Implementation & Testing of Grid Infrastructure

Index

Abstract

This thesis studies the fundamental technologies lying under *Grid Computing*. Another objective is to study how these technologies can be implemented and used in a variety of current applications and how they can collaborate with relevant projects and research performed within the framework of the Technical University of Crete (TUC).

Grid Computing allows people to share computing power, databases and other on-line tools securely across institutions without sacrificing local autonomy.

Globus Toolkit has emerged as the de facto standard middleware for Grids. It constitutes of open-source code that has been developed by the Global Research Community as means to implement the principles of Grid Computing. Although it was conceived mainly to achieve a better scientific collaboration and resource sharing in solving complex problems, it has almost been deployed in every sector of our life.

In this thesis four Linux Kernel workstations have been used. More details can be found in Section 4 regarding deployment and implementation. On those machines, we have installed the Globus Toolkit 4.01 and its components: GridFTP for data transfer, Reliable File Transfer (RFT), and OGSA-DAI database management. In Section 4 we show some GridFTP application tests that have been run in cooperation with the University of Plymouth. Similar to GridFTP, RFT is a service used to transfer data. Using RFT service, a user can control many data transfer aspects such as block size, TCP buffer size, etc. which GridFTP doesn't provide. RFT has also been tested successfully with the University of Plymouth. OGSA-DAI is a middleware, which supports the exposure of data resources such as relational databases and their remote management in Grid environments. In an attempt to enrich our database management system we had to install the java servlet Tomcat container. We have finally installed and configured a Condor job management system.

Some Facts about Grid Computing

The aim of this thesis is to study the concept of "Grid Computing" [1] [2] [3]. Nowadays, one can say that Grids constitute the evolution step after the internet.

Let us see some useful facts:

Period 1986 to 2000:

- Computers: 500 times faster
- Networks: 340000 times faster

Period 2001 to 2010:

- Computers: 60 times faster
- Networks: 4000 times faster

It is concluded that:

- Computer speed doubles every 18 months
- Network speed doubles every 9 months.

A question arises: how can someone benefit from so fast CPUs and also so fast networks in an efficient way?

The Globus project [2] provides a complete software toolkit (Globus Toolkit) that makes it easier to build computational grids and grid-based applications.

Globus Toolkit [1] has emerged as the de facto standard middleware for Grids. The Globus Toolkit is a result of scientific work trying to identify mechanisms and to develop software code that makes scientific collaboration and resource sharing possible in the field of complex problems solving.

The Globus Toolkit is freeware and open-source. It has been developed under the requirement to manage:

- resources
- data
- users

Its main development stages since late 1990s are:

- GT1 - 1998
- GT2 - 2002
- GT3 - 2003-2004
- GT4 - 2005

Until May 2005, GT3 was the latest available release of Globus Toolkit although not so stable and documented as GT4 which has been released afterwards.

Moreover, as Ian Foster (one of the inventors of Globus) said in an interview, to represent the release of GT4, “GT3 was not of production quality”.

Having the experience of dealing with GT3, it helped a lot in the installation and configuration of GT4 release when it was made available.

1 Introduction

It is practically impossible nowadays to do science without computers. Scientists are facing increasingly complicated problems which require much more than a single computer's capabilities [4]. Often, a single computer, a cluster of standard computers or even a special-purpose supercomputer is not enough for the calculations scientists really want to do. Although, computers are improving incredibly fast, they still do not keep up with the increased demands of the scientists. Due to that, it's very difficult, very expensive and sometimes impossible to achieve certain scientific goals with current computer technology [4].

Here are some issues that do concern scientists:

- The amount of data scientists need is huge. Moreover, data is stored in different locations. Satellite images of the Earth can be a good example. It might take a lot of time to copy the data one needs to one central computer in order to analyze it. So ideally the scientist wants to do the computation where the data is.
- The amount of calculations the scientist has to do is huge. For example, simulating the effect of thousands of potential drug molecules on a protein related to some disease.
- A scientific team with members around the globe wants to share large amounts of data and do complex analysis of the data rapidly online together.

To solve these problems the following idea was put forth:

1. of having huge storage space so they would never have to worry where to put the data,
2. of having nearly large computing power available for their institution whenever they need it,
3. of being able to collaborate with distant colleagues easily and efficiently, safely sharing with them resources, data, procedures and results.

If we take a look at the world today, we can see that it contains a lot of computers (several million). These could be desktop PCs, workstations, mainframes and supercomputers. Of course, these computers belong to many different people (students, doctors, secretaries…) and institutions (companies, universities, hospitals…). Most of these computers are probably connected to the internet. Imagine now, that all these computers around the world are connected together. It's here where the idea of the Grid comes out.

If a scientist wants to run a colleague's molecular simulation program, (s)he would no longer need to install the program on his machine. Instead, he could just ask the Grid to run it remotely on his colleague's computer. In fact, he wouldn't need to ask the Grid anything. It would find out the best place to run the program, and install it there [4].

If one needs to analyze a lot of data from different computers all over the globe, he could ask the Grid to do this. Again, the Grid could find out where the most convenient source of the data is without me specifying anything and do the analysis on the data wherever it is. Moreover, if one needs to do some analysis interactively in collaboration with several colleagues around the world, the Grid would link computers up and figure out who should be able to take part in this common activity.

Here is a look at how the evolution of computing has naturally led to the concept of the Grid.

Distributed computing

Nowadays, whenever there is a problem due to lack of computing power (a complicated calculation or an application that requires more computing power than a single computer can provide) the solution is to link computer resources from across a business, a company or an academic institution. The network of computer is then used as a single, unified resource. This solution is called “distributed computing” [4], and this term refers to just about any system where many computers solve a problem together. Grid computing is, in a sense, just one species of distributed computing. There are many others, a few of which are listed below.

Metacomputing

Metacomputing [5] generally refers to building a system out of several disjoint systems, possibly located in different administrative domains. Running a parallel computation over several high performance computers, or building a system to visualize the output from data coming from a weather satellite can be considered examples of metacomputing. Metacomputing was very popular in the early nineties, which involved linking up supercomputer centers with what was, at the time, high speed networks.

Cluster Computing

Many years ago, back in the last century, scientists put some PCs together and got them to communicate. Their aim was trying to kill the expensive “supercomputer”. Many commercial companies now offer clusters of PCs as a standard off-the-shelf solution.
Clusters [6] can have different sizes. One of the big advantages of this approach is "scalability": a cluster can grow simply by adding new PCs to it. Of course there are limits, because somehow the computers have to communicate with each other, and this starts to get pretty hairy when there are many computers. But clusters of hundreds of computers are not uncommon nowadays.

Peer to Peer computing

You must have heard also about “Napster” [7], the website that used to let music fans share music files from all over the world. By downloading a piece of software onto your hard drive, you could connect to a network of other users who have downloaded the

same software. Users only had to specify which information on their hard drive was public, and could access what others had made public.
In this way, computers can share files and other data directly, without going through a central server.

Local Grid computing

Nowadays, for problems that can be divided into many smaller problems, all independent of each other, the solution is to link computer resources from across a business, a company or an academic institution. The network of computers is then used as a single, unified resource.

Dedicated software efficiently matches the processing power required by any application with the overall availability. One popular type of software for linking computers in institutions like universities is Condor [8][9]. Condor is a type of software often referred to as "middleware" [1][3] , because it is not the operating system - the program that runs the computer - nor is it an application program running on the computer, but it is "between" these two, making sure that the application can run optimally on several computers, by automatically checking which computers are available. No more wasting time waiting for available computing power while systems in the next office remain idle.
Like clusters, local Grid computing is scalable - you can keep adding more PCs and workstations, within reasonable limits. Often the connection between the computers in such a system is a local area network, although it can also be via Internet. Usually the computers are geographically close together, for instance in the same building, and belong to the same administrative domain.
Local Grid computing is limited to a well-defined group of users, a department or several departments inside a corporate firewall, or a few trusted partners across the firewall. Also, such systems typically pool the resources of some dedicated PCs as well as others whose primary purpose is not distributed computing.

Grid computing

Grid computing [3] can be seen as the evolution of local Grid computing to the global scale, made possible by the advent of very high-speed Internet connections, and of powerful computer processors that are able to run quite complex middleware in the background without disturbing the task that the computer is trying to handle.

As Internet connect speed increases, the difference between having two PCs in the same office, the same building, the same city or the same country shrinks. By developing sophisticated middleware which makes sure that widely distributed resources are used effectively, Grid computing gives the user the impression of shrinking the distances further still. In addition, as the middleware gets more sophisticated, it can deal with the inevitable differences between the types of computers that are being used in a highly distributed system, which are harder to control than within one organization. One of the most popular middleware packages today is called Globus [2], and it is essentially a software toolkit for making Grids. With such middleware, the aim is to couple a wide variety of machines together effectively, including supercomputers, storage systems, data

sources and special classes of devices such as scientific instruments and visualization devices.

What makes "Grid Computing" different from "Local Grid Computing" is that Grid Computing focuses more on large scale sharing, which goes beyond institutional boundaries. Also, Grid computing leans more to using dedicated systems, such as scientific computer centers.

Finally, Grid computing aims to use resources that are not centrally controlled. The sharing is across boundaries - institutional and even national - which adds considerable complexity, while bringing also huge potential benefits.

One definition of Grid computing, by Ian Foster, one of the persons who helped coin the term, distinguishes it from other forms of computing.

The definition is that Grid must satisfy three criteria:

1. no central administrative control of the computers involved (that eliminates clusters and farms, and also local Grid computing),
2. use of general-purpose protocols,
3. high quality of service (that eliminates peer-to-peer and means that Grids should not rely on cycle scavenging from individual processors, but rather on load balancing between different independent large resources, such as clusters and local Grids).

Another distinction is that a Grid could in principle have access to parallel computers, clusters, farms, local Grids, even Internet computing solutions, and would choose the appropriate tool for a given calculation. In this sense, the Grid is the most generalized, globalized form of distributed computing one can imagine.

The five big ideas behind the Grid

Of course, there are many big ideas behind the Grid. And of course, some of them have been around long before the name Grid appeared. Nevertheless, if you look at where the software engineers and developers who are building the Grid are spending their time and effort, then there are five big areas.

The most important is the **sharing of resources** on a global scale. This is the very essence of the Grid. Then, although it is hardly a novelty, **security** is a critical aspect of the Grid, since there must be a very high level of trust between resource providers and users, who will often never know who each other are. Sharing resources is, fundamentally, in conflict with the ever more conservative security policies being applied at individual computer centers and on individual PCs. So, getting Grid security right is crucial.

If the resources can be shared securely, then the Grid really starts to pay off when it can **balance the load** on the resources, so that computers everywhere are used more efficiently, and queues for access to advanced computing resources can be shortened. For this to work, however, communications networks have to ensure that **distance no longer matters** - doing a calculation on the other side of the globe, instead of just next door, should not result in any significant reduction in speed.

Finally, underlying much of the worldwide activity on Grids these days is the issue of **open standards**, which are needed in order to make sure that R&D worldwide can contribute in a constructive way to the development of the Grid, and that industry will be prepared to invest in developing commercial Grid services and infrastructure [4].

The Grid takes its name from, similar in concept, electrical "power grid".

Electrical power grid	**The Grid**
When you plug-in your toaster your only concern is if you have got any bread. You usually never worry about where the electricity you are using comes from, if it is from coal, from the wind or from a nuclear plant. You know that whatever appliance you plug into a wall socket, it will get the electrical power you need to do the job.	When you sit in front of your computer your only concern will be if you have got a smart idea of what you want to do (a useful calculation, the design of a new engine, …).You will know that whatever computer you plug into the Internet, you will get the computing power and storage capacity you need to do the job.
The infrastructure that makes this possible is called "the power grid". It links together power plants of many different kinds with your home, through transmission stations, power stations, transformers, power-lines and so forth.	The infrastructure that makes this possible is called "the Grid". It links together computing resources such as PCs, workstations, servers, storage elements, and provides the mechanism needed to access them.
The power grid is transparent: you needn't worry about how and where the electrical power you are using is generated.	The Grid will be transparent: you needn't worry what computer processes you request and where the data is that it needs. So-called middleware will assign your task to the resource best suited to perform it and will work out the best way to locate and retrieve the data you need.
The power grid is pervasive: electricity is available essentially everywhere and you can simply access it through a standard wall socket.	The Grid will be pervasive: remote computing resources will be accessible from different platforms, including desktop, laptop, but also PDAs and mobile phones, and you will simply access the Grid through a web browser.
The power grid is a utility: you ask for electricity, and you get it. You also pay for what you get.	The Grid is a utility: you ask for computer power or storage capacity and you get it. Ultimately, you will also pay for what you get, although in an initial phase, scientists may donate resources in an effort to get the technology in wide circulation.

Table 1.1: Power Grid vs. Grid Computing

The structure of this thesis is formulated as follows:

Section 1 is this introduction.

Section 2 presents the principles and goals of Grid security. It starts with the introduction to Cryptography and symmetric/asymmetric cryptosystems, where Grid Security Infrastructure (GSI) is based on. The asymmetry is the result of the existence of a trusted Certificate Authority (CA) which is responsible for issuing the users' certificates. In a few words, Section 2 describes the steps that should be followed by users in order to become authenticated (ensure that they are who they say they are) and authorized (decide who may access data).

Section 3 introduces Condor. Condor is a workload management system for compute-intensive jobs. Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Jobs are submitted to run serially or in parallel to a Condor pool through a queue. Condor has to choose when and where the jobs should run based upon current workload conditions and job specified requirements. It also monitors their progress and ultimately informs the user upon completion.

Section 4 constitutes the deployment and implementation part of this thesis. We present the hardware that has been used, as well as all the software components that have been installed, configured and tested. Specifically, we emphasize on the following basic grid application tools like GridFTP, Reliable File Transfer (RFT), OGSA-DAI database management and Grid Resource & Allocation Management (GRAM).

Section 5 gathers all the conclusions of our engagement with Grid computing so far.

Section 6 attempts a short definition of some interesting areas which could be researched and worked on in the future.

2 Grid Security

2.1 Goals of Security

The goals of security are three:

1) prevention – prevent attackers from violating security policy,
2) detection - detect attackers' violation of security policy and
3) recovery - stop an attack, access and repair damage and continue to function correctly even if the attack succeeds.

Obviously prevention is the ideal scenario. In this case there would be no successful attacks. Detection occurs only after someone violates the security policy. It is important when a violation has occurred that it is reported swiftly. The system must then respond appropriately. Recovery means that the system continues to function correctly [9].

Security concerns deal principally with data. Additional concerns deal more with people and their actions.

Talking about data, there are three concerns:

1. Confidentiality,
2. Integrity,
3. Availability.

Confidentiality means that data is only available to those who are authorized.
Confidentiality is aimed at different issues. The content of a packet during a communication has to be secure to prevent external users from stealing the data. In order to prevent unauthorized users retrieving secret information, a common approach is to encrypt data by the user before sending. On the receiving end, the receiver can extract the original information by decrypting the encrypted text [9].

Integrity means that data is not changed except by controlled processes. In other words, integrity is the protection of data from modification by unauthorized users. This is not the same as data confidentiality. Data integrity requires that no unauthorized users can change or modify the data concerned.

Availability means that data is available when required. The term availability addresses the degree to which a system is operable and in a usable state.

Security concerns dealing with people can be the following:

- Authentication: Ensuring that users are who they say they are;
- Authorization: Making a decision about who may access data;
- Assurance: Being confident that the security system functions correctly;

- Non-repudiation: Ensuring that a user cannot deny an action;
- Auditability: Tracking what a user did to data.

  - ❖ Authentication implies ensuring the right user executes the granted services or the origin of the data was from the real sender. There are a large number of techniques that may be used to authenticate a user – passwords, biometric techniques, smart cards or certificates. Before starting services between a server and client, there should be a mechanism to identify the privilege of the user.

  - ❖ Authorization is often a first step towards providing the service of authentication. Authorization enables the decision to allow a particular operation when a request to perform the operation is received.

  - ❖ Assurance is the counterpart to authorization. Authorization mechanisms allow the provider of a service to decide whether to perform an operation on behalf of the requester of the service. Assurance mechanisms allow the requester of a service to decide whether a candidate service provider meets the requesters' requirements for security, trustworthiness, reliability or other characteristics. Assurance can be implemented through certificates signed by a trusted third party.

  - ❖ Non-repudiation is the concept of ensuring that a contract, especially one agreed to via the Internet, cannot later be denied by one of the parties involved. Non-repudiation means that it can be verified that the sender and the recipient were, in fact, the parties who claimed to send or receive the message, respectively.

  - ❖ Auditability is about keeping track of what is happening on a system. The idea is that if there is an intrusion, then the system operator can find out exactly what has been done and in whose name.

Other security concerns relate to:

- Trust: People can justifiably rely on computer-based systems to perform critical functions securely, and on systems to process, store and communicate sensitive information securely;
- Reliability: The system does what you want, when you want it to;
- Privacy: Within certain limits, no one should know who you are or what you do.

2.2 Cryptography

Cryptography [9] [10] is the most commonly used means of providing security; it can be used to address four goals:

- Message confidentiality: Only an authorized recipient is able to extract the contents of a message from its encrypted form;
- Message integrity: The recipient should be able to determine if the message has been altered during transmission;
- Sender authentication: The recipient can identify the sender, and verify that the purported sender did send the message;
- Sender non-repudiation: The sender cannot deny sending the message.

There are two kinds of cryptosystems: *symmetric* [11] and *asymmetric* [12]. Symmetric cryptosystems use the same key (the secret key) to encrypt and decrypt a message. By this data confidentiality and data integrity are succeeded.
Symmetric cryptosystems have a problem: how do you transport the secret key from the sender to the recipient securely? If you could send the secret key securely, then, in theory, you wouldn't need the symmetric cryptosystem in the first place -- because you would simply use that secure channel to send your message.

In asymmetric cryptosystems, encryption and decryption are performed using a pair of keys. One key (the public key) is used to encrypt a message and a different key (the private key) is used to decrypt it. Both keys are numbers and are mathematically related.
The public key is published where the private key is kept private. A person can prove that he or she holds the private key simply by encrypting a message. If the message can be decrypted using the public key, the person must have used the private key to encrypt the message. The main advantage of asymmetric cryptography (also called *public key* cryptography) is that secrecy is not needed for the public key.

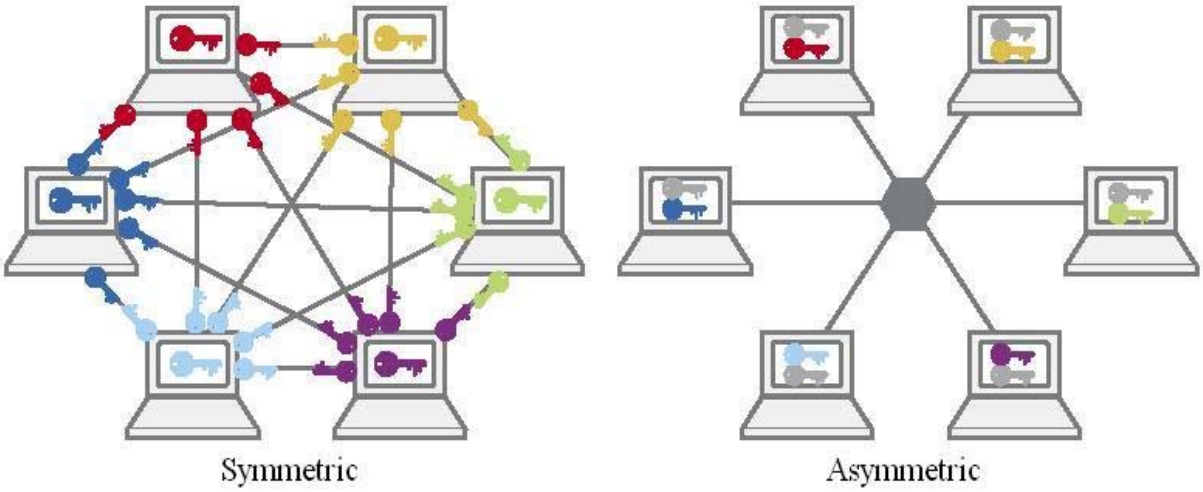


Figure 2.1: Symmetric vs. Asymmetric Encryption

Symmetric cryptography has an equation of $\frac{n \times (n-1)}{2}$ for the number of keys needed. For example, if we have 1000 users, that means that 499500 keys are needed.
Whereas in asymmetric cryptography (using key pairs for its users), the number of keys needed is the same number of the users. In the case of 1000 users, that means 1000 key pairs are needed.

If anyone wants to send me a message that only I can read, they can encrypt the message using my public key. Anyone can access my public key. The only way the message can be decrypted is by using the corresponding private key. So as long as I make sure I keep my private key private, only I will be able to read the message.

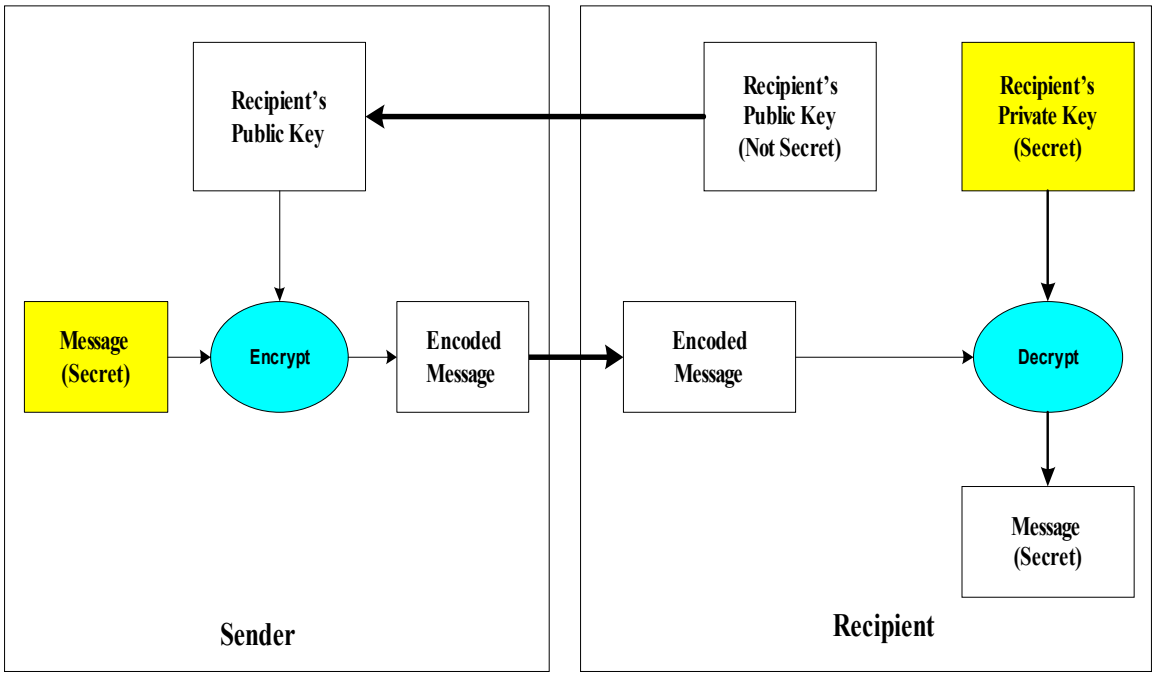


Figure 2.2: Communication between 2 parties

Important: The private key must be kept private. It should not be sent to others and it should not be stored on systems where others may be able to find it. A good storage device would be a smart card. The stored keys should always be password-protected.

Keeping the private secure is one of the concerns of public key cryptography. Another concern is when sending an encrypted message, the recipient must be sure that the message has not been changed during transmission.
Using public key cryptography, it is possible to digitally "sign" a piece of information [9]. Signing information essentially means assuring a recipient of the information that the information hasn't been changed since it left your hands.

To sign a piece of information, first a mathematical hash [13] of the information is computed. A hash is a condensed version of the information. The algorithm used to compute the hash must be known to the recipient of the information. Second, using the sender's private key, the hash is encrypted and attached to the message.
To verify that the signed message is authentic, the recipient of the message will compute the hash of the message using the same hashing algorithm the sender used, and will then decrypt the encrypted hash attached to the message. If the newly-computed hash and the decrypted hash match, then it proves that the message hasn't been changed since the sender signed it.

A cryptographically hash function is applied to the message and the resulting message digest is encrypted instead of the entire message, which makes the signature significantly shorter than the message and saves time since hashing is generally much faster than public-key encryption.

2.3 Public Key Infrastructure (PKI)

Since anybody can create a public-private-key pair (using existing algorithms), a certificate is required that guarantees that the public key belongs to a named entity. More briefly, if the originator is sending private information encrypted with the recipient's public key, a malicious user can fool the originator into using their public key, and so get access to the information, since it knows its corresponding private key. But if the originator only trusts public keys that have been signed ("certified") by an authority, then this type of attack can be prevented.
A public-key certificate is a file that contains a public key, together with identity information, such as a person's name, all of which is signed by a Certification Authority (CA) [9].

Public Key Infrastructure (PKI) [9][14] refers to the software that manages certificates in a large-scale setting.
A certificate typically includes:

- The public key.
- A name, which can refer to a person, a computer or an organization.
- A validity period.
- The location (URL) of a revocation list.

The most common certificate standard is the X.509 [9] [15]. An X.509 certificate is a generally a plaintext file that includes information in a specific syntax (see the example below):

- Subject: This is the name of the user;
- Subject's public key: This includes the key itself and other information such as the algorithm used to generate the public key;
- Issuer's subject: CA's distinguished name*;
- Digital signature: The certificate includes a digital signature of all the information in the certificate.

Distinguished Names (DN) have several different attributes such as:

- OU: Organizational Unit
- L: Location
- CN: Common Name (the user's name)

An example of a X.509 certificate (taken from our CA):

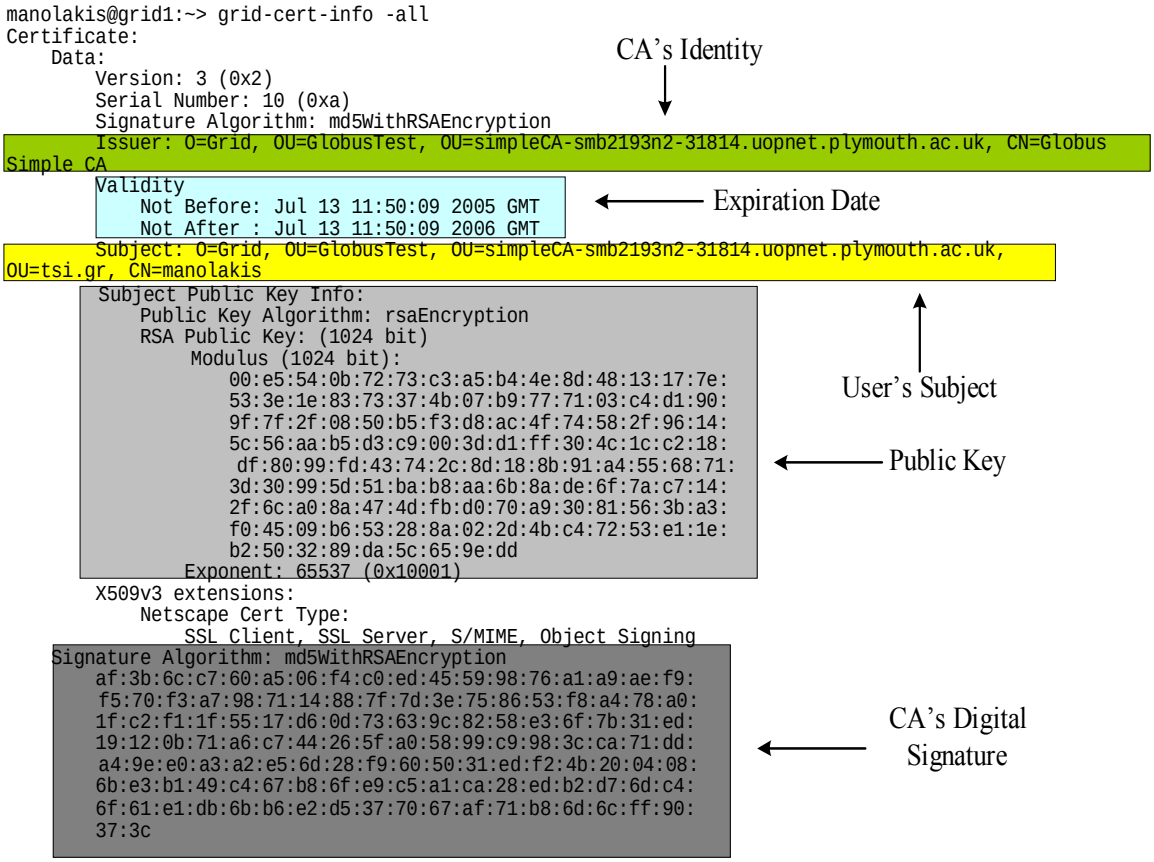


Figure 2.3: X509 Certificate

Certification Authority (CA)

The CA exists to provide entities with a trustable digital identity which they can use to access resources in a secure way. Its role is to issue (create and sign) certificates and make the valid certificates publicly accessible.

A CA can issue personal certificates to users; the purpose of the certificate is to allow users to identify themselves to remote entities. A personal certificate can also be used for digital signatures. A CA can also issue host (server) and service certificates. Each host and service connected to a network must be able to identify itself.

A CA issues public-key certificates that state that the CA trusts the owner of the certificate and they are who they claim to be.

2.4 Grid Security Infrastructure (GSI)

The Grid needs an efficient way to keep track of information regarding who is authorized to use the Grid, and which resources on the Grid are they authorized to use.

Who authenticates that a given user is who he says he is. What are the usage policies of the different resources?

All these things may change from day to day, so the Grid needs to be extremely flexible, and have a reliable accounting mechanism. Ultimately, the accounting will be used to decide pricing policy for the Grid. In IT security, it is common to talk about the "three A's", Authorization, Authentication and Accounting, and this is certainly true for the Grid.

In General, IT security is concerned with ensuring that critical information is not compromised or put at risk by external agents. Here, the external agent might be anyone that is not authorized to access the critical information.

The Grid Security Infrastructure (GSI) [9][16] is the part of the Globus Toolkit that provides the fundamental security services needed to support Grids. GSI provides libraries and tools for authentication and message protection that use standard X.509 public key certificates, public key infrastructure (PKI), the SSL/TLS protocol [17] and X.509 Proxy Certificates, an extension defined for GSI to meet the delegation requirements of Grid communities.
The primary motivations behind GSI are:

- The need for secure communication between elements of a computational Grid.
- The need to support security across organizational boundaries.
- The need to support "single sign-on" for users of the Grid.

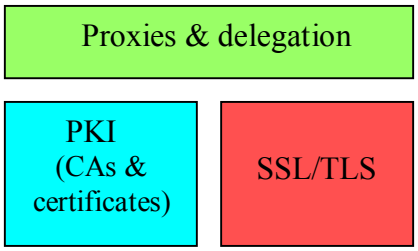


Figure 2.4: GSI structural elements

GSI uses public key cryptography (asymmetric cryptography) as the basis for its functionality. GSI is based on a PKI, with certificate authorities and X.509 certificates.
GSI provides:

- A public-key system.
- Mutual authentication through digital certificates.
- Credential delegation and single sign-on.

GSI uses X.509 public key certificates and Secure Socket Layer (SSL) [17] for authentication not only because these are well-known technologies with readily available, well-tested open source implementations, but because the trust model of X.509 certificates allows an entity to trust another organization's certification authority CA without requiring that the rest of its organization do so or requiring reciprocation by the trusted CA.

GSI is a collection of well-known and trusted technologies. It can be configured to provide privacy, integrity and authentication. Not all of these features are always needed during communication. Typically, a GSI-based secure conversation must at least be authenticated. Integrity is desirable, but can be disabled and encryption can also be activated, when needed, to ensure privacy.

GSI certificates are encoded in the X.509 certificate format. Mutual authentication means that both parties in a secure conversation authenticate the other. So, when an originator wants to communicate with a remote party, the originator must establish trust in the remote party and vice versa. Here trust means that each party must trust the certificate of the CA that signed the other's certificate. GSI uses the Secure Sockets Layer (SSL) for its mutual authentication protocol.

GSI lets a user create and delegate proxy credentials [18] to processes running on remote resources; this allows remote processes and resources to act on a user's behalf. This feature is important for complex applications that need to use a variety of remote resources. Proxy credentials are short-lived credentials created by a user; in essence they are a short-term binding of the user's identity to an alternate private key. It allows users to be authenticated once and then perform multiple actions. The proxy credential is stored unencrypted for ease of repeated access and has a short lifetime. Single sign-on is an important feature as it simplifies the coordination of multiple resources.

2.4.1 Authorization modes in GSI

GSI supports three authorization modes on both the server and client.

Server-side authorization

- None: No authorization will be performed. (this is the simplest type of authorization)
- Self: A client will be allowed to use a grid service if the client's identity is the same as the service's identity.
- Grid map: The "gridmap file" contains a list of authorized users where only the users listed in the service's gridmap file may invoke it.

Client-side authorization

- None: No authorization will be performed.
- Self: A service can be invoked if the client's identity is the same as the service's identity.
- Host: The client will authorize a security request if the host returns an identity containing the hostname. This is done using host certificates.

2.4.2 Mutual Authentication

As mentioned above, GSI uses the SSL protocol for its mutual authentication [1] [9] [19].

Before mutual authentication can occur, the two parties involved in the communication must first trust the CAs that signed each other's certificate. Each entity will have a copy of the other CA's certificate (that includes the public key).

1) The first entity (A) establishes a connection with the second entity (B). To start the authentication process, A gives B its certificate. The certificate tells B who A is claiming to be (the identity), A's public key and which CA is being used to certify the certificate.
2) First, B makes sure that the certificate is valid by checking the CA's digital signature to ensure that the CA actually signed the certificate and that the certificate has not been changed. This is the point where B must trust the CA that signed A's certificate.
3) Once B has checked out A's certificate, B must make sure that A really is the entity identified in the certificate. To do this, B generates a random plaintext message and sends it to A, asking A to encrypt it.
4) A encrypts the message using its private key and sends it back to B. B decrypts the message using A's public key. If the result is the same as the original random message, then B knows that A is who they say they are and B trusts A's identity.
5) The same operation now happens in reverse. B sends A its certificate. A validates the certificate and sends a challenge message to be encrypted. B encrypts the message using its private key and sends it back to A. A decrypts it and it compares it with the original. If it matches, then A knows that B is who they say they are.
6) At this point, A and B have established a connection to each other and are certain that they know each others' identities, i.e. they trust each other.

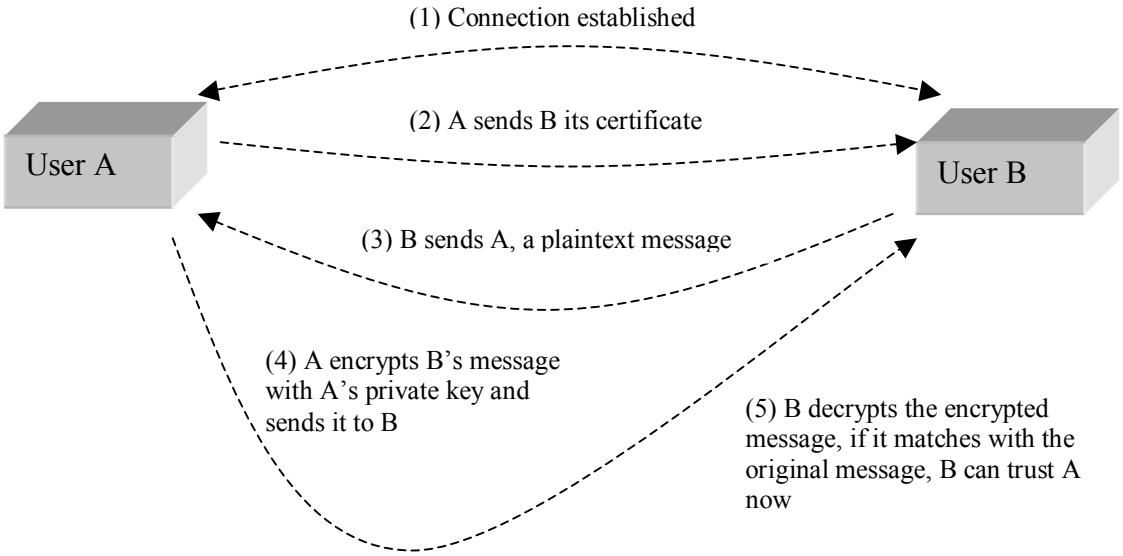


Figure 2.5: Mutual Authentication

2.4.3 Confidential Communication

By default, the GSI does not establish confidential (encrypted) communication between entities. Once mutual authentication is performed, the GSI gets out of the way so that communication can occur without the overhead of constant encryption and decryption.
The GSI can easily be used to establish a shared key for encryption if confidential communication [1][9] is desired.
Another security feature is communication integrity. Integrity means that an eavesdropper may be able to read communication between two entities but is not able to modify the communication in any way. The GSI provides communication integrity by default. (It can be turned off if desired). Communication integrity introduces some overhead in communication, but not as large an overhead as encryption.

2.4.4 Securing Private Keys

The core GSI software provided by the Globus Toolkit expects the user's private key to be stored in a file in the local computer's storage. To prevent other users of the computer from stealing the private key, the file that contains the key is encrypted via a password (also known as a pass phrase). To use the GSI, the user must enter the pass phrase required to decrypt the file containing their private key.
The use of cryptographic smartcards in conjunction with the GSI is also possible. This allows users to store their private key on a smartcard rather than in a file system, making it still more difficult for others to gain access to the key [1][9].

2.4.5 Delegation and Single Sign-On

The GSI delegation capability is an extension of the standard SSL protocol which reduces the number of times the user must enter the pass phrase. If a Grid computation requires that several Grid resources be used (each requiring mutual authentication), or if there is a need to have agents (local or remote) requesting services on behalf of a user, the need to re-enter the user's pass phrase can be avoided by creating a proxy.

A proxy consists of a new certificate (with a new public key in it) and a new private key. The new certificate contains the owner's identity, modified slightly to indicate that it is a proxy. The new certificate is signed by the owner, rather than a CA.

Proxy Certificates allow an entity holding a standard X.509 public key certificate to delegate some or all of its privileges to another entity which may not hold X.509 credentials at the time of delegation. This delegation can be performed dynamically, without the assistance of a third party, and can be limited to arbitrary subsets of the delegating entity's privileges. Once acquired, a Proxy Certificate is used by its bearer to authenticate and establish secured connections with other parties in the same manner as a normal X.509 end-entity certificate.

The certificate also includes a time notation after which the proxy should no longer be accepted by others. Proxies have limited lifetimes.

GSI and Proxy Certificates have been used to build numerous middleware libraries and applications that have been widely deployed in large production and experimental Grids. This experience has proven the viability of proxy delegation as a basis for authorization within Grids, and has further proven the viability of using X.509 Proxy Certificates.

The proxy's private key must be kept secure, but because the proxy isn't valid for very long, it doesn't have to be kept quite as secure as the owner's private key. It is thus possible to store the proxy's private key in a local storage system without being encrypted, as long as the permissions on the file prevent anyone else from looking at them easily. Once a proxy is created and stored, the user can use the proxy certificate and private key for mutual authentication without entering a password.

When proxies are used, the mutual authentication process differs slightly. The remote party receives not only the proxy's certificate (signed by the owner), but also the owner's certificate. During mutual authentication, the owner's public key (obtained from her certificate) is used to validate the signature on the proxy certificate. The CA's public key is then used to validate the signature on the owner's certificate. This establishes a chain of trust from the CA to the proxy through the owner.

2.5 Managing credentials in Globus

Once a user has been authenticated to a system and proven his GSI identity, he must be authorized before he gains any access.
Globus comes with an authorization mechanism, which uses a file called "*grid-mapfile*".
The "*grid-mapfile*" is the main point of control for the local system administrator to control who can access their system using Globus.

The "*grid-mapfile*" is a plaintext file containing a list of authorized GSI identities and mappings from those identities to local user identities (e.g. UNIX account names).
The local system administrator, when requested by a user, who presents their GSI identity to the administrator, adds entries to it. The administrator then determines what the local account name is for the user and adds this mapping to the "*grid-mapfile*".
The "*grid-mapfile*" is used by GSI after authenticating a user. A "*grid-mapfile*" at each site specifies grid-id to local-id mapping. GSI first checks to see if the user's grid identity is listed in the "*grid-mapfile*". If the user is not listed, they are denied access to the resource.

An example of a "*grid-mapfile*" entry is (taken from our grid-mapfile):

"/O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-31814.uopnet.plymouth.ac.uk/OU=tsi.gr/CN=manolakis" haris
"/O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-31814.uopnet.plymouth.ac.uk/OU=tsi.gr/CN=ioannis" haris
"/O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-31814.uopnet.plymouth.ac.uk/OU=tsi.gr/CN=haris" haris
"/O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-31814.uopnet.plymouth.ac.uk/OU=tsi.gr/CN=nikos" haris

The Globus command "*grid-mapfile-add-user*" can be used by an administrator to add entries to the gridmap file.

The Globus command "*grid-cert-info –subject*" can be used to get information form the user's certificate.

2.6 Generate a client proxy

Before running a client, it is necessary to generate a proxy for the client. Proxies are intended for short-term use, when the user is submitting many jobs and does not want to repeatedly enter their password for every job. The subject of a proxy is the same as the subject of the certificate that signed it.

Proxies provide a convenient alternative to constantly entering passwords, but are also less secure than the user's normal security credential. Therefore, they should always be **user-readable only**, and should be deleted after they are no longer needed (or after they expire).

The command to create a proxy in Globus is:

```
grid-proxy-init
```

Generating a proxy uses the "*$HOME/.globus/usercert.pem*" and "*$HOME/.globus/userkey.pem*" (e.g. for $HOME in this thesis is: "*/home/nikos/*").
A pass phrase should be entered.

After running the command "*grid-proxy-init*", a file in "*/tmp*" should be created. The file should look like this "*/tmp/x509up_u$UID*". For example, with an UID of 1002 the proxy file will be "*/tmp/x509up_u1002*". The default expiry time of the proxies is 12 hours.

The Globus command "*grid-proxy-destroy*" deletes a proxy that was previously created with "*grid-proxy-init*".

2.7 Firewall traversal

The Globus Toolkit provides a way to restrict the ports which client and server programs can use. This is done by the use of the "*GLOBUS_TCP_PORT_RANGE*" environment variable. Setting this variable to a range of ports will cause Globus tools to only use ports in this range.

TCP ports	
Ephemeral	A non-deterministic port assigned by the system in the untrusted port range (>1024)
Controllable ephemeral	An ephemeral port selected by the Globus Tooolkit libraries that can be constrained by use of the "*GLOBUS_TCP_PORT_RANGE*" environment variable to a given port range. In our care, we set the latter from 32000 to 32100.
Grid service	Static ports for well-known Grid services For example: ▪ 22/TCP (GSI-enabled OpenSSH) ▪ 2119/TCP (Globus Gatekeeper) ▪ 2135/TCP (MDS) ▪ 2811/TCP (GridFTP server)

2.8 Grid authentication and authorization mechanisms

Below is the description of the Grid authentication and authorization mechanisms:

1. A user uses signs a Proxy certificate using his certificate.
2. The proxy certificate is sent together with job submission or file transfer requests to grid servers.
3. On the server side, grid services (e.g. Globus-gatekeeper [20] and GridFTP [21]) authenticate the requests by extracting a Distinguished Name (DN) out of the Proxy certificate and comparing it against a "*grid-mapfile*" file, which is comparable to an access-control-list of a grid server.
4. Finally, the grid services authorize the client requests by executing them under the extracted DN. For example, supposed the client submits requests and proxy certificates with the DN of: "*"/O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-31814.uopnet.plymouth.ac.uk/OU=tsi.gr/CN=nikos" haris*". The grid services will execute the request under the user account "haris" on the server machine.

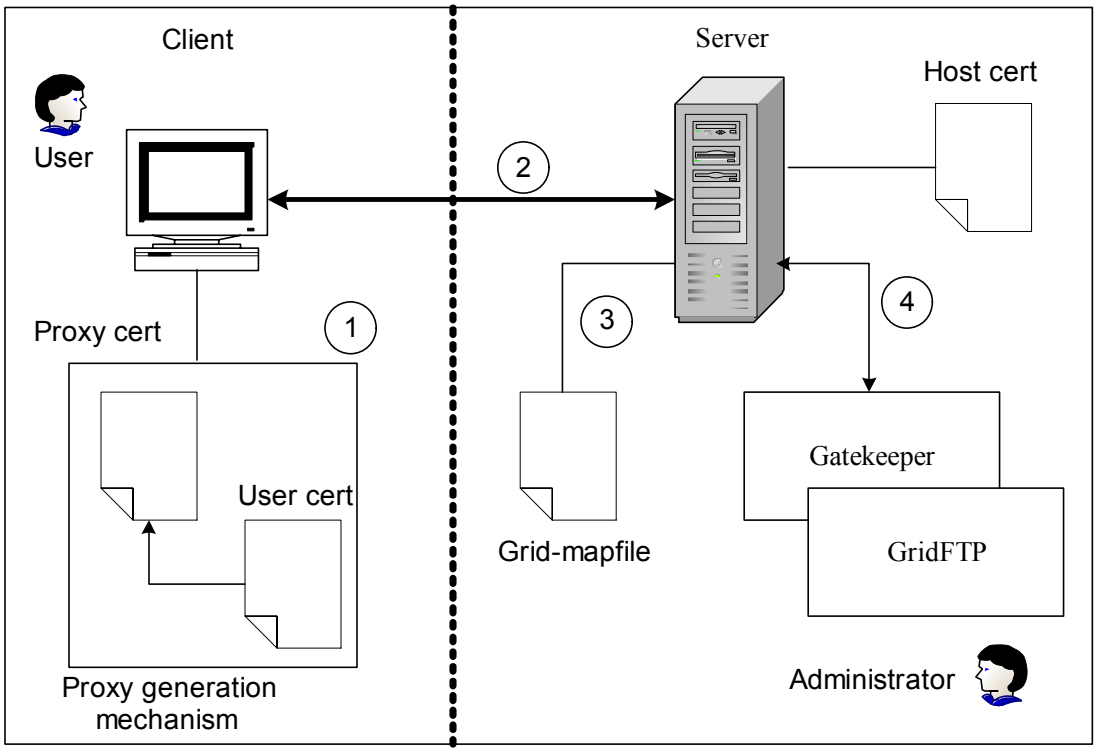


Figure 2.6: Grid Authentication & Authorization Mechanisms

2.9 Possible Vulnerabilities

Security mechanisms are not fool proof. There always tends to be tension between security and usability. So, there will always be vulnerabilities, which range from technical to physical ones.

Authentication

On the Grid, both sides achieve authentication by using certified public keys. An authorized CA decides who should be certified. An obvious concern here is the amount of checking done to confirm the identity of someone before issuing the certificate.

Proxies

The proxy mechanism is useful as it provides local authorization without the need to contact the user site for every remote action or service access.

If the remote system were compromised, the user's proxy private key would no longer be private, and even if the user were aware of this there is no revocation process. Globus attempts to minimize the risk by recommending that proxies are short-lived, so the stolen credential can only be misused for a short time.

Authorization

Authorization is the step of deciding what access rights a user has to services and resources. In the Grid, most systems assert that authorization to a service should be generated by the service owner. Most Grid systems do this by mapping a remote Grid user's identity onto a local account. The local system administrator can then define the services the local account is able to access. The greatest problem with this methodology is that for every user, service or resource, there need to be entries in the "*grid-mapfile*". The number of such entities to be managed by the "*grid-mapfile*" file can be huge, and system administrators will be swamped and start giving all users default rather than particular access rights, which may mean that vulnerabilities are introduced. This model also means that it is likely that it takes time to add or remove user access rights, which can cause problem also.

In general, it is clear that the greater the access rights that a user is granted, the more likely that a service or resource will be exposed to security compromise. If a user's credentials are stolen and if they are not invoked quickly, services and resources that the user had access right to are at risk. This risk is minimized by enabling the dynamic allocation of local UNIX usernames to Grid users.

The latter is done by applying pool account mechanism. The pool account mechanism enables the dynamic allocation of local UNIX usernames to Grid users. The account is allocated to the Grid user only for the time needed to access the local resources; then it can be de-allocated and reassigned to the pool of available accounts. It is possible to define multiple pools in order to implement policies at local level. This mechanism makes negligible the probability of exhausting the set of standard UNIX credentials (normally the maximum number of possible user accounts on a UNIX system is 65536).

3 Condor

3.1 Introduction

Condor [8] [9] is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring and management. Users submit their serial or parallel jobs to Condor, Condor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

Condor's architecture allows it to succeed in areas where traditional scheduling systems fail. In addition, unique mechanisms enable Condor to effectively harness wasted CPU power from otherwise idle desktop workstations. For instance, Condor can be configured to only use desktop machines where the keyboard and mouse are idle. Should Condor detect that a machine is no longer available (such as a key press detected), in many circumstances Condor is able to transparently produce a checkpoint and migrate a job to a different machine which would otherwise be idle. Condor does not require a shared file system across machines (if no shared file system is available), Condor can transfer the job's data files on behalf of the user, or Condor may be able to transparently redirect all the job's I/O requests back to the submit machine. As a result, Condor can be used to seamlessly combine all of an organization's computational power into one resource [8].

The idea behind Condor is simple. It matches any computational jobs that computer users have with spare power in other owners' computers. Computer owners don't have to modify their programs to use Condor. They just have to agree to become part of a Condor network, a group of computers connected in such a way that messages can pass between them [8].

Meeting the varied demands of computer owners, however, is not so easy. Condor must keep track of activity on every computer on networks that can include hundreds of computers. It must know how soon it is allowed to start using a computer after the owner stops using it. It must work both in environments where each individual uses just a single computer and in environments where any individual can log on to any computer. And it must know how to share the resources of the network fairly [8].

Researchers who use Condor gain two major advantages: they can have many tasks running at once, and they can run big projects that otherwise might be too expensive to conduct.

But Condor's advantages have a price: each task takes longer that it would run on a single machine. Projects move from computer to computer. Projects often must wait their turn. Information must be moved in and out of electronic memory and the magnetic storage drives. And if a computer's owner returns-signaled either by tapping a key or moving the

computer's mouse-the project must be interrupted and moved either to another workstation or, if no other machine is available back to its home workstation [8].

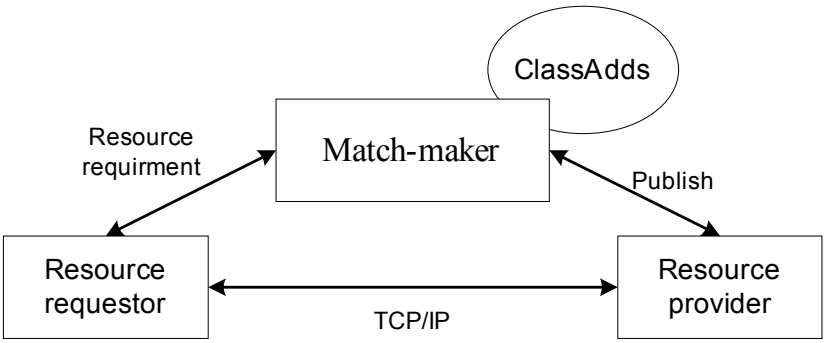


Figure 3.1: The Condor Model

The "*ClassAd*" [59] mechanism (see Figure 3.1) in Condor provides an extremely flexible and expressive framework for matching resource requests (jobs) with resource offers (machines). Jobs can easily state both job requirements and job preferences. Likewise, machines can specify requirements and preferences about the jobs they are willing to run. These requirements and preferences can be described in powerful expressions, resulting in Condor's adaptation to nearly any desired policy.

3.2 The architecture of a Condor pool

Machines in Condor are normally organized in a Condor pool [8]. A pool is an administrated domain of hosts. A Condor pool normally has one Central Manager (master hosts) and an arbitrary number of execution hosts. The Central Manager host can be any computer in the network. It watches for inactive computers. When it senses an idle machine, Condor accesses it and run a project on it. When the owner resumes using the computer, Condor moves the project somewhere else. In other words, the Central Manager host is used to manage resources and jobs in a Condor pool. If the Central Manager host in a Condor pool crashes, jobs that are already running will continue to run unaffected. Queued jobs will remain in the queue unharmed, but they cannot begin running until the Central Manager host is restarted. A Condor Execution host can be configured as a job execution host or a job Submission host or both [9].

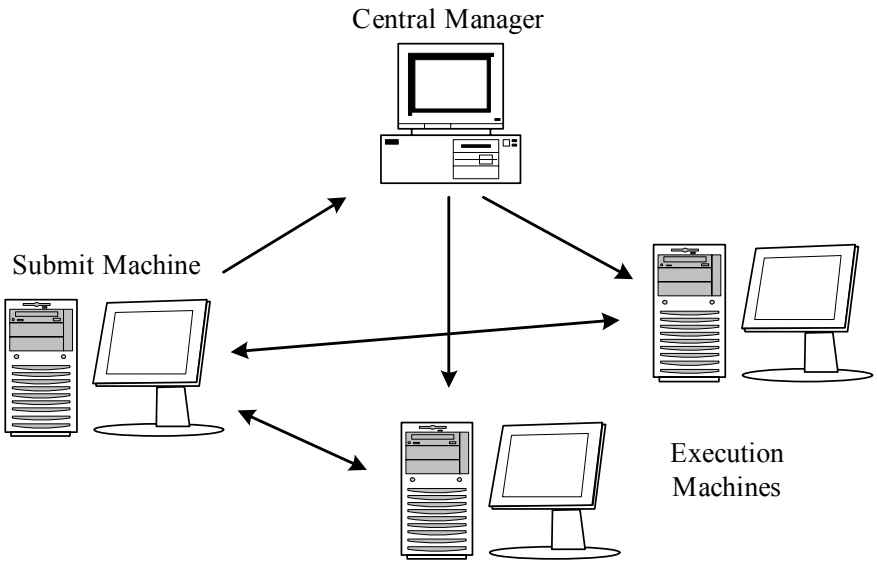


Figure 3.2: The architecture of Condor pool

3.2.1 Daemons in a Condor pool

A daemon is a program that runs in the background once started. To configure a Condor pool, the following Condor daemons need to be started.

condor_master

- Runs on each host in the Condor pool
- Starts up all other Condor daemons
- If there are any problems and a daemon exits, it restarts the daemon and sends email to the administrator
- Checks the time stamps on the binaries of the other Condor daemons, and if new binaries appear, the master will gracefully shutdown the currently running version and start the new version
- Acts as the server for many Condor remote administration commands:
  - *condor_reconfig, condor_restart, condor_off, condor_on, condor_config_val*, etc.

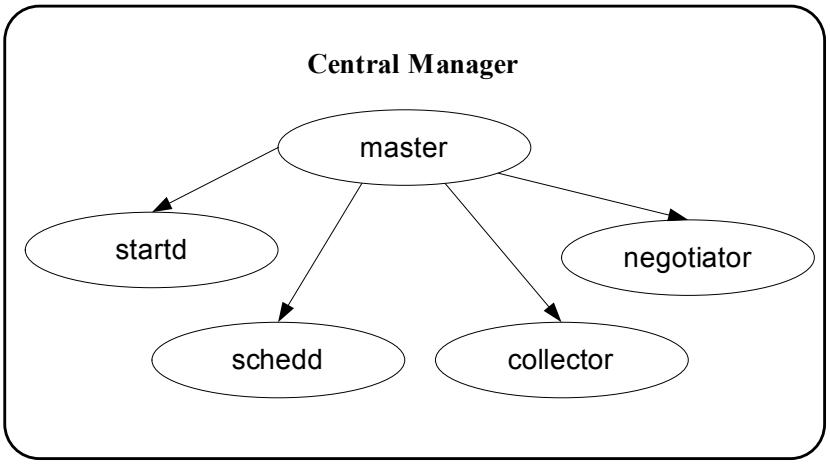


Figure 3.3: Daemons running in the Central manager

condor_startd

- Runs on each host in the Condor pool
- Advertises information related to the node resources
- Responsible for starting, suspending, and stopping jobs
- Enforces the policies of the resource owner

condor_starter

- Runs only on Execution hosts
- Passes a remote job on a given host in a Condor pool
- Sets the execution environment
- Monitors the job once it is running

condor_schedd

- Runs on each host in the Condor pool
- Represents users to the Condor system
- Maintains the persistent queue of jobs
- Responsible for contacting available machines and sending them jobs
- Services user commands which manipulate the job queue:
  - *condor_submit ,condor_rm, condor_q, condor_hold, condor_release, condor_prio, ...*

condor_shadow

- Runs only on Submission hosts
- Allows jobs submitted to be check-pointed

- Responsible for making decision about a user job submission request, such as where check-pint file should be stored.

condor_collector

- Runs only on the Central Manager
- Collects information from all other Condor daemons in the pool
  - "Directory Service" / Database for a Condor pool
- Each daemon sends a periodic update called a "ClassAd" to the collector
- Services queries for information:
  - Queries from other Condor daemons
  - Queries from users (***condor_status***)

condor_negotiator

- Runs only on the Central Manager
- Performs "matchmaking" in Condor
- Gets information from the collector about all available machines and all idle jobs
- Tries to match jobs with machines that will serve them
- Both the job and the machine must satisfy each other's requirements

condor_kbdd

Runs only on Execution hosts which have Digital Unix or IRIX. On these platforms, the "condor_startd" daemon cannot determine console (keyboard or mouse) activity directly from the operating system.

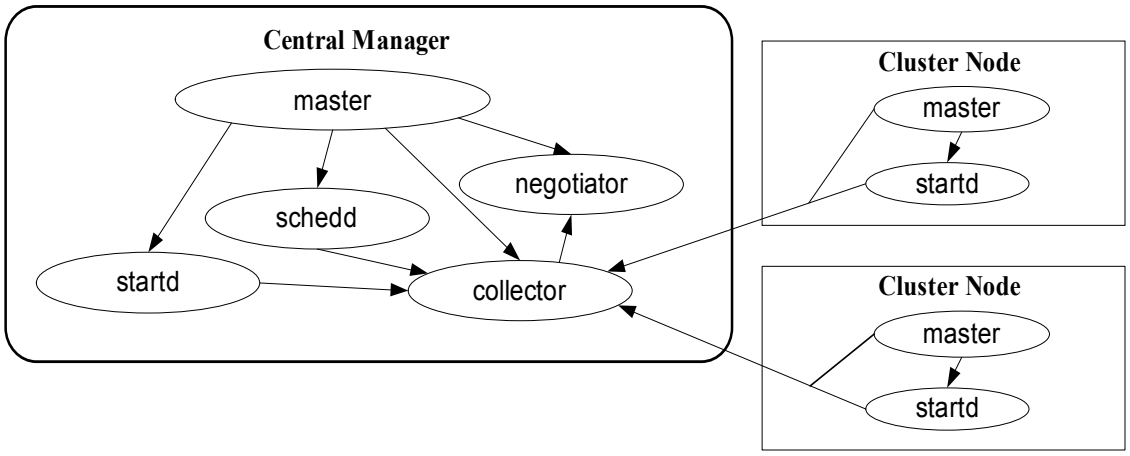


Figure 3.4: Layout of the Condor Pool

3.2.2 Job life cycle in Condor

A job submitted to a Condor pool will go through the following steps as shown in the Figure 3.5 below.

1. Job submission: A job is submitted by a Submission host with "*condor_submit*" command (Step 1).

2. Job request advertising: Once it receives a job request, the "*condor_schedd*" daemon on the Submission host advertises the request to the "condor_collector" daemon running on the Central Manager host (Step 2).

3. Resource advertising: Each "*condor_startd*" daemon running on an Execution host advertises resources available on the host to the "*condor_collector*" daemon running on the Central Manager host (Step 3).

4. Resource matching: The "*condor_negotiator*" daemon running on the Central Manager host periodically queries the "*condor_collector*" (Step 4) to match a resource for a user job request. It then informs the "*condor_schedd*" daemon running on the Submission host of the matched Execution host (Step 5).

5. Job execution: The "*condor_schedd*" daemon running on the job Submission host interacts with the "*condor_startd*" daemon running on the matched Execution host (Step 6), which will spawn a "*condor_starter*" daemon (Step 7). The "*condor_shcedd*" daemon on the Submission host spawns a "*condor_sahdow*" daemon (Step 8) to interact with the "*condor_starter*" daemon for job execution (Step 9). The "*condor_starter*" daemon running on the matched Execution host receives a user job to execute (Step 10).

6. Return output: When a job is completed, the results will be sent back to the Submission host by the interaction between the "*condor_shadow*" daemon running on the Submission host and the "*condor_starter*" daemon running on the matched Execution host (Step 11).

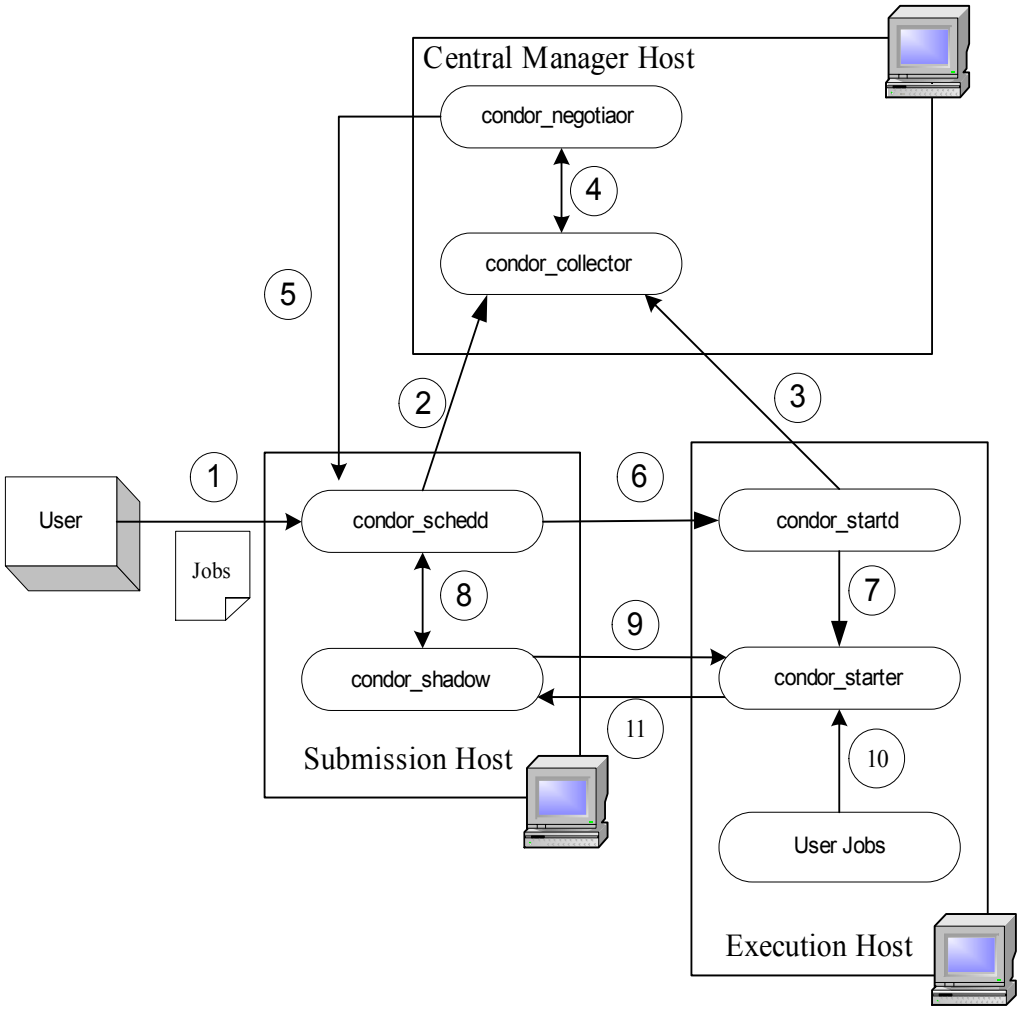


Figure 3.5: Job life cycle in Condor

3.2.3 Job management in Condor

Condor manages jobs in the following aspects.

Job
A Condor job is a work unit submitted to a Condor pool for execution.
Job types
Jobs that can be managed by condor are executable sequential or parallel codes. A job submission may involve a job that runs over a long period, a job that needs to run many times or a job that needs many machines to run in parallel.
Queue
Each Submission host has a job queue maintained by the "*condor_schedd*" daemon running on the host. A job queue can be removed and placed on hold.
Job status
A job can have one of the following statuses:

- Idle: There is no job activity.

- Busy: A job is busy running.
- Suspended: A job is currently suspended.
- Vacating: A job is currently check-pointing.
- Killing: A job is currently being killed.

Job run-time environments

The condor universe specifies a Condor execution environment [22].

- The default universe is the "*Standard Universe*". It tells Condor that this job has been re-linked via "*condor_compile*" with Condor libraries and therefore supports checkpointing and remote system calls.

- The "*Vanilla Universe*" is an execution environment for jobs which have not been linked with condor libraries; and it is used to submit shell scripts to Condor.

- The "*PVM Universe*" is used for a parallel job written with PVM [23].

- The "*Globus Universe*" is intended to provide the standard Condor interface to users who wish to start Globus jobs from Condor. Each job queued in the job submission file is translated into the "Globus Resource Specification Language" (RSL) and subsequently submitted to Globus via the Globus GRAM protocol.

- The "*MPI Universe*" is used for MPI [24] jobs written with the MPICH [25] package.

- The "*Java Universe*" is used for programs written in Java.

- The "*Scheduler Universe*" allows a Condor job to be executed on the host where the job is submitted. The job does not need matchmaking for a host and it will never be preempted.

Jobs can be submitted directly to a Condor pool from a Condor host or via Globus as shown in Figure 3.6 below. The Globus host is configured with "*Condor jobmanager*" provided by Globus. When using a "*Condor jobmanager*", jobs are submitted to the Globus resource, e.g. using "*globusrun-ws*" [26]. However, instead of forking thye jobs on the local machine, jobs are re-submitted by Globus to Condor using the "*condor_submit*" tool.

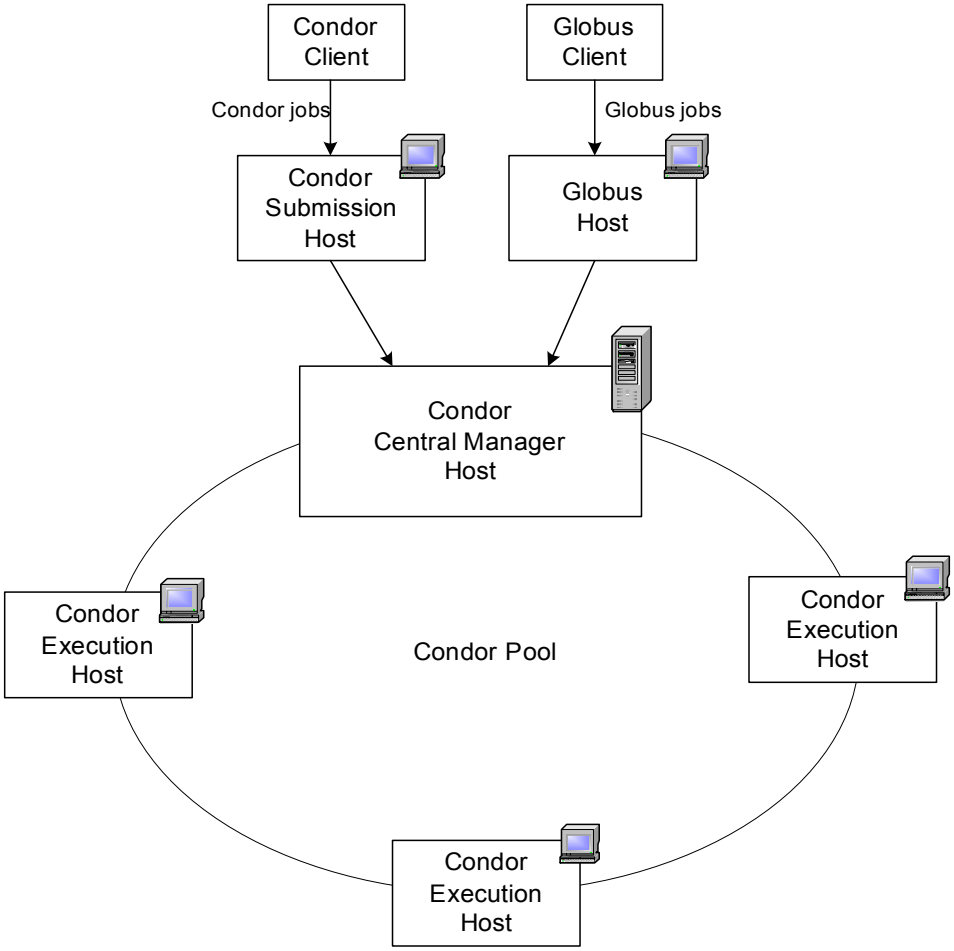


Figure 3.6: Submitting jobs to a Condor pool via Condor or Globus

3.3 Condor-G

Condor-G [27] is a version of Condor that has the ability to maintain contact with a Globus gatekeeper, submitting and monitoring jobs to Globus. Condor-G allows users to write familiar Condor job-submission scripts with a few changes and run them on Grid resources managed by Globus, as shown in Figure 3.7.

To use condor-G, we do not need to install a Condor pool. Condor-G is only the job management part of Condor. Condor-G can be installed on just one machine within an organization and the access to remote Grid resources using a Globus interface can be done through it.

Submitting Globus jobs using condor-G provides a much higher level of service than simply using "*globusrun-ws*" [26] command provided by Globus.

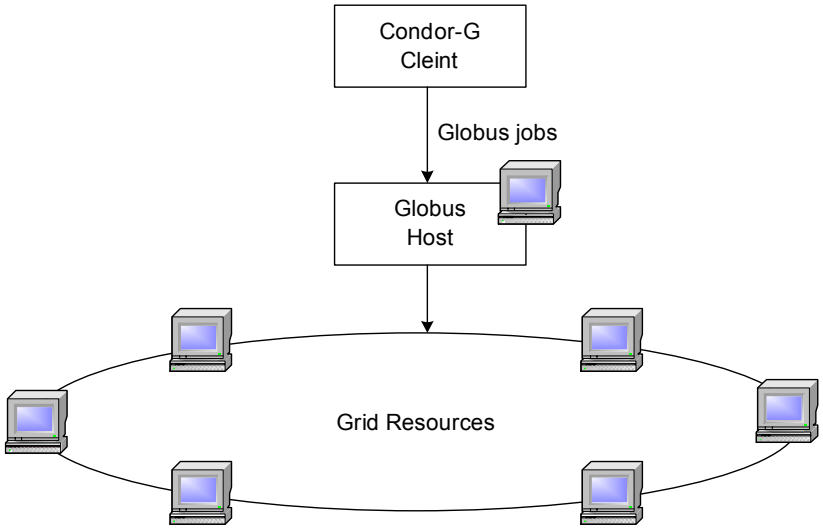


Figure 3.7: Submitting jobs to Globus via condor-G

As mentioned above, a machine can be both a submit machine and an execute machine in a Condor pool. Same with Condor-G, you can have Condor-G and Globus interface in the same pool. Users will submit jobs through Condor-G in order to have their jobs execute on resources outside the Condor pool. External users will use the Globus interface to submit jobs to be executed inside the Condor pool.

First, jobs submitted to Globus with Condor-G enter a local Condor queue that can be effectively managed by Condor.
Secondly, jobs remain in the Condor queue until they are completed. Therefore, should the job crash while running remotely, Condor-G can re-submit it again without user intervention.

The difference between Condor and Condor-G will become clearer with a brief overview of the software that forms a Condor pool. The software of a Condor pool is divided into two parts. The first part does **job management**. It keeps track of a user's jobs. You can ask the job management part of Condor to show you the job queue, to submit new jobs to the system, to put jobs on hold and to request information about jobs that have completed. The other part of the Condor software does **resource management**. It keeps track of which machines are available to run jobs, how the available machines should be utilized given all the users who want to run jobs on them, and when a machine is no longer available.

A machine with the job management part installed is called a submit machine. A machine with the resource management part installed is called an execute machine. Each machine may have one part or both. No one need to login to a central server to use the thousand machines in the pool, and every machine is potentially available run jobs.

Notes:

- **Condor-G does not "*create*" a grid service. It only deals with "*using*" remote grid services.**
- **Condor-G does not have a GUI.**

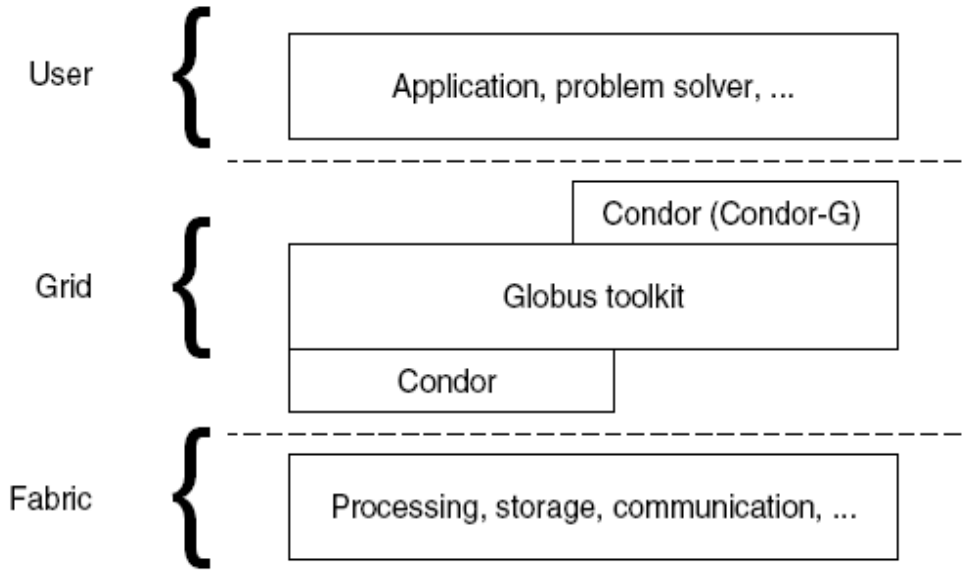


Figure 3.8: Condor technologies in Grid middleware. Grid middleware consisting of technologies from both Condor and Globus sit between the user's environment and the actual fabric (resources).

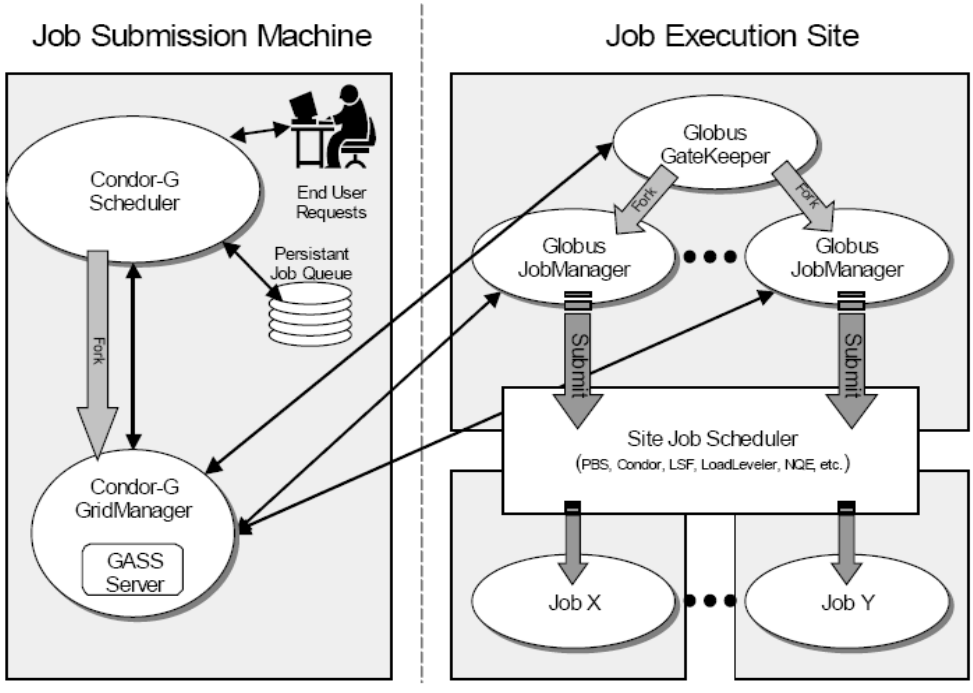


Figure 3.9: Remote execution by Condor-G on Globus-managed resources

4 Deployment & Implementation

4.1 Introduction

In this section we present the hardware that has been used as well as all the software components that have been installed, configured and tested during this thesis. Specifically, four Linux Kernel workstations have been deployed. On these machines we have installed the Grid middleware Globus Toolkit 4.01 with its components: GridFTP for data transfer, Reliable File Transfer (RFT) and OGSA-DAI database management. Furthermore, we have installed and configured the Tomcat java servlet container, the Condor job management system, and the PostgreSQL and MySQL database management systems. In addition to the above software implementations, we have worked on java language to write code for accessing databases and on XML language to write code for submitting jobs in globus stand-alone container for execution. Moreover, we emphasize on the following basic grid application tools like GridFTP, Reliable File Transfer (RFT), OGSA-DAI database management, Grid Resource & Allocation Management (GRAM) and Condor job scheduler.

The hardware/software deployed in this thesis is shown in the figure 4.1 below:

Grid0

Hardware:

- Pentium 4 CPU 3.00 Ghz
- 512 MB of RAM
- 100 GB of Storgage
- IP address:147.27.1.240
- Host: grid0.tsi.gr

Software:

- OS: SuSe Linux 9.2
- Globus Toolkit 4 & components*
- Java Software Development Kit J2SE 1.4.2
- Apache Ant 1.5.1
- JDBC compliant database.
- MySQL 4.1.12
- PostgreSQL 7.4.8
- Condor 6.6.10
- OGSA-DAI WSRF 2.1
- Java CoG Kit 4.1.12

Grid1

Hardware:

- Pentium 4 CPU 3.00 Ghz
- 512 MB of RAM
- 120 GB of Storage
- IP address:147.27.1.241
- Host: grid1.tsi.gr

Software:

- OS: SuSe Linux 9.2
- Globus Toolkit 4 & components*
- Java Software Development Kit J2SE 1.4.2
- Apache Ant 1.5.1
- JDBC compliant database.
- MySQL 4.1.12
- PostgreSQL 7.4.8
- Condor 6.6.10
- OGSA-DAI WSRF 2.1
- Java CoG Kit 4.1.12

Grid2

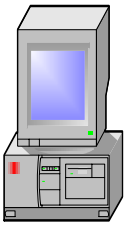
Hardware:

- Pentium II CPU 400 Mhz
- 128 MB of RAM
- 20 GB of Storgage
- IP address:147.27.1.242
- Host: grid2.tsi.gr

Software:

- OS:SlackWare 10.1
- Globus Toolkit 4 & components*
- Java Software Development Kit J2SE 1.4.2
- Apache Ant 1.5.1
- JDBC compliant database.
- PostgreSQL 7.4.8
- Condor 6.6.10
- Java CoG Kit 4.1.12

Grid3



Hardware:

- Pentium 4 Xeon CPU 2.00 Ghz
- 512 MB of RAM
- 153 GB of Storage
- IP address:147.27.1.243
- Host: grid3.tsi.gr

Software:

- OS: SuSe Linux 9.2
- Globus Toolkit 4 & components*
- Java Software Development Kit J2SE 1.4.2
- Apache Ant 1.5.1
- JDBC compliant database.
- MySQL 4.1.12
- PostgreSQL 7.4.8
- Condor 6.6.10
- OGSA-DAI WSRF 2.1
- Java CoG Kit 4.1.12
- Apache Tomcat 5.0

*: GT 4 Components: GridFTP, RFT, RLS, GRAM, etc.

Figure 4.1: Hardware/Software List

4.2 Grid testbed hardware/software deployment

After establishing the testbed hardware, we worked on the required software. The first thing that should be installed was the Operating System (OS). As listed above, the OS used was either SuSe 9.2 (grid0, grid1 and grid3) [28] or Slackware 10.1 (grid2) [29]. After this step, we moved on installing and configuring the Globus Toolkit [1] [2] software (visit https://grid2.tsi.gr for more details). We first installed the Java Software Development Kit (JSDK) [30] as a prerequisite of GT and as an environment to compile and run java code. Secondly, we installed Apache Ant [31]. Apache Ant is a java-based build tool (similar with the "make" command in Linux). Thirdly, two different databases (MySQL and PostgreSQL) were installed.

PostgreSQL [32] is needed by the Reliable File Transfer protocol (RFT) [33] to work properly. PostgreSQL can be configured along with the RFT configuration (see GT4 Installation chapter). During the configuration of PostgreSQL database server, a user "globus" has been created in order to connect to the database (rftDatabase) which is also created and populated with the appropriate files through this configuration ("globus" user account is created for managing the stand-alone container). The "*rftDatabase*" is charged with tracking of all the requests and checkpoints of the RFT server. PostgreSQL was also needed for the configuration of the Replica Location Server (RLS) [34].

MySQL [35] on the other hand has been used along with OGSA-DAI WSRF [36], to create, query and update grid wide databases. In the appendix you can find a summary of the basic steps to install MySQL (server & client).

At this point we should mention that it is desirable to install and configure Condor [8] [9] (job scheduler) before compiling-installing Globus Toolkit. Well, due to lack of experience regarding Globus Toolkit and Grids in general, we installed first the Globus Toolkit and afterwards we moved to Condor. That was not the proper decision because some serious problems occurred which led to the malfunction of Condor-Globus. We learned the hard way that installing Condor first would alleviate us from all these problems.

Before moving to the installation of Condor, the most important thing was to distinguish which machine should play the central manager role (see Condor chapter for details).In our case, grid1 is the central manager, grid0 (used also as a submit machine), grid2 and grid3 are the execution machines.

Condor should be first installed and configured on the central manager and then on the other machines. The first step is to create a linux-account-user "*condor*". This user is needed for Condor installation. In the home directory of Condor, i.e. "*/home/condor*" you can find the configuration files of Condor (created by the installation command).
The second step is to unzip the zip archive in some location, e.g. "*/usr/local/condor*". After that, the command "*/usr/local/condor/condor_install*" should be run. Following the

installation instructions, Condor installation is completed. The same steps should be followed on the other machines. It's very important, during the installation, to enter the right hosts in the appropriate options.

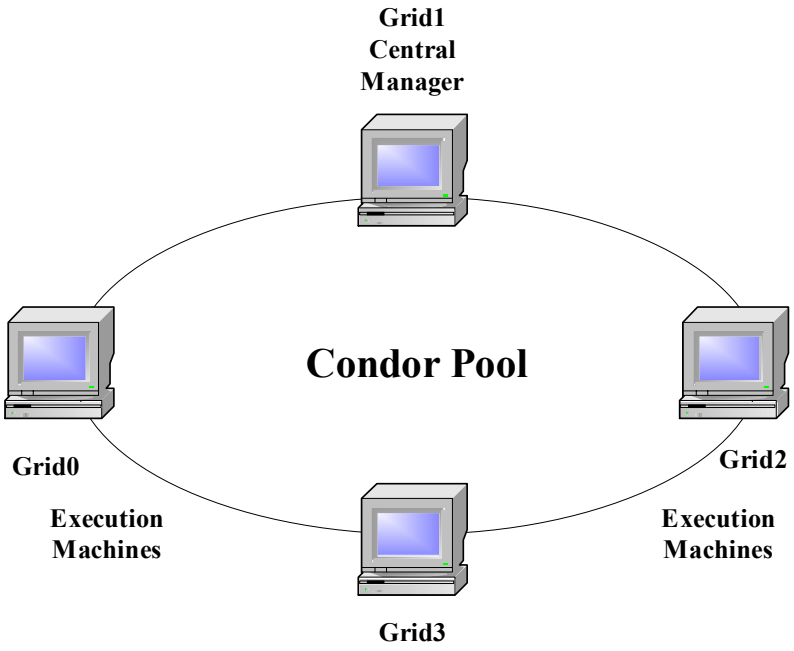


Figure 4.2: Condor Pool at TUC

That was one of the issues that should be taken in mind when installing Condor. Another important issue in Condor is to configure the idle CPU/keyboard parameters. These parameters should be configured wisely because they determine the idle time of every machine. Idle time is the time where a machine should change from "wait" to "active" state. By default, the Condor "*config file*" sets these parameters to some values which should be altered to fulfill the user's needs.

After editing the "*/home/condor/config_file*", we moved on testing the Condor pool we had just installed and configured. This was done by starting the Condor on the central manager first and then on the other machines by running the command "*condor_master*". Running the command "*condor_status*" returns information about the machines running in the condor pool. Every machine will be in some state (claimed, active, and idle). The status of every machine depends on the idle CPU/keyboard time and whether it has been executing a job or not. When a machine is in the idle state, a job can be submitted to that machine. If more than one machine is in the idle state then the central manager decides where to run the job depending on the job specifications. For example, if the job needs a storage capacity more than 10 GB, it can't be run on machines having declared less than 10GB storage capacity. If all machines in the Condor pool are claimed and a job is submitted to the pool, the job will be queued until a machine changes to "unclaim" state.

After setup of Condor pool has been completed, one can compile GT4 with the option to use Condor as the underlying job scheduler. GT4 Installation section (see Appendix B) describes in detail the steps.

After the compiling GT4 and installing all it's components we moved to OGSA-DAI. OGSA-DAI [36] is a middleware product that allows data resources, such as relational databases, to be accessed via web services. An OGSA-DAI web service allows data to be queried, updated, transformed and delivered. OGSA-DAI web services can be deployed within a Grid environment, thereby OGSA-DAI provides means for users to publish their data resources in the Grid. OGSA-DAI WSRF 2.1 is used in this thesis in order to be compatible with WSRF-based Globus Toolkit 4 [37]. For more information on the OGSA-DAI project visit http://www.ogsadai.uk.org.

When we first started working on OGSA-DAI WSRF 1.0, the installation guide was perplexed and not so well-documented. It took us much time to rearrange the documentation and convert it in a tangible form. The software can be downloaded either in executable or in source distribution type.

OGSA-DAI is deployed either under Globus SOAP [38] Container or Apache Tomcat [39]. In this thesis, it has been deployed under Globus SOAP Container therefore giving the ability to grid-users to access distributed database resources exposed through web services. Just deploying the service onto the container is not enough because initially this service exposes no resources. Therefore, resources (more precisely data resources) should be deployed to the service.

The data service we deployed under the container was "*/ogsadai/DataService*". This service has been deployed on grid1. The same data service could be deployed on the other machines but there is no need to have a container on every machine.

The data resource has been exposed to the "*/ogsadai/DataService*" under the resource id "*grid1db*". For simplicity, a config file has been used for deploying the data resource. This file includes all the information needed about the database server and version (e.g. MySQL), the resource id and the database used.

After configuring OGSA-DAI, we installed Apache Tomcat. Apache Tomcat is a servlet container as well as a web server. Mainly it is used in implementation for the Java Servlet and Java Server Pages (JSP) technologies.

Java Development Kit (JDK) is needed to install Tomcat. An environment variable pointing to the base path of the Tomcat installation has been configured under the name "CATALINA_HOME". In order to make HTML pages, JSPs and other files available on the Tomcat server, they should be placed in "*webapps/ROOT/*" which is located at "CATALINA_HOME".

After all, a grid testbed has been deployed for testing purposes and research within the Technical University of Crete (TUC).

The figure 4.3 below shows the features of Grid1 and Grid3.

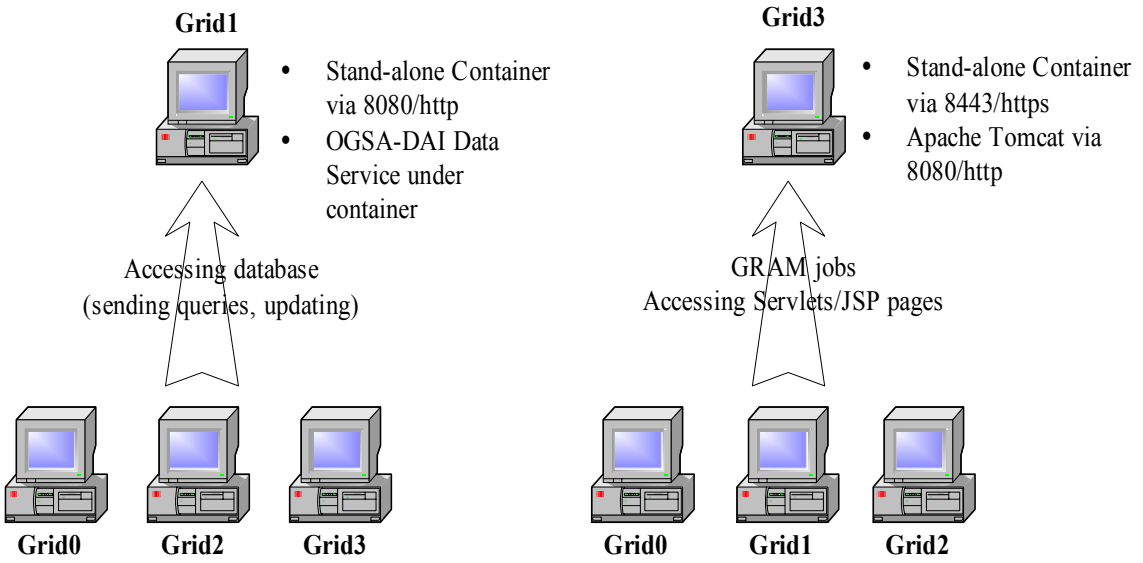


Figure 4.3: Grid1 & Grid3 features

4.3 Implementation

In this part we are going to present the various implementations which have been done through this thesis. These implementations had to do with GridFTP for data transfer, Reliable File transfer (RFT) for more parameterized file transfer, Condor job management system, OGSA-DAI database management and GRAM resource management. Briefly, all the services mentioned above have been tested locally at the Grid lab at TUC and remotely with the cooperation of the University of Plymouth. These tests helped debug service misconfiguration problems and ensure their integrity.

4.3.1 GridFTP

Here we are going to describe how a GridFTP server should handle data transfers (locally and remotely), how GridFTP works and few guide lines when working with the GridFTP service.

GridFTP is a common data transfer and access protocol that provides secure, efficient data transfer in Grid environments. This protocol is based on the well-known, standard FTP protocol [40]. The FTP protocol has been chosen because it is the most commonly used protocol for data transfer on the Internet. The GridFTP protocol includes the following features:

- Grid Security Infrastructure (GSI)
- Third-party control of data transfer: the GridFTP separates control and data channels, enabling third-party transfers, that is, the transfer of data between two end hosts, mediated by a third host
- Parallel data transfer (multiple TCP streams between 2 hosts): improves aggregate bandwidth relative to that achieved by a single stream

- Stripped data transfer (1 or more TCP streams between *m* hosts on the sending side and *n* hosts on the receiving side)
- Partial file transfer: Some applications can benefit from transferring segments of files rather than complete files: for example, analyses that require access to subsets of massive object-oriented database files. GridFTP allows transfer of the remainder of a file starting at a specified offset
- Manual control of TCP buffer/window sizes improves data transfer performance dramatically
- Support for reliable and restartable data transfer (in case of failure)

The Globus GridFTP provides secure authentication to the control channel requests (obligatory) and for data channel integrity and confidentiality (optional). The Grid Security Infrastructure (GSI) authentication mechanism is supported.

A session is established when the client (user) initiates a TCP connection to the port on which the server is listening (2811/TCP). This can be done by typing the command "*globus-url-copy*" [41]. The first thing to be checked is authentication. By default, the client (user) presents a delegated proxy certificate [18] which is signed by a "user certificate", which in turn has been issued by a CA [18] trusted by the client. If authentication is not successful, the connection is dropped. The delegated proxy certificate is password-protected and of limited lifetime by default.

If authentication is successful, an authorization callout is invoked to verify authorization of the Distinguished Name (DN), which included in the proxy certificate, and determine the corresponding local user id consulting the server grid-mapfile. This file is typically located in "*/etc/grid-secutiry/grid-mapfile*". In this file, there are pairs of DNs for authorized users and local user ids. If the pair (DN-user id) exists then the control channel is established. The control channel is encrypted and integrity protected by default.

To establish the data channel, where the actual data will flow, there are several scenarios.

Scenario 1:

A file is transferred from 'host A' (source), where the client application runs, to 'host B': 2811 (destination) where the GridFTP server runs. In this case the 'host A' data port is chosen arbitrarily.

Scenario 2:

A file is transferred from 'host B': 2811 (source) where the GridFTP server runs to 'host A' (destination), where the client application runs. In this case the 'host A' data port is chosen from the "*GLOBUS_TCP_PORT_Range*" as defined in 'host A'.

Scenario 3:

A file is transferred from 'host B' (source) to 'host C' (destination), where the GridFTP server runs on both. In this case the 'host B' data port is chosen arbitrarily and the 'host C' data port is chosen from the "*GLOBUS_TCP_PORT_Range*" as defined in 'host C'.

The following command is used in order to transfer files:

```
user>globus-url-copy –dbg gsiftp://source_ip:2811/path_file
        gsiftp://destination_ip:2811/path_file
```

Example:

The following example has been run by user "*nikos*" at grid0.

Verify first, that a proxy certificate is valid. This can be checked by running the following command:

```
nikos@grid0:~> grid-proxy-info
```

The output should look like this:

```
nikos@grid0:~> grid-proxy-info
subject  : /O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-
        31814.uopnet.plymouth.ac.uk/OU=tsi.gr/CN=nikos/CN=proxy
issuer   : /O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-
        31814.uopnet.plymouth.ac.uk/OU=tsi.gr/CN=nikos
identity : /O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-
        31814.uopnet.plymouth.ac.uk/OU=tsi.gr/CN=nikos
type     : full legacy globus proxy
strength : 512 bits
path     : /tmp/x509up_u1000
timeleft : 6:54:59
```

After that, run the following command:

```
nikos@grid0:~> globus-url-copy gsiftp://grid0.tsi.gr:2811/home/nikos/test.pdf   \
                gsiftp://grid3.tsi.gr:2811/home/ioannis/test_done.pdf
nikos@grid0:~>
```

You can choose the "*-dbg*" option in a case of failure to detect errors.

Instead of running command-lines to use GridFTP ("*globus-url-copy*" command), the Java CoG provides a service called "Simple GridFTP Queue".

You can choose "Simple GridFTP Queue" from the toolbar at left.

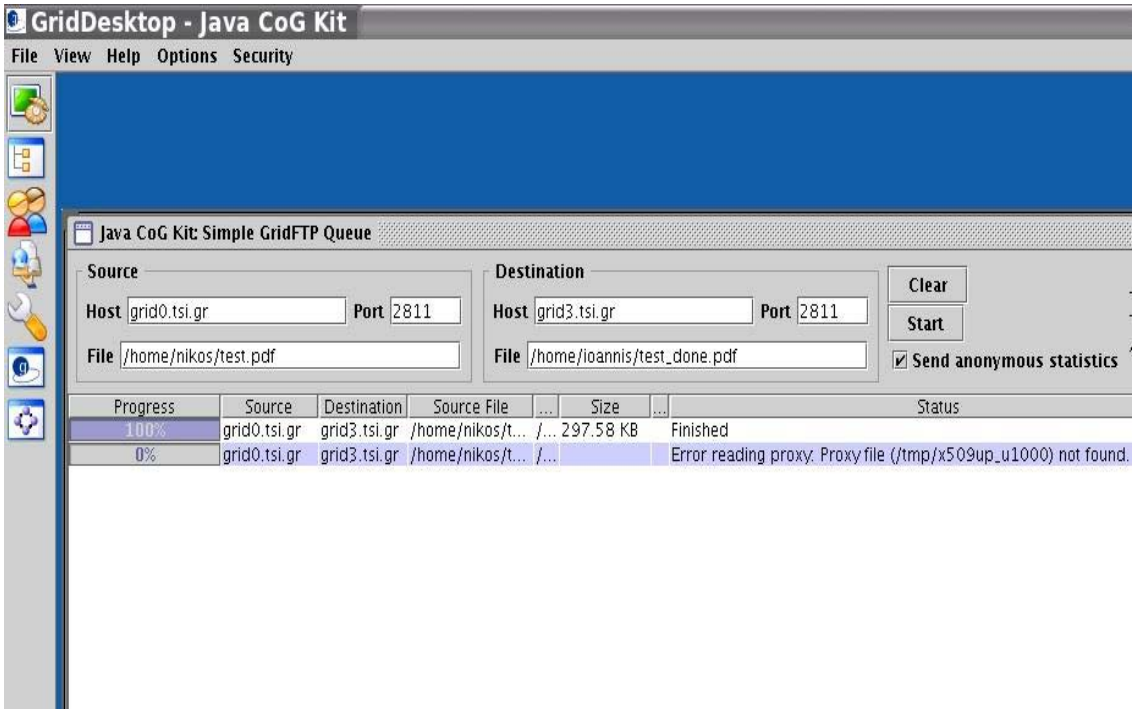


Figure 4.4: CoG screenshot

As shown above, you can submit the source's/destination's host and the file to be transmitted. In the figure 4.4 you can see two tries to transfer "*test.pdf*" from "*grid0.tsi.gr*" to "*grid3.tsi.gr*". In the first try, the transfer is a success, and in the second one is a failure. The second try fails because a proxy was not found at the source side. (I destroyed the proxy certificate in order to check its consistency).

Besides the tests done locally in the grid lab at TSI, further tests have been done in cooperation with the University of Plymouth in England.

The test aimed to check the GridFTP server functionality using the command "*globus-url-copy*".

The test took place in two steps:

1. The first step was to transfer data from UoP to TSI.
2. The Second step was to transfer data from TSI to UoP.

Three matters were to be checked before the test took place:

1. GridFTP server running
2. Valid proxy certificate
3. Open ports through firewall

Contents & schedule of "*globus-url-copy*" test

Test Name	**Test purpose**	
globus-url-copy	Test CA & GridFTP Configurations	
Time and Date	**Hostname & IP address**	**GridFTPServer Port**
British Summer Time 11:00, 26 July 2005	UoP:smb2193n2-31814, 141.163.121.45	2811
Greek Summer Time 13:00, 26 July 2005	TSI:grid3.tsi.gr, 147.27.1.243	2811
Test File	**Data Source Directory**	**Data Destination Directory**
Testfile_1_M.txt (1 Mbytes)	UoP:/home/g-user/BGTSource TSI:/home/ioannis/Source	UoP:/home/g-user/BGTSource TSI:/home/ioannis/Source
Testfile_5_M.txt (5 Mbytes)	//	//
Testfile_10_M.txt (10 Mbytes)	//	//
Testfile_100_M.txt (100 Mbytes)	//	//
Testfile_1024_M.txt (1 Gbytes)	//	//
Testfile_5120_M.txt (5 Gbytes)	//	//

The GridFTP server can be checked by running the following:

user>telnet localhost 2811

The message you get tells you that the server is ready.

To check if you have a valid proxy certificate, run the following:

user>grid-proxy-info

The output should look like this:

subject : subject goes here
issuer : issuer goes here
identity : identity goes here
type : full legacy globus proxy
strength : 512 bits
path : /tmp/x509up_u1000
timeleft : 6:54:59

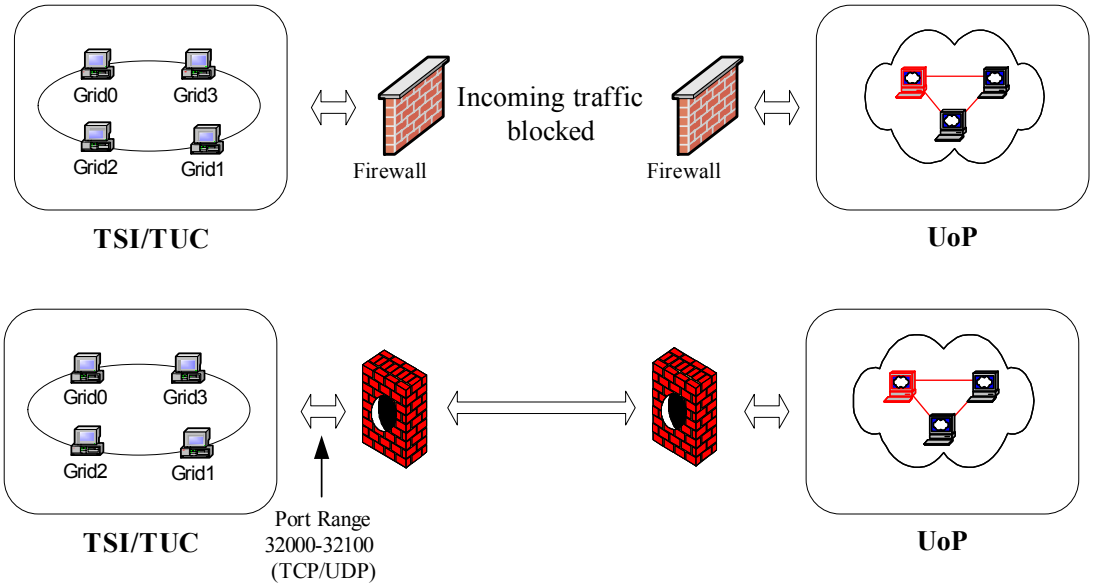


Figure 4.5: Test between TUC & UoP

We have to mention that both TSI and UoP, are behind a firewall. Being behind a firewall means that data, information, etc. can't be received at any ports if they are closed. Sending data is not an issue. The problem appears when someone has to receive incoming traffic data at a specific port. The GridFTP server is listening at port 2811/TCP. When you run the "*globus-url-copy*" command, the GridFTP server invokes some data ports. Data will be transferred within these data ports. A range of ports should be open in the firewall so the transfer may take place. As mentioned above, GridFTP server is based on the FTP protocol which means that files will be transferred in the form of packets (1 to n packets). In order to receive the next packet, the receiver sends back to the sender an acknowledgement telling the sender whether the transfer is done successfully or should be sent again in case of failure. The acknowledgment should find an open port at sender's side in order to be received.

We conclude form the above that at both sides (sender and receiver), a range of "data" ports should be open in the firewall in order for the server to function properly.

You have to configure GT4 to operate to a given port range by setting the environment variable "*GLOBUS_TCP_PORT_RANGE*".

Examples of test commands:

1. Copy data from UoP to TSI

   *user>globus-url-copy file:///home/g-suer/BGTSource/testfile_10_M.txt *
   gsiftp://grid3.tsi.gr:2811/home/ioannis/Destination/testfile_10_M_done.txt

2. Copy data from TSI to UoP

   *user>globus-url-copy file:///home/ioannis/Source/testfile_10_M.txt *

gsiftp://141.163.121.45:2811/home/g-user/GBTDestination/testfile_10_M_done.txt \

4.3.2 Reliable File Transfer (RFT)

The Reliable File Transfer protocol (RFT) [33] is a service that is used to reliably transfer files between different hosts. In order for RFT to work properly, the database system PostgreSQL [32] should be installed and configured.

During the configuration of PostgreSQL database, a "globus" account has been created in order to connect to the database under the name "*rftDatabase*" (The "globus" user account has been created for managing the stand-alone container). The "*rftDatabase*" has been created and populated with the appropriate files through the initial configuration. The "*rftDatabase*" is charged with recording all the requests and checkpoints of the RFT server.

Having configured the RFT service, grid-users can transfer files to grid-certified hosts in a more advanced and reliable transfer mode. That means a grid-user who wishes to transfer files through RFT, can control the following protocol aspects:

1. Data encoding scheme (Binary or ASCII)
2. Block size
3. TCP buffer size
4. Number of parallel streams
5. Data Channel Authentication (DCAU)
6. Number of retries

All of the above protocol aspects are stated in one single file, which has the "*.xfr*" extension.

Example:

```
#true=binary false=ascii
true
#Block size in bytes
16000
#TCP Buffer size in bytes
16000
#Notpt (No thirdPartyTransfer)
false
#Number of parallel streams
1
#Data Channel Authentication (DCAU)
true
# Concurrency of the request
1
#Grid Subject name of the source gridftp server
        subject goes here
#Grid Subject name of the destination gridftp server
```

subject goes here
#Transfer all or none of the transfers
false
#Maximum number of retries
10
#Source/Dest URL Pairs
gsiftp://source_ip:2811/path_file
gsiftp://destination_ip:2811/path_file

After editing the "*.xfr*" file, a grid-user can make the file transfer by running the command "*rft*" along with some arguments. This is shown below:

user>rft -h host_where_container_is_running -f transfer.xfr

Example:

nikos@grid0:~> rft -h grid0.tsi.gr -f transfer.xfr
Number of transfers in this request: 1
Subscribed for overall status
Termination time to set: 60 minutes

Overall status of transfer:
Finished/Active/Failed/Retrying/Pending
0/1/0/0/0

Overall status of transfer:
Finished/Active/Failed/Retrying/Pending
1/0/0/0/0
All Transfers are completed
nikos@grid0:~>

The "*transfer.xfr*" file used in the example above was:

#true=binary false=ascii
true
#Block size in bytes
16000
#TCP Buffer size in bytes
16000
#Notpt (No thirdPartyTransfer)
false
#Number of parallel streams
1
#Data Channel Authentication (DCAU)
true
 # Concurrency of the request
1
#Grid Subject name of the source gridftp server
/O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-
31814.uopnet.plymouth.ac.uk/CN=host/grid0.tsi.gr
#Grid Subject name of the destination gridftp server

/O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-
31814.uopnet.plymouth.ac.uk/CN=host/grid3.tsi.gr
#Transfer all or none of the transfers
false
#Maximum number of retries
10
#Source/Dest URL Pairs
gsiftp://grid0.tsi.gr:2811/home/nikos/testfile.txt
gsiftp://grid3.tsi.gr:2811/home/ioannis/testfile_done.txt

Notes:

- **Ensure that there is a container running.**
- **Ensure that the user who runs the transfers has a valid proxy certificate.**

Besides transferring files, a grid-user can also delete files using "*rft-delete*" command of the RFT service. An "*.xfr*" file is also used in this situation which looks like:

 # Subject name of the gridftp server (if null then defaults to host subject)
subject goes here
#URL
gsiftp://ip_address:2811/path_file_to_be_deleted

Example:

 # Subject name (if null then defaults to host subject)
/O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-
31814.uopnet.plymouth.ac.uk/CN=host/grid3.tsi.gr
#URL
gsiftp://grid3.tsi.gr:2811/home/ioannis/testfile_done.txt

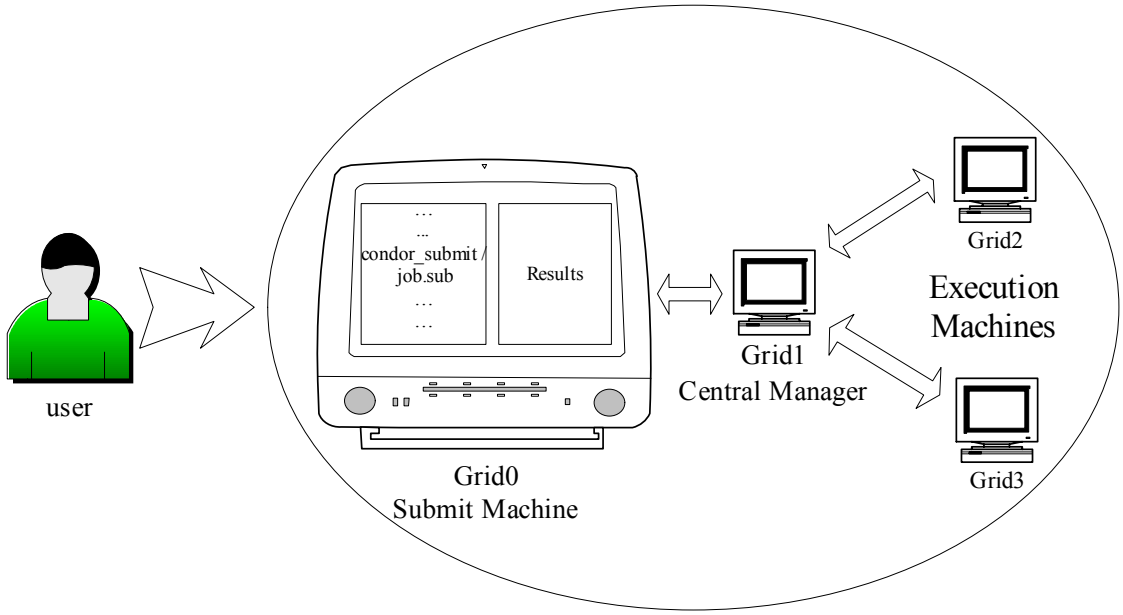
The above "*.xfr*" file deletes "*testfile_done.txt*" which is located in the directory "*/home/ioannis/*" at host "*grid3.tsi.gr*"

To run the example, run the following:

nikos@grid0:~> rft-delete -h grid0.tsi.gr -f delete.xfr
creating a transfer
https://grid0.tsi.gr:8443/wsrf/services/ReliableFileTransferFactoryService
https://grid0.tsi.gr:8443/wsrf/services/ReliableFileTransferService
Subscribed for overall status
Termination time to set: 60 minutes
nikos@grid0:~>

4.3.3 Condor

When a job is sent directly to the Condor pool for negotiation and execution, the whole procedure is illustrated in the following figure:



Example:
Multiple job to a java environment.

universe = java ← Condor Execution Environment (vanilla, globus, java,...)

executable = /home/manolakis/java_tests/Hello.class ← Executable's Path

requirements = (OpSys == "LINUX") && (Arch == "INTEL") ← In order to run on the given machine

arguments = Hello ← The name of the class containing main in the java program

output = /home/manolakis/java_tests/Hello$(PROCESS).output ← Output's Path

log = /home/manolakis/java_tests/Hello$(PROCESS).log ← Log's Path

error = /home/manolakis/java_tests/Hello$(PROCESS).error ← Error's Path

should_transfer_files = YES ← Condor transfers files to and from the remote machine where the job runs

when_to_transfer_output = ON_EXIT ← Transfer job's output only when the job completes

queue 10 ← Specifies how many times to submit the job into the queue

Figure 4.6: Condor Implementaion

As shown above, Grid1 plays the role of the "Central Manager" which means that it is responsible for checking out the state (active/inactive) of each member machine, Grid0 is the submission machine and Grid2 and Grid3 are the execution machines. The job is submitted by running the "*condor_submit file.sub*" command, where "*file.sub*" constitutes the description file of the job as shown in the above figure.

4.3.4 OGSA-DAI

The figure 4.7 below shows the process of accessing a database (e.g. EEG data) through the OGSA-DAI service via the globus stand-alone container.

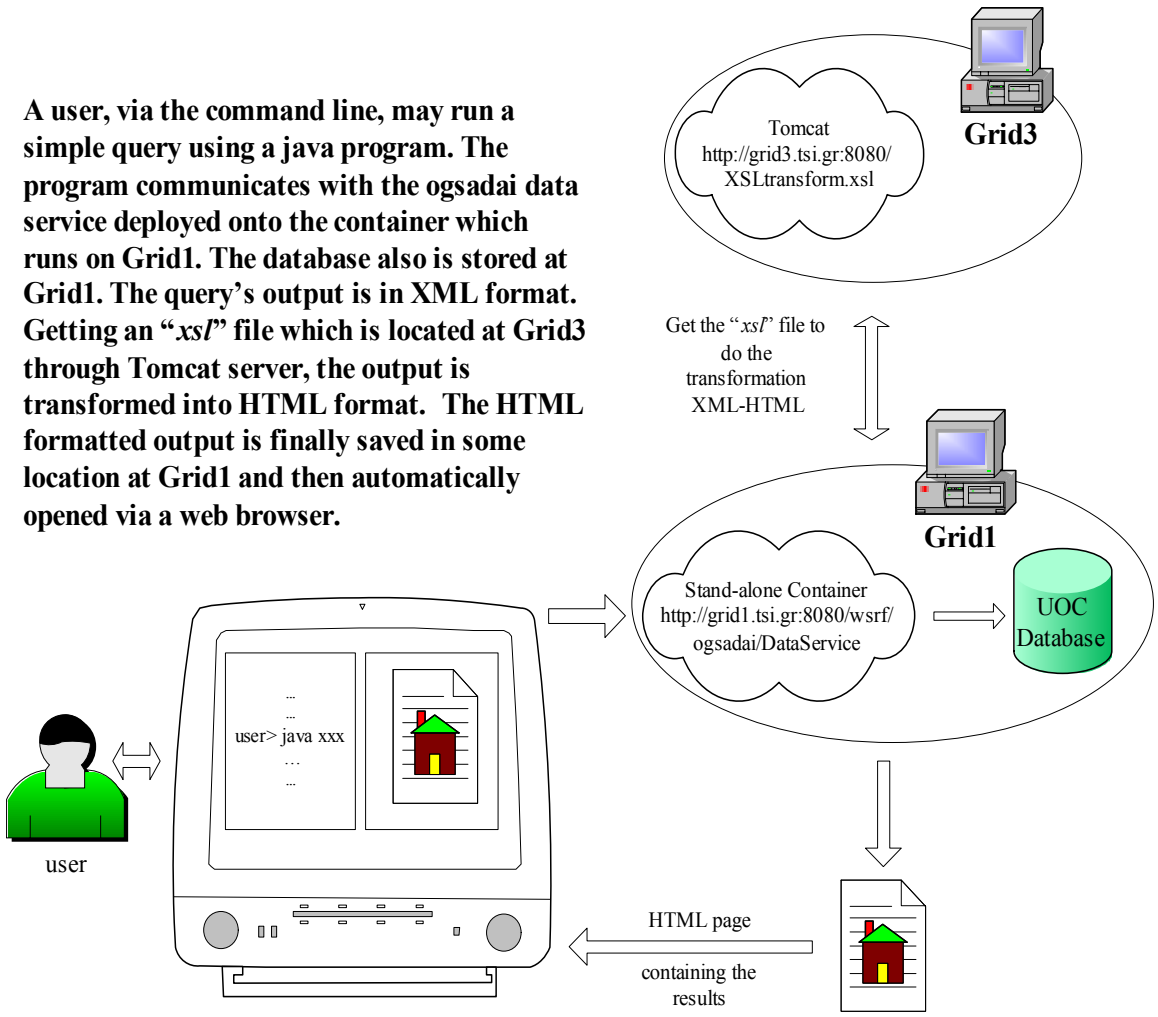


Figure 4.7: OGSA-DAI Implementation

Below we quote the java-code program used in the above description (comments are highlighted for better understanding):

```
import java.io.*;
import java.lang.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import uk.org.ogsadai.client.toolkit.GenericServiceFetcher;
import uk.org.ogsadai.client.toolkit.activity.ActivityRequest;
import uk.org.ogsadai.client.toolkit.activity.delivery.DeliverFromURL;
import uk.org.ogsadai.client.toolkit.activity.sql.SQLQuery;
import uk.org.ogsadai.client.toolkit.activity.sql.WebRowSet;
import uk.org.ogsadai.client.toolkit.activity.transform.XSLTransform;
import uk.org.ogsadai.client.toolkit.service.DataService;
import uk.org.ogsadai.common.FileUtilities;
```

← Classes

```
public class XSLTransformWithDelivery
{
   public static void main(String[] args) throws Exception
   {
        String resourceId = args[0];

        // Factory handle
        String handle = "http://localhost:8080/wsrf/services/ogsadai/DataService";

        // Connect to data service
        DataService service = GenericServiceFetcher.getInstance().getDataService(handle, resourceId);
        System.out.println("\nReady to connect to factory at " + handle);

        // Create a delivery object
        String url = "http://147.27.1.243:8080/tutorial/transformRowSet.xsl";
        DeliverFromURL deliver = new DeliverFromURL( url );

        // Construct a query
        String sql = "select * from clinical where Step = '4'";
        System.out.println("\nPerforming query:  " + sql);
        SQLQuery query  = new SQLQuery(sql);
        WebRowSet rowset = new WebRowSet( query.getOutput() );

        // Construct the transformation activity
        XSLTransform transform = new XSLTransform();
        transform.setXMLInput( rowset.getOutput() );
        transform.setXSLTInput( deliver.getOutput() );

        // Construct the request
        ActivityRequest request = new ActivityRequest();
        request.add(deliver);
        request.add(query);
        request.add(rowset);
        request.add(transform);
        System.out.println("\nPerforming request...\n");
        service.perform( request );

        // Write results to an HTML file
        String toFile = "/usr/local/ogsadai-wsrf-2.1/results.html";
        FileUtilities.getInstance().writeStringToFile(toFile, transform.getOutput().getData());
        System.out.println("The output will open in a Web browser.\n");
        System.out.println("This will take few seconds...\n");

        // Open the HTML file in a web browser
        String[] open_browser = {"konqueror", "/usr/local/ogsadai-wsrf-2.1/results.html"};
        Runtime.getRuntime().exec( open_browser );                    }
}
```

Figure 4.8: Java code

4.3.5 GRAM Implementation

"*globusrun-ws*" [41] (WS GRAM client) is a program for submitting and managing jobs to a local or remote job host. WS GRAM provides secure job submission to many types of job scheduler for users who have the right to access a job hosting resource in a Grid environment. In other words, the jobs are submitted to the Job Managing Factory Service in the Globus stand-alone container at https port 8443/TCP.

"*globusrun-ws*" offers additional features to fetch job output files incrementally during the run as well as to automatically delegate credentials needed for certain optional WS GRAM features.

Example:

user>grid-proxy-init
user>globusrun-ws –submit –J –S –f file.xml

Here is an example of an "*.xml" that is used to submit jobs (read the notes for further information):

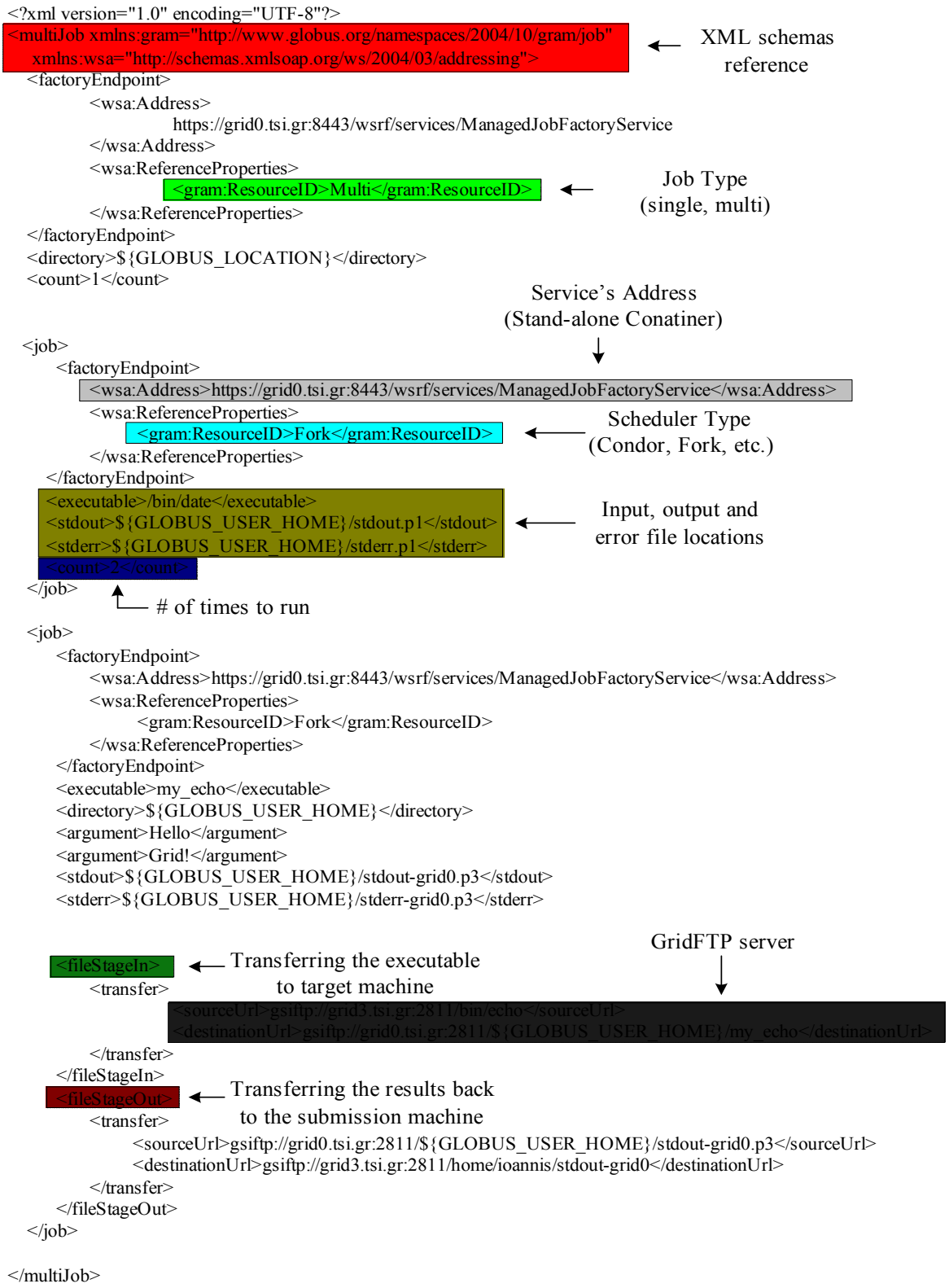


Figure 4.9: XML code

5 Conclusions

The Grid is emerging as a new mean for solving problems in science, engineering, industry and commerce. A computing node or even more an entire single site can no longer meet all the resource needs of today's demanding applications. Moreover, using distributed resources can bring many benefits to application users.

During this thesis, we have made an extensive use of the Globus Toolkit and its components like GridFTP, RFT, GRAM and others. Besides that, OGSA-DAI and Condor job scheduler proved to be prerequisites during this thesis.

Almost all of our interest has been exhausted on understanding the aspects of Grids in general, on setting up a Grid testbed consisting of 4 nodes and on the evaluation of the potential as well as the actual benefits Grid environments can provide the scientific community nowadays.

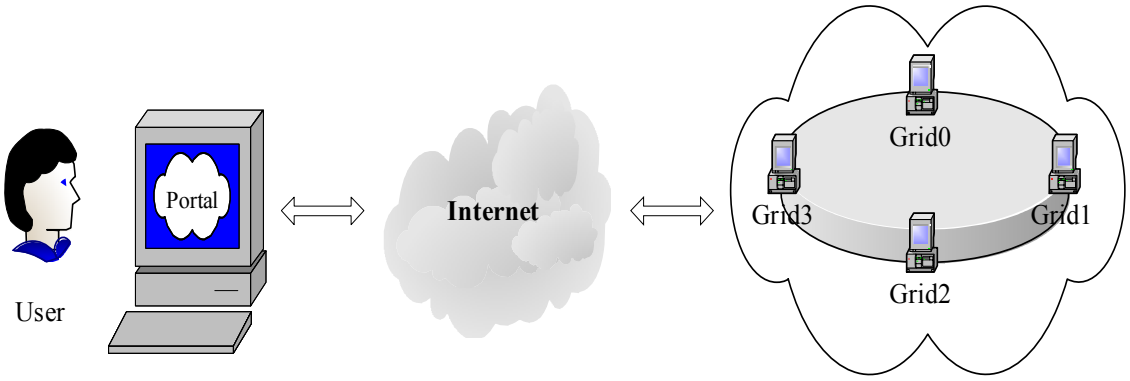
6 Future Work

During the last decade, a lot of people engaged in Grid technologies have adopted the principle of Ian Foster (recognized as the "father of grid computing"): "Basically, we were looking to establish a set of techniques or standards that would enable the federation or sharing of resources in a distributed environment. By establishing a new way of computing, we believed we could transform the way that scientists around the globe worked. In turn, we thought, this would ultimately improve the research process itself".

Moreover, Ian Foster explains: "One area that will be particularly interesting to watch is how enterprise service providers incorporate grid into their own production environments, thereby extending the benefits of grid to their customers via their products and services".

Here are some interesting areas that could be researched and worked on in the near future:

- Resource allocation on demand.
- Continuous availability of resources.
- Efficient allocation of physical resources.
- Access to any resources, for anyone, anywhere, anytime, from any platform (portals, see the figure below), keeping in mind security.
- End-user tools.
- Monitoring and managing applications through wireless devices (PDAs etc.).
- Solutions provision in real-time.
- Standardization and industry adoption.



Grid at TSI

7 Acknowledgments

I am grateful to Dr. Ioannis Barbounakis for his help and input in this thesis.

8 References

[1] Globus, http://www.globus.org/toolkit

[2] The Globus Alliance, http://www.globus.org

[3] Grid Computing, http://www.gridcomputing.com/

[4] Grid Café, http://gridcafe.web.cern.ch/gridcafe

[5] Metacomputing, http://www.hpti.com/tblTopicsHomeTemplate.asp?ID=114 , http://www.epcc.ed.ac.uk/DIRECT/grid/node10.html

[6] Cluster Computing, http://en.wikipedia.org/wiki/Computer_cluster

[7] Peer-to-Peer, http://en.wikipedia.org/wiki/Peer-to-peer

[8] Condor, http://www.cs.wisc.edu/condor

[9] "The Grid: Core Technologies" by Maozhen Li and Mark Baker

[10] Cryptography, http://www.cryptography.org/

[11] Symmetric Cryptography, http://www.maths.mq.edu.au/~steffen/old/PCry/report/node8.html

[12] Asymmetric Cryptography, http://www.maths.mq.edu.au/~steffen/old/PCry/report/node8.html

[13] Hash function, http://www.nist.gov/dads/HTML/hash.html

[14] Public Key Infrastructure (PKI), http://www.pki-page.org/

[15] X.509 Certificates, http://java.sun.com/j2se/1.3/docs/guide/security/cert3.html

[16] Grid Security Infrastructure (GSI), http://www-unix.globus.org/toolkit/docs/3.2/gsi

[17] Secure Sockets Layer (SSL), http://ww.openssl.org/

[18] Proxy Certificate, http://www.globus.org/security/proxy.html

[19] Mutual Authentication, http://www.growl.org.uk/portal_guide/node18.html

[20] Globus Gatekeeper, http://egee.in2p3.fr/html/Public/Documentation/Organigram-0.1/node6.html

[21] GridFTP, http://www-unix.globus.org/toolkit/docs/3.2/gridftp/

[22] Condor Job Submission, http://www.cs.wisc.edu/condor/manual/v6.2/condor_submit.html

[23] Parallel Virtual Machine (PVM), http://www.csm.ornl.gov/pvm/pvm_home.html

[24] Messaging Passing Interface (MPI), http://www-unix.mcs.anl.gov/mpi

[25] MPICH, http://www-unix.mcs.anl.gov/mpi/mpich/

[26] WS-Gram client (globusrun-ws), http://globus.org/toolkit/docs/4.0/execution/wsgram/rn01re01.html

[27] Condor-G, http://www.cs.wisc.edu/condor/condorg/

[28] SuSe Linux, http://www.novell.com/linux/suse/

[29] Slackware Linux, http://www.slackware.com/

[30] Java Software Development Kit (JSDK), http://mindprod.com/jgloss/jsdk.html

[31] Apache Ant, http://ant.apache.org/

[32] PostgreSQL, http://www.postgresql.org/

[33] Reliable File Transfer (RFT), http://www-unix.globus.org/toolkit/docs/3.2/rft/

[34] Replica Location Service (RLS), http://www.globus.org/rls/

[35] MySQL, http://www.mysql.org

[36] OGSA-DAI, http://www.ogsadai.org.uk

[37] Web Services Resource Framework (WSRF), http://www.globus.org/wsrf/

[38] Simple Object Access Protocol (SOAP), http://www.webopedia.com/TERM/S/SOAP.html

[39] Apache Tomcat, http://tomcat.apache.org

[40] File Transfer Protocol (FTP), http://www.w3.org/Protocols/rfc959/

[41] globus-url-copy, http://www.westgrid.ca/support_old/topics/globusurlcopy.php

[42] "The Anatomy of the Grid: Enabling Scalable Virtual Organizations" by Ian Foster, Carl Kesselman and Steven Tuecke Intl. Journal of Supercomputing Applications 2001

[43] Open Grid Services Architecture (OGSA), http://www.globus.org/ogsa/

[44] Quality of Service (QoS), http://wwwfp.globus.org/retreat00/presentations/i_volker_qos_slides/sld001.htm

[45] Web Services http://www.w3.org/2002/ws/

[46] Open Grid Services Infrastructure (OGSI), http://www-unix.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf

[47] Extensible Markup Language (XML), http://www.w3.org/XML/

[48] Web Services Description Language (WSDL), http://www.w3.org/TR/wsdl

[49] Global Grid Forum (GGF), http://www.gridforum.org/

[50] Service-Oriented Architecture (SOA), http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

[51] Grid Resource Allocation and Management (GRAM), http://www-unix.globus.org/toolkit/docs/3.2/gram/key/index.html

[52] OGSADAI, http://www.ogsadai.org.uk

[53] Web Services Addressing (WS-Addressing), http://www.w3.org/Submission/ws-addressing

[54] Java API for XML-Based RPC (JAX-RPC), http://java.sun.com/webservices/jaxrpc/index.jsp

[55] Monitoring & Discovery System (MDS), http://www.globus.org/mds/

[56] Portable Batch System (PBS), http://www.openpbs.org/

[57] Load Sharing Facility (LSF), http://accl.grc.nasa.gov/lsf/

[58] Replica Location Service (RLS) Glossary, http://www.globus.org/toolkit/docs/4.0/data/rls/RLS_Glossary.html

[59] ClassAd Condor, http://www.cs.wisc.edu/condor/classad/

[60] iODBC, http://www.iodbc.org/

[61] GT 4 Glossary, http://www.globus.org/toolkit/docs/4.0/security/key-index.html

[62] “A Study of Site Security and Grid Computing Policies” by Kasidit Chanchio and Al Geist and Meili Chen

[63] “What Is the Grid? A Three Point Checklist” by Ian Foster

[64] "Grids: The Top Ten Questions" by Jennifer M. Schopf

[65] "Globus Security Architect", Personal Communication by WELCH, V., 2001.

[66] "A Web Services Data Analysis Grid" by William A. Watson, Ian Bird, Jie Chen, Bryan Hess, Andy Kowalski, Ying Chen

[67] "Reliable Data Transport: A Critical Service for the Grid" by William E. Allcock, Ian Foster, Ravi Madduri

[68] "Performance and Scalability of a Replica Location Service" by Ann L. Chervenak, Naveen Palavalli, Shishir Bharathi, Carl Kesselman, Robert Schwartzkopf

[69] "Replica Management in Data Grids" by L. Guy, 2002

[70] "Worldwide Fast File Replication on Grid Datafarm" by Osamu Tatebe 2003

[71] "Mapping Abstract Complex Workflows onto Grid Environments" by E. Deelman Journal of Grid Computing, vol. 1, pp. 25-39, 2003.

[72] "Open Grid Services Architecture: a framework for commercial" by Jeff Nick

[73] "The commercial power of Grid" by Andreas Kerstan

[74] "Secure, efficient data transport and replica management for high-performance data-intensive computing" by B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. In IEEE Mass Storage Conference, April 2001

[75] "Globus: A metacomputing infrastructure toolkit" by I. Foster and C. Kesselman Intl. Journal of Supercomputing Applications 11(2):115-128, 1997

9 Appendix A: Globus Toolkit

The Globus Toolkit [1] [2] is an open source software toolkit used for building grids. It is being developed by the Globus Alliance [1] [2]and many others all over the world.

The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. Every organization has unique modes of operation, and collaboration between multiple organizations is hindered by incompatibility of resources such as data archives, computers, and networks. Its core services, interfaces and protocols allow users to access remote resources as if they were located within their own machine room while simultaneously preserving local control over who can use resources and when to use them [1].

A.1 Open Grid Services Architecture OGSA

Grid systems and applications aim to integrate, virtualize and manage resources and services within distributed dynamic "virtual organizations" (VOs) [42]. The realization of this goal requires the disintegration of the numerous barriers that normally separate different computing systems within and across organizations, so that computers, application services, data, and other resources can be accessed as and when required, regardless of physical location [1].

Key to the realization of this Grid vision is "standardization" [42], so that the several components that make up a modern computing environment can be discovered, accessed, allocated, monitored, accounted for and, in general managed as a single virtual system - even when provided by different vendors and/or operated by different organizations. Standardization can contribute into the development of secure, robust, and scalable Grid systems by facilitating the use of good practices.

Open Grid Services Architecture (OGSA) [43] is the outcome of such a standardization process. OGSA is a set of core capabilities and behaviors that address key concerns in Grid systems.

These concerns include issues such as:

- How do I establish identity and negotiate authentication?
- How is policy expressed and negotiated?
- How do I discover services?
- How do I negotiate and monitor service level agreements?
- How do I manage membership of, and communication within, virtual organizations?

- How do I integrate data resources into computations? How do I monitor and manage collections of services?

The objectives of OGSA are to:

- Manage resources across distributed heterogeneous platforms.
- Deliver seamless quality of service (QoS) [44]. The topology of grids is often complex. Interaction of grid resources is usually dynamic. It's important that the grid provide robust, behind-the-scenes services such as authorization, access control, and delegation.
- Provide a common base for autonomic management solutions. A grid can contain many resources, with numerous combinations of configurations, interactions, and changing state and failure modes. Some form of intelligent self regulation and autonomic management of these resources is necessary.
- Define open, published interfaces. OGSA is an open standard managed by the GGF [13] standards body. For interoperability of diverse resources, grids must be built on standard interfaces and protocols.
- Exploit industry-standard integration technologies. The authors of OGSA had foresight to leverage existing solutions where appropriate. The foundation of OGSA is Web services [45].

Four main layers comprise the OGSA architecture:

- Resources - physical resources and logical resources
- Web services, plus the OGSI [46] extensions that define grid services
- OGSA architected services
- Grid applications

Physical and Logical Resources Layer

Resources comprise the capabilities of the grid, and are not limited to processors. Physical resources include servers, storage, and network. Above the physical resources are logical resources. They provide additional function by virtualizing and aggregating the resources in the physical layer. General purpose middleware such as file systems, database managers, directories, and workflow managers provide these abstract services on top of the physical grid.

Web Services Layer

The second layer in the OGSA architecture is Web services [45]. Both logical and physical resources are modeled as services. The Open Grid Services Infrastructure (OGSI) [46] specification defines grid services and builds on top of standard Web services technology. OGSI exploits the mechanisms of Web services like XML [47] and WSDL [48] to specify standard interfaces, behaviors, and interaction for all grid resources. OGSI extends the definition of Web services to provide capabilities for

dynamic, stateful, and manageable Web services that are required to model the resources of the grid.

OGSA Architected Grid Services Layer

The Web services layer, with its OGSI extensions, provides a base infrastructure for the next layer - architected grid services. The Global Grid Forum (GGF) [49] is currently working to define many of these architected grid services in areas like program execution, data services, and core services. Some are already defined, and some implementations have already appeared. As implementations of these newly architected services begin to appear, OGSA will become a more useful service-oriented architecture (SOA) [50].

Grid Applications Layer

Over time, as a rich set of grid-architected services continues to be developed, new grid applications that use one or more grid architected services will appear. These applications comprise the fourth main layer of the OGSA architecture.

The base component of that architecture is the "Open Grid Service Infrastructure" (OGSI). This is a grid software infrastructure standardization effort based on the emerging Web service standards to provide maximum interoperability among OGSA software components.

Difference between OGSA and OGSI

Architecture like OGSA defines a grid service, and an infrastructure like OGSI can be implemented by individual tools to create an actual grid service.
For example, if you wanted a bridge (grid service) built, you'd hire an architect (OGSA). This architect would draw up all the plans, but he would also work with a structural engineer (OGSI), who would be in charge of making sure the right materials were used in the building of the bridge. Finally, all of these plans and specifications would be given to a team of construction workers (Globus Toolkit), who would build the bridge according to the blueprints and specifications.
HTTP is stateless. A web application based on a stateless protocol means that you send in a request, you get back a response. End of story. The server doesn't know if a particular request is the first, twentieth, or the millionth request you've made. It certainly doesn't know if a particular request is somewhere in the middle of a long series of requests, all of which need to be handled in order.
In order to add statefulness to applications, a combination of tools - databases, XML files, cookies, and session handling - outside the HTTP protocol should be used. One of the best-known (and most-used) examples of stateful applications on the Web is shopping carts. They keep track of what you add or remove from the shopping cart. The data values are stored from page view to page view and even evolve as the interaction increases (for example, the price goes up the more things you add to the cart). Many extranets, intranets, and portals have to rely on a combination of database records and cookies/sessions to control who sees what, when they get kicked off the system, and so on.

When Web services first appeared, they were stateless. The only way to provide state information/security to web services is the insertion of a token or some other identifier into the envelopes. However, there still remains a bit of dissent in the whole matter.

Afterwards grid standardized technology known as "Open Grid Services Infrastructure" (OGSI). OGSI recasts grid services as Web services by using Web Services Definition Language (WSDL) and other available XML tools.
The Web Services Description Language (WSDL) is an XML-type description language for Web services as a set of "endpoints" operating on messages containing either document-oriented or remote procedure call (RPC) payloads. Service interfaces are defined abstractly in terms of message structures and sequences of simple message exchanges (or operations, in WSDL terminology) and then bound to a concrete network protocol and data-encoding format to define an endpoint. Related concrete endpoints are bundled to define abstract endpoints (services). WSDL is extensible to allow description of endpoints and the concrete representation of their messages for a variety of different message formats and network protocols.

Grid Services needed state.

A.2 Web Services Resource Framework (WSRF)

The WS-Resource Framework [37] consists of six Web services specifications that define the "WS-Resource approach" to modeling and managing state in a Web services context.

Web services [45] must often provide their users with the ability to access and manipulate state, i.e. data values that persist across, and evolve as a result of, Web service interactions. Also, while Web services successfully implement applications that manage state today, it is desirable to define Web service conventions to enable the discovery of, introspection on, and interaction with stateful resources in standard and interoperable ways. WSRF defines these conventions and does so within the context of established Web services standards.

The WSRF specifications

- **WS-ResourceLifetime** defines mechanisms for WS-Resource destruction, including message exchanges that allow a requestor to destroy a resource, either immediately or by using a time-based scheduled resource termination mechanism.
- **WS-ResourceProperties** defines how the type definition of a WS-Resource can be associated with the interface description of a Web service, and message exchanges for retrieving, changing, and deleting WS-Resource properties.
- **WS-Notification** defines mechanisms for event subscription and notification using a topic-based publish/subscribe pattern.
- **WS-RenewableReferences** defines a conventional decoration of a WS-Addressing [32] endpoint reference with policy information needed to retrieve an updated version of an endpoint reference when it becomes invalid.

- **WS-ServiceGroup** defines an interface to heterogeneous by-reference collections of Web services.
- **WS-BaseFaults** defines a base fault XML type for use when returning faults in a Web services message exchange.

WSRF retains essentially all of OGSI concepts, and introduces only modest changes to OGSI messages and their associated semantics.

Services implemented using OGSI-based tools, such as the Globus Toolkit's OGSI Core, are likely to require some changes to exploit WSRF-based tools, but the changes should be modest due to the similarities between WSRF and OGSI.

Applications that use higher-level interfaces, such as the Globus Toolkit's GRAM [51] or emerging standards such as OGSA-DAI [52], will be only minorly affected by these changes.

More generally, the fact that WSRF is based on mainstream Web services standards and has been embraced by major vendors means that we can expect to see rapid integration into commercial Web services products, enabling a much richer choice of products upon which WSRF compliant services can be built.

WSRF specifications build directly on core Web services standards, in particular WSDL, SOAP [38], and XML, and exploit capabilities provided by WS-Addressing [53].

WSRF represents a refactoring and evolution of OGSI that delivers essentially the same capabilities in a manner that is more in alignment with the Web services community. As such, it represents an important next step towards the larger goal of a comprehensive Open Grid Services Architecture (OGSA) that supports on-demand, utility computing, collaborative and other "Grid" scenarios within a Web services setting.

The most valuable aspect of WSRF is that it effectively completes the convergence of the Web service and Grid computing communities.

OGSI Grid Services have several drawbacks which have blocked Web Services and Grid Services convergence such as:

- ***Too much stuff in one specification***. Many would like to use parts but not all of OGSI, and while most of OGSI v1.0 is optional, some feel that use of parts obligates use of all.

  Response: WSRF partitions OGSI v1.0 functionality into a family of composable specifications.

- ***Does not work well with existing Web services tooling***. OGSI v1.0 uses XML Schema fairly aggressively, for example with substantial use of "*xsd:any*",

attributes, etc., and “document-oriented” WSDL operations. These features cause problems with, for example, JAX-RPC [54].

Response: WSRF tones down the usage of XML Schema somewhat.

- ***Much oriented***. OGSI v1.0 models a stateful resource as a Web service that encapsulates the resource's state, with the identity and lifecycle of the service and resource state coupled. This approach has spurred anxiety among some Web services purists who argue that "Web services do not have state or instances." In addition, some Web services implementations do not accommodate dynamic service creation and destruction.

  Response: WSRF re-articulates the underlying OGSI architecture to make an explicit distinction between the “service” and the stateful entities acted upon by that service. WSRF calls these entities “resources”, and says that a service that acts upon resources through a conventional use of WS-Addressing [53] exhibits the “implied resource pattern”.

  The change from OGSI to WSRF represents a change in the fundamental message exchanges and XML definitions that underlie GT. Thus, while these changes are for the most part minor and syntactic, the effect is that a WSRF-based GT4 cannot interoperate (at the message level) with GT3.

  Nevertheless, GT4 will in all other respects be designed to maximize portability of applications and services, via the use of the same programming model and client-side APIs wherever possible. Further, all of the capabilities embodied in the GT3 OGSI-compliant services will evolve into GT4 WSRF-compliant services.

The “Web Services Resource Framework” (WSRF) came to solve OGSI’s problems:

- WSRF aims to be integrated into the family of WS standards
- OGSA will be based on Web Services instead of OGSI Grid Services.

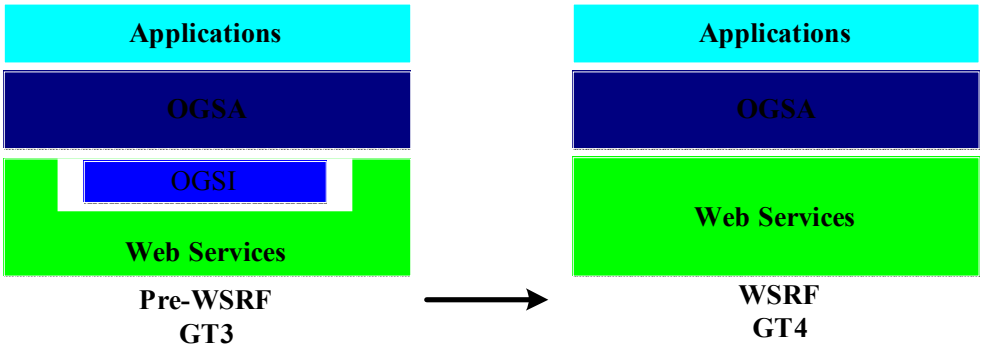


Figure A.1: OGSI vs. WSRF

A.3 Globus Toolkit 3 (GT3)

GT3 is OGSA/OGSI compatible and its major features are:

- Security (GSI),
- Remote job submission and control (GRAM),
- High-performance secure data transfer (GridFTP),
- Reliable File Transfer (RFT),
- Replica Location Service (RLS),
- Interfaces to system and service information (MDS),
- Web Services

Grid Services are the core of GT3.

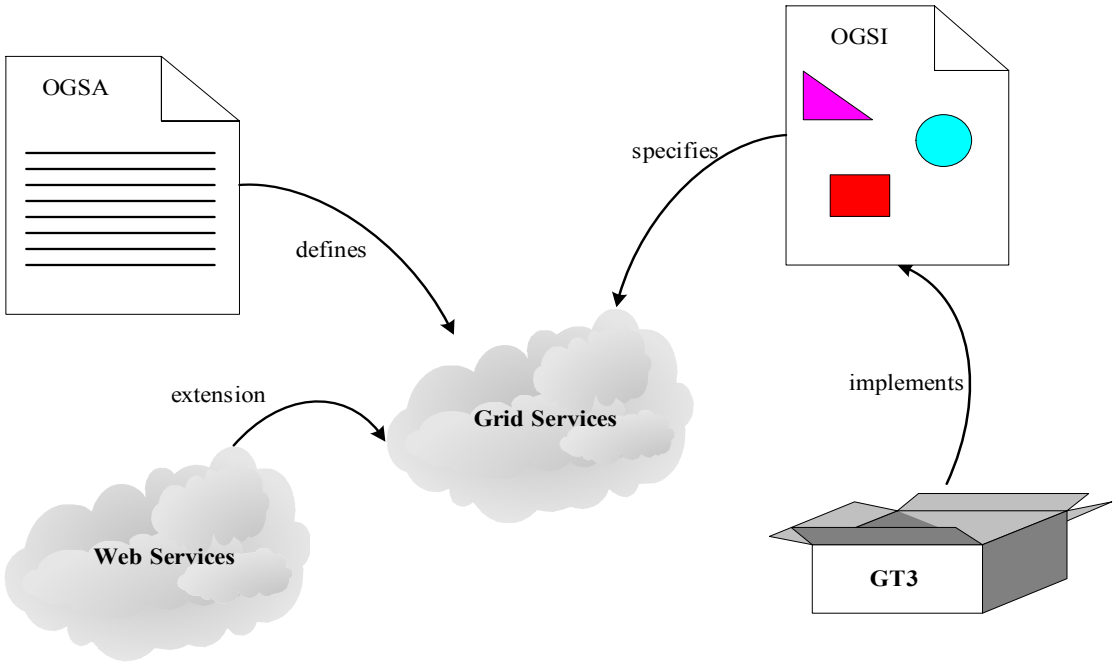


Figure A.2: Relation between Web Services and Grid Services

A.4 Globus Toolkit 4.0 (GT4)

Globus Toolkit 4 is the first WSRF-based version.
It has almost the same set of major features as in GT3:

- Security (GSI),
- Remote job submission and control (GRAM),
- High-performance secure data transfer (GridFTP),
- Reliable File Transfer (RFT).

GT4 includes also the following:

- WSRF
- OGSA-DAI WSRF version
- Web MDS
- Updated Data Replication Service

Probably the most noteworthy feature of GT4 is the inclusion of a much wider spectrum of Web services capabilities and standards than the previous GT versions. GT4 is also greatly advanced beyond previous versions in terms of:

1. Quality of testing
2. Documentation
3. Performance
4. Robustness.

A.5 Differences between GT3 and GT4

GT3 was the first Globus Toolkit to support Web Services, via Open Grid Services Infrastructure (OGSI), a predecessor of WS-Resource Framework (WSRF). Unfortunately, OSGI wasn't an industry-recognized standard, and the GT3 software was not of production quality. The latter is proved during this thesis where GT3 had a lot of bugs. One is concerning the stand-alone container that started with a lot of errors, where in GT4 there was not any problem. Another problem concerning GT3 is the release notes. The installation guides are ambiguous, where in GT4 the installation guides are written much better, i.e. less ambiguous.

GT4 uses WSRF mechanisms instead of OGSI that is used in GT3.

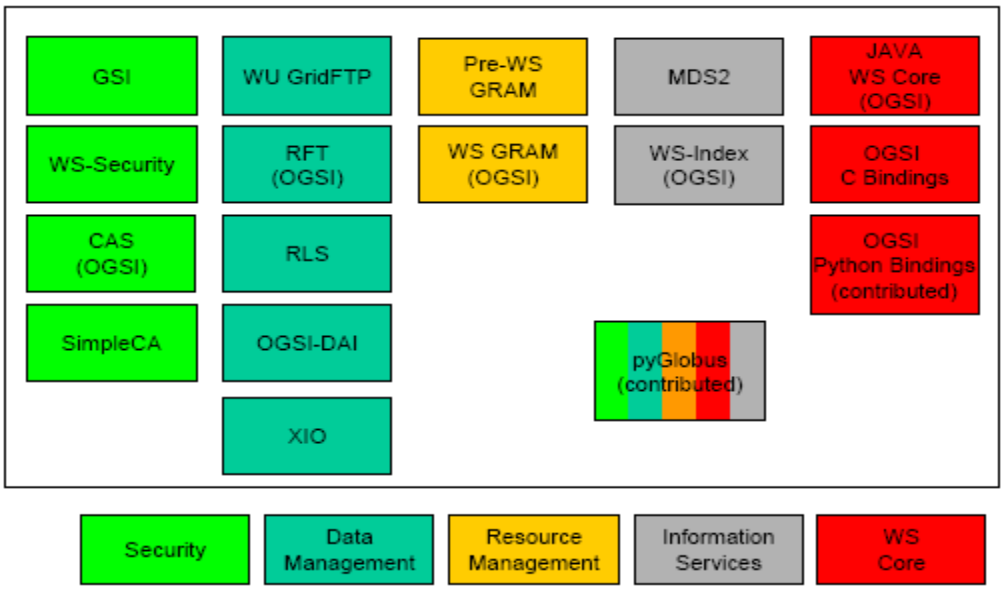


Figure A.3: GT3.2 Components

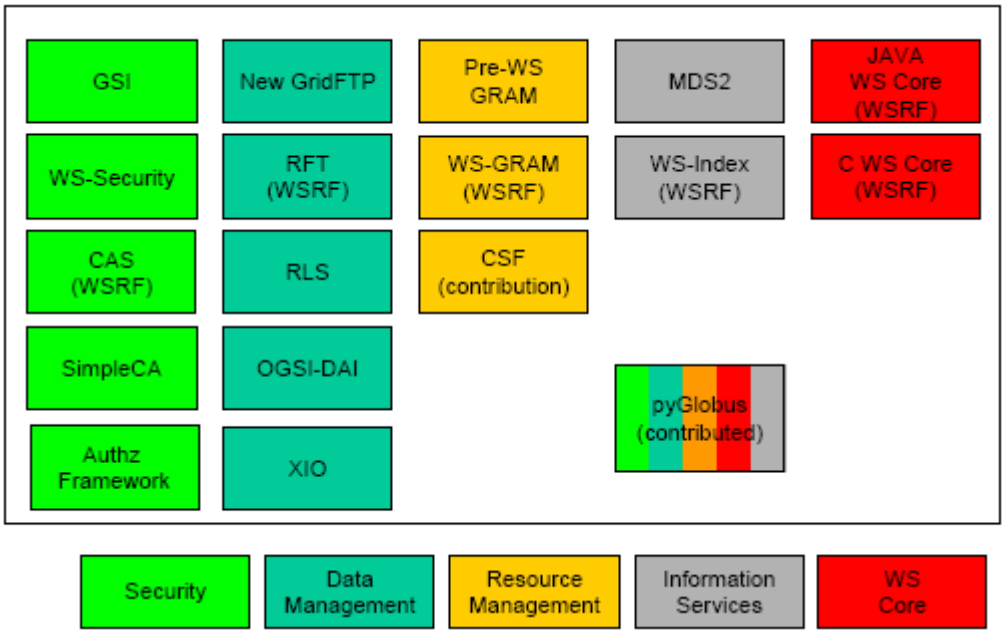


Figure A.4: GT4 Components

A.6 GT4 Traffic Ports

Application	Network Ports
GRAM Gatekeeper	2119/tcp
GRIS/GIIS	2135/tcp
Container (SOAP server)	8080/tcp-8443/tcp
GridFTP	2811/tcp (control port) *GLOBUS_TCP_PORT_RANGE* (data ports)
GSI-SSH	22/tcp
MyProxy	7512/tcp
MDS	8080 via container
GRAM	8443 via container
PostgreSQL	5432/tcp
MySQL	3306/tcp

Table A.1: GT4 Traffic Ports

If you are a member of a research or academic institution, then you are behind a firewall. If you want to cooperate with other institutions that are also behind a firewall, then you have to configure GT4 to operate to a given port range that is configured "open" in the firewall at each side.

This is done by setting the environment variable "*GLOBUS_TCP_PORT_RANGE*" [21] in GT4.

For instance, in order to configure the GridFTP server to use "*GLOBUS_TCP_PORT_RANGE*", the "*inetd/xinted*" daemons need to be configured.

For "*xinetd*" add a line to your "*/etc/xinet.d/gsiftp*" file. For example:

```
service gsiftp
{
        instances          = 100
        socket_type        = stream
        wait               = no
        user               = root
        env               += GLOBUS_LOCATION=/usr/local/gt4
        env               += LD_LIBRARY_PATH=/usr/local/gt4/lib
        server             = /usr/local/gt4/sbin/globus-gridftp-server
        server_args        = -i
        log_on_success    += DURATION
        nice               = 10
        disable            = no
        env               += GLOBUS_TCP_PORT_RANGE=32000, 32100
}
```

The above string would need to be customized to reflect your configuration by replacing "32000, 32100" with you choice of port range. Alternatively, the "*GLOBUS_TCP_PORT_RANGE"* variable should be set in "*/etc/profile.local*".

The Gatekeeper can be configured the same way.

MDS [55] does not need to accept connections so it does not need to be configured to use "*GLOBUS_TCP_PORT_RANGE"*.

A.7 Grid Scheduling and Resource Management

The Grid is emerging as a new paradigm for solving problems in science, engineering, industry and commerce. Increasing numbers of applications are utilizing the Grid infrastructure to meet their computational, storage and other needs. A single site can simply no longer meet all the resource needs of today's demanding applications and using distributed resources can bring many benefits to application users. The deployment of Grid systems involves the efficient management of heterogeneous, geographically distributed and dynamically available resources. However, the effectiveness of a Grid environment is largely dependent on the effectiveness and efficiency of its schedulers, which act as localized resource brokers [9].

Figure A.5 [9], shows that user tasks, for example, can be submitted via Globus to a range of resource management and job scheduling systems, such as Condor [3], the Portable Batch System (PBS) [56] and the Load Sharing Facility (LSF) [57].

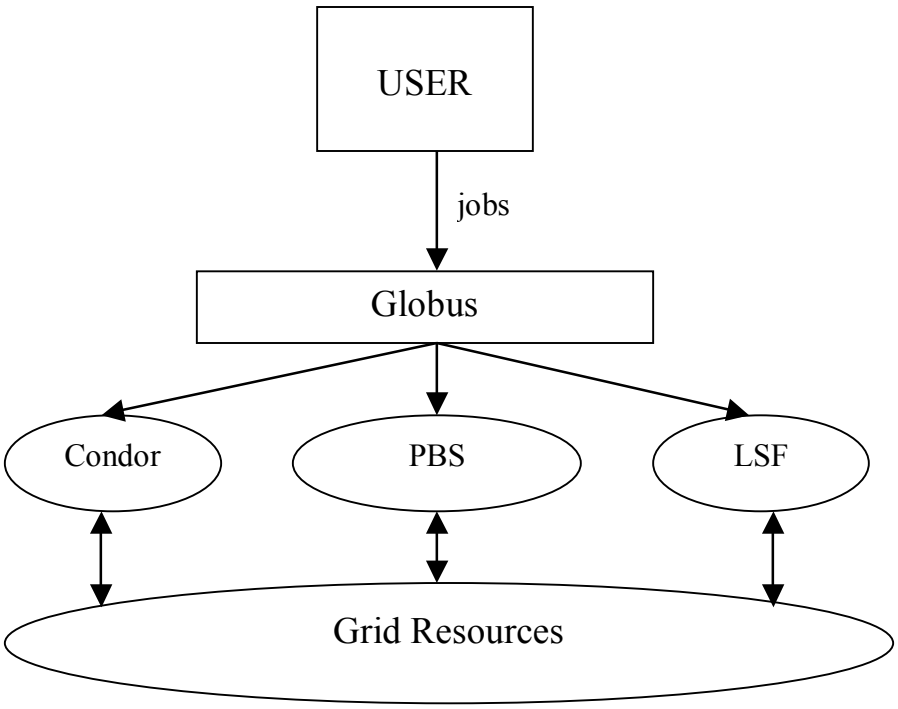


Figure A.5: Jobs, via Globus, can be submitted to systems managed by Condor, PBS and LSF.

Grid scheduling is defined as the process of mapping Grid jobs to resources over multiple administrative domains. A Grid job can be split into many small tasks. The scheduler has the responsibility of selecting resources and scheduling jobs in such a way that the user and application requirements are met, in terms of overall execution time (throughput) and cost of the resources utilized [9].

Grid scheduling involves four main stages:

1. resource discovery
2. resource selection
3. schedule generation
4. job execution

Resource discovery

The goal of resource discovery is to identify a list of **authenticated** resources that are available for job submission. In order to cope with the dynamic nature of the Grid, a scheduler needs to have some way of incorporating dynamic state information about the available resources into its decision-making process [9].

This decision-making process is somewhat analogous to an ordinary compiler for a single processor machine. The complier needs to know how many registers and functional units exist and whether or not they are available or busy. It should be also aware of how much memory it has to work with, what kind of cache configuration has been implemented and the various communication latencies involved in accessing these

resources. It is through this information that a compiler can effectively schedule instructions to minimize resource idle time [9].

Similarly, a scheduler should continuously know what resources it can access, how busy they are, how long it takes to communicate with them and how long it takes for them to communicate with each other. With this information, the scheduler optimizes the scheduling of jobs to make more efficient and effective use of the available resources [9].

Resource selection

Once the list of possible target resources is known, the second phase of the scheduling process is to select those resources that best suit the constraints and conditions imposed by the user, such as:

- CPU usage
- RAM available
- Disk storage

Schedule generation

The generation of schedules involves two steps, selecting jobs and producing resource selection strategies. Once all resources could meet the minimum requirements imposed by the job, an algorithm is needed to choose the best resource(s) to execute the job. The resource selection algorithm should take into account the current state of resources and choose the best one based on a quantitative evaluation, for example, a resource selection algorithm can take CPU and RAM into account [9].

The goal of job selection is to select a job from a job queue for execution. Three strategies that can be used to select a job are given below:

- First come first serve: the scheduler selects jobs for execution in the order of their submissions. If there is no resource available for the selected job, the scheduler will wait until the job can be started. The other jobs in the queue have to wait. There are two main drawbacks with this type of job selection. It may waste resources when, for example, the job selected needs more resources to be available before it can start, which results in a long waiting time. And jobs with high priorities cannot get dispatched immediately if a job with a low priority needs more time to complete.
- Random selection: The next job to be scheduled is randomly selected from the job queue. Jobs selection by this strategy in not fair and job submitted earlier may not be scheduled until much later.
- Priority-based selection: Jobs submitted to the scheduler have different priorities. The next job to be scheduled is the job with the highest priority in the job queue. A job priority can be set when the job is submitted. One drawback of this strategy is that it is hard to set an optimal criterion for a job priority. A job with the highest priority may need more resources than available and may also result in a long waiting time and inability to make good use of the available resources.

Job execution

Once a job and a resource are selected, the next step is to attach the job to the resource for execution. Job execution may be presented as running a single command or as complicated as running of scripts that may, or may not, include file for I/O data purposes.

A.8 Replica Location Service (RLS)

RLS Overview

The Replica Location Service (RLS) [34] is one component of data management services for Grid environments. RLS is a tool that provides the ability to keep track of one or more copies, or replicas, of files in a Grid environment. This tool, which is included in Globus Toolkit 4 [1], is especially helpful for users or applications that need to find where existing files are located in the Grid.

RLS is a simple registry that keeps track of where replicas exist on physical storage systems. Users or services register files in RLS when the files are created. Later, users query **RLS servers** to find these replicas.

RLS is a distributed registry, meaning that it may consist of multiple servers at different sites. By distributing the RLS registry, we are able to increase the overall scale of the system and store more mappings than would be possible in a single, centralized catalog. We also avoid creating a single point of failure in the Grid data management system. If desired, RLS can also be deployed as a single, centralized server.

Before explaining RLS in detail, we need to define a few terms:

- A *logical file name* is a unique identifier for the contents of a file.
- A *physical file name* is the location of a copy of the file on a storage system.

The job of RLS is to maintain associations, or mappings, between logical file names and one or more physical file names of replicas. A user can provide a logical file name to an RLS server and ask for all the registered physical file names of replicas. The user can also query an RLS server to find the logical file name associated with a particular physical file location.

In addition, RLS allows users to associate attributes or descriptive information (such as size or checksum) with logical or physical file names that are registered in the catalog. Users can also query RLS based on these attributes.

Replica Location Service Components

The RLS design consists of two types of servers: the Local Replica Catalog (LRC) [58] and the Replica Location Index (RLI) [58].

The **Local Replica Catalog (LRC)** stores mappings between logical names for data items and the physical locations of replicas of those items. Clients query the LRC to discover replicas associated with a logical name. The simplest RLS deployment consists of a single LRC that acts as a registry of mappings for one or more storage systems. Typically, when an RLS is deployed on a site, an administrator populates it to reflect the contents of a local file or storage system. If new data files are produced by a workflow manager or a data publishing service, these services typically register newly created files with the RLS as part of their publication process. Our performance studies for an LRC deployed on a moderately powerful workstation with a MySQL [35] relational database back end show that the catalog can sustain rates of approximately 600 updates and 2,000 queries per second.

For a **distributed RLS deployment**, we also provide a higher-level Replica Location Index (RLI) server. Each RLI server collects information about the logical name mappings stored in one or more LRCs. An RLI also answers queries about those mappings. When a client wants to discover replicas that may exist at multiple locations, the client will pose that query to an RLI server rather than to an individual Local Replica Catalog. In response to a query, the RLI will return a list of all the LRCs it is aware of that contain mappings for the logical name contained in the query. The client then queries these LRCs to find the physical locations of replicas.

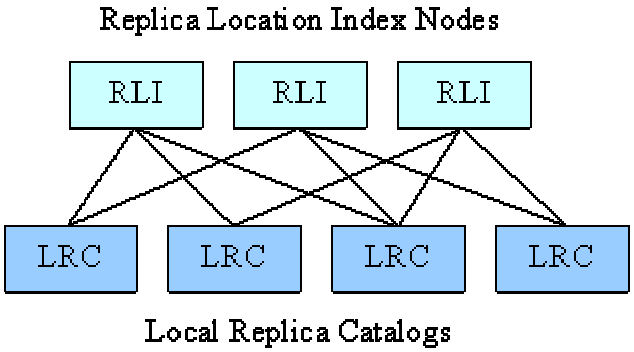


Figure A.6: deployment of a Replica Location Service

The figure A.6 above illustrates a distributed RLS deployment. Information is sent from the LRCs to the RLIs using soft-state update protocols. Each LRC periodically sends information about its logical name mappings to a set of RLIs. The RLIs collect this mapping information and respond to queries regarding the mappings. Information in RLIs times out and gets periodically refreshed by subsequent updates. An advantage of using such soft-state update protocols is that if an RLI fails and later resumes, its contents can be reconstructed using these updates.

Because each LRC may hold millions of logical file name mappings, updates from LRCs to RLIs can become large. Sending them across the network may be slow, especially in the wide area; when updates arrive at an RLI, they may consume considerable storage space there. One option for making these updates more efficient is to compress their contents. Various compression strategies are available. The one that we chose to implement for the RLS is based on Bloom filter compression. Each Local Replica Catalog periodically creates a bit map that summarizes its contents by applying a series of hash functions [13] to each logical name registered in the LRC and setting the corresponding bits in the bit map.

Configuration Options for RLS Deployment

We can determine the performance and reliability of a distributed RLS by the number of Local Replica Catalog (LRC) and Replica Location Index (RLI) servers we deploy and the way we configure updates among LRCs and RLIs. For example, we can improve reliability by configuring the system so that every LRC updates multiple RLI indexes.
In practice, current deployments of RLS systems are relatively small. For deployments at a scale of ten, the RLS is often deployed in a fully connected configuration, with a Local Replica Catalog and a Replica Location Index server deployed at each site, and all LRCs sending updates to all RLIs. Such a configuration has the advantage that every site has a complete picture of the replicas in the distributed system.

For larger deployments, however, this configuration is unlikely to scale well because it requires a large number of updates to be sent among servers and stored at each RLI. For example, in a system with hundreds of RLS sites, a fully connected deployment would require every site to send and receive hundreds of updates. In such large deployments it is likely that the system would be partitioned so that each RLI would receive updates from only a subset of the LRCs.

Ideally, the management of a distributed RLS should be automated and self-configured, so that LRCs and RLIs discover one another and updates among them are redistributed automatically to balance the load when servers join or leave the system. While new approaches for automated membership management, including peer-to-peer self-organization of RLS servers are under study, the current RLS implementation uses a simple static configuration of LRC and RLI servers. An administrator must manually change the pattern of updates among servers.

The RLS and Replica Consistency Services

RLS is a fairly simple tool that provides registration and discovery of files. **It is intended to be just one part of an overall system for managing data in Grids**. Those considering whether to use an RLS should understand what functionality the system does and does not provide.

One of the most common assumptions new users make about RLS is that, when a new replica is registered in the RLS, the system checks to make sure that the registered entry is a valid replica of an existing file. Actually, the RLS does not perform such correctness or consistency checks on new entries. The RLS allows users to register mappings between logical names and physical locations without any verification that the physical files that are registered as replicas are actually copies of one another. Likewise, if registered replicas are modified so that they are no longer valid copies of one another, the RLS will not detect these changes or take action. Instead, the RLS relies on higher-level data management or consistency services to perform these functions.

There are several reasons for this design choice. One is the simplicity and efficiency of providing a registry that is not required to perform consistency checks on the registered files, which can be time-consuming. For example, a service that guarantees consistency might calculate checksums for all replicas and verify that they match. Providing consistency guarantees therefore creates additional overhead for the system that can limit its performance.

In practice, providing a high level of consistency checking during replica registration is not required for many applications because often only a small set of highly trusted users is allowed to publish data. These privileged users do not need to verify that registered files are actually replicas because the users control the entire publishing process. They typically need to publish a large number of files quickly and cannot tolerate extra overheads associated with performing consistency operations such as checksum calculations on every file registration.

Another reason why we do not enforce replica consistency in the RLS is that users may have different definitions of what constitutes a "valid" replica. For some users, replicas are exact byte-for-byte copies of one another. For others, two files may be considered replicas if they are different versions of the same file or if the files contain the same content but in different formats, for example, compressed and uncompressed versions of the same data.

For these reasons we leave consistency-checking operations to higher-level services, which may perform these checks before or after replicas are registered in the RLS. If these services find that registered replicas are invalid, they will remove the replica mappings from the RLS.

10 Appendix B: GT4 Installation

B.1 Software Prerequisites

Required software

- Globus Toolkit installer
- J2SE 1.4.2+ SDK
- Ant 1.5.1+
- C compiler. If gcc, avoid version 3.2.
- GNU tar
- GNU sed
- zlib 1.1.4+
- GNU Make
- sudo
- JDBC compliant database. For instance, PostgreSQL 7.1+
- MySQL RPM packages (client, server, max, administrator and query browser)
- PostgreSQL
- psqlODBC (the ODBC driver for PostgreSQL)
- IODBC. The iODBC package is used to interface to the ODBC layer of the RDBMS.

Optional software

- Tomcat (required by WebMDS, optional for other services)

For further information and where to download the appropriate software visit www.globus.org.

B.2 Setting environment variables

In order for the system to know the location of the Globus Toolkit commands you just installed, you must set an environment variable and source the "*globus-user-env.sh*" script.

As "root" do the following:
Open the "*/etc/profile.local*" and add the following lines:

```
# In order for the system to know the location of the Globus Toolkit commands
export GLOBUS_LOCATION=/usr/local/gt4
export GPT_LOCATION=$GLOBUS_LOCATION
```

source /usr/local/gt4/etc/globus-user-env.sh

 # In order for the system to know the location of the ANT commands
export ANT_HOME=/ant/directory

 # In order for the system to know the location of the JAVA commands
export JAVA_HOME=/java/directory

 # Setting the CLASSPATH; define all the jar files within the CLASSPATH in order to run java applications
export
PATH=${PATH}:${ANT_HOME}/bin:${JAVA_HOME}/jre/javaws:/usr/local/condor/sbin

#Set the installation directory of the iODBC package
export GLOBUS_IODBC_PATH=$GLOBUS_LOCATION

#Indicates the path toteh file odbc.ini
export ODBCINI=$GLOBUS_LOCATION/var/odbc.ini

#Setting the rate of the GLOBUS_TCP_RANGE
export export GLOBUS_TCP_RANGE=max, min

 # In order for the system to know the location of the OGSA-DAI WSRF commands
source /ogsadai-wsrf_location/setenv.sh

B.3 Installing GT 4.0

1. Create a user named "globus" (This non-privileged user will be used to perform administrative tasks such as starting and stopping the container, deploying services, etc).

2. The directory where GT4 will be installed is the following: "*/usr/local/gt4*".
   In order to create this directory, run the following as "root":

   root>mkdir /usr/local/gt4
   root>chown globus:globus /usr/local/gt4

   After that change to user "globus" and then unzip the "gt4.allsource.tar.gz" into the target directory i.e. "*/usr/local/gt4*".
   To install the GT4, as "globus" run:

   globus>export GLOBUS_LOCATION=/usr/local/gt4
   globus>./configure --prefix=$GLOBUS_LOCATION --enable-prewsmds --
   enable-wsgram-condor

globus>make
globus>make install

Notes:

"--enable-prewsmds" is needed for a successful configuration of RLS service.
"--enable-wsgram-condor" is used in order for the gram to be able to cooperate with Condor scheduler.

B.4 Authentication & Authorization

Authentication in the Globus Toolkit is based on X.509 certificates [15]. This part describes the configuration steps required to:

- Determine whether or not to trust certificates issued by a particular Certificate Authority (CA),
- Provide appropriate default values for use by the "*grid-cert-request*" command, which is used to generate certificates,
- Specify identity mapping information (grid-mapfile).

B.4.1 Configuring Globus to Trust a Particular Certificate Authority

Important: The Globus tools will trust certificates issued by a CA if (and only if) it can find information about the CA in the trusted certificates directory.

In our case, the location of the trusted certificates directory is: "*/etc/grid-security/certificates*".

This directory contains information about which CAs are trusted (including the CA certificates themselves).

The following two files must exist in this directory for each trusted CA:

1. cert_hash.0
2. cert_hash.signing_policy

The first file is the trusted CA certificate and the second file is a configuration file defining distinguished names of certificates signed by the CA.

Globus components will "honor" a certificate only if:

- its CA certificate exists (with the appropriate name) in the "*/etc/grid-security/certificates*" directory, and
- the certificate's distinguished name matches the pattern described in the signing policy file.

The "*cert_hash*" that appears in the file names above is the hash of the CA certificate.

Some CAs provide tools to install their CA certificates and signing policy files into the trusted certificates directory, i.e. "*/etc/grid-security/certificates*".

The signing policy file should have the following format:

```
access_id_CA X509 'CA Distinguished Name'
pos_rights globus CA:sign
cond_subjects globus '"Distinguished Name Pattern"'
```

In the above, the "*CA Distinguished Name*" is the subject name of the CA certificate, and the "*Distinguished Name Pattern*" is a string used to match the distinguished names of certificates granted by the CA. Some very simple wildcard matching is done: if the Distinguished Name Pattern ends with a '*', then any distinguished name that matches the part of the CA subject name before the '*' is considered a match.

Note: the "*cond_subjects*" line may contain a space-separated list of distinguished name patterns.

Example:

```
# ca-signing-policy.conf, see ca-signing-policy.doc for more information
# This is the configuration file describing the policy for what CAs are
# allowed to sign whoses certificates.
# This file is parsed from start to finish with a given CA and subject
# name.
# subject names may include the following wildcard characters:
#    *    Matches any number of characters.
#    ?    Matches any single character.
# CA names must be specified (no wildcards). Names containing whitespaces
# must be included in single quotes, e.g. 'Certification Authority'.
# Names must not contain new line symbols.
# The value of condition attribute is represented as a set of regular
# expressions. Each regular expression must be included in double quotes.
#
# This policy file dictates the following policy:
#   -The Globus CA can sign Globus certificates
# Format:
#-----------------------------------------------------------------------
#  token type  | def.authority |                value
#--------------|---------------|---------------------------------------
# EACL entry #1|

 access_id_CA      X509        '/O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-
31814.uopnet.plymouth.ac.uk/CN=Globus Simple CA'

 pos_rights        globus        CA:sign

 cond_subjects     globus       '"/O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-
31814.uopnet.plymouth.ac.uk/*"'
```

 # end of EACL

B.4.2 Configuring Globus to Create Appropriate Certificate Requests

The "*grid-cert-request*" command, which is used to create certificates, uses the following:

1. "*globus-user-ssl.conf*" Defines the distinguished name to use for a user's certificate request. The format is described here.
2. "*globus-host-ssl.conf*" Defines the distinguished name for a host (or service) certificate request. The format is described here.
3. "*grid-security.conf*" A base configuration file that contains the name and email address for the CA.

The above files, in our case, are located in the directory: "*etc/grid-security*".

Many CAs provide tools to install configuration files. These are located in "*etc/grid-security/certificates*". These files are:

1. "*globus-user-ssl.conf.cert_hash*"
2. "*globus-host-ssl.conf.cert_hash*"
3. "*grid_security.conf.cert_hash*"

Note: The "*cert_hash*" above, is the same hash of the CA certificate.

The command:

grid-cert-request -ca cert_hash

will create a certificate request based on the specified CA's configuration files (i.e. the file found in "*etc/grid-security/certificates*") .

The command:

grid-cert-request –ca

will list the available CAs and let the user choose which one to create a request for.

You can specify a default CA for certificate requests (i.e. a CA that will be used if "*grid-cert-request*" is invoked without the "*-ca*" flag) by making the following symbolic links:

```
root>ln -s /etc/grid-security/certificates/globus-user-ssl.conf.cert_hash      \
      /etc/grid-security/globus-user-ssl.conf
root>ln -s /etc/grid-security/certificates/globus-host-ssl.conf.cert_hash      \
      /etc/grid-security/globus-host-ssl.conf
```

*root>ln -s /etc/grid-security/certificates/grid_security.conf.cert_hash *
/etc/grid-security/grid_security.conf

B.4.3 Configuration Phase

In this phase we perform the required configuration of the Globus Toolkit 4. We are in the phase of creating a simple Certificate Authority (CA), host certificates and user certificates. This configuration phase consists of 20 steps.

The first two steps set the environment variable "*$GLOBUS_LOCATION*" and source the globus user environment variables through the file "*globus-user-env.sh*", in order for the system to be aware of the Globus Toolkit environment status.

Steps 3 through 9 create a simple Certificate Authority (CA) (could be any host) who is responsible for creating and accepting host certificates and user certificates. The CA is responsible for authorization of users and hosts.

Step 10 finalizes the setup of GSI.

Steps 11 through 13 refer to a host certificate. To do this, a request is needed (step 11, DNS is required here). A request is done by executing the following: "*grid-cert-request -host 'host_name'*". This creates the following files:

- */etc/grid-security/hostkey.pem*
- */etc/grid-security/hostcert_request.pem*
- *(an empty) /etc/grid-security/hostcert.pem*

When the request is done a sign process follows (step 12). A file named "*hostsigned.pem*" is created (actually it is the host certificate). Finally the host certificate is moved to the root directory "*/etc/grid-security/hostcert.pem*" as "*hostcert.pem*". The host certificate is owned by the root and read-only by other users.

Users also must request user certificates, which you will sign using the "globus" user. For this purpose:

A request is done (step 14) by executing "*grid-cert-request*" as a user (not root). This creates:

- *~$USER/.globus/usercert.pem (empty)*
- *~$USER/.globus/userkey.pem*
- *~$USER/.globus/usercert_request.pem*

Then, the "*usercert_request.pem*" file is e-mailed to the simple CA maintainer.

In order to sign the user certificate (step15), as the simple CA owner “globus”, the following is executed: "*grid-ca-sign -in usercert_request.pem -out signed.pem*", (the private key of the CA certificate will be entered). At last, the signed copy "*signed.pem*" is sent back to the user who requested the certificate.

As user now (step 16), the signed user certificate is copied into "*~/.globus/*" as "*usercert.pem*", thus replacing the empty file. The certificate should be owned by the user, and read-only for other users.

Step 17 refers to the testing of the simple CA certificate. In order to test that the simple CA certificate is installed in "*/etc/grid-security/certificates*" and that your certificate is in place with the correct permissions, the following is executed: "*grid-proxy-init -debug -verify*".

Step 18 refers to the changing of the ownership. Actually this step allows resource management tools to run as “root”.

Step 19 refers to an addition of an authorization for users.

Step 20 refers to the next steps to be done including the test of the gt3 and other optional configurations.

B.4.4 Specifying Identity Mapping Information

In general, Globus services map distinguished names (found in certificates) to local identities (e.g., unix logins). These mappings are maintained in the *grid-mapfile* file.

A gridmap line of the form:

"Distinguished Name" local_name

maps the distinguished name “*Distinguished Name*” to the local name “*local_name*”.

A gridmap line of the form:

"Distinguished Name" local_name1,local_name2

maps “*Distinguished Name*” to both “*local_name1*” and “*local_name2*”; any number of local user names may occur in the comma-separated local name list.

Example:

```
"/O=Grid/OU=GlobusTest/OU=simpleCA-grid1.tsi.gr/OU=tsi.gr/CN=nick" manolakis,nick
"/O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2                       \
31814.uopnet.plymouth.ac.uk/OU=tsi.gr/CN=ioannis" nick,manolakis
"/O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-                      \
31814.uopnet.plymouth.ac.uk/OU=tsi.gr/CN=manolakis" manolakis,nick
```

"/O=Grid/OU=GlobusTest/OU=simpleCA-smb2193n2-31814.uopnet.plymouth.ac.uk/OU=tsi.gr/CN=nikos" nick,manolakis \

To add an entry to the grid-map file, run:

root>grid-mapfile-add-entry -dn "Distinguished Name" -ln local_name

To delete an entry from the grid-map file, run:

root>grid-mapfile-delete-entry -dn "Distinguished Name" -ln local_name

To chech consistency of the grid-map file, run,

root>grid-mapfile-check-consistency

These commands recognize several useful options, including a "*-help*" option, which lists detailed usage information.

The location of the gridmap file, in our case, is: "*/etc/grid-security/grid-mapfile*" (This is by default because user "root" has a "uid" 0).

B.5 Verify Basic Security

When a user acquires a user certificate he has to verify if security setup is complete. To do so, as a user account, run the following command:

user>grid-proxy-init –verify –debug

You will be asked to enter a pass phrase (the same pass phrase you entered when you created user certificate).

There are a few things you can notice from this command. Your "usercert" and "key" are located in "*$HOME/.globus/*". The proxy certificate is created in the "*/tmp/*" directory. The file should look like: "*/tmp/x509up_uxxxx*" The "up" stands for "user proxy" and the "_uxxxx" should be your UNIX "userId" (UID). For example, a proxy certificate "*/tmp/x509up_u1002*" tells you that the user who created the proxy certificate has a userId 1002. It also prints out your distinguished name (DN).

When a user runs the command "*grid-proxy-init*" a proxy certificate will be created as described above. The proxy certificate, by default, is valid for 12 hours. A user has the ability to extend the validation time of the proxy certificate. This is done by running the following:

user>grid-proxy-init –hours 240

The above command creates a proxy certificate which is valid for 10 days (240 hours).

B.6 Configuring the Globus Container

The container will be always running as "globus" user.

The hostkey security file "*/etc/grid-security/hostkey.pem*" is only "root" readable. The container is started by "globus" user. During this process the host credentials should be checked. Since the host credentials are owned by "root", we have to create a copy that will be owned by user "globus". Otherwise, the container cannot be started.

As "root", run:

```
root>cd /etc/grid-security
root>cp hostkey.pem containerkey.pem
root>cp hostcert.pem containercert.pem
root>chown globus.globus containerkey.pem containercert.pem
```

Starting the container

To start the container, run the following as "globus':

```
globus>globus-start-container
```

The container by this way will be started at the port 8443 i.e. with security (https://).

To start the container without security, tun the followaing as globus:

```
globus>globus-start-container -nosec
```

By this, the container will be running at the port 8080 (http).

Stoping the container

In order to stop the container do the following:

As "globus" user, create a script "*stopcontainer*" in the directory "*$GLOBUS_LOCATION/bin*" and add the following lines:

```
#!/bin/sh

grid-proxy-init –cert /etc/grid-security/containercert.pem -key   \
/etc/grid-security/containerkey.pem –out                          \
/usr/local/gt4/bin/containerproxy.pem
 export X509_USER_PROXY=/usr/local/gt4/bin/containerproxy.pem
globus-stop-container
unset X509_USER_PROXY
rm /usr/local/gt4/bin/containerproxy.pem
```

Note: The symbol "\" means that the following lines are in the same line with the one earlier.

In order to run the script, it should be executable (+x) .
This is done by running the following as "globus":

```
globus>cd $GLOBUS_LOCATION/bin
globus>chmod +x stopcontainer
```

B.7 Configuring GridFTP GT 4.0

This configuration will be done as "root". The following steps should be done:

1. Add the following entries to "*/etc/services*":

```
#GridFTP Service
gsiftp 2811/tcp
```

2. This step concerns the configuration of the "*Inted/Xinetd*". (Inetd/Xinetd allows running one daemon to invoke several others, reducing load on the systems).
The distribution of the OS SuSe on grid0, grid1 and grid3 has the daemon Xinetd and on grid2 the daemon Inetd.

The appropriate instructions for both of the daemons:

For **Inetd**, open "*/etc/inetd.conf*" and add the following entry.

```
gsiftp  stream  tcp     nowait  root /usr/bin/env env
LD_LIBRARY_PATH=/usr/local/gt4/lib /usr/local/gt4/sbin/in.ftpd -l -a -G
=/usr/local/gt4
```

For **Xinetd**, create a file "*grid-ftp*" in the directory "*/etc/xinted.d/*" with the following contents:

```
service gsiftp
{
        instances           = 100
        socket_type       = stream
        wait                 = no
        user                 = root
        env                 += GLOBUS_LOCATION=/usr/local/gt4
        env                 += LD_LIBRARY_PATH=/usr/local/gt4/lib
        server               = /usr/local/gt4/sbin/globus-gridftp-server
        server_args        = -i
        log_on_success += DURATION
        nice                 = 10
        disable              = no
}
```

The advantage of this setup is that when you apply a security update to your installation, the GridFTP server will pick it up dynamically without your having to rebuild it.

3. This step concerns notification of Inetd/Xinetd to the changes that has been done. To do so the following should be run:

Inted

root>killall –HUP inted

Xinted

root>/etc/init.d/xinted restart

Testing

- To check if the GridFTP is running you can simply run the following:

  telnet localhost 2811

  The message you get will tells you that the server is ready.

- Another test may be done using the command "*globus-url-copy*".
  As user run the following: (**read as a single line**)

  *user>globus-url-copy file:///path_to_file *
  gsiftp://destination_host/destination_directory

Check if the file was copied. If so, then your GridFTP server is running well.

B.8 Configuring Reliable File Transfer (RFT) GT 4.0

PostgreSQL (version 7.1 or greater) needs to be installed and configured for Reliable File Transfer (RFT) to work.
RPM packages can be used for this purpose.
PostgreSQL should be installed as "root". After installation a "postgres" user should be created.

Configuring PostgreSQL

1. Run the following as "root":

root>/etc/init.d/postmaster start

The above will create the directory "*data*" and its contents in the directory "*/var/lib/pgsql/*".

2. Security must be set on the database you are about to create. In order to do so run the following:

root>su postgres
postgres>vi /var/lib/pgsql/data/pg_hba.conf

and add the following line to the file:

host rftDatabase "globus" "host-ip" 255.255.255.0 trust

3. A postgresql user should now be created in order to connect to the database. This is usually the account under which the container is running i.e. "*globus*".

postgres>createuser globus

4. To create the database that is used for RFT, as "*globus*" run:

globus>createdb rftDatabase

5. To populate the RFT database with the appropriate schemas, run:

*globus>psql –d rftDatabase –f *
$GLOBUS_LOCATION/share/globus_wsrf_rft/rft schema.sql

6. In order for the RFT to find the database you have just created to store RFT's state open the file "*jndi-config.xml*" in the directory "*$GLOBUS_LOCATION/share/globus_wsrf_rft*" and change "*username*" to "*globus*" (without quotes)

7. Finally, as root, open the "*postgresql*" in "*/etc/init.d*" and in the line where the daemon postgre starts, **add the option "*-i*" (without quotes)**, so that the postgresql daemon starts automatically.

Note: The PostgreSQL server runs at the port 5432.

"PGAdmin3" has been installed at "*grid1.tsi.gr*" under Windows XP OS to graphically administer PosrgeSQL database systems in machines configured to allow it, e.g. "grid0.tsi.gr".

Testing

In order to test the RFT server, files with "*.xfr*" extensions are needed. These files are used to transfer files locally and non-locally.

As user, run the following:

user>rft –h host_where_to_run_rft –f name_of_file.xfr

The ".xfr" files should be look like:

```
#true=binary false=ascii
true
#Block size in bytes
16000
TCP buffer size in bytes
16000
Notpt (No thirdPartyTransfer)
false
Number of parallel streams
1
Data Channel Authentication (DCAU)
true
Concurrency of the request
1
#Grid subject name of the source gridftp server
/O=Grid/OU=GlobusTest…/CN=host/CN=grid2.tsi.gr
#Grid subject name of the destination gridftp server
/O=Grid/OU=GlobusTest…/CN=host/CN=grid3.tsi.gr
#Transfer all or none of the transfers
false
#Maximum number of retries
10
#Source/Destination URL pairs
gsiftp://source_host:2811/path_of_file_to_transfer
gsiftp://destination_host:2811/path_of_file_to_transfer_to
```

B.9 Setting up a Pre-WS GRAM Server

Setting up a Pre-WS GRAM server is actually setting up a gatekeeper [20].
In order to use one gatekeeper, "globus-gatekeepeer" must be run as "root".

To set up a full gatekeeper, the following steps must be done:

1. Add the following entries to "*/etc/services*":

```
#Globus Gatekeeper Service
gsigatekeeper 2119/tcp
```

2) This step concerns the configuration of the Inted/Xinetd. (Inetd/Xinetd allows running one daemon to invoke several others, reducing load on the systems).
The distribution of the OS SuSe on grid0, grid1 and grid3 has the daemon Xinetd and on grid2 the daemon Inetd.

The appropriate instructions for both of the daemons:

Inetd
As root, find "*/etc/inetd.conf*" and add the following entry, all on one line:

```
gsigatekeeper stream tcp nowait root
/usr/bin/env env LD_LIBRARY_PATH=/home/globus/gt3/lib
/home/globus/gt3/sbin/globus-gatekeeper
-conf /home/globus/gt3/etc/globus-gatekeeper.conf
```

Xinetd
Go to the "*/etc/xinetd.d/*" directory and add a file called "*globus-gatekeeper*" with the following contents.

```
service gsigatekeeper
{
socket_type = stream
protocol = tcp
wait = no
user = root
env = LD_LIBRARY_PATH=/home/globus/gt3/lib
server = /home/globus/gt3/sbin/globus-gatekeeper
server_args = -conf /home/globus/gt3/etc/globus-gatekeeper.conf
disable = no
}
```

The advantage of this setup is that when you apply a security update to your installation, the gatekeeper will pick it up dynamically without your having to rebuild it.

3) This step concerns notification of Inetd/Xinetd to the changes that has been done. To do so the following should be run:
Inted

```
root>killall –HUP inted
```

Xinted

```
root>/etc/init.d/xinted restart
```

At this point, your gatekeeper will start up when a connection comes in to port 2119, and will keep a log of its activity in "$GLOBUS_LOCATION/var/globus-gatekeeper.log".

B.10 Configuring WS GRAM

In order to use WS GRAM , the container must be started with Transport Level security. In other words the "*-nosec*" option should **not** be used with the command "*globus-start-container*".

WS GRAM depends on a local mechanism for starting and managing jobs. In other words it depends on scheduler adapters to translate the WS GRAM [51] job description document into commands understood by the local job scheduler, as well as monitor the jobs.
Included in the WS GRAM software is Fork scheduler, which requires no special software installed to execute jobs on the local host. However to enable WS GRAM to execute and manage jobs to a batch scheduler, the scheduler software must be installed and configured prior to configuring WS GRAM.
Condor scheduler has been used in our case.

Configuring sudo

As "root", add the following lines to the "*/etc/sudoers*":

```
# Globus GRAM entreis
globus ALL=(users that start the jobs, example manolakis, nikos )          \
NOPASSWD: $GLOBUS_LOCATION/libexec/globus-gridmap-and-execute              \
-g /etc/grid-security/grid-mapfile                                         \
$GLOBUS_LOCATION /libexec/globus-job-manager-script.pl *
globus ALL=(users that start the jobs, example manolakis, nikos )          \
NOPASSWD: $GLOBUS_LOCATION/libexec/globus-gridmap-and-execute              \
-g /etc/grid-security/grid-mapfile                                         \
$GLOBUS_LOCATION /libexec/globus-gram-manager-proxy.tool *
```

Note: the symbol "\" means that all in the same line.

Configuring Condor Scheduler Adapter

```
globus>cd $GLOBUS_LOCATION/gt4.0.0-all-source-installer
globus>make gt4-gram-condor
globus>make install
```

B.11 Configuring Replica Location Service (RLS) GT 4.0

B.11.1 Requirements

A Relational Database Server (RDBMS) that supports ODBC is required for this configuration. PostgreSQL or MySQL can be used.
From our experience, it was much easier to use PostgreSQL in the configuration than MySQL. So, we provide information on how to configure RLS with PostgreSQL.

Because we are going to use PostgreSQL, we will need "psqlODBC" (the ODBC driver for PostgreSQL). Otherwise, if you are using MySQL instead of PostgreSQL, you will need the "MyODBC" (connector/ODBC) packages.

The location of iODBC and the odbc.ini file must be specified before installing the RLS server.

Before proceeding in the configuration of Replica Location Service (RLS), some environment variables should be added to "*/etc/profile.local*".
The environment variables needed for this configuration are:

```
export GLOBUS_IODBC_PATH=$GLOBUS_LOCATION
export ODBCINI=$GLOBUS_LOCATION/var/odbc.ini
```

B.11.2 Installing iODBC

The iODBC [60] package is used to interface to the ODBC layer of the RDBMS.

Download first the "iODBC" package (tar file). After that, untar the file in a directory ("*$GLOBUS_LOCATION/iodbc*").

The following commands have to be used during RLS development to install iODBC:

```
globus>export IODBCSRC=$GLOBUS_LOCATION/iodbc
globus>export IODBCINIDIR=$GLOBUS_LOCATION/var
globus>cd $IODBCSRC
globus>./configure --prefix=$GLOBUS_IODBC_PATH --with-pthreads \
        --disable-gui --with-iodbc-inidir=$ODBCINIDIR
globus>make
globus>make install
```

where:

- $IODBCSRC is the directory where you untarred the iODBC sources.
- $ODBCINIDIR is the directory where you plan to install the odbc.ini file (which you will create in the next step).

B.11. 3 Create the "odbc.ini" file

Data source definitions and option settings are stored in the "*odbc.ini*" file. The main role of the "*odbc.ini*" file is to map a data source name (DSN) to a particular ODBC driver and options.

As "*globus*", go to "*$GLOBUS_LOCATION/var*" and create the "*odbc.ini*".
The contents of the file should be as follows:

[ODBC Data Sources]
lrc1000=lrc database
rli1000=rli database
[lrc1000]
Description=LRC database
DSN=lrc1000
Servertype=postgres
Servername=localhost
Database=lrc1000
ReadOnly=no

[rli1000]
Description=RLI database
DSN=rli1000
Servertype=postgres
Servername=localhost
Database=rli1000
ReadOnly=no
[Default]
Driver=/path/to/psqlodbc.so
Port=5432

Note: the path to the "*psqlodbc.so*", will be obtained after installing psqlODBC and it should be something like "*$GLOBUS_LOCATIOB/lib/psqlodbc.so*"

A symbolic link should be created in the root's home directory to the location of "*odbc.ini*" which is "*$GLOBUS_LOCATION/var/odbc.ini*". The symbolic link is needed because psqlodbc will not find a Data Source Name (DSN) in the system odbc.ini file "*$GLOBUS_LOCATION/var/odbc.ini*". It will find DSNs in the user's odbc.ini file if it exists at "*$HOME/.odbc.ini*".
To create the symbolic link, run the following:

root>cd root
root>ln –s .odbc.ini $GLOBUS_LOCATION/var/odbc.ini

B.11.4 Installing psqlODBC

If your relational database of choice is PostgreSQL, you need to install and configure both PostgreSQL and psqlODBC (the ODBC driver for PostgreSQL).

PostgreSQL must have been installed and configured during RFT Configuration.

Download first the "psqlODBC" package (tar file). As "*globus*", untar the tar file in "*$GLOBUS_LOCATION/psqlODBC*".

To install psqlODBC run the following commands:

```
globus>cd $GLOBUS_LOCATION/psqlODBC
globus>./configure –prefix=$GLOBUS_LOCATION
globus>make
globus>make install
```

B.11.5 Installing the RLS server

To configure the RLS server run the following as "*globus*"

```
globus>cd /home/globus/gt4.0.0-all-source-installer
globus>make globus-rls-server
globus>make install
```

B.11.6 Configuring the RLS Database

Configuration settings for the RLS are specified in the "*$GLOBUS_LOCATION/etc/globus-rls-server.conf*" file.
This file contains some information where the RLS server will use, to connect to the DBMS. (database user, odbcini path, etc.)
The "*globus-rls-server.conf*" should like:

```
#Database connection options
db_user                    globus
db_pwd                     *****
odbc.ini                   /usr/local/gt4/var/odbc.ini
update_immediate    true

#LRC options
lrc_server                 true
lrc_dbname                 lrc1000

#RLI options
rli_server                 false
rli_dbname                 rli1000
#Comments#
#Comments#
acl                        .*: all

#Here are all the configuration options with default values:
#acl                       all
#authentication            true
#db_pwd                    xxxx
#db_user dbuser            globus
#idletimeout               900
#loglevel                  3
#lrc_bloomfilter_numhash   3
#lrc_bloomfilter_ratio     10
#lrc_dbname                lrc1000
#lrc_server                true
#maxbackoff                300
#maxconnections      100
#maxfreethreads     5
#maxthreads         30
myurl                      rls://147.27.1.24x:39281
#odbcini            /usr/local/gt4/var/odbc.ini
```

```
#pidfile                    /usr/local/gt4/var/<progamname>.pid
port                        39281
#rli_bloomfilter            false
#rli_bloomfilter_dir        none
#rli_dbname                 rli1000
#rli_expire_int             28800
#rli_expire_stale           86400
#rli_server                 false
#rlscertfile                /etc/grid-security/hostcert.pem
#rlskeyfile                 /etc/grid-security/hostkey.pem
#startthreads               3
#timeout            30
#update_bf_int              900
#update_buftime     30
#update_factor              10
#update_ll_int              86400
#update_retry               300
```

Notes:

- **The "*rlscerfile*" and "*rlskeyfile*" should point to "*hostcert.pem*", "*hostkey.pem*" respectivley. These files are found in "*/etc/grid-security*" and have permissions "root". If the RLS server starts as "root" then the "*rlscertfile*" and "*rlskeyfile*" should point to "*/etc/grid-security*". If the RLS server starts as "non-root" user it should point to "*hostcert.pem*" and "*hostkey.pem*" with the "non-root" user permissions. This is done by copying "*hostcert.pem*" and "*hostkey.pem*" from "*/etc/grid-security*" to a "non-root" user directory and then changing the permissions to the "non-root" user.**

- **During the test of RLS, we figured out that LRC and RLI doesn't work together, i.e. one server should be "true" and the other one "false."**

B.11.7 Creating the databases

To create the databases "*lrc1000*" and "*rli1000*", run the following as "*globus*":

```
globus>creatdb –O globus –U globus –W lrc1000
globus>creatdb –O globus –U globus –W rli1000
globus>psql –W –U globus –d lrc1000 –f                \
        $GLOBUS_LOCATION/setup/globus/globus-rls-lrc-postgres.sql
globus>psql –W –U globus –d rli1000 –f                \
        $GLOBUS_LOCATION/setup/globus/globus-rls-rli-postgres.sql
```

Note: The command "creatdb" creates a new database. It's a Linux command.

Finally, we have to give permissions for "globus" user to use the databases you have just created. To do so, as "root", run the following:

```
root>su postgres
postgres>pico /var/lib/pgsql/data/pg_hba.conf
```

and add the following lines:

host lrc1000 "globus" "host-ip" 255.255.255.0 trust
host lrc1000 "globus" "host-ip" 255.255.255.0 trust

B.11.8 Starting the RLS server

Start the RLS server by running:

root>SXXrls start

B.11.9 Stoping the RLS server

Stop the RLS server by running:

root>SXXrls stop

Note: the RLS server runs at port: 2135.

B.11.10 Configuring the RLS server for the WS MDS Index Service

To enable Index Service reporting, add the contents of the file "*$GLOBUS_LOCATION/setup/globus/rls-ldif.conf*" to the WS MDS Index Service configuration file: "*$GLOBUS_LOCATION/etc/grid-info-resource-ldif.conf*".

B.11.11 Testing

You can use the programs "globus-rls-admin" and "globus-rls-cli" to test functionality.
Start the server in debug mode with the command:

root>$GLOBUS_LOCATION/bin/globus-rls-server -d [-N]

The -N option is helpful: if you do not have a host certificate for the server host, or a user certificate for yourself, it disables authentication.

Ping the server using globus-rls-admin:

user>$GLOBUS_LOCATION/bin/globus-rls-admin -p rls://serverhost

If you disabled authentication (by starting the server with the -N option), then use this command:

user>$GLOBUS_LOCATION/bin/globus-rls-admin -p rlsn://serverhost

B.12 Troubleshooting

Here are some points to check if the GT4 is installed successfully and it's ready for use.

- **"globus-condor.conf" file**

  Go to "*$GLOBUS_LOCATION/etc/*" and check if the file "*globus-condor.conf*" exists.
  If it does not exist, then as "*globus*", run the following:

  globus>cd $GLOBUS_LOCATION/setup/globus/
  globus>./setup-seg-condor.pl

  The above command-line creates the necessary file which is needed for the Condor job manager to cooperate with Globus.

- **"gram-service-Condor" directory**

  Go to "GLOBUS_LOCATION/etc/" and check if the directory "*gram-service-condor*" does exist.
  If it does not exist, then as "*globus*", run the following:

  globus>cd $GLOBUS_LOCATION/setup/globus/
  globus>./setup-gram-service-condor

  The above command-line creates the necessary file needed for the Condor job manager to cooperate with Globus.

- **Gatekeeper**

  The gatekeeper will map your certificate against a local user and start a job manager as that user.
  To run pre-web services, the Gatekeeper should be configured first.
  To submit jobs from "Condor" to "globus" you can use the command "*condor_submit*" with "*universe = globus*".
  The Gatekeeper will be running at "2119/tcp".
  Submitting jobs will be done by "users" (neither "root", nor "globus").
  To start a personal gatekeeper, as "user" run:

  user>globus-personal-gatekeeper –start –jmtype=condor

  Once you run the above command, a directory "*personal-gatekeeper.machine-host.xxxx*"

will be created in "*/home/user/.globus*". This directory contains the appropriate files such as: .log, .conf and jobmanager.conf. These files are responsible for running the job using the "*condor_submit*" command.

- **"rls-ldif.conf" / "grid-info-resource-ldif.conf"**

  Go to "*$GLOBUS_LOCATION/setup/globus/*" and "*$GLOBUS_LOCATION/etc/*" and check if "*rls-ldif.conf*", "*grid-info-resource-ldif.conf*" are found respectively.
  If not, run the following as "*globus*":

  globus>cd /home/globus/gt4.0.0-all-source-installer
  globus>make gt4-prewebmds
  globus>make install

- **"gluerp.xml"**

  The file above is located in "*$GLOBUS_LOCATION/etc/globus_wsrf_mds_usefulrp*". This file is responsible to configure which "information provider" (Ganglia, Hawkeye, Condor, etc.) to be used by MDS.

- A critical problem in setting up a grid environment is that every machine has to be synchronized with the others. The problem was fixed by setting up the NTP client (Network Time Protocol) in every machine to a common NTP server.

11 Appendix C: OGSA-DAI WSRF Configuration

The OGSA-DAI [52] project is concerned with constructing middleware, e.g. Globus Toolkit [1] [2] to assist with access and integration of data from separate data sources via the grid.

OGSA-DAI WSRF is used in this thesis in order to be compatible with WSRF-based GT4 [37].

This OGSA-DAI distribution provides a WSRF based service interface, using the GT4 WSRF implementation.

C.1 Download

Get first the OGSA-DAI WSRF 1.0 release distribution (source or binary) and untar the zip in the directory "*/usr/local/ogsadai_wsrf*" as "globus" user.

Note: if you need to download the release form "www.ogsadai.uk.org", a registration is needed. Use this nickname and password:

nickname=ogsadai
password=db9452

Prerequisites and Dependencies

Ensure that GT4 Java WS Core is properly installed.

To use OGSA-DAI WSRF you will need the following software:

- Java 1.4.0 (already installed before GT4 configuration)
- Jakarta ANT 1.5 (already installed before GT4 configuration)
- A database management system (DBMS) (MySQL is used)

Ensure you have downloaded the following external JARs (if required):

- jakarta-oro-2.0.7.jar
- xmldb.jar
- lucene-1.4-final.jar
- xindice.jar
- MySQL JDBC drivers

Ensure "GLOBUS_LOCATION" variable is set correctly.

Ensure "CATALINA_HOME" variable is set correctly if using GT4 [1] on Tomcat [39].

Note: All the "*.jar" files must be inserted in "*/usr/local/ogsadai_wsrf/lib*", where "*/usr/local/ogsadai_wsrf*" is the installation direcroty.

C.2 Pre Installation

If you have downloaded the binary distribution then skip to part 7.3.

If you have the OGSA-DAI source distribution you will need to generate a binary release. You can do this as follows:

1) Ensure you have downloaded the JARs required (**see above**).

2) To build the OGSA-DAI binary distribution, run the following:

globus>ant createBinaryDistribution

Note: If you build your OGSA_DAI WSRF from a source distribution you will already have these JARs and they will have been inserted into the binary lib directory as part of the build ("*/usr/local/ogsadai_wsrf/lib*").

C.3 Installing OGSA-DAI WSRF

Ensure you have downloaded the prerequisite JARs in the OGSA-DAI distribution lib directory "*/usr/local/ogsadai_wsrf/lib*".

The OGSA-DAI WSRF can be installed either onto the GT4 container or onto GT4 on Tomcat.

The installation could be done by:

- Either via the **Command-line**
- Or using a **GUI**

Installing OGSA_DAI WSRF via the Command-line onto GT4/Tomcat

From OGSA-DAI distribution directory "*/usr/local/ogsadai_wsrf*", run the following as "globus":

globus>ant deploy[GTContainer|Tomcat]

OR

globus>ant deploy[GT4|Tomcat]

OGSA-DAI WSRF will then be installed.

Installing OGSA_DAI WSRF using a GUI onto GT4/Tomcat

Run the following from within the OGSA-DAI binary distribution directory "*/usr/local/ogsadai_wsrf*":

globus> ant guiDeploy[GT4|Tomcat]

A GUI will appear where you can select between the options.

Uninstalling OGSA-DAI WSRF from GT4

The OGSA-DAI WSRF can be uninstalled:

- Either via the **Command-line**
- Or using a **GUI**

Uninstalling OGSA_DAI WSRF via the Command-line from GT4/Tomcat

From OGSA-DAI distribution directory "*/usr/local/ogsadai_wsrf*", run the following as "globus":

globus>ant undeploy[GT4|Tomcat]

OGSA-DAI WSRF will then be uninstalled.

Uninstalling OGSA_DAI WSRF using a GUI from GT4/Tomcat

Run the following from within the OGSA-DAI binary distribution directory "*/usr/local/ogsadai_wsrf*":

globus> ant guiUndeploy[GT4|Tomcat]

A GUI will appear where you can select among the options to uninstall OGSA-DAI WSRF.

C.4 Deploying Data Services & Exposing Data Resources using OGSA-DAI WSRF

Below there follows a three-step process:

1. Deployment of an OGSA-DAI data service. This data service initially exposes 0 data service resources.
2. Deployment of a data service resource. The data service resource contains information about a data resource and the activities clients can perform on that data resource via a data service.

3. Addition of the data service resource to the data service. This instructs the data service to expose the data service resource and so allows clients to interact with the data service resource - thereby interacting with the data resource.

Under GT4, OGSA-DAI resides in the following subfolders within the folder "*/usr/local/ogsadai_wsrf*" :

- share/schema/ogsadai/ - OGSA-DAI XML Schema and WSDL.
- lib/ - OGSA-DAI JAR files.
- etc/ogsadai_wsrf/ - OGSA-DAI configuration files - see below.

The OGSA-DAI WSRF configuration files residing in the "*$GLOBUS_LOCATION/etc/ogsadai_wsrf*" directory, are as follows:

- server-config.wsdd - deployment descriptors for the current OGSA-DAI services.
- jndi-config.xml - JNDI deployment descriptors for the current OGSA-DAI services
- Data service resource files for each current data service. There will be one of these XML files for each current OGSA-DAI service. These files maintain a list of the current data service resources for each data service. The file name is derived from the service path e.g. a service with relative service path ogsadai/DataService has a file _ogsadai_DataService.dsr.xml.
- Data service resource directories. There is one such directory for each deployed data service resource - the directory shares the name of the data service resource e.g. MyRelationalResource Each directory contains:
- dataResourceConfig.xml - data resource configuration file for the data service resource.
- activityConfig.xml - activity configuration file for the data service resource.
- DatabaseRoles.xml - roleMaps file for the data service resource.

C.4.1 Deployment of a Data Service

This part describes how to deploy an OGSA-DAI data service which initially will expose 0 data service resources.

You can:

- Deploy data services using the command-line.
- Deploy data services using a GUI.
- Check that a data service deployed correctly.

Deploying Data Services using the Command-line

To deploy a data service onto GT4:

Run the following command from within the OGSA-DAI WSRF binary distribution directory "*/usr/local/ogsadai_wsrf*":

globus> ant deployDataService[GT4/Tomcat] -Ddai.service.path=service/path

where "*dai.service.path*" specifies the local URL of the service. For example -Ddai.service.path=ogsadai/DataService.

Deploying Data Services using a GUI

To deploy a data service onto GT4:

Run the following command from within the OGSA-DAI WSRF binary distribution directory "*/usr/local/ogsadai_wsrf*":

globus> ant guiDeployDataService[GT4|Tomcat]

Checking the OGSA-DAI Data Service Deployment

To check that a service has been deployed onto GT4:

- Start the GT4 container:

  globus>globus-start-container -nosec

Note: The OGSA-DAI Data Service works only with no security (i.e. http, port 8080).

Eventually a list of service URLs will be displayed. You should see one corresponding to your service. For example:

[22]: http://host-ip:8080/wsrf/services/ogsadai/DataService

An alternative way is to test the service using the OGSA-DAI WSRF listResourcesClient client (described below).

ListResources Client

OGSA-DAI comes with a client that allows you to list the data service resources exposed by an OGSA-DAI WSRF data service.

To list the data service resources exposed by a data service:

- Ensure that your SOAP container is running.
- Run the following command from within the OGSA-DAI WSRF binary distribution directory "*/usr/local/ogsadai_wsrf*" :

```
globus> ant listResourcesClient -Ddai.url=                \
        http://localhost:8080/wsrf/services/ogsadai/DataService
```

GetProperty Client

GT4 comes with a client that allows you to get the properties of data service resources exposed by an OGSA-DAI WSRF data service.

To get the properties of a data service resource exposed by a data service:

- Ensure that your SOAP container is running.
- Invoke the Globus GetProperty Client:

```
globus>wsrf-get-property -s DATA-SERVICE-URI
        -k {http://ogsadai.org.uk}DataServiceResourceKey DATA-
        RESOURCE-NAME PROPERTY-NAME
```

For example:

```
globus> wsrf-get-property –s                                    \
        http://localhost:8080/wsrf/services/ogsadai/DataService  \
        -k {http://ogsadai.org.uk}DataServiceResourceKey grid1db  \
        {http://ogsadai.org.uk/namespaces/2004/05/gdsf}databaseschema
```

From the example above, "*grid1db*" is the DATA-RESOURCE-NAME and "*databaseschema*" is the PROPERTY-NAME.

C.4.2 Deployment of a Data Service Resource

This part describes how to deploy a data service resource.

The characteristics of a data service resource are specified in a data services resource file. A data service resource file is used specify the properties of your data service resource. An example of this file is found within the OGSA-DAI WSRF distribution directory, "*/usr/local/ogsadai_wsrf*". This file is called "*data.service.resource.properties*".

The properties are as follows:

- dai.resource.id= - name for the data service resource.
- dai.data.resource.type=[Relational | XML | Files] - the type of data resource to which the data service resource provides access.
- dai.product.name= - data resource product name (optional).
- dai.product.vendor= - data resource product vendor (optional).
- dai.product.version= - data resource product version (optional).
- dai.data.resource.uri= - data resource URI. This must be compatible with the driver class specified next.

- dai.driver.class= - data resource driver class name. This is optional only if dai.data.resource.type=Files.
- dai.credential= - Grid certificate credentials of a user permitted to access the data resource. If omitted then any user will be allowed access.
- dai.user.name= - data resource user name. Optional only if there is no user name required for a database.
- dai.password= - corresponding data resource password. Optional if there is no user name required, or if the password is null.
- dai.driver.jars= corresponding database connector jar file.

An example of this file without comments:

```
dai.resource.id=grid1db
dai.data.resource.type=Relational
dai.product.name= MySQL
dai.product.vendor= MySQL
dai.product.version= 4.0
dai.data.resource.uri= jdbc:mysql://localhost:3306/ogsadai
dai.driver.class= org.gjt.mm.mysql.Driver
dai.credential=
dai.user.name= root
dai.password=
dai.driver.jars=/usr/local/ogsadai_wsrf/lib/mysql-connector-java-3.1.10-bin.jar
```

Deploying Data Service Resources using the Command-line

To deploy a data service resource onto GT4:

- Take a copy of "*data.service.resource.properties*" within the OGSA-DAI WSRF distribution directory "*/usr/local/ogsadai_wsrf*".
- Load the file into an editor and provide values specifying your data service resource - the comments in the file should help you.
- When done, save the file.
- Put all the JARs implementing the data resource driver within the drivers directory within the OGSA-DAI WSRF binary distribution directory "*/usr/local/ogsadai_wsrf*".
- Run the following command from within the OGSA-DAI WSRF binary distribution directory:

```
globus> ant deployResource[GT4|Tomcat]
        -Ddai.service.resource.file=DAI-SERVICE-RESOURCE-FILE
```

where:
"*dai.service.resource.file*" specifies the location of a data service resource properties file.

For example:

```
globus> ant deployResourceGT4 -Ddai.service.resource.file=my.config
...
[echo] Reading properties file my.config
[echo] Data service resource ID:grid1db
[echo] Data resource type: Relational
[echo] Data resource product: MySQL
[echo] Data resource vendor: MySQL
[echo] Data resource version: 1.0
[echo] Data resource URI: jdbc:mysql://myhost.example.com:3306/ogsadai/DATABASE
[echo] Data resource credential:
[echo] Data resource user name: ogsadai
[echo] Data resource password: ogsadai
[echo] Data driver class: org.gjt.mm.mysql.Driver
[echo] Deploying data service resource MySQLResource...
...
[echo] Data service resource deployed!
```

Deploying Data Service Resources using a GUI

To deploy a new data service resource onto GT4:

Run the following command from within the OGSA-DAI WSRF binary distribution directory "*/usr/local/ogsadai_wsrf*":

```
globus> ant guiCreateResource[GT4|Tomcat]
```

C.4.3 Addition of a Data Service Resource to the Data Service

This part describes how to add a data service resource to a data service, thereby allowing clients to access and interact with the data service resource. It assumes that both the data service and data service resource have been deployed.
You can:

- Add data service resources using the **command-line**.
- Add data service resources using a **GUI**.
- Dynamically add a data service resource using the **command-line**.

Adding Data Service Resources using the Command-line

To add data service resources to data services onto GT4:

Run the following command from within the OGSA-DAI WSRF binary distribution directory "*/usr/local/ogsadai_wsrf*":

```
globus> ant addResourceGT4 -Ddai.service.path=service/path
        -Ddai.resource.id=ResourceID
```

where:

"*-Dai.service.path*" specifies the local URL of the service. For example – Ddai.service.path=ogsadai/DataService specifies a data service whose URL will be http://HOST:PORT/wsrf/services/ogsadai/DataService.
"*-Ddai.resource.id*" is the ID of the data service resource that the data service is to expose. (In the example above the ResourceID is "MySQLResource").

For example,

globus> ant addResourceGT4 -Ddai.service.path= ogsadai/DataService -Ddai.resource.id= grid1db

You will need to shutdown and restart your container before clients are able to access the new data service resource via the data service.

After restarting your container, you can test whether the new data service resource was successfully added using the OGSA-DAI WSRF listResourcesClient client (see above).

Adding Data Service Resources using a GUI

To add data service resources to data services onto GT4:

Run the following command from within the OGSA-DAI WSRF binary distribution directory "*/usr/local/ogsadai_wsrf*":

globus> ant guiAddResourceGT4

Dynamically Add a Data Service Resource using the Command-line

If your data service is configurable then you can request that the data service immediately expose the data service resource, without having to restart your Web services container.

This is done as follows:

- Run the following command from within the OGSA-DAI WSRF binary distribution directory "*/usr/local/ogsadai_wsrf*":

  globus> ant dataServiceClient -Ddai.url=SERVICE-URL -Ddai.resource.id=RESOURCE-ID -Ddai.action=deploy

where:
"*-Ddai.url*" specifies the URL of the data service. If omitted then a default URL of http://localhost:8080/wsrf/services/ogsadai/DataService is used.
"*-Ddai.resource.id*" specifies the name of the data service resource.

For example:

globus> ant dataServiceClient
-Ddai.url=http://localhost:8080/wsrf/services/ogsadai/MyDataService
-Ddai.resource.id=MyNewResource -Ddai.action=deploy

The request will be forwarded to the data service. For example:

[echo] Deploying resource MyNewResource...
[java] Contacting ... http://localhost:8080/wsrf/services/ogsadai/DataService
[java] Service version: OGSA-DAI WSRF 1.0
[java] Number of resources: 1
[java] Resource: MySQLResource
[java] Data Service Resource: MyNewResource
[java] About to invoke Deploy...
[java] Deploy completed!

You can now use the listResourcesClient client (described above) to check if the deployment was successful.

Removing Rsources

In order to remove data service resources run the following from the OGSA-DAI distribution directory "*/usr/local/ogsadai_wsrf*":

globus> ant removeResource[GT4|Tomcat]
-Ddai.service.path=service/path
-Ddai.resource.id=ResourceID

where:
"*-Dai.service.path*" specifies the local URL of the service. For example – Ddai.service.path=ogsadai/DataService specifies a data service whose URL will be http://HOST:PORT/wsrf/services/ogsadai/DataService.
"*-Ddai.resource.id*" is the ID of the data service resource that the data service is to expose.

Ensure to restart the GT4/Tomcat container before the changes take effect.

C.5 End-to-end Client

OGSA-DAI comes with a client that allows you to:

- Contact a data service and get a list of the data service resources it exposes.
- Submit an XML-type perform document to the data service and print the resulting Response document.

To submit a Perform document to a data service resource exposed by a data service and so performs some data resource-related activites (e.g. queries a database):

- Ensure that your Web services container is running.
- Ensure that the GLOBUS_LOCATION environment variable is set.

Run the following command from within the OGSA-DAI WSRF binary distribution directory "*/usr/local/ogsadai_wsrf*":

```
globus> ant dataServiceClient -Ddai.url=SERVICE-URL        \
        -Ddai.resource.id=RESOURCE-ID                      \
        -Ddai.action=PERFORM-DOC                           \
```

where:
"*-Ddai.url*" specifies the URL of the data service. If omitted then a default URL of http://localhost:8080/wsrf/services/ogsadai/DataService is used.
"*-Ddai.resource.id*" specifies the data service resource at which the Perform document is targeted.
"*-Ddai.action*"= specifies the location of an OGSA-DAI Perform document.

For example:

```
globus> ant dataServiceClient -Ddai.url=                   \
  http://localhost:8080/wsrf/services/ogsadai/MyDataService \
-Ddai.resource.id= MySQLResource                           \
-Ddai.action= examples/JDBC/query/select1Row.xml           \
```

The Perform document will be forwarded to the data service and executed by the specified data service resource. For example:

```
[echo] Executing Perform document on resource One...
[java] Contacting ... http://localhost:8080/wsrf/services/ogsadai/MyDataService
[java] Service version: OGSA-DAI WSRF 1.0
[java] Number of resources: 2
[java] Resource: MySQLResource
[java] Resource: AnotherResource
[java] Data Service Resource: grid1db
[java] About to invoke Perform...
[java] Perform completed!
[java] Response:
[java] <?xml version="1.0" encoding="UTF-8"?>
[java] <ns1:response xmlns:ns1="http://ogsadai.org.uk/namespaces/2005/03/types">
[java] <request status="COMPLETED"
xmlns="http://ogsadai.org.uk/namespaces/2005/03/service/types"/>
...
```

C.6 OGSA-DAI Java Client Toolkit

Introduction

This part introduces the OGSA-DAI Client Toolkit, a Java API providing the basic building blocks for OGSA-DAI client development. Using these building blocks, a developer can construct anything from a basic query client to a complex distributed data integration client with relative ease.

The main concept of the Client Toolkit is that of the activity. An activity dictates an action to be performed by an OGSA-DAI service. OGSA-DAI provides many different types of activity to perform operations such as SQL queries, XSL transformations and FTP data delivery. A sequence of one or more activities can be chained together to form a request. The Client Toolkit provides a simple mechanism for constructing and processing requests. The steps involved in a typical interaction are summarized below:

1. Locate an OGSA-DAI data service
2. Construct a number of activities
3. Chain them together to form a request
4. Process the request using the data service
5. Process the results of the activities

Prerequisites

1) From OGSA-DAI distribution directory "*/usr/local/ogsadai_wsrf*", run the following as "globus":

globus> source ./setenv.sh

Note: In order not to run the command above every time a client wants to build a database, it can be added to the file "*/etc/profile.local*".

2) Ensure that OGSA-DAI services are running on **http://localhost:8080/ogsa/services**.

The MySQL database will be accessed via:

- a WSRF service located at http://localhost:8080/wsrf/services/ogsadai/DataService and resource ID grid1db

A Simple Example: Running an SQL Query

In this example, you will run a simple SQL query across a table "littleblackbook".

The following example shows you how to write a Java client that creates a Grid Data Service (GDS), queries the database and prints out the results.

SimleClient.java:

```
package tutorial.clienttoolkit;

import uk.org.ogsadai.client.toolkit.Response;
import uk.org.ogsadai.client.toolkit.activity.sql.SQLQuery;
import uk.org.ogsadai.client.toolkit.wsrf.WSRFServiceFetcher;
import uk.org.ogsadai.client.toolkit.service.DataService;

public class SimpleClient {
```

```
    /** Copyright statement */
    private static final String COPYRIGHT_NOTICE =
       "(c) IBM Corp. 2002 - 2005. (c) University of Edinburgh 2002 - 2005.";

    public static void main(String[] args) throws Exception {
       String handle = "http://localhost:8080/wsrf/services/ogsadai/DataService";
       String resourceId = args[0];
            DataService service = WSRFServiceFetcher.getInstance().getDataService(handle, resourceId);
       SQLQuery query = new SQLQuery("select * from littleblackbook where id=10");
       Response response = service.perform( query );
       System.out.println(response.getAsString());
    }
}
```

Notes:

- **Ensure that all the java packages are declared in the CLASSPATH.**
- **Be careful where the file "*SimpleClient.java*" is located.**

After creating the file, you have to compile and run it. To do so, tun the following:

globus>javac tutorial/clienttoolkit/SimpleClient.java
globus>java tutorial/clienttoolkit/SimpleClient resourceID

Notes:

- **To successfully run the above command let us assume that the "*SimpleClient.java*" is found in the directory "*/usr/local/ogsadai_wsrf/tutorial/clienttoolkit*".**
  Reading the file, a line looks like:
  package tutorial.clienttoolkit;
  From the line above, "*SimpleClient.class*" exists in the directory "*/tutorial/clienttoolkit*".
  In order to run "javac" and "java", the directory you should be in is: "*/usr/local/ogsadai_wsrf/*".
- **The first line does the compiling and generates "*SimpleClient.class*".**
- **The second line runs the "*SimpleCleint.class*" with an argument "*resourceID*" which is the same with "*dai.resource.id*".**

C.7 Setting Up a Test Database

In this page you will learn how to configure your database so that you can run some examples on the database (query the database, create tables, etc.).

To see a client's options and default settings (a client is needed in order to build a database and build a table):

globus> java uk.org.ogsadai.client.dbcreate.CLIENT -help

For example:

globus> java uk.org.ogsadai.client.dbcreate.CreateTestMySQLDB -help

"*CreateTestMySQLDB*" is a java file which is found in the subdirectory "installation_directory_of_ogsadai/doc/api-docs/uk/org/ogsadai/client/dbcreate/"

The "CreateTestMySQLDB" Client also supports the creation of the database itself, as well as the creation and population of the table within it.
If you wish to create both a database and a table within it, you can run the following:

globus> java uk.org.ogsadai.client.dbcreate.CreateTestMySQLDB
-host localhost -port 3306 -database ogsadai -username ogsadai
-password ogsadai -tablename littleblackbook -rows 1000000
-rootusername root -rootpassword passwd

The result of the above should look like:

MySQLDriverClass: org.gjt.mm.mysql.Driver
MySQLHostName: localhost
MySQLPortNumber: 3306
MySQLDatabaseName: ogsadai
MySQLUserName: ogsadai
MySQLPassword: ogsadai
NameOfTableToCreate: littleblackbook
NumberOfRowsToCreate: 10000
MySQLRootUserName: root
MySQLRootPassword:

12 Appendix D: MySQL Documentation

D.1 Installing MySQL (on Linux)

The recommended way to install MySQL on Linux is by using the RPM packages. To obtain RPM packages, click here. Otherwise, you can use a binary or source distributions to install MySQL. Visit www.mysql.org for more information.

In most cases, you only need to install the "*MySQL-server*" and "*MySQL-client*" packages to get a functional MySQL installation. The other packages are not required for a standard installation. If you want to run a MySQL-Max server that has additional capabilities, you should also install the "*MySQL-Max*" RPM.

In order to check the contenets of an RPM package (for example, a MySQL-server RPM), run:

root> rpm -qpl MySQL-server-VERSION.rpm

To perform a standard minimal installation, run:

root> rpm -i MySQL-server-VERSION.rpm
root> rpm -i MySQL-client-VERSION.rpm

To install just the client package, run:

root> rpm -i MySQL-client-VERSION.rpm

The installations from the RPM packages result in files under the following system directories:

Directory	Contents
/usr/bin	Client programs and scripts
/usr/sbin	The "*mysqld*" server
/var/lib/mysql	Log files, databases
/usr/share/doc/packages	Documentation
/usr/include/mysql	Include (header) files
/usr/lib/mysql	Libraries
/usr/share/mysql	Error message and character set files
/usr/share/sql-bench	Benchmarks

The server RPM places data under the "*/var/lib/mysql*" directory. The RPM also creates a login account for a user named "**mysql**" to use for running the MySQL server, and

creates the appropriate entries in "*/etc/init.d/*" to start the server automatically at boot time.

If you want to install the MySQL RPM on Linux distributions that do not support initialization scripts in "*/etc/init.d*", you should create a symbolic link that points to the location where your initialization scripts actually are installed. For example, if that location is "*/etc/rc.d/init.d*", use these commands before installing the RPM to create "*/etc/init.d*" as a symbolic link that points there:

```
root> cd /etc
rootl> ln -s rc.d/init.d .
```

If the RPM files that you install include MySQL-server, the mysqld server should be up and running after installation. You should be able to start using MySQL.

Note: The accounts that are listed in the MySQL grant tables initially have no passwords.

After installing MySQL on Unix, you need to initialize the grant tables, start the server, and make sure that the server works satisfactorily. You may also wish to arrange for the server to be started and stopped automatically when your system starts and stops. You should also assign passwords to the accounts in the grant tables.

On Unix, the grant tables are set up by the "*mysql_install_db*" program which runs automatically using the RPM distribution. This program sets up the initial MySQL grant tables containing the privileges that determine how users are allowed to connect to the server.

Note: The "mysql_install_db" program needs to be run only the first time you install MySQL.

To initialize the grant tables, run the following::

```
root> cd /usr/local/bin/mysql
rootl> ./mysql_install_db
```

The "mysql_install_db" script creates the data directory, the mysql database that holds all database privileges, and the test database that you can use to test MySQL. The script also creates privilege table entries for "**root**" accounts and anonymous-user accounts. The accounts have no passwords initially. Briefly, these privileges allow the MySQL "**root**" user to do anything.

Moreover, the "mysql_install_db" script creates several tables in the "*mysql*" database: "*user, db, host, tables_priv, columns_priv, func*", and possibly others depending on your version of MySQL.

If you do not want to have the "*test*" database, you can remove it by running the following:

```
root>mysqladmin -u root drop test
```

In the instructions shown above, the server runs under the user ID of the "***mysql***" login account. This account will be created automatically during the RPM installation.

D.2 Starting & Stopping MySQL

To start and stop MySQL automatically on your server, you need to add start and stop commands to the appropriate places in your "*/etc/rc**" files.

The Linux RPM package (MySQL-server-VERSION.rpm) installs the "mysql.server" in the "*/etc/init.d*" directory with the name "*mysql*". No need to install it manually.

You can add your own options to the server, simply by, creating a global file "*my.cnf*" in the "*/etc*" directory.

A typical "*/etc/my.cnf*" file might look like this:

```
[mysqld]
#Location of the data directory
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql

[mysql.server]
# Base directory under which MySQL is installed
basedir=/usr/local/mysql
```

Note: When the "*mysqld*" server starts, it changes location to the data directory "*/var/lib/mysql*". This is where it expects to find databases and where it expects to write log files. On Unix, the server also writes the pid (process ID) file in the data directory.

If the data directory is located somewhere else on your system, the server does not work properly. To solve this, just add the new locations to the option file "*/etc/my.cnf*".

In order to start the server manually, run the following:

```
root>mysqld_safe
```

or

```
root>mysqld_safe --user=mysql &
```

Run “mysqladmin” to verify that the server is running.

```
root>mysqladmin version
```

The output from “*mysqladmin*” version varies slightly depending on your platform and version of MySQL, but should be similar to that shown here:

```
shell>mysqladmin version

    mysqladmin  Ver 8.40 Distrib 4.1.13, for linux on i586
    Copyright (C) 2000 MySQL AB & MySQL Finland AB & TCX DataKonsult AB
    This software comes with ABSOLUTELY NO WARRANTY. This is free software,
    and you are welcome to modify and redistribute it under the GPL license

    Server version          4.1.13-max
    Protocol version        10
    Connection              Localhost via Unix socket
    TCP port                3306
    UNIX socket             var/lib/mysql/mysql.sock
    Uptime:                 5 days 19 hours 19 min 0 sec

    Threads: 1  Questions: 163  Slow queries: 0
    Opens: 11  Flush tables:1  Open tables: 0  Queries per second avg: 0.007
    Threads: 1  Questions: 9  Slow queries: 0
```

You can also use the “*--help*” with “*mysqladmin*” to see what else you can do.

Verify that you can shut down the server:

```
root>mysqladmin shutdown
```

Verify that you can restart the server. Do this by using “mysqld_safe”:

```
root>mysqld_safe --user=mysql --log &
```

To connect to the server, run the following:

```
shell> mysql
```

If that works, you should see some introductory information followed by a “*mysql>*” prompt:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 4.1.16-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

The "*mysql>*" prompt tells you that "*mysql*" is ready for you to enter commands.

After you have connected successfully, you can disconnect any time by typing "*quit*" (or "*\q*") at the "*mysql>*" prompt:

```
mysql> quit
Bye
```

You can also disconnect by pressing "*Control-D*" (on Unix).

Run some simple tests to verify that you can retrieve information from the server. The output should be similar to what is shown here:

```
shell>mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 4.1.16-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> mysqlshow
  +-----------+
  | Databases |
  +-----------+
  | mysql     |
  | test      |
  +-----------+

mysql>mysqlshow mysql
  Database: mysql
  +--------------+
  |    Tables    |
  +--------------+
  | columns_priv |
  | db           |
  | func         |
  | host         |
  | tables_priv  |
  | user         |
  +--------------+

mysql>mysql -e "SELECT Host,Db,User FROM db" mysql
  +------+--------+------+
  | host | db     | user |
  +------+--------+------+
  | %    | test   |      |
  | %    | test_% |      |
  +------+--------+------+
```

D.3 Entering Queries

In order to run some queries, make sure that you are connected to the server.

To see a list of options provided by "*mysql*", invoke it with the "*--help*" option:

```
shell> mysql --help
```

Here's a simple command that asks the server to tell you its version number and the current date.

```
mysql> SELECT VERSION(), CURRENT_DATE;
+------------+--------------+
| VERSION()  | CURRENT_DATE |
+------------+--------------+
| 4.1.14-Max | 2005-09-03   |
+------------+--------------+
1 row in set (0.01 sec)
mysql>
```

This query illustrates several things about mysql:

- A command normally consists of an SQL statement followed by a semicolon. (There are some exceptions where a semicolon may be omitted, e.g. "QUIT").
- When you issue a command, "*mysql*" sends it to the server for execution and displays the results, then prints another "*mysql>*" prompt to indicate that it is ready for another command.
- "*mysql*" displays query output in tabular form (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), "*mysql*" labels the column using the expression itself.
- "*mysql*" shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency.

Note: Keywords in MySQL can be entered in any letter case.

D.4 MySQL Administrator

MySQL Administrator is a graphical program for performing administrative operations, such as configuring your MySQL server, monitoring its status and performance, starting and stopping it, managing users and connections, performing backups, and a number of other administrative tasks.

You can perform most of those tasks using a command line interface such as that provided by “mysqladmin” or “mysql”, but MySQL Administrator is advantageous in the following respects:

1. Its graphical user interface makes it more intuitive to use.

2. It provides a better overview of the settings that are crucial for the performance, reliability, and security of your MySQL servers.

3. It displays performance indicators graphically, thus making it easier to determine and tune server settings.

MySQL Administrator runs on Linux machines that have a graphical desktop installed.

In order to install MySQL Administrator, download the RPM package and run the folllwing:

shell> rpm -i MySQL-administrator_version.rpm

To run MySQL Administrator just type the following:

shell>mysql_administrator

D.5 MySQL Query Browser

The MySQL Query Browser is a graphical tool for creating, executing, and optimizing queries in a graphical environment. Where the MySQL Administrator is designed to administer a MySQL server, the MySQL Query Browser is designed to help you query and analyze data stored within your MySQL database.

While all queries executed in the MySQL Query Browser could also be performed in the mysql command-line utility, the MySQL Query Browser allows for the querying and editing of data in a more intuitive, graphical manner.

To install MySQL Query Browser, first download the RPM package and then run the following:

shell> rpm -i MySQL_query_browser_version.rpm

To run MySQL Administrator just type the following:

shell>mysql_query_browser