ И ДАННЫМИ В ГРИДЕ

Галактионов В. В. Р10-2011-55
GridCom, Grid Commander: графический интерфейс
для работы с задачами и данными в гриде

Программный пакет GridCom обеспечивает автоматизацию доступа к задачам и данным в гриде. Графическая клиентская часть, выполненная в виде Java-апплетов, реализует Web-интерфейс доступа к гриду через стандартные браузеры. Исполнительная часть — Lexor (LCG Executor) — запускается в работу пользователем на машине для выполнения грид-операций UI (User Interface).

Работа выполнена в Лаборатории информационных технологий ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна, 2011

Galaktionov V. V. P10-2011-55
GridCom, Grid Commander: Graphical Interface
for Grid Jobs and Data Management

GridCom — the software package for maintenance of automation of access to means of distributed system Grid (jobs and data). The client part, executed in the form of Java-applets, realises the Web-interface access to Grid through standard browsers. The executive part Lexor (LCG Executor) is started by the user in UI (User Interface) machine providing performance of Grid operations.

The investigation has been performed at the Laboratory of Information Technologies, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna, 2011

WLCG (Worldwide LHC Grid Computing) — программно-вычислительный комплекс распределенной в планетарном масштабе архитектуры предназначен для сбора, хранения и обработки гигантского объема данных, генерируемых ускорителем LHC (Large Hadron Collider) [1].

В данной публикации описывается программный пакет**GridCom**, предназначенный для обеспечения автоматизации доступа к средствам распределенной системы WLCG (задачам и данным). Для понимания и применения этого пакета ниже будут приведены основные сведения и основы функционирования и архитектуры WLCG, иногда для упрощения называемого просто гридом. Полное руководство для пользователя грида находится в документации ЦЕРН [2].

Все операции с задачами и данными выполняются пользователем в командном режиме **CLI** (Command Language Interface). Громоздкие, неудобные и многословные конструкции языка управления задачами и данными (типа 40-символьных hash-кодов в идентификаторах задач и наименованиях файлов) представляют собой совершенно недружественный интерфейс пользователя к WLCG. Поэтому одной из главных целей данной работы было обеспечить более удобные средства для пользователя в форматах уже известных современных графических средств оперирования данными и задачами, в частности с использованием средств WEB-доступа к программному обеспечению WLCG. Надо отметить, что подобная проблема решается для частных задач, как, например, программа **AtCom** (Atlas Commander) специализировалась сугубо для работы с пакетами программ эксперимента ATLAS [4].

1. ВВЕДЕНИЕ В WLCG. ТЕРМИНОЛОГИЯ И ОСНОВНЫЕ СВЕДЕНИЯ

SE (Storage Elements) — вычислительные средства для хранения данных, **CE** (Computing Elements) — вычислительные средства для обработки данных, **WMS** (Workload Management System) — вычислительные средства для приема и распределения задач пользователя и контроля за их выполнением.

VOMS (Virtual Organization Management Service) — система организации пользователей WLCG в тематические группы (виртуальные организации) и распределения вычислительных ресурсов (квот) для счета задач и объемов данных. Функции VOMS выполняются VOMS-серверами, причем один VOMS-сервер может обслуживать несколько виртуальных организаций

и одна виртуальная организация может обслуживаться несколькими VOMS-серверами. Примеры названий виртуальных организаций: dteam — для разработчиков программного обеспечения WLCG; alice, atlas, cms — для пользователей экспериментальных групп.

gLite — программное обеспечение промежуточного слоя (Middleware), обеспечивающее взаимодействие (интерфейс) пользователя с WLCG.

UI-машина — компьютер, обеспечивающий доступ (User Interface) к средствам gLite.

Грид-сертификат — «входной билет» пользователя в WLCG, который, с одной стороны, однозначно идентифицирует пользователя, а с другой — обеспечивает безопасность WLCG от несанкционированного доступа. Грид-сертификат имеет два формата: **pem**-формат для применения в грид-среде и формат **pk12** для применения в стандартных WEB-браузерах, в частности для контроля доступа к документации и сервисам в ЦЕРН. Обоюдная конвертация форматов выполняется процедурой **openssl** (заметим, что формат pk12 применяется в ЦЕРН для облегчения процедуры генерации грид-сертификатов). Запрошенный пользователем грид-сертификат выдается (подписывается) сроком на один год централизованной службой **CA** (Certificate Authority) с подтверждением локальной службой **RA** (Registration Authority). Сертификат содержит важный атрибут его владельца — его имя в формате **SN** (Subject Name). Пример SN в формате, принятом в RDIG:

/C=RU/O=RDIG/OU=users/OU=jinr.ru/CN=Victor Galaktionov.

Для непосредственного выполнения операций с данными и задачами в WLCG используется **proxy-сертификат**, генерируемый самим пользователем и имеющий ограниченный срок (несколько часов) действия. Proxy-сертификат сопровождает задачи и данные пользователя в WLCG и определяет принадлежность его к виртуальной организации и, соответственно, права доступа к ресурсам грида. Как правило, генерация proxy-сертификата является начальной процедурой вхождения в WLCG и выполняется на UI-машине командами типа **grid-proxy-init**, **voms-proxy-init**, **glite-voms-proxy-init**.

LFC (LCG File Catalogue) — каталог размещенных в SE файлов. LFC для каждой виртуальной организации обслуживается отдельным LFC-сервером (централизованным или локальным). Некоторые виртуальные организации могут иметь несколько LFC-серверов. По этой причине имя LFC-сервера на UI-машине должно задаваться самим пользователем в переменной **LFC_HOST**.

JDL (Job Definition Language) — язык описания задачи для WLCG. Содержит требования задачи для выбора потенциального CE для ее счета, перечень программ и данных задачи [3].

GUID (Grid Unique IDentifier) — генерируемое уникальное имя файла, расположенного в SE. Уникальность имени обеспечивается 40-значным hash-кодом в идентификаторе имени, что по понятным причинам затрудняет его использование.

LFN (Logical File Name) — логическое имя файла, которое имеет простую иерархическую структуру, применяемую в операционных системах — цепочку директорий и простого идентификатора, назначаемого самим пользователем.

Существуют и другие названия файлов: **SURL** (Storage URL) и **TURL** (Transport URL), в той или иной степени отражающие местонахождение файла и средств доступа к нему, **alias** — альтернативное логическое имя файла, **replica** — копия (реплика) файла. Реплики файлов должны размещаться на разных SE.

Далее будут описаны некоторые стандартные технологические приемы для управления задачами и данными в WLCG, которые положены в основу при проектировании и программировании GridCom.

1.1. Job Management. Управление задачами. Работа с задачами в WLCG выполняется в несколько этапов.

Подготовка задачи заключается в ее описании на языке JDL в текстовом файле, как правило, с расширением имени **.jdl**. Это описание содержит основные параметры задачи:

• тип задачи;

• название исполнительного модуля;

• перечень файлов с программами, которые составят передаваемый в грид пакет;

• перечень файлов с данными, которые будут передаваться в грид в пакете (StdInput, InputSandbox);

• перечень файлов для результатов счета задачи (StdOutput, StdError, OutputSandbox);

• перечень грид-файлов, необходимых задаче для обработки данных и размещения результатов обработки (InputData и OutputData);

• другие требования задачи для выбора подходящего для счета CE.

В приложении 1 приведены примеры jdl-файлов.

Запуск задачи и контроль ее состояния выполняется пользователем в режиме CLI на UI-машине.

Запуск задачи: glite-wms-job-submit –a <jdl-файл>.

Задача передается в WMS, и после успешного синтаксического и содержательного анализа jdl-файла пользователь получает следующее сообщение: Connecting to the service https://lcgwms01.jinr.ru:7443/glite_wms_wmproxy_server, с именем WMS-сервера, принявшего задачу, а также идентификатор задачи **jobID** типа https://lcglb11.jinr.ru:9000/HdG_Pv8WwtVoVo0HyBK0pA.

Запрос *состояния* переданной задачи выполняется командой

glite-wms-job-status <jobID>.

Перечень возможных состояний задачи приведен в приложении 3.

При появлении статуса задачи типа **Done** (Done (Success) или Done (Failed)) можно затребовать на UI-машину результаты счета, описанные JDL-параметрами StdError и StdOut:

glite-wms-job-output <jobID> или

glite-wms-job-output –dir <dir_name> <jobID>.

В случае успешного выполнения команды получаем сообщение с указанием размещения результатов на UI-машине:

Output sandbox files for the job:
https://lcglb11.jinr.ru:9000/HdG_Pv8WwtVoVo0HyBK0pA
have been successfully retrieved and stored in the directory:
/afs/jinr.ru/user/g/gvv/glite31/gvv_HdG_Pv8WwtVoVo0HyBK0pA.

Результаты счета записываются на UI-машине в установленной UI-конфигуратором директории (первый вариант) либо в указанной пользователем команде директории (второй вариант). После этого задача принимает состояние **Cleared**.

1.2. Data Management. Управление данными. В WLCG используются несколько типов операций с файлами.

• *Массовая пересылка* данных применяется специальными уполномоченными администраторами для перемещения больших объемов данных между основными хранилищами данных, например, между Tier0 и Tier2. Эта задача выполняется сервисом **FTS** (File Transfer Service).

• *Пользовательский уровень* работы с файлами. Эти операции выполняются двумя группами команд: **lfc-*** для работы с LFC-каталогом и LCG Data Management Tools (**lcg_util**) для непосредственных операций с файлами (заведение, копирование, удаление и др.). В приложении 4 находится перечень основных опраций этого уровня.

• *Специальный* (нижний уровень) работы с файлами: **GSIFTP** (edg-gridftp-* и globus-url-copy).

1.3. Операции с файлами пользовательского уровня. Upload — копирование файла из локальной UI-машины на SE в WLCG и включение его имени в LFN-формате в LFC-каталог. Обязательные параметры команды — название виртуальной организации (для правильного выбора LFC-каталога) и имя локального файла в UI-машине. Остальные атрибуты грид-файла (SE и LFN-имя файла) устанавливаются автоматически (по умолчанию) либо могут быть определены пользователем. В любых операциях с LFC-каталогом должно быть установлено значение переменной LFC_HOST.

Примеры:

```
lcg-cr --vo dteam file:/home/doe/file1
lcg-cr --vo dteam -d lxb0710.cern.ch file:/home/doe/file1
lcg-cr --vo dteam -d lxb0710.cern.ch -l lfn:my{\_}alias1
file:/home/doe/file1
```

После успешного завершения копирования файла выдается сообщение о присвоении ему имени в формате GUID типа

```
guid:baddb707-0cb5-4d9a-8141-a046659d243b
```

Download — копирование файла из хранилища SE в локальную на UI-машине файловую систему.

Примеры:

```
lcg-cp --vo dteam -t 100 -v lfn:/grid/dteam/doe/myfile file:/tmp/
myfile
```

В приложении 4 находится перечень основных операций пользовательского уровня. Наиболее часто используемые операции с файлами (кроме вышеописанных):

- lfc-ls — просмотр LFC-каталога;
- lfc-mkdir — создание директории в LFC;
- lfc-rename — переименование директорий и файлов;
- lfc-chmod — изменение доступа к файлу или директории;
- lcg-rep — создание копии (реплики) файла;
- lcg-lr — просмотр списка реплик файла;
- lcg-aa — создание альтернативного логического имени файла (альяса);
- lcg-la — просмотр списка альясов.

2. JAVA-ТЕХНОЛОГИИ, ПРИМЕНЯЕМЫЕ В GridCom

Программы пакета GridCom написаны на языке Java [7], включают в себя практически все его технологические возможности и обеспечивают многоплатформенность их применения.

Java-апплеты. Java-апплеты используются для выполнения Java-программ в большинстве стандартных WEB-приложений — браузеров. Установка поддержки Java-программ в браузерах **JRE** (Java Runtime Environment) выполняется почти автоматически. Наличие же пакета **JDK** (Java Development Kit), включающего JRE, практически является обязательным в вычислительных системах общего пользования. Вызов и загрузка апплетов на персональной машине пользователя выполняются браузерами стандартными средствами языка HTML. Средства программирования в Java (классы **JApplet** и интерфейс **ActionListener**) обеспечивают подготовку программ такого типа. Использование апплетов открывает доступ пользователю и обеспечивает выполнение программ практически любой сложности через стандартный WEB-интерфейс. Пример вызова Java-апплета для GridCom:

```
    <applet code="appletGridCom.class" archive="GridCom.jar,
mysql.jar" width=450
```

```
height=40>
    <param Name="Log" value="0" >
    <param Name="CERN" value="/C=CH/O=CERN/OU=GRID/CN=
Rod Walker" >
    <param Name="RDIG" value="/C=RU/O=RDIG/OU=users/OU=jinr.ru/
CN=Ivan Petrov" >
    </applet>
```

Пользователь задает браузеру всего лишь задание на загрузку HTML-страницы с кодами вызова апплета:
http://wwwinfo.jinr.ru/∼gvv/GridCom

JDBC. Компонент JDBC (Java Database Connectivity) с объектами/классами библиотеки **java.sql.*** обеспечивает работу со стандартными реляционными базами данных, используя стандартные схемы подключения к базам и выполнения SQL-запросов. В данной реализации GridCom используется база данных **MySQL** из стандартного обеспечения LINUX-кластера ЛИТ ОИЯИ. Используются основные SQL Java-классы и их методы (функции): **Connection** (методы getConnection(), createStatement(), close()), **Statement** (методы executeQuery(), getStatement(), executeUpdate(), getResultSet(), close()), **PreparedStatement** (метод prepareStatement()), **Resultset** (методы getMetaData(), getString(), next(), getInt()), **ResultSetMetaData** (методы getColumnCount(), getColumnLabel()).

Для обработки нештатных ситуаций при выполнении SQL-запросов используется механизм исключений (перехват аварийных прерываний): **try/catch** и **SQLException**.

В GridCom используются 7 таблиц (для всех пользователей):

gridTasks — заготовки-шаблоны задач (templates);

gridJobExe — таблица запущенных на счет задач;

CEstatis — статистика использования CE;

jobData — таблица выполнения операций с данными;

gridRegister — таблица зарегистрированных пользователей GridCom;

gridCMD — таблица запросов и результатов UNIX-команд;

gridJobs — таблица задач со всеми их атрибутами (статус, jobID, CE и др.).

В каждой из таблиц пользователь GridCom *идентифицируется* с его **SN**-именем в грид-сертификате.

ИПИ — инструменты пользовательского интерфейса. Клиентская часть GridCom обеспечивает графический интерфейс пользователя с исполнительной частью пакета современными средствами графики языка Java ИПИ-компонентов **awt** и **swing** (классы java.awt.* и javax.swing.*), а также средствами компоновки графических объектов.

ИПИ содержит набор изобразительных средств типа *окон, кнопок, меток, полей ввода, списков* и многих других объектов: JFrame, JPanel, JButton, JLabel, JTextField, JTextAreaa, JList.

Средства компоновки ИПИ используются для размещения графических компонент ИПИ в окнах и панелях. Наиболее популярные из них BorderLayout, CardLayout, BoxLayout, GridLayout.

Меню. Важнейшей частью графического интерфейса пользователя является применение средств выбора действий — меню. В GridCom используются меню нескольких типов.

• **JMenuBar** — верхняя планка для меню в окне типа JFrame с последующим созданием ниспадающего меню **JMenu** с уточняющими элементами подменю **JMenuItem** и **JCheckBoxMenuItem.**

• **popup** — всплывающее меню с элементами подменю **JMenuItem**. Общепринято в графических системах вызывать popup-меню *правой* клавишей компьютерной мыши.

Обработка событий. Неотъемлемой и самой сложной частью любой интерактивной графической системы является реакция программ *обработчика событий* на предусмотренные изменения состояния объектов ИПИ — выбор элемента меню, нажатие (кликание) графической кнопки и др. В Java предусмотрен механизм такой реакции и написания программ обработчика событий:

• необходимо при создании главного графического объекта — окна типа JFrame — установить *возможность* создания механизма реакции на изменение статуса интерактивного объекта в этом окне. Это означает задать типы «прослушивания» для объектов этого окна, т. е. класс данного графического объекта должен *наследовать* (термин объектного программирования) Java-интерфейс обработчика событий, например:
public class GridCom extends JFrame **implements ActionListener, ItemListener**

• При создании конкретного графического объекта (кнопки или меню) задать тип «прослушивания», например:

```
menuItem = new JMenuItem("dteam");
menuVO.add(menuItem);
menuItem.addActionListener(this).
```

• Подключение обработчика событий к конкретному графическому объекту может выполняться двояко: можно добавить коды обработки при создании (описании) этого объекта либо организовать групповую обработку событий (для группы объектов). Естественно, обработка групповых событий более сложна, поскольку требует идентификации объекта. В этом случае пишутся программы-методы со стандартными именам, которые, следуя правилу полиморфизма, заменяют соответствующие заготовки в объектах «прослушки», указанных как параметры наследования **ActionListener, ItemListener** для суперкласса JFrame (см. приведенный выше пример).

Пример для первого типа:

```
JButton mybtn = new JButton("Do Something");
mybtn.addActionListener(
        new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        doMyAction();
                }
        }
).
```

Примеры для второго типа:

```
public void actionPerformed(java.awt.event.ActionEvent e) {
   String Command = e.getActionCommand();
}
public void itemStateChanged(ItemEvent e) {
     JMenuItem source = (JMenuItem)(e.getSource());
}
public boolean action (Event evt, Object what) {
}.
```

• Существуют отдельные методы обработчиков события, отслеживающие перемещение курсора мыши: mouseEnter(), mouseExit(), mouseDown(), mouseDrag() и др.

• Наиболее сложной является обработка событий с объектами типа **JTable** и **JTree** (таблицы и графические изображения иерархических структур, таких как файловая система) в частности для вызова popup-меню правой клавишей мыши.

• В GridCom реализовано множество из перечисленных выше типов обработки событий и их комбинаций.

Таблицы. Представление структур данных в виде таблиц **JTable** выполняется уже упоминавшейся выше подсистемой **swing** (классами javax.swing.*). Объекты типа JTable обеспечивают создание табличных данных, динамическое их изменение (добавление и удаление строк и столбцов), выделение (одиночная или групповая селекция), расцвечивание элементов таблицы, перераспределение столбцов вручную с помощью мыши. Важной особенностью объектов этого типа является возможность подключения *обработчиков событий* с помощью всплывающих popup-меню. Надо отметить, что создание таких обработчиков событий для объектов JTable является более сложной процедурой, чем для вышеупомянутых графических средств ИПИ. Таблицы типа JTable применяются для отображения объектов в компоненте GridCom «**Управление задачами**» для индикации состояния задач, что особенно важно для режима **polling** — наблюдения динамического, быстро меняющегося состояния запущенных в счет задач.

Иерархические структуры. Современная файловая система в общепринятом представлении имеет иерархическую структуру с неопределенной заранее глубиной вложения элементов. Элементами являются директории и файлы. Такие структуры хорошо описываются объектами типа JTree и представляются в виде *дерева*, отдельные элементы которого (*ветви*) могут динамически закрываться или открываться нажатием кнопки мыши. Как и для табличных данных, создание обработчиков событий и всплывающего popup-меню для объектов JTree является достаточно сложной процедурой. Эти объекты используются в компоненте GridCom «**Управление данными**» для обслуживания *LFC-каталогов* в WLCG и каталогов *локальной файловой системы* на UI-машине.

Thread — параллельные потоки. Программа, реализующая Java-поток, должна быть выполнена в виде класса, *наследующего* объект **Thread**, либо, по причине запрета в Java множественности наследования, реализовывать стандартный Java-интерфейс **Runnable**. Примеры декларации программ для организации потоков: class PrimeThread extends **Thread**; class PrimeRun implements **Runnable**.

Объект типа **Thread** содержит несколько методов организации и существования потоков: **start(), stop(), suspend(), resume()**; регулирования приоритетов: **getPriority()**, **setPriority()**. Однако важнейшим методом выполнения потоком алгоритма задачи по существу является метод **run()**. Схема построения такой программы-метода:

```
public void run() {
        try {
                  <коды программы>
                  sleep(1000);
        } catch (InterruptedException e) {}

}.
```

Важную функцию выполняет метод **sleep()** — установка *спящего* режима на заданное время для текущего потока.

Вся исполнительная часть GridCom (см. ниже) построена именно на выполнении пользовательских запросов в режиме потоков. Всего организовано девять потоков:

```
Merlin,     Command,      lcgSTATUS,     lcgDataCommand,
lcgSUBMIT,  lcgExecutor,  lcgJobCANCEL,  lcgOUTPUT,
lcgUnixCommand.
```

Все они выполняют независимо свою задачу с равными приоритетами. Для всех потоков установлено правило динамического перерасчета времени спящего режима (паузы): при повышении активности задачи потока пауза уменьшается и, наоборот, при уменьшении активности пауза увеличивается.

Граничные значения пауз и величина изменения паузы устанавливаются параметрически.

Runtime — системные процессы. Такие процессы создаются для выполнения команд операционной системы (ОС), в которой запущены в работу Java-программы (Windows или UNIX). Каждое Java-приложение имеет единственный класс **Runtime** для связи со средой (environment) ОС, в которой выполняется это приложение. Метод этого класса **Runtime.exec** формирует специальный процесс (объект) **Process**, специфический для данной платформы — *native process*. Весь стандартный ввод/вывод процесса (stdin, stdout, stderr) перенаправляется *родительскому* процессу через три потока (Process.getOutputStream(), Process.getInputStream(), Process.getErrorStream()). Метод **exec(String com)** этого объекта готовит *native process*, которому передается его входной параметр как команда для выполнения операционной системе. Процесс запускается по схеме **try/catch** для обработки возможных *исключений* при его выполнении. Результаты выполнения можно получить через вышеуказанные потоки данных.

Пример и схема выполнения команд:

```
   Runtime r;
   Process p;
   DataInputStream in;
   PrintStream out;
   String currentLine;
   String ERROR;
    String result = "";
   r = Runtime.getRuntime();
   try {
            String command = com;
            Process p = r.exec(command);
            in = new DataInputStream(p.getInputStream());
            BufferedReader err = new BufferedReader( new Input
StreamReader(p.getErrorStream()));

            while ((currentLine = in.readLine()) != null) {
                     result += currentLine + "\n";
            }
            in.close();
            while ((currentLine = err.readLine()) != null) {
                        ERROR += currentLine + "\n";
            }
             err.close();
   }catch (IOException e) {
                e.printStackTrace();
             }.
```

3. АРХИТЕКТУРА GridCom

Программный пакет GridCom состоит из двух частей: *клиентской* части GridCom и *исполнительной* **Lexor** (LCG Executor).

Клиент GridCom не требует установки каких-либо специальных программ на конечных пользовательских компьютерах, выполняется в операционных системах Windows или UNIX стандартными сетевыми приложениями-браузерами, обеспечивает интерактивное взаимодействие с пользователем в графическом режиме с применением Java-технологий.

• Апплет-структура пакета обеспечивает WEB-интерфейс пользователя к исполнительной части. Этим обеспечивается популярная в Сети парадигма *тонкого* клиента: минимум нагрузки и инсталляций для машин потребителя.

• Средства **swing** и **awt** используются для конструирования графических форм и объектов.

• Разнообразные интерактивные средства (меню, обработчики событий) обеспечивают динамическую схему формирования запросов пользователя.

• Средства **JDBC** обеспечивают обмен информацией с базой данных MySQL.

• Графические средства **JTable** и **JTree** обеспечивают в наглядной форме представление данных.

Две части клиентской программы **Job Management** (управление задачами) и **Data Management** (управление данными) выполняются в разных окнах и работают совершенно независимо. Программа формирует запросы пользователя в виде записей в базу данных *(семафоры* и данные), запоминает в очереди и периодически в режиме polling сканирует состояние запроса. Состояние семафора может быть следующим:

• есть запрос,

• запрос принят,

• есть результат,

• ошибка выполнения,

• time-out.

Обращение пользователя к клиентской программе GridCom в браузере: http://wwwinfo.jinr.ru/∼gvv/GridCom.

Серверный компонент Lexor выполняется только в специальных вычислительных системах с программным обеспечением доступа к WLCG — в UI-машинах. Таким, например, является LINUX-кластер ЦИВК ЛИТ ОИЯИ. Lexor получает запросы пользователя и формирует из них наборы команд для работы с локальной файловой системой и с задачами и данными WLCG. Для эффективного функционирования серверная программа выполняет запросы, используя *потоковый* механизм распараллеливания процессов, группируя их

по типам (работа с задачами, данными и др.). В Lexor применяются Java-технологии:

- JDBC — для работы с базой данных MySQL;
- Thread — для выполнения распараллеливания потоков;
- Runtime — для выполнения команд ОС Unix и операций с WLCG.

Каждый поток кроме выполнения заданной операции в режиме polling проверяет наличие в базе данных новых запросов (состояние семафора) для своей группы операций, по окончании выполнения операции записывает ее результаты в базу данных, проставляет значения семафоров типа «есть результат» либо «ошибка».

Lexor запускается программой shell-script **Lexor.sh** (см. приложение 5).

Пример:

```
export GC_LOC=/afs/jinr.ru/user/g/gvv/public/GridCom
glite-voms-proxy-init
$GC_LOC/Lexor.sh
```

Примечания.

1. Обе части GridCom (клиентская и исполнительная) находятся в публичной директории автора и доступны для копирования и установки в собственной директории даже для неискушенного потенциального пользователя.

2. Для выполнения операций GridCom пользователь должен иметь Grid-сертификат, а для запуска Lexor на UI-машине должен быть сгенерирован proxy-сертификат.

База данных. В GridCom используется база данных общего пользования MySQL из стандартного программного обеспечения UNIX-кластера ЛИТ ОИЯИ. В принципе, возможно использование любой другой реляционной базы данных, поскольку Java-средство JDBC для баз данных нейтрально к их производителю и требует лишь наличия соответствующих драйверов. Для MySQL JDBC-драйверы содержатся в свободно распространяемом пакете **mysql-connector-java-3.1.12-bin.jar.** При загрузке апплетов клиентской программы на машину пользователя этот пакет с драйверами также пересылается с WEB-сервера.

База данных в GridCom *невидима* для пользователя и применяется для синхронизации двух его компонент с помощью семафоров и обмена данными — запросов и их результатов, несмотря на то, что их совместная работа выполняется в принципиально асинхронном режиме. Каждый компонент GridCom можно в любой момент времени остановить (завершить) и продолжить также в любой момент без потери нити взаимодействия и результатов. В этом и заключается стабилизирующая роль применения базы данных. Существуют и другие изящные алгоритмы синхронизации клиентских и серверных приложений, такие как Web Services [6], Java RMI (Remote Method Invocation) и существовавшая до недавнего времени CORBA [5], однако для

их работы требуются неподъемные для пользователя громоздкие конструкции, такие как WEB-серверы типа **Tomcat** или же **rmiregistry** (для RMI).

На рис. 1 показана схема взаимодействия компонентов GridCom.

Рис. 1. Взаимодействие компонентов GridCom

4. УПРАВЛЕНИЕ ЗАДАЧАМИ В GridCom

Как уже упоминалось выше, управление задачами в GridCom выполняется загружаемым через Интернет Java-апплетом (WEB-интерфейс) графическими изобразительными средствами и интерактивными запросами пользователя. Все операции сводятся к выбору объекта (операция Select) в окне (мышью), минимальным действиям с ним (редактированию) и к выбору действия с ним в popup-меню. Выбранный пользователем тип действия над задачей трансформируется в команды gLite, которые с использованием механизма Runtime выполняются серверным компонентом Lexor в операционной системе UI-машины. Основные операции с задачами.

Подготовка задачи.

• Использование шаблона с табличными значениями:

Templates → Select → Create Job.

• Табличные значения в формате JDL: Jobs → Select → New → Create.

• Копирование и редактирование существующей задачи:

Jobs → Select → Clone/Edit → Create.

• Полный набор JDL-описания: Jobs → JDL → Example → Create.

• Использование существующего на UI-машине JDL-файла: Jobs → JDL → UI AFS.

Передача задачи в WMS (Workload Management System).

• Выбор задачи в таблице (Select).

• Передача задачи: Jobs → Select → Submit. Это одна из сложных процедур в Lexor, поскольку выполняется формирование JDL-описания задачи на основе табличных данных о ней.

• Групповая передача задач: Jobs → Submit All.

• Проверочная (режим match-list) передача задачи:

Jobs → Select → Math-list → Send.

• Результат проверки: Jobs → Select → Math-list → Show.

Контроль состояний задачи. Состояние (статус) задачи высвечивается автоматически (по умолчанию) в окне задач, при этом можно сделать следующее:

- обновить состояния всех задач в окне: Jobs → Refresh;
- установить или отменить режим polling для автоматического обновления: Jobs → Polling;
- получить подробности состояния задачи: Jobs → Select → View → STATUS;
- изменить тип высвечиваемой информации о задачах; иногда полезно знать идентификатор задачи, CE (computing element) и др. Меню: Jobs → Show columns;
- прекратить выполнение переданной задачи: Jobs → Select → Cancel;
- удалить задачу: Jobs → Select → Delete;
- удалить все задачи в таблице (очистить таблицу): Jobs → Delete All;
- посмотреть состояние прохождения задачи в WLCG: задание уровня информации: Jobs → Select → Logging-info → N (0|1|2); просмотр состояния: Jobs → Select → Logging-info → Show.

Результаты счета задач. После успешного завершения задачи результаты ее счета (выходные данные типа stdout и stderror), описанные в OunputSandbox, можно получить командами меню двух типов:

Jobs → Select → OUTPUT → get OUTPUT to: → default dir
Jobs → Select → OUTPUT → get OUTPUT to: → current dir

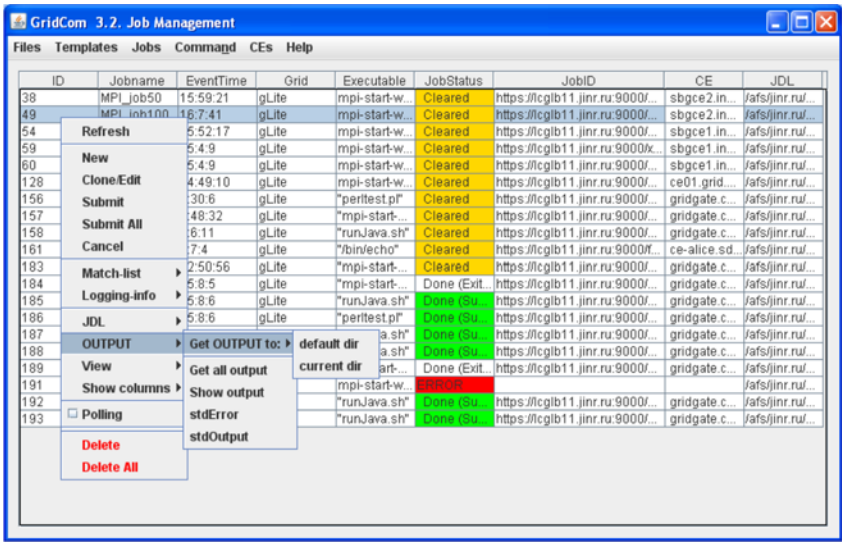

Рис. 2. Окно управления задачами и popup-меню

Посмотреть результаты выполнения этих операций можно командами меню:

Jobs → Select → OUTPUT → Show output
Jobs → Select → OUTPUT → stdError
Jobs → Select → OUTPUT → stdOutput

Примечание. После приема результатов счета состояние задачи принимает значение **Cleared**.

На рис. 2. приведен пример окна управления задачами со всплывающим popup-меню.

5. УПРАВЛЕНИЕ ДАННЫМИ В GridCom

Управление данными в GridCom выполняется в отдельном окне **Data management** (рис. 3) независимо от окна управления задачами. Основные типы операций с данными в WLCG описаны выше. Как и в случае окна Job Management, в клиентской части GridCom графические средства используются для выбора объекта, типа операции над ним и формирования запроса в исполнительную часть — Lexor, в которой эти запросы формируются в систему gLite-команд и выполняются на UI-машине.

Файловые системы WLCG и локальные файлы пользователя, каталоги которых являются иерархическими системами, хорошо представляются в виде

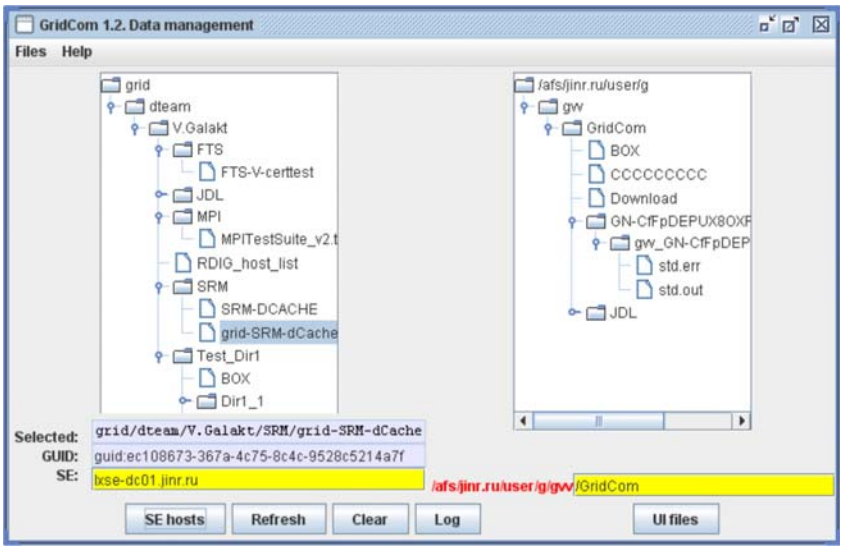

Рис. 3. Окно управления данными

графических древовидных структур. Окно Data Management содержит две главные панели:

• **LFC** (левая панель) — для представления структуры LFC-каталога;

• **UI** (правая панель) — для представления каталога файлов пользователя в HOME-директории на UI-машине.

Имеются также дополнительные поля для уточнения операций, например:

• SE — для выбора сервера хранения данных (Storage Element) при операциях создания Grid-файла или его репликаций (копий);

• GUID — для хранения имени файла в формате GUID, который существенно необходим в ряде операций, например при удалении файла.

Стандартная схема операций с файлами (директориями):

• select — выбор объекта в одной или в обеих панелях (выполняется компьютерной мышью);

• вызов popup-меню в одной из панелей для выбранного объекта (выполняется правой клавишей мыши);

• выбор операции в меню.

Пример последовательности действий для операции Upload — загрузки файла из локальной файловой системы в WLCG:

• панель LFC: операция select для директории (куда будет записан элемент LFC-каталога);

• панель UI: операция select для исходного файла на UI-машине;

• панель UI: вызов popup-меню;

• выбор меню Upload.

Примечания.

1. Для выбора SE-сервера (хранилища для нового файла) используется поле SE. Если это поле пустое, то выбирается стандартный для данного UI SE-сервер. Самостоятельно выбрать желаемый SE можно из списка, выдаваемого по кнопке **SE hosts**.

2. В некоторых случаях может выдаваться диалоговое окно с предупреждением либо предложением ввести дополнительную информацию, например, новое имя файла при переименовании.

Управление данными в GridCom выполняет минимально необходимый набор действий с файлами LFC и UI.

Перечень операций для Grid-файлов (popup-меню для панели LFC):

• для директорий: **Add alias, Add directory, Remove, Rename**;

• для файлов: **Download, get GUID, get alias, Remove, Rename, add Replica, list Replicas, del Replica, Refresh**.

Перечень операций для локальных файлов (popup-меню для UI-панели):

Upload, View, Remove, Rename.

ЗАКЛЮЧЕНИЕ

Первоначально пакет GridCom представлял собой коллекцию разнообразных наработок автора (работа с задачами и данными в LCG), которые использовались как средства автоматизации тестирования различных компонентов LCG и gLite. Эти работы выполнялись в ЦЕРН в рамках проекта EGEE/SA3. Кроме того, у автора был опыт применения средств языка Java в распределенных вычислительных системах, для работы с базами данных и для WEB-приложений. В настоящее время пакет является цельным программным продуктом, объединенным единым синхронизирующим механизмом для его клиентской и серверной частей, и свободным для применения и распространения.

Работа подтверждена Свидетельством об официальной регистрации программы для ЭВМ № 2006614056, зарегистрированным в реестре программ для ЭВМ Федеральной службы по интеллектуальной собственности, патентам и товарным знакам 27 ноября 2006 г.

Автор благодарен Калмыковой Л. А. за редакционную помощь в подготовке публикации.

ЛИТЕРАТУРА

1. *Фостер Я. и др.* Grid-службы для интеграции распределенных систем // Открытые системы. 2003. № 1. С. 20.
2. gLite 3.1 User Guide. https://edms.cern.ch/file/722398/1.3/gLite-3-UserGuide.html.
3. https://edms.cern.ch/file/555796/1/EGEE-JRA1-TEC-555796-JDL-Attributes-v0-4.doc
4. ATLAS Commander: an ATLAS production tool. http://www.slac.stanford.edu/econf/C0303241/proc/papers/MONT002.PDF.
5. *Галактионов В. В.* Java-технологии в распределенных системах с CORBA-архитектурой. Сообщ. ОИЯИ Р10-2002-63. Дубна, 2002.
6. *Галактионов В. В.* Web Services — сервис-ориентированная технология для распределенных вычислительных систем. Основные концепции, протоколы и спецификации. Сообщ. ОИЯИ Р10-2003-140. Дубна, 2003.
7. *Барлетт Н., Лесли А., Симкин С.* Программирование на Java. Путеводитель. Киев: Изд-во НИПФ «ДиаСофт Лтд.», 1996.

ПРИЛОЖЕНИЕ 1.
ПРИМЕРЫ ФАЙЛОВ С ОПИСАНИЕМ ЗАДАЧ НА ЯЗЫКЕ JDL

1.1. Пример простейшей задачи — файл hello.jdl.

```
Executable = "/bin/echo";
Arguments = "Hello World";
StdOutput = "Message.txt";
StdError = "stderr.log";
OutputSandbox = {"Message.txt","stderr.log"}.
```

1.2. Пример задачи для параллельных вычислений MPI — файл myMPI.jdl.

```
    Type = "Job";
     JobType = "MPICH";
     CPUNumber=10;
     VirtualOrganisation = "dteam";
     Executable = "mpi-start-wrapper.sh";
     Arguments = "mpi-test MPICH";
     StdOutput = "std.out";
     StdError = "std.err";
     RetryCount = 3;
     InputSandbox={"$HOME/gliteMPI/mpi-start-wrapper.sh",
     "$HOME/gliteMPI/mpi-hooks.sh", "$HOME/gliteMPI/mpi-test.c"};
     OutputSandbox = { "std.out", "std.err" };
     Requirements=
     other.GlueCEHostingCluster == "gridgate.cs.tcd.ie"
    && Member("MPI-START", other.GlueHostApplicationSoftware
RunTimeEnvironment)
    && Member("OPENMPI", other.GlueHostApplicationSoftware
RunTimeEnvironment).
```

ПРИЛОЖЕНИЕ 2. КОМАНДЫ УПРАВЛЕНИЯ ЗАДАЧАМИ

```
glite-wms-job-submit -a <jdl_file>
glite-wms-job-list-match -a <jdl_file>
glite-wms-job-status <jobID>
glite-wms-job-cancel <jobID>
glite-wms-job-output <jobID>
glite-wms-job-logging-info <jobID>
```

ПРИЛОЖЕНИЕ 3.
ПЕРЕЧЕНЬ ВОЗМОЖНЫХ СОСТОЯНИЙ ЗАДАЧИ В WLCG

Состояние задачи	Значения состояний
SUBMITTED	Задача принята в WMS
WAITING	Задача в состоянии ожидания в WMS
READY	Задаче назначен CE
SCHEDULED	Задача в состоянии ожидания на CE
RUNNING	Задача выполняется
DONE	Задача выполнена
ABORTED	Аварийное завершение задачи в WMS
CANCELED	Задача прекращена пользователем
CLEARED	Результаты счета задачи переданы на UI-машину

ПРИЛОЖЕНИЕ 4.
ОПЕРАЦИИ С ФАЙЛАМИ ПОЛЬЗОВАТЕЛЬСКОГО УРОВНЯ

Группа команд **lfc-***:

`lfc-chmod` — изменение типа доступа к файлам и директориям в LFC;
`lfc-ls` — содержимое или состояние файлов/директории;
`lfc-rm` — удаление файла/директории;
`lfc-mkdir` — создание директории;
`lfc-rename` — переименование файла/директории.

Группа команд **lcg_util**:

`lcg-cp` — копирование файлов в WLCG;
`lcg-cr` — операция Upload;
`lcg-del` — удаление файла или реплик;
`lcg-rep` — создание реплик файла;
`lcg-aa` — добавление альяса к файлу по заданному GUID;
`lcg-ra` — удаление альяса в LFC по заданному GUID;
`lcg-la` — список альясов для заданного LFN, GUID или SURL;
`lcg-lg` — запрос GUID для заданного LFN или SURL;
`lcg-lr` — список реплик для заданного LFN, GUID или SURL;
`lcg-ls` — информация о файле, заданном как SURL или LFN.

ПРИЛОЖЕНИЕ 5. SHELL-СКРИПТ ЗАПУСКА LEXOR НА UI-МАШИНЕ

```
    #/bin/bash
    if [ -z $1 ]
    then
     echo "Usage: Lexor.sh <vo> [0,1,2]"
     exit 1
    else
     vo=$1
    fi
    if [ -z $2 ]
    then
      log=0
    else
      log=$2
    fi
    #Check if exists directory GridCom
    d_gridcom="$HOME/GridCom"

    if [ -d $d_gridcom ]; then
     echo "$d_gridcom directory - exists"
    else
           echo "Directory $d_gridcom not exists, create it"
       mkdir $d_gridcom
    fi
    cd $d_gridcom

    pwd
    export USR=‘echo $USER‘
    export HM=‘echo $HOME‘

    ProxyExist=‘$GLITE_LOCATION/bin/glite-voms-proxy-info 2>/
dev/null | grep timeleft | wc -l‘
    ProxyExpired=‘$GLITE_LOCATION/bin/glite-voms-proxy-info 2>/
dev/null | grep "timeleft : 0:00:00" | wc -l‘
    echo "ProxyExist={\$}ProxyExist, ProxyExpired={\$}ProxyExpired"
    #ProxyExist=1
    #ProxyExpired=0
    if [ $ProxyExist -gt 0 -a $ProxyExpired -eq 0 ]; then
        #nop
        :
    else
       echo "Valid proxy is needed for Lexor execution!"
       $GLITE_LOCATION/bin/glite-voms-proxy-info
       if [ $ProxyExpired -gt 0 ]; then
         echo "Proxy credential expired!"
       fi
       exit 1
    fi

    # Set LFC server
```

```
    #     for special VO set var LFC{\_}HOST manually
    setlfc="lcg-infosites --vo $vo lfc"
    export LFC_HOST=‘$setlfc‘
    export LCG_CATALOG_TYPE=lfc
    echo "LFC_HOST=$LFC_HOST"
    echo "LCG_CATALOG_TYPE=$LCG_CATALOG_TYPE"
    lfc-ls -l /grid/$vo/V.Galakt

    export CLASSPATH={\$}GC{\_}LOC/mysql-connector-java-3.1.12-bin.
jar:{\$}GC{\_}LOC/Lexor.jar

    SN=‘grid-proxy-info -subject‘
    TL=‘grid-proxy-info -timeleft‘

    echo "VO=$vo"
    echo "Subject name=$SN"
    echo "Time left: $TL"
    java LEXOR "$SN" $TL $vo $log $USR $HM
```

Получено 10 июня 2011 г.

Редактор *Е. В. Сабаева*

Подписано в печать 12.09.2011.
Формат 60 × 90/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 1,5. Уч.-изд. л. 1,85. Тираж 250 экз. Заказ № 57418.

Издательский отдел Объединенного института ядерных исследований
141980, г. Дубна, Московская обл., ул. Жолио-Кюри, 6.
E-mail: publish@jinr.ru
www.jinr.ru/publish/