Accessing Grid Resources from Portals and Applications

FELICIA IONESCU, SILVIU POPESCU
Department of Electronics, Telecommunications and Information Technology
University Politehnica of Bucharest
Spl. Independentei Nr. 313 Sector 6 Bucharest - 60042
ROMANIA

Abstract: - The users of Grid-based applications and portals must be able to access Grid resources from different contexts, using high-level abstraction and full control of access parameters. Globus Toolkit, which is the most important solution for building Grid infrastructures, offers command line clients for access different resources such as data transfer using Grid FTP protocol and job submission using Gram services. But these clients are difficult to be used from different application and portals. This paper presents a complete and concise solution for the problem of access Grid resources from portals and applications through development of context-independent Java clients for accessing data transfer and execution management, and these clients can be used from applications and portals more efficiently and simpler than using command line clients offered by Globus Toolkit.

Key-Words: - Grid technology, Globus Toolkit, GridFTP, GRAM, Data transfer, Job submission

1 Introduction

Grid technology offer powerful computing resources for parallel and distributed applications and is of a great importance for research and education. The specific characteristic of the Grid towards conventional distributed systems is *"coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations"* [1]. A virtual organization is considered a set of users and institutions grouped together by specific sharing rules of resources. A resource is considered here a generic processing element which can be particularized in data (that is stored in files) and processing power (that applications need to run).

Grid technologies are now moving towards a service-oriented view by the definition of the Open Grid Service Architecture (OGSA) [2], which aims to standardize practically all the services one finds in a Grid application (job management services, resource management services, security services, etc.) by specifying a set of standard interfaces for these services. The base for the OGSA could, in theory, be any distributed middleware, but, for many reasons, Web Services [3] were chosen as the underlying technology.

Currently, the biggest role in development of Grid applications has open source Globus Toolkit [4], which includes a full range of core Grid services, commands and programming tools to quickly develop Grid applications or to run existing applications in a Grid environment. Globus Toolkit version 4 (GT4) implements WSRF (Web Services Resource Framework) specifications of the OGSA architecture [5].

In GT4, data transfer is done through specialized protocols, such as GridFTP [6], build upon the FTP protocol (File Transfer Protocol). GT4 also includes a command line client for data transfer between Grid nodes using GridFTP protocol. In Section 3 of this paper we will present Java implementation of a client program that makes Grid data transfer more flexible and simpler for programmers than corresponding GT4 command line client. This module can be included in different applications and portals, allowing user access to Grid resources.

A special role in Grid systems is played by execution management which refers to scheduling and running execution tasks (jobs) on processing nodes. GT4 offers a command line client for execution of jobs in Grid environment, using GRAM (Grid Resource Allocation and Management) services [6]. As for data transfer, we have developed a Java program for access GT4 execution management services, and this module can be included in different applications and portals.

This paper presents the development of context-independent Java programs for accessing Grid resources (data transfer and execution management) from applications and portals more efficiently and simpler than using command line clients offered by GT4. The paper is organized as follows: Section 2 presents a general overview of Globus Toolkit; in Section 3, the command line client and Java implementation of a client that controls data transfer in Grid environment are described; in Section 4, the command line client and Java implementation of a client that controls execution management of jobs in the Grid are described; Section 5 presents conclusions.

2 The Globus Toolkit 4 overview

Globus Toolkit version 4 (GT4) is a set of utilities and libraries built for serving Grid systems needs and requirements. A special part of the Globus architecture is a web service container which contains most important services needed in Grid infrastructure. Using Service Oriented Architecture (SOA) [7] leads to a great flexibility and large perspective of development. In a SOA new services can be created on top of the existing ones in a loosely coupled fashion.

The main components of the GT4 that can be used to develop Grid applications are:

- Security management (GSI - Grid Security Infrastructure);
- Data transfer (GridFTP, RFT – Reliable File Transfer);
- Execution management (GRAM – Grid Resource Allocation and Management);
- Data replication (RLS – Replica Location Service, DRS – Data Replication Service);
- Resource monitoring and discovery (MDS – Monitoring and Discovery System);

The Globus Alliance continues to improve this tools and services for better usage and integration in Grid infrastructure.

3 GT4 File Transfer System (GridFTP)

A major component in the Globus Toolkit 4 is represented by the file transfer system: GridFTP. This system is safe, robust, efficient, protected and optimized for data transfers in broadband WANs.

3.1 Overview of the GridFTP system

GridFTP extends the FTP (File Transfer Protocol) protocol so that it could face the problems in Grid infrastructures where special security policies are required and large quantities of data have to be transferred. From the security point of view, GridFTP has full support for Grid Security Infrastructure (GSI).

The GridFTP architecture model works on the same principles of the FTP model. It uses at least two independent socket connections: one for control, which transmits the commands and receives the answers and a channel for the actual data transfer. For connecting to a server, a control connection is open which is used for client-server communication through short command messages. A mutual authentication is accomplished between the client and the server based on X.509 certificates. When a data transfer is required one or more data channel connections are open which are used only for data flow.

GridFTP supports advanced features, like striping and parallelism, which can significantly improve data transfer performance for large data sets. Parallelism refers to the number of data connections used for transfer between one source and one destination. Striping [8] refers to transfers which use more than one data source. Data connection exists between each source and the destination and different stripes of the necessary data are transferred from each source reassembling the contents at the destination.

Other important features added to GridFTP protocol are automatically negotiation of TCP buffer size and data channel authentication which improve data security.

GT4 contains a command line client for GridFTP transfer operation (*globus-url-copy*), which supports most of the features described above. For executing file transfers, the client authenticates to the server with a proxy certificate and receives authorization for transfer.

3.2 Implementation of a GridFTP Java client

We wanted to implement a Java module to extend the possibilities of *globus-url-copy* command, and, also to be used in different contexts (JavaServer Pages or applications).

The access to the Grid resources can be addressed using Java Commodity Grid (CoG) Kit [9], that provides the basic APIs to the Grid to allow access to GridFTP servers, the classic GRAM services, MyProxy server [10] and other Grid components. Java CoG Kit allows Grid users, Grid application developers, and Grid administrators to use, program, and administer Grids from a higher-level framework.

Using Java COG (more specifically the COG-jglobus. library), we developed *GridTransferBean* module, which can be used for GridFTP access.

The module permits doing data transfers from a server to local host or controlling the transfer between to server (third-party transfer). It supports features like data connection parallelism and striping. The striped transfer depends on the starting mode of the server. The server must control several data nodes which will be used for data striping.

In this module we define an “appended” transfer for appending the contents of a file when the file exists at the destination. If appended option is not used, the destination file is overwritten.

For transfer of multiple files between a source and a destination, we defined a “multiple” transfer that doesn’t reinitialize the data connections and use those already created.

For a transfer to be started, a mutual authentication must be done between server and client based on X509 certificates signed by a trusted certificate authority. We use authentication based on a proxy file, which contains

a proxy certificate. The proxy file must exist on the host that is running a client implemented with *GridTransferBean* class. This proxy file should be already created during users' authentication to the Grid infrastructure.

In Fig. 1, the activity diagram of a GridFTP transfer using *GridTransferBean* class is shown. For initializing a transfer we need to set the parameters needed for source and destination like host names, server ports and file names. A typical transfer starts by invoking the *doTransfer()* method which returns a *TransferOk* string if the operation was successful, otherwise an exception is thrown and an error message is returned. Based on the type of transfer selected through the parameters, the different methods from the activity diagram are called by the method *doTransfer()*.

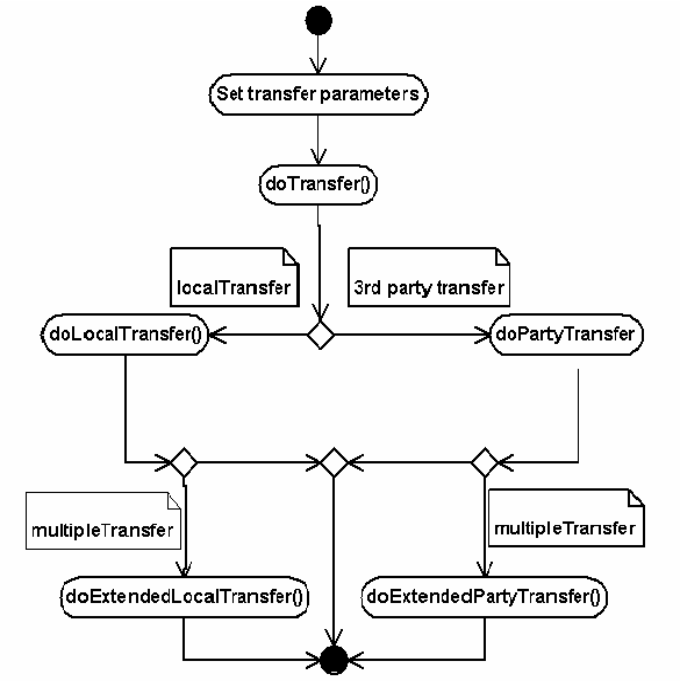

Fig.1 GridFTP transfer using *GridTransferBean* class.

The type of transfer can be one of those summarized below:

- *localTransfer* – a local transfer is considered a transfer from the server to the local machine;
- *appendTransfer* – this type of transfer has relevance if the destination files already exist and is used to append the content of a source file to a destination file;
- *stripedTransfer* – specifies that a server started in "stripe" mode is used, transferring data stripes from more data nodes;
- *multipleTransfer* – this is used when more files are needed for transfer and will reuse the connections already created to transfer the files.

When this module is used from a Java application, the class *GridTransferBean* can be included in application package and transfer parameters can be obtained from different variables of the program. When the module is used from a JSP page, the desired transfer parameters can be obtained from a HTML form.

We have used this module embedded in JSP pages, which are resources (assets) of shareable learning objects in a Grid-based e-learning platform [11].

4 GT4 Execution Management

Execution management tools are concerned with the initiation, monitoring, management, scheduling, and/or coordination of remote computations. GT4 supports the Grid Resource Allocation and Management (GRAM) interface as a basic mechanism for these purposes.

4.1 Overview of the GT4 GRAM services

GRAM is composed of a set of services deployed in GT4 Java container (Java WS Core) that can be used in conjunction with different local schedulers (*Fork*, PBS, LSF, *Condor*, etc.), which control execution over local resources. *Fork* is a scheduler designed by Globus team for local use purpose, which uses standard Unix *fork()* method to spawn simple processes.

GRAM uses very well defined structures named jobs. A *job* is an execution structure which passes through a number of states during its lifetime. For example, states like *StageIn* or *StageOut* refers to the process of transferring files prior or after the job execution. When a job executable is executed, the job stays in *Active* state; when a job has finished execution it is put in *Done* state, which refers to a successful execution or *Failed* state, which shows that the execution was unsuccessful.

GRAM services use several others GT4 services to accomplish different tasks (Fig. 2). GRAM uses RFT (Reliable File Transfer) service for file transfer prior and after the job execution process; RFT needs GridFTP server access for explicitly call data transfer commands. GRAM also needs Delegation service for credential delegation to other services like RFT.

For job submission operation, Globus Toolkit provides a command line client, *globusrun-ws*, which requires different parameters such as: the job executable program, execution parameters, files to be transferred prior or after job execution. Also, a complex cleanup operation to remove unnecessary files created by the job executable could be needed.

All this parameters can be specified in an XML job description file. The Globus Alliance defines a job description file XML Schema [12], which can be used to create from simple to complex job files in XML format.

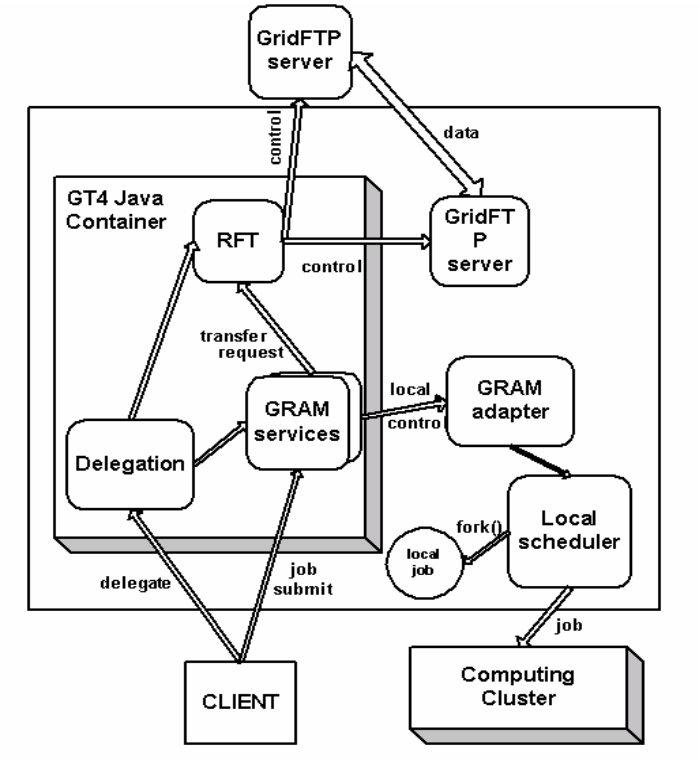

Fig.2 GRAM architecture.

The job description files are translated to a format readable for the local scheduler by using GRAM scheduler adapter.

4.2 Implementation of a GRAM Java client

We wished to implement a GRAM client as a Java module (package) that can be integrated in an application or a web interface (like a JSP page).

We have developed *JobSubmit* class that uses an instance of *org.globus.exec.client.GramJob* class provided by GT4 API, which allows job operations such as: creating, submitting and destroying. This class implements *GramJobListener* interface, which specifies methods for notifications about job status.

In order to access GRAM services and execute further delegations, the client needs a proxy certificate that is given through a file. The delegation is needed when the job has to access other Grid services or when file staging is required.

A job can be submitted by calling the *submit()* method after initializing specific parameters like destination host, the scheduler that will be used (we use by default *Fork*) and the job description file. If the job has been sent with success, a job uniquely identifier is returned, otherwise an exception occurs an error message is returned.

Fig. 3 shows the steps executed by different software components for job submission using Java client (*JobSubmit* class) included in a JSP page and using GRAM services, Delegation, RFT service and GridFTP servers.

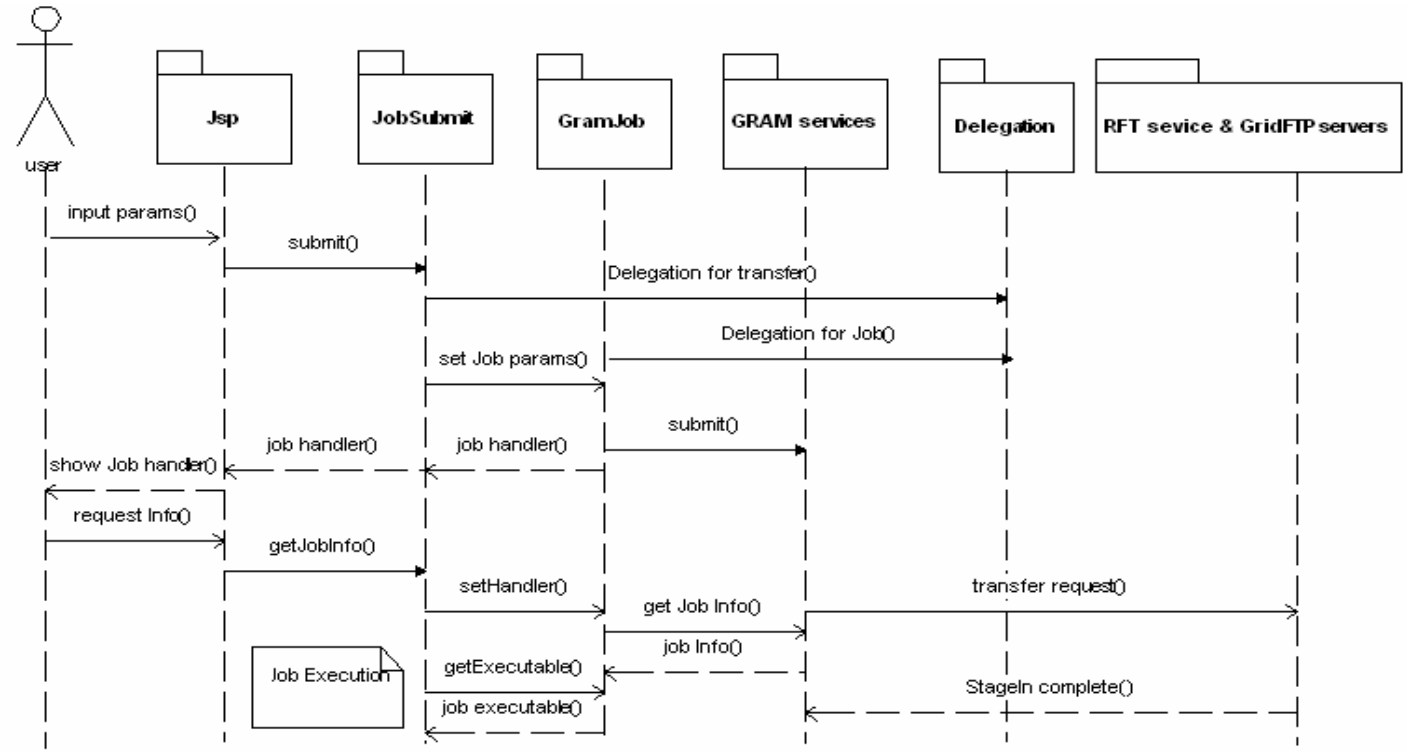

Fig. 3. Job Submission using Java client – Sequence Diagram

In the *submit()* method are accomplished most of the operations with the job entity. After creating the security context using credentials obtained from the proxy file a *GramJob* instance is created and a job is submitted to the destination's *ManagedJobFactoryService*, which is the service used as endpoint for job submission. By default https protocol is used for transport and port 8443 for connection.

By implementing the *GramJobListener* interface, a *stateChanged()* method is provided that is called every time a job status changed. We use this to store the job status in a class parameter. At any time, the job status can be retrieved calling *getJobStatus()* method. The job resource is destroyed when the job state is *Done* or *Failed*, which means the job has finished. This behavior is intended for interactive jobs for which the user have to wait to finish and is not always a desirable scenario. We could also use "batch mode" for submitting the job and without waiting for the job to finish. In this way the job status can be retrieved at a later time based on the job handler returned by the *submit()* method.

The job submission process presented above uses a job description file to set job parameters. The class *JobFileCreate* can be used to create a job description file in the XML schema format provided by Globus. This class contains a *createJobFile()* method, which receives different input parameters. Thus different job files can be created: with an executable using a variable number of parameters, using a variable number of *stageIn*, *stageOut* or cleanup elements.

As for GridFTP client, this Java GRAM client can be used from a Java application, or from a JSP page, allowing flexible execution context and simpler programs than using command line client provided in Globus Toolkit.

5. Conclusion

The access of Grid resources from applications and portals is an important operation in Grid environments. In this paper we have presented the implementation of two context-independent Java clients, for GridFTP data transfer and for GRAM job execution that make Grid data transfer and job execution more flexible and simpler for programmers than corresponding GT4 command line clients.

We have included these clients in the portal of a Grid-based e-learning platform and in shareable learning objects. Thus, after proper authentication, users can easily submit execution jobs or starting GridFTP transfers through friendly interfaces.

Acknowledgment

This work was developed under Romanian Grant CNCSIS code 44 (UPB part 14) started in 2006.

References:

[1] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *Intern. Journal of Supercomputer Applications and High Performance Computing*, Vol. 15 No. 3, 2001, pp. 200-222.

[2] I. Foster, C. Kesselman, J.M. Nick, S. Tuecke. The Physiology of the Grid: An Open Grid Service Architecture for Distributed System Integration, Proc. of Open Grid Service Infrastructure WG, Global Grid Forum, June 2002

[3] H. Kreger. Web Services Conceptual Architecture, IBM Technical Report WCSA 1.0, 2001.

[4] Globus Project, www.globus.org

[5] M. Little, J. Webber, S. Parastatidas, Stateful Interactions in Web Services: A Comparison of WS-Context and WS-Resource Framework, *Web Services Journal. April 30, 2004*.

[6] Grid FTP protocol, http://www.globus.org/toolkit/docs/4.0/

[7] http://www-306.ibm.com/software/solutions/soa/

[8] W. Allcockm J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, *The Globus Striped GridFTP FrameWork and Server*, SC`05, ACM Press, 2005.

[9] G. von Laszewski, J. Gawor, S. Krishnan, and K. Jackson, Commodity Grid Kits Middleware for Building Grid Computing Environments in Grid Computing, *Communication Networking and Distributed Systems*, John Wiley and Sons, 2003.

[10] Jim Basney, Marty Humphrey2, Von Welch, *The MyProxy online credential repository*, Software – Practice and Experience, 2005, 1-17

[11] Felicia Ionescu, Vlad Nae, Elena-Cristina Stoica, *Development of a Grid-based Learning Management System*, Proceedings of the 11th WSEAS International Conference on Computers, Agios Nikolaous, Crete Island, Greece, July 26-28, 2007, pp. 294-298

[12] http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/gram_job_description.html