The Globus Toolkit

Lecture of the course of Complements of Enabling Platforms

Master in Computer Science and Networking - SSSUP, UNIPI

Gianmarco Saba

Contents

1

Introduction to the Globus Toolkit

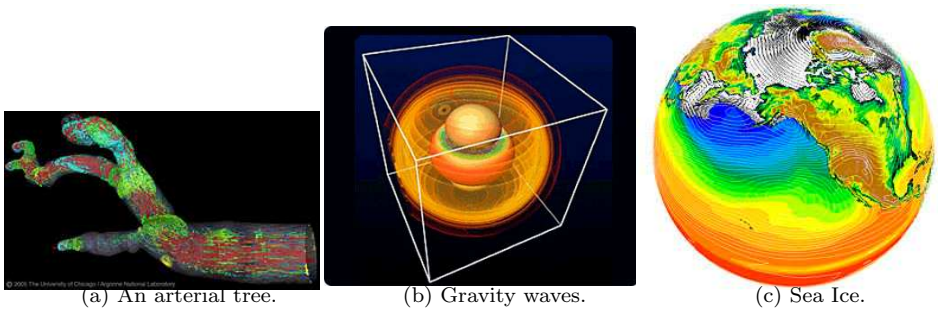
Gianmarco Saba
gianmarco.saba@gmail.com

The *Globus Toolkit*[1] is a software toolkit addressing key technical problems in the development of Grid Enabled Tools, services and applications. It offers a modular "bag of technologies", enables incremental development of Grid-enabled tools and applications and implements standard Grid protocols and API's (the "core" of the so called hourglass). The inventors decided to provide not just an operating system of which you don't know anything of what is going on inside, but they provided a small set of independent technologies in such a way that if I provide 3, 4 basic technologies, you can use just the one you need for your particular grid and you can completely forget the others; you don't have to install the whole middleware to have a grid. If you are just interested in grid security, you can completely forget *Resource Management*, *File Transfers*, *Information Management*: just build the grid. This has been decided and according to the hourglass model that we have seen, on this few components we build some more components in such a way to provide an incremental association of tools according to this model. For sure, they wanted to implement the very core of the hourglass. In this way they achieved few standardized protocols needed to build everything up. Globus is available under the open source licence and, Created by I. Foster and C. Kesselman, it was the *de facto* standard for grid computing middleware in the golden age of grid computing.

1.1 Examples of the Globus Impact

The Globus Alliance and the Globus Toolkit have enabled many exciting new scientific and business applications. The images here showcase just a few of the advances that have been helped by Globus technology. Computational scientists at Brown University are using the Globus Toolkit and MPICH-G2[2] to simulate the flow of blood through human arteries. This image, prepared at Argonne National Laboratory, shows velocity (red arrows) and pressure

[1] `http://www.globus.org/toolkit/`
[2] `http://www3.niu.edu/mpi/`



(a) An arterial tree. (b) Gravity waves. (c) Sea Ice.

FIGURE 1.1: Example of Globus Employments

(surface color) within a branched, three-dimensional arterial structure. The simulation was conducted using Nektar (software developed at Brown University) and was the first high-performance simulation to run in a distributed fashion using systems at multiple TeraGrid sites. Physicists used the Globus Toolkit and MPICH-G2 to harness the power of multiple supercomputers to simulate the gravitational effects of black hole collisions. The team, which included researchers from Argonne National Laboratory, the University of Chicago, Northern Illinois University, and the Max Planck Institute for Gravitational Physics in Germany, was awarded a prestigious Gordon Bell prize for its work. Again, scientists in the Earth System Grid[3] (ESG) are producing, archiving, and providing access to climate data that advances our understanding of global climate change. This image displays data from ESG and shows sea ice extent (white/gray), sea ice motion, sea surface temperatures (colors), and atmospheric sea level pressure (contours). ESG uses Globus software for security, data movement, and system monitoring.

1.2 General Approach

The authors defined the grid protocols and the relative APIs. The protocols mediated access to remote sources integrating and extending existing standards. Globus has been developed as first reference implementation, providing SDKs, services, tools etc. The very base protocols were FTP[4], SSH[5], Condor[6], SRD, MPI[7] and others. Without Globus toolkit, grids would not exist since there were no standards, no protocols. On top of Globus, the *European Data Grid Project*[8] was built, the *Egee Project*[9] was built, the *Glite Middleware*[10] was built, the *CoG Middleware*[11] , the *Xtreme OS*[12] middleware in some way was built. All the middleware that have been implemented here for the *ASSIST Programming Envi-*

[3] `http://www.earthsystemgrid.org/`
[4] `http://tools.ietf.org/html/rfc959`
[5] `http://tools.ietf.org/html/rfc4252`
[6] `http://www.cs.wisc.edu/condor/`
[7] `http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf`
[8] `http://eu-datagrid.web.cern.ch/eu-datagrid/`
[9] `http://www.eu-egee.org/`
[10] `http://glite.web.cern.ch/glite/`
[11] `http://www.globus.org/toolkit/cog.html`
[12] `http://www.xtreemos.eu/`

ronment [13] were built on top of this technologies. The set of all the basic protocols needed to implement whatever grid were this four:

- **Security**. This is the most important protocol. We must provide a secure layer in such a way that it can be used by all the remaining protocols from the upper level on. Over this layer, they decided to provide only three families of protocols. As we will see some protocols families are composed by just one single protocol; other families have two or three protocols implementing several layer because here we are at the resource level and we may have something at the collective layer because, remember, the *source* layer means a single resource; *collective* means coordination of several resources. In some of this protocols we will have a protocol for the resource level and another protocol built on top of the previous one for the collective layer, but the three main families are:
  - **Resource Management**. It mainly means, from the point of view of the grid, desktop machines or clusters or in general computation power;
  - **Information System**. In order to floadthe grid with information about everything: the status of a resource, who is in charge of distributing the data, what is the CPU power of the resource we are looking for. For example, if you have any kind of performace model for your application, in some way if you want to see a result in less then one month wou would need one hundred CPUs working at 2 GHz of speed with 4 GB of memory. How do you look for this information in the grid? Someone has to provide this information. Who?
  - **Data Management**. This is a base protocol for grids working with huge amount of data (LHC[14] experments for example).

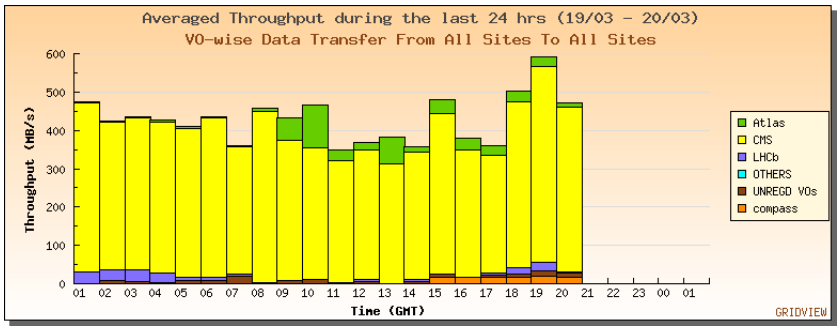


FIGURE 1.2: Throughput of the LHC experiments.

[13] http://www.di.unipi.it/Assist.html
[14] http://lcg.web.cern.ch/LCG/

1.3 Evolution of the Globus Toolkit

Now The Globus Toolkit 5 has just been released; it is a standard for cloud computing middleware. The first version (1.0) was released in 1995 as a bunch of software written in C providing implementations of basic protocols, then the CERN in Geneva provided a financial assistance to the project and in this way the Globus Toolkit version 2 was released.

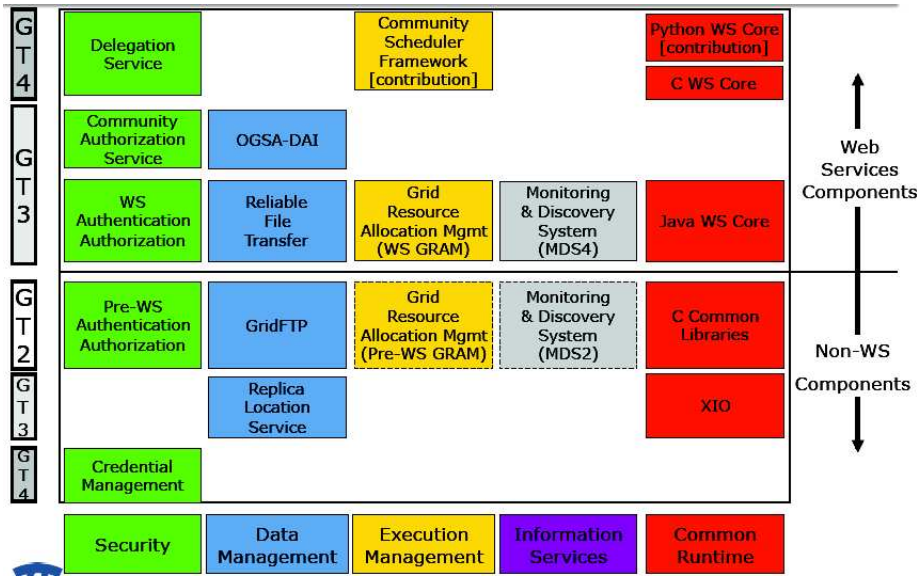


FIGURE 1.3: Globus Open Source Grid Software.

In this periond I. Foster and C. Kesselman proposed the concept of *Grid Service*. The structure of the Globus Toolkit version 2 was composed by:

- **Security**: in the version 2 we have protocols for *authentication* and *authorization*
- **Data Management** where the main protocol was named *GridFTP*;
- **Execution Management**: with as main protocol *GRAM* (*Grid Resource Allocation Protocol*) that provided access to the resources
- **Information Services** with the *Monitoring and Discovery System* (MDS2)
- **Common Runtime**: a set of common functionalities written in C.

In the next version (3) the implementors decided to provide a common resemblance of every single kind of service used in grid: they didn't want to implement a specific kind of server for GridFTP (an high performance tranfser protocol not so reliable) because otherwise it would be necessary to compile the implementation in each architecture, they thought to a way to provide a common set of guidelines to implement this set. This was exactly the scope of the Grid Service. Some concepts were born in the wrong way since they have never been used basically (mainly regarding the version 3). In GT3 everything was based on Grid Service which was a kind of messy evolution of Web Service. They provided four protocols on top of the Web Service Protocol. At Web Service level, due to backward compatibility

reasons, the old 4 protocols were encapsulated, exploited by new implementation so the Run Time Support was moved from C to Java. So they reimplemented the MDS providing MDS3 which was completely refactored from scratch. Then Web Service Grid Allocation Manager was a protocol built inside a web service that has a back-end which was called the *PRE-WS* protocol. They added to the Data Management a *Replica Location Manager* and a *Community Authorization Service*. In April 2005 the GT4 version was released. Why GT4? At the time, the messy *Grid Service* was something more than a common Web Service. This something more was something used from the grid services but was not standardized, it was used in an ad-hoc way just within a grid service and not in the all services area. Some of the facilities required from the grid services could be potentially used for web services in general. What is this feature? The *state*. A web service, by definition is *stateless* (see the next section). They said "we need a state!", for example to give information. We use this information written in some custom files on the MDS tree in different machines. They thought that it was a mess and they ended up on the WSRF (*Web Service Resource Framework*[15]) which was the standard for web services with state. This means that at this point we had a completely new standard that we must use in order to implement all the services. Not using web services plus grid services, but completely from scratch we use WSRF. Since they could provide a status they provided another protocol to exchange status information between web services so WSN (*Web Service Notification*[16]) was born. Our web service can, in this way, publish his status to another web service in such a way that the second one can exploit this information, used for example when we have a network of WSN servers sharing the data all around.

So they started to write everything and the back-end was always the same.

1.4 Web Services

A web service works in this way. I have a service running somewhere using WSDL[17], UDDI[18]. Imagine the web service is providing two services: a way to calculate the fast Fourier Transform[19] and one to do some spectral correlation analysis. In order to use these functionalities, we have just to provide an array of Java doubles (double samples). We send a SOAP[20] message with this data to the web service. Here the web service performs the calculation (half an hour if we are lucky!) we will receive back another array of Java doubles containing the frequency components of the spectrum. At this point I want to do some calculations, I want to send it back to the same web service, asking to another calculation. Considering the size of a double, one million of doubles means 8 MB and if they are represented in XML[21] format they can take up to 80 MB. The transmission of such amount of data can last a long period. I have to receive the data and to transmit back the same data to the web service. This means that when a computation is done, the communication is over. I am not using the memory of the server machine to keep the data. The job at server side is finished, the web server has no idea that we are going to send him the same set of doubles to compute another operation.

[15] `http://www.globus.org/wsrf/`
[16] `http://www.ibm.com/developerworks/library/specification/ws-notification/`
[17] `http://www.w3.org/TR/wsdl`
[18] `http://uddi.xml.org/`
[19] `http://en.wikipedia.org/wiki/Fast_Fourier_transform`
[20] `http://www.w3.org/TR/soap/`
[21] `http://www.w3.org/XML/`

2

Grid Security

Gianmarco Saba
gianmarco.saba@gmail.com

Speaking about security, we will use some terminologies, like:

- **Authentication**: I must be sure that you are who you are saying to be;
- **Authorization**: We need to know what each entity is allowed to do;
- **Integrity**: I must be sure that when data are sent (this is always a problem in data transmission), the data receive correct and not corrupted data. I must have a way to check that the data is correct because for example I can lose some bit of data during the transmission;
- **Confidentiality** When a message is sent from a source to a destination, I don't want that any other entity listens the message. This was one of the main problem in grids because a lot of companies wanted to use the grid computers to run experiments in order to analyze markets; such information represents a competitive advantage: they wanted to be sure 200% that the data they were distributing among the computers was not spoof and readed by no one else except from the receiver.
- **Non-repudiation**: when I send my data I don't want, after a while, to change my mind and say "This is not my data";
- **Delegation**: if I send a job to an entity, and this one delegates subtasks to other entities, I have to adopt some security mechanisms to propagate my authorization and my authentication to all the resources involved in the computation
- **Single Sign-On**: I want to provide my password only once during the login phase and I don't want to do this many times in order to have access to different resources. I need a way to propagate my password and my certificate around the grid;
- **Digital Signature**: we want some digital mechanisms in order to do this, if someone is in charge of checking my ID, I want that a software can manage automatically this signature in order to establish my identity.

2.1 Public Key Infrastructure

The core protocol and the associated infrastructure of the Grid Security is the PKI (*Public Key Infrastructure*). Who needs an identity certificate, an ID? Users, administrators and resources as well. To provide this ID we exploit the *Public Key Infrastructure*[1] which works in this way: every single entity needing an ID request to the authorized entities a public and a private key. The data that these entities want to transmit to someone else is encrypted using its private key. The public key is distributed around the world. How can we distribute the puclic keys around the world and being sure that the identities are true? This information is encapsulated inside a *certificate*. A certificate is like an ID or a passport. In this certificate we have the name of the entity, its public key, who emitted this certificate and in some way the signature of the owner. There is a standard in order to put all these information inside a certificate which is the *X.509* [2]. The public authorities are demanded to keep the file of all the IDs. There is not just one but a small set of legal entities known as *Certificate Authorities*[3] which can put their signature on the certificates stating that the certificate is correct. How to obtain a certificate?

2.1.1 Certificate Issuance

To request a certificate a user starts by generating a key pair. The private key is stored encrypted with a pass phrase the user gives; the public key is then put inside a certificate request. The user takes the certificate to the CA which usually include a *Registration Authority* (RA) that verifies the uniqueness with respect to the CA and the given name is the real name of the user (through ID/passport check). The CA finally signs the certificate request and release the certificate for the user.

2.2 Why Grid Security is hard?

Grid security is the most difficult part of the protocols. In grids we have *Virtual Organizations*, we have several administrative domains with several different certificate authorities providing certificates which bring with them critical issues. Each resource has its own policies & procedures; again, set of resources used by a single computation may be large, dynamic and unpredictable: for example within a virtual organization a member can can give up, join the group or membership subgroups. We have to cop with dynamic environments: the machines fail, if you are storing the authorizations in a certificate server and the server goes down, you can't access the resources. How to solve this problem? We have to replicate these servers several times. How do we connect and how do we recover the information in case of failures?. In clouds the security is not a problem. In fact, Google has all its machines in its buildings.

2.2.1 Grid security Requirements

The requirements of grid security can be bounded depending on the family of user we want to take into account:

[1] `http://www.pki-page.org/#CA`
[2] `http://www.ietf.org/dyn/wg/charter/pkix-charter.html`
[3] `https://www.tractis.com/help/?p=3670`

- From a **user view**, the requirements are:
  1. ease of use;
  2. single sign-on;
  3. possibility to run applications
     - FTP, SSH, MPI, Condor, Web, etc.
  4. user based trust models;
  5. proxies, agent(delegation).
- From the **Resource owner view** we need:
  1. Specification of local access control;
  2. auditing, accounting, etc.;
  3. integration with local systems:
     - Kerberos[4], license migration
  4. protection from compromized resources.
- Finally, from the **developer view** we need:
  1. API/SDK with authentication, flexible message protection, flexible communication, delegation,...
     a) direct calls to various security functions (e.g. GSS-API);
     b) security integrated into higher-level SDKs

2.3 Grid Security Infrastructure (GSI)

All these requirements have been implemented inside the GSI (*Grid Security Infrastructure*[5]) which is an extension of a set of standard protocols(SSL[6] /TLS [7], X.509[8] & CA, GSSAPI[9]) used to provide all the requirement we saw. Globus is a reference implementation of GSI, so it provides:

- SSLeay[10]/OpenSSL[11]+GSSAPI+single sign-on/delegation;
- tools and services to interface to local security: simple ACLs[12], SSLK5/PKINT
- tools for credential management:
  - login, logout, etc.;
  - smartcards; MyProxy[13];
  - KScert.

[4] http://web.mit.edu/Kerberos/
[5] http://www.globus.org/security/overview.html
[6] http://info.ssl.com/article.aspx?id\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{1\global\mathcha
[7] http://www.ietf.org/dyn/wg/charter/tls\discretionary{-}{}{}charter.html
[8] http://www.ietf.org/dyn/wg/charter/pkix\discretionary{-}{}{}charter.html
[9] http://www.ietf.org/rfc/rfc2743.txt
[10] http://www.columbia.edu/~ariel/ssleay/
[11] http://www.openssl.org/
[12] http://en.wikipedia.org/wiki/Access_control_list
[13] http://grid.ncsa.illinois.edu/myproxy/

2.3.1 Delegation Proxies

What is a proxy? A proxy is the key component of *delegation*, it is an extension introduced in grids. Delegation is basically implementing the remote creation of a proxy credential. During this process a new pair of keys is generated on the server, the proxy certificates and publics the key sent to the client; then, the clients sign the proxy certifications and returns it. The server (usually) puts proxy in the temporary directory. The delegation allows remote processes to authenticate on behalf of the user so the remote processes "impersonate" the user. Which kind of proxies can we create? Basically two. A **limited proxy** that is a proxy that has a reduced set of rights because is not really used but is something that acts on the behalf of the user and, for example, if I am able to start a process on a remote resource with a limited proxy, this is not allowed to start a new job somewhere, but tries to access information on my behalf. Why? Because if you have a full proxy you can start spreading jobs on the grid. That's why we want limited rights, it is a security requirement of the resource owner of the grid. We can have a local policy to manage limited proxies everywhere. A **restricted proxy** is a superset of a limited proxy. With a limited proxy you cannot start processes. A restricted proxy is a kind of proxy such that you can store inside it the rules which the proxies must adere to, for example, as before, limited set of IP adresses that can be used for transferring proxy. Another example is that you can use just resources from IBM and not from Google, you can use only 16 bit architectures and not the 32 bit ones. In this you can implement the **proxy usage rules**. The Grid Security infrastructure can be summarized in three parts: **Public Key Infrastructure**, **Secure Socket Layer** (SSL), **Transport Layer Security** (TLS) for authentication and message protection and the new mechanism for delegation and single sign-on.

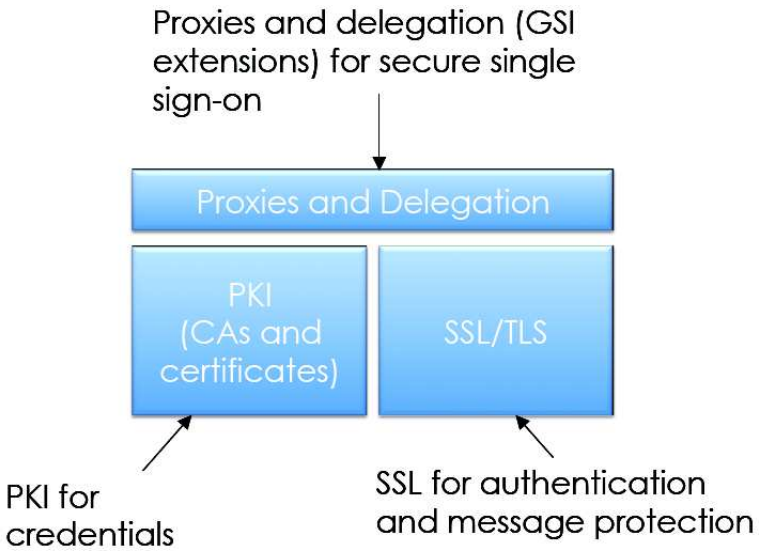


FIGURE 2.1: Security layer structure.

Let's see in action how all this works.

Example

I want to create two processes A and B communicating and at a certain point A needs a file stored in another resource C. If I am a user, which are the steps that I must take in order to start this job from the point of view of security? First I need a single sign-on to a grid identifier that is my name on the grid generating the proxy for this job: I am not going to give to the machines the ID but the proxy of the ID (I can use a proxy for storage somewhere on the network, an online repository). Then I generate the proxy certificate and the proxy credential.

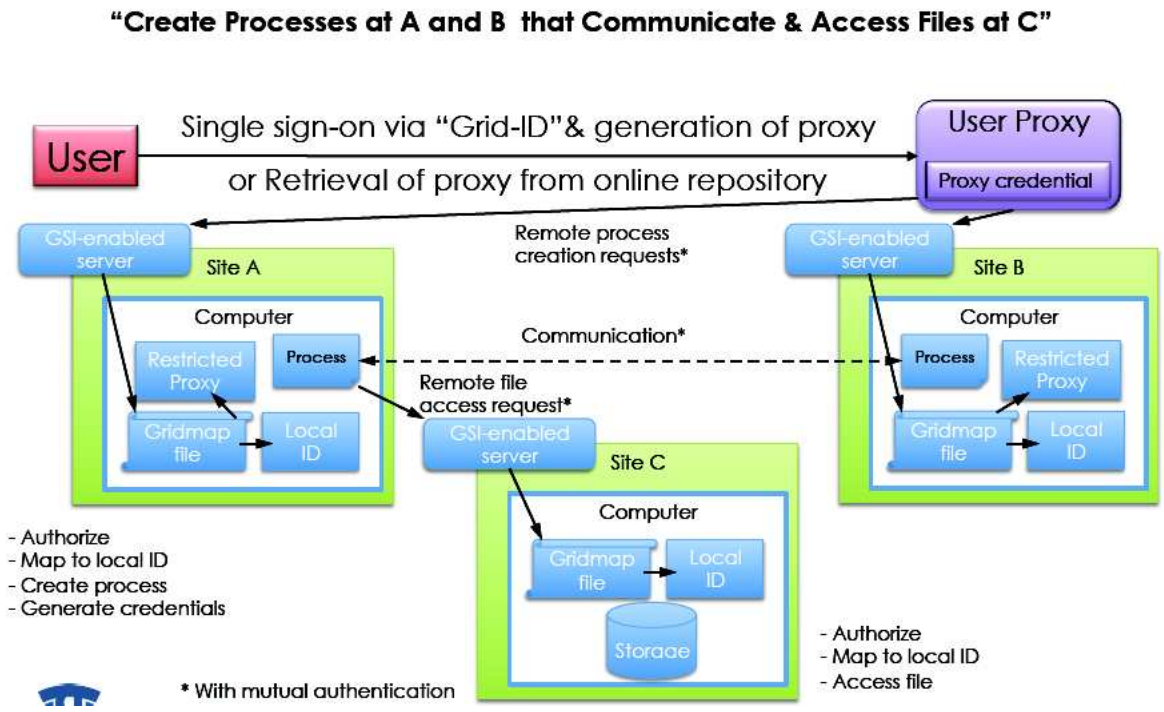


FIGURE 2.2: Example of GSI execution.

My proxy credentials are sent to the computer on side A and to the computer on side B. Here we have a kind of server depending implementing: for example GRAM or GridFTP that has GSI capabilities. This is in charge of checking the identity of the users sending me the stuff, but it must send back to the proxy the certificate of the computer in the communication because the user proxy wants to be sure that is the right source. I want the resource A, not someone saying that it is A. So this is a double direction exchange. Let's see at the single side what happens.

After the GSI server authenticate my ID and this was authenticated by the user proxy, such credential are stored somewhere and are used to access the local security mechanism of the computer. In order to do so, in Globus, you have the concept of GridMap file which is just a simple text file that is protected by root access on the local resource that is a mapping from a grid ID to a local user. This means that when the computer receives a

request to do some work on another computer behalf a user with a given ID, we will have a map of such ID in the local user of the machine. That means that in the local machine every work is done as by the local user. The local administrator can change file everytime because the final control of the local resource is given by the local resourse manager. After that, the communication is started.

2.4 Configuration

2.4.1 Obtaining a certificate

The program *grid-cert-request* is used to create a public/private key pair and an unsigned certificate where:

- *usecert_request.pem* is the unsigned certificate file and
- *userkey.pem* is the encrypted private key file

then, it is necessary to send an email with the *usercert_request.pem* to *ca@globus.org* and wait for the Globus-signed certificate. Other organizations usually use different approaches, for example NCSA[14], NPACI[15] and NASA[16] have their own CA. The received certificate will be something like:

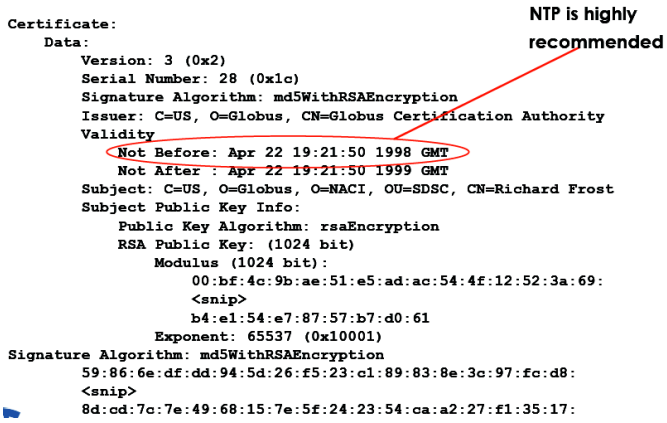


FIGURE 2.3: Security layer structure.

2.4.2 Logging-on to the Grid

In order to run a program on the grid it is necessary to authenticate on Globus before. This can be done by the command *grid-proxy-int* and entering the proper *PEM pass phrase*. This creates a local, short-lived proxy credential (a file) for usage by our computations.

[14] http://www.ncsa.nato.int/
[15] http://npacigrid.npaci.edu/
[16] http://www.nasa.gov/

Now the user has to insert his pass phrase which is used to decrypt the private key. The private key is used to sign a proxy certificate with its own, new private/public pair. In this way the private key is no more exposed after proxy has been signed and no network traffic is needed. The default lifetime of a proxy is 12 hours, so if needed we have to specify an adequate duration.

2.4.3 Logging-off to the Grid

In order to destroy our local proxy that has been created by grid-proxy-init we use the command *grid-proxy-destroy*. We have anyway to pay attention to the fact that this does not destroy any proxies that were delegated from this proxy since we have no control on remote proxies. That's why proxies with short lifetimes are created.

2.4.4 Important files

Some important files are listed below grouped by origin directory:

- **/etc/grid-security**
  - *hostcert.pem*: certificate used by the server in mutual authentication;
  - *hostkey.pem*: private key corresponding to the server's certificate (read only by root);
  - *grid-mapfile*: maps grid subject names to local user accounts (really part of gatekeeper);
- **/etc/grid-security/certificates**
  - *CA certificates*: certs that are trusted when validating certs, and thus need not be verified;
  - *ca-signing-policy.conf*: defines the subject names that can be signed by each CA
- **$HOME.globus**
  - *usercert.pem*: Users certificate (subject name, public key, CA signature);
  - *userkey.pem*: Users private key (encrypted using the users pass phrase);
- **/tmp**
  - *Proxy file(s)*: Temporary file(s) containing unencrypted proxy private key and certificate (readable only by user accounts)

3

Globus Upper Layers

Gianmarco Saba
gianmarco.saba@gmail.com

3.1 Security and upper layers

In the previous chapters we started discussing the de facto standard implementation of grid middleware named Globus and the software tools provided by the Globus Consortium which are generally known as the Globus Toolkit. We discussed about the key protocols that are implemented, the security protocols that are the foundation for any other additional protocol of the grid. We have now to discuss the other three pillars: the *Resource Management*, the *Data Management* and the *Information Management*. In Globus they are implemented by the *Grid Resource Allocation Manager* that is a protocol implementing what is needed to manage the computational resources on the grid, the *Monitoring and Discovering Service* that is a set of protocols (we will see two of them) to manage the information for several uses on a grid and the *GridFTP* protocol that is an high-performance protocol to transfer huge amount of files. This is a general picture of everything working in the Globus Toolkit. Let's recall some ideas that we discussed previously about the security. We saw that from the client point of view security is just a proxy credential generated from an assigned certificate to work with grid resources with potentially restricted tools to access resources and limited time scope to use such resources the user working from a machine, which is part of the grid, just need to authenticate against this certificate, to generate the proxy and all the remaining problems regarding security are directly managed by the grid middleware.

So this is used by the client tools provided by the Globus Toolkit to interact with the servers implementing the additional services provided by the Toolkit. The proxy is used by some tools which can be command line tools for applications to allocate and manage jobs on the grid, to find resources on the grid and to transfer and control the transfers of data on the grid. As usual, these protocols (the protocols for resource information and data) are built on top of existing protocols to avoid to reinvent the weel. In particular the GRAM protocol is built on top of HTTP protocol and an additional protocol that we will see is the RSL *Resource Specification Language* which is a clear syntax to write job submission specifications; we will see that we have different proposals for a protocol to describe jobs and relative requirements. The protocols for the monitoring and discovering services are built on top of the LDAP protocol regarding Globus version 2; from the version 3 on, the protocol has been reimplemented from scratch. The GridFTP server is built, as you can imagine from the name, on the FTP protocol but we can use other services that are provided by Globus that are on top on GSI-FTP with Grid Security Infrastructure as a back end,

HTTP Secure and just file transfer between local resources. We start now with the *Grid Information Services*.

4

Grid Information Services

Gianmarco Saba
gianmarco.saba@gmail.com

4.1 Resource Discovery and Monitoring

Why are Grid Information Services needed? We are in an environment where resource discovery and monitoring is a critical issue because we have a distributed set of users and resources. We have a lot of external enties that can modify the virtual organization layout because these events are in some way not forecastable. As we know, virtual organizations are dynamic entities that are created and destroyed according to the rules of the virtual organizations. So the layout can change accordingly to the changes in virtual organizations. Then, since everything is distributed, we can't have a single point to access information, we have the problem that resources can fault (network links, a computer can be powered off, used by a local user and consequently its characteristics for the grid change).

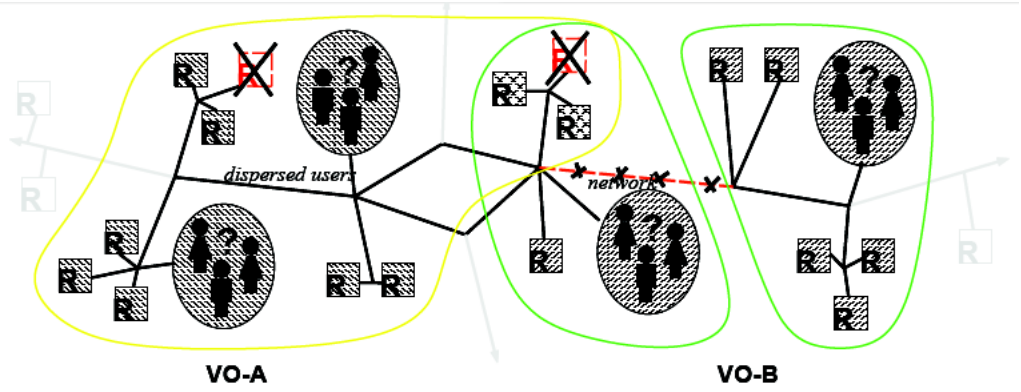


FIGURE 4.1: Example of a Virtual Organization structure.

We need a way to monitor all these changes. Even if we have to admit from the very beginning that is impossible to have at a certain point in time the global status of the grid because to collect information we need a time and even if it is 15 minutes, this means that the information is old because you don't know if after 15 minutes a given computer is still on or not. You need some time to collect such amount of information. Morover everything is distributed and a user simply does not know which are the resources in the grid, even if the grid is stable we need a way to collect all this information and to provide this information to the user, to clients or to tools that are used by the user to interact with the grid. For example, if I need to run a job that requires 1000 machines, we will have a program written to specify which machines you need in terms of computing power and memory and then something should be in charge of finding these resources on the grid. How? Checking somewhere the characteristics of the resources, collecting the right resources and for example their IP addresses. Providing this information is useful in order to distribute the code among the resources and to connect the communications between your application modules.

4.2 Grid Discovery

Resource Discovery. What does it mean grid discovery? Discovery is intended as resource discovering. Which kind of resources could they use to run my application?

Resource Inquiry I need now a way to compare them in such a way I can select the best one according to the requirements of my job.

Resource Control How can I take control of the resources?

Let's see what kind of information we need to store at this step. For Resource Discovery we need to store information about the search. When we are looking for a research we describe the hipothetically resource that you would like and we want they to be collected. At the second point we need information on mechanisms to compare such descriptions. At the third level, even if how to take control of the resources is not part of the information service, we have anyway information that is used at a certain point. Which kind of information? For example if you take control of a resource, somewhere should be written that the resource is not available from this time point to another time point. This is another kind of information about the status of the resource that must be published.

In grids we can create thousands of taxonomies, we can create taxonomies for about everything, some of them are meaningful and some of them are completely useless. What is useful is the distinction between *static* information and *dynamic* information. Some of the information stored in a Grid Information Service is static in the sense that it does not change with time or it requires a lot of time to change. Tipical examples of such information are hard disk capacities, the identities of the partners of an organization, or the partners of the CNR users because you can use users as well in an information system, or again the kind of a CPU. Even if the machine goes down or the connection fails, the architecture (Power PC, x86, etc.) does not change. On the other end, tipical dinamic information is the current load on a machine because if I need to run an application I need to know the load of the machine, if the machine is idle is good, if it is too busy it is almost useless. An Information Service needs to manage these two very heterogeneous sets of data because these data are completely different.

How do we organize this information? We organize this information at the two levels we know of the grid stack, at *Resource level* and at *Collective Layer.* At Resource Level we are going to provide information in a coherent and standardized way using a common syntax for every resource for a single resource. I am just providing access to the information

of the CPU type and load of a computer in a standard way, for example an XML file or any other kind of different syntax that must be the same for every resource but up to now we have that every single resource is giving us its information. We stop at this level: to obtain information we must contact every single resource of the world to know their characteristics. This is something that is not enough for us so in a Collective Layer we put something different that is going to collect the information from the several resources and providing to the users of this service a whole set of information regarding several resources in such a way that if I want to discover something I just need to connect to something inside the collective layer that is going to do the job to collect the data from the low level resources dinamically.

4.3 Requirements

Here we have a list of problems that arise and require that a generic user could put on an Information Service for the grid.

- **Performance**. An Information Service must be performant in the sense that I need answers in milliseconds, not in hours, it is like a search engine, it must be performant;
- **Scalability**. It must be scalable in the sense that with 10000 resources more, it should give me the same response time;
- **Cost**. It must not be costly to install and administrate, it should be easy;
- **Uniformity**. It should be also uniform in the sense that we don't have to connect with 3 different protocols to obtain information used by the same application; we just use one protocol;
- **Expressiveness**. It must be expressive in the sense that we want to express a lot of heterogeneous information with the same service;
- **Extensibility**. It must be exstensible because we can't forecast all the potentially information that can be managed in the grid from now to the end of the world. At a certain point you can imagine that you need some additional information service and you need a mechanism such that you don't need to recompile everything.
- **Multiple information sources**.
- **Dynamic data**. We have to provide a lot of data while the information are changing quickly in time;
- **Access Flexibility**.
- **Security**.
- **Easy to employ**.
- **Decentralized Maintainability**.

The monitoring and discovery service has the following structure:

The base of everything is LDAP, it is composed by server interfaces and client interfaces. The LDAP client interfaces are directory mapped in the MDS client interfaces. Ho do we use these interfaces from the server side to implement the monitoring and discovery service? As shown in the figure, at the bottom we have a local resource, a computer with several resources: memories, CPUs, network. For every resource inside this computer there is an information provider that is a particular piece of software that is in charge of interrogate in some way the local resource and collect information. We will have an information provider

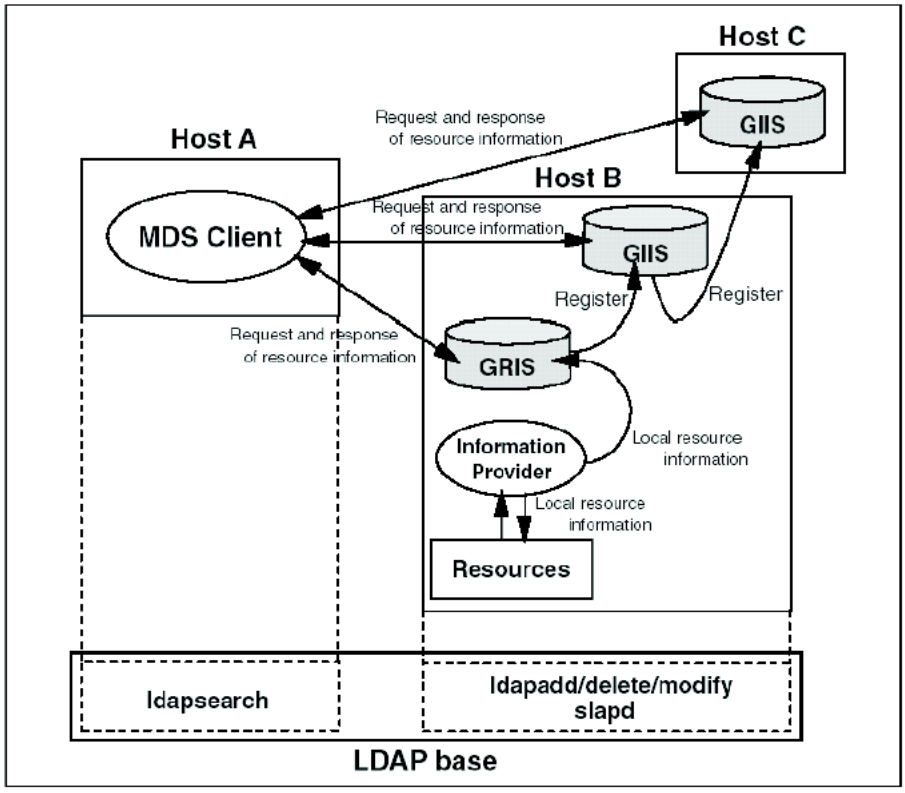


FIGURE 4.2: Structure of the Monitoring and Discovery Service.

for the CPU attributes, another for the load of the machine, for the memory, for the network and so on. How to collect? We need an information provider for every single resource. For example, in the Linux filesystem, we know that the information about the machine is contained in the */proc/* filesystem. It is a particular directory on each machine where inside there are some files that are written by the system and that contain information about how much memory do you have, the brand, the connection, etc. . Every half an hour, this information are read and broadcast in the middleware.

Each information provider is resource dependent and that is why want it to be extensible. All the information provider are providing from several sources the information to the GRIS (*Grid Resource Information Service*), it is a modified LDAP server that is going to store all the information collected from the local information provider on a resource, providing mechanisms to implement a distributed directory service like the one in LDAP. How? We are not going to reimplement everything, we are just leveraging the LDAP mechanism to propagate information and to distribute queries. At which level of the stack we think the service is going to be put? At Resource level.

Then we are at the point in which each resource is able to publish some data, but now we need something at collective layer we can use as an entry point of the directory service and this is implemented by the GIIS (*Grid Indexing Information Service*). This is something at higher level that is just collecting all the information from local resources and, for example, is saying that the resource of CNR are being monitored. The power of the GIIS is that

we don't have only registration from grids, but we can have hierarchical registrations from other GIIS services in order to built a distributed tree structure.

The client using the LDAP mechanism can connect freely with the same interface and the same protocol to a first level GRIS server or at upper level servers. The GRIS needs the IP address of the server it wants to connect to. If we connect to GRIS we can have just information stored in a close area, but if we connect to one of an upper level we will have access to information about everything. Of course, the performaces at such higher level won't be as high as the one in the lower levels because a lot of users will try to connect to the top level server.

4.4 MDS-2 Information Model

Following we have an example of information Model of MSD-2:

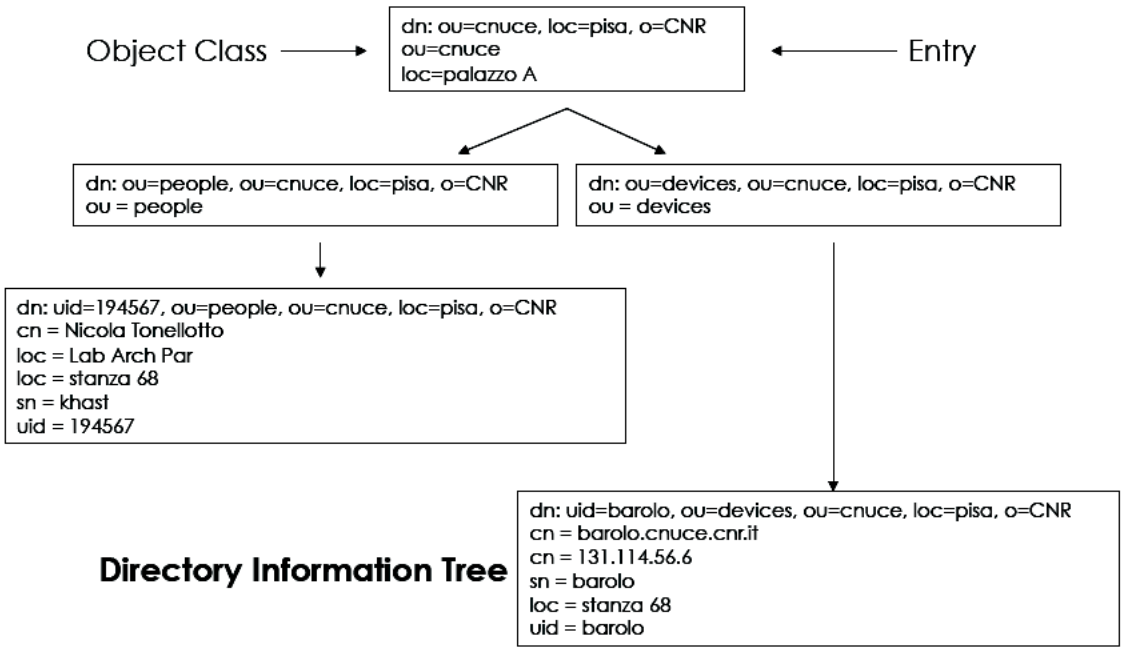


FIGURE 4.3: MSD-2 Information Model.

Every box is a single file containing its related information. How in a single entry the information is structured? We have a list of names and a list of values. The specification of the list of names with their type is given somewhere by the designer of the tree in somethingh that is called *Object Class*. What is the name of an entry? The name of an entry is the list of all its attributes. Moving down we can have additional attributes.

4.5 MDS-2 Functional Model

The functional model is very simple. Ho do we interact with LDAP? In the first way we submit a search operation, we have a returned entry and a code describing how the operation was performed. In the second way, you can ask for more than one result and then you will have a result submitted every time a single answer is found; then the result code is submitted. Finally, the third way consists in a secure connection: we can open a connection and keep the connection open because we want to submit several queries. After the work is finished, the connection is closed.

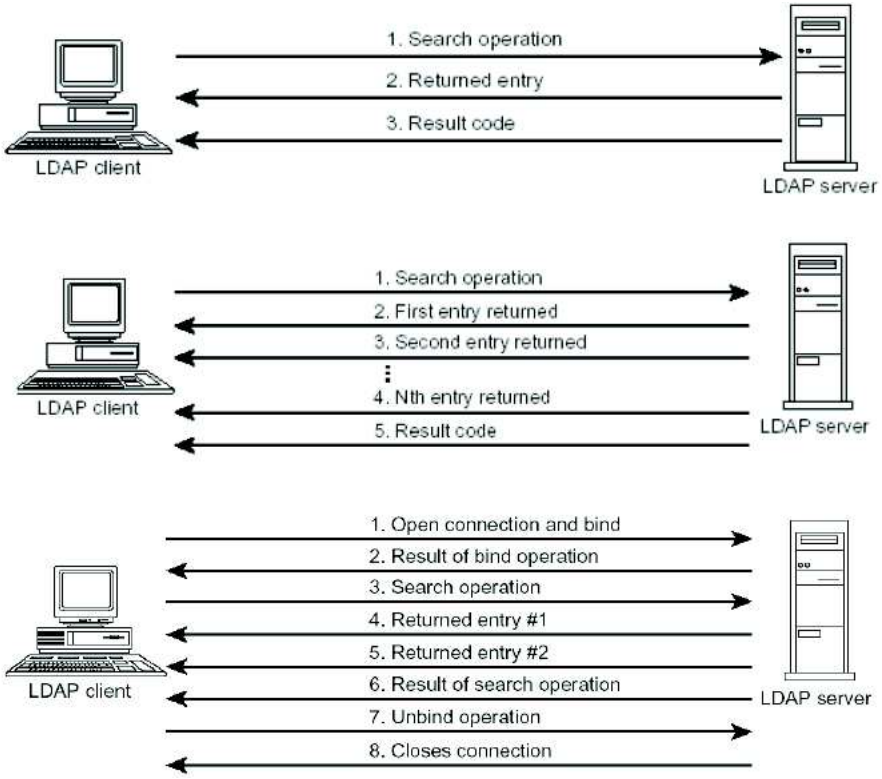


FIGURE 4.4: MDS-2 Functional Model.

4.6 MDS-2 Data Representation

What kind of information do we put inside an LDAP resource directory service? Every grid can select the information needed so every virtual organization potentially can select which kind of information must be provided to MDS. At the beginning each entry of the representation system was grouping resources depending on their kind: disk, memory, CPUs, OSs, etc. A resource is an entry but we know that inside a computer we have several resouces, in fact we will have leaves node for network interfaces grouped at the next level in the set of all the network interfaces of a computer, the same for disks, the same for memory

banks, the same for CPUs the same for the OS and then all these groups of resources are collected in another node that is the description of the host. This model was not well structured, not really describing everything, so it was too simple and they decided to complicate it. This is the Glue Schema from the High Phisics community:

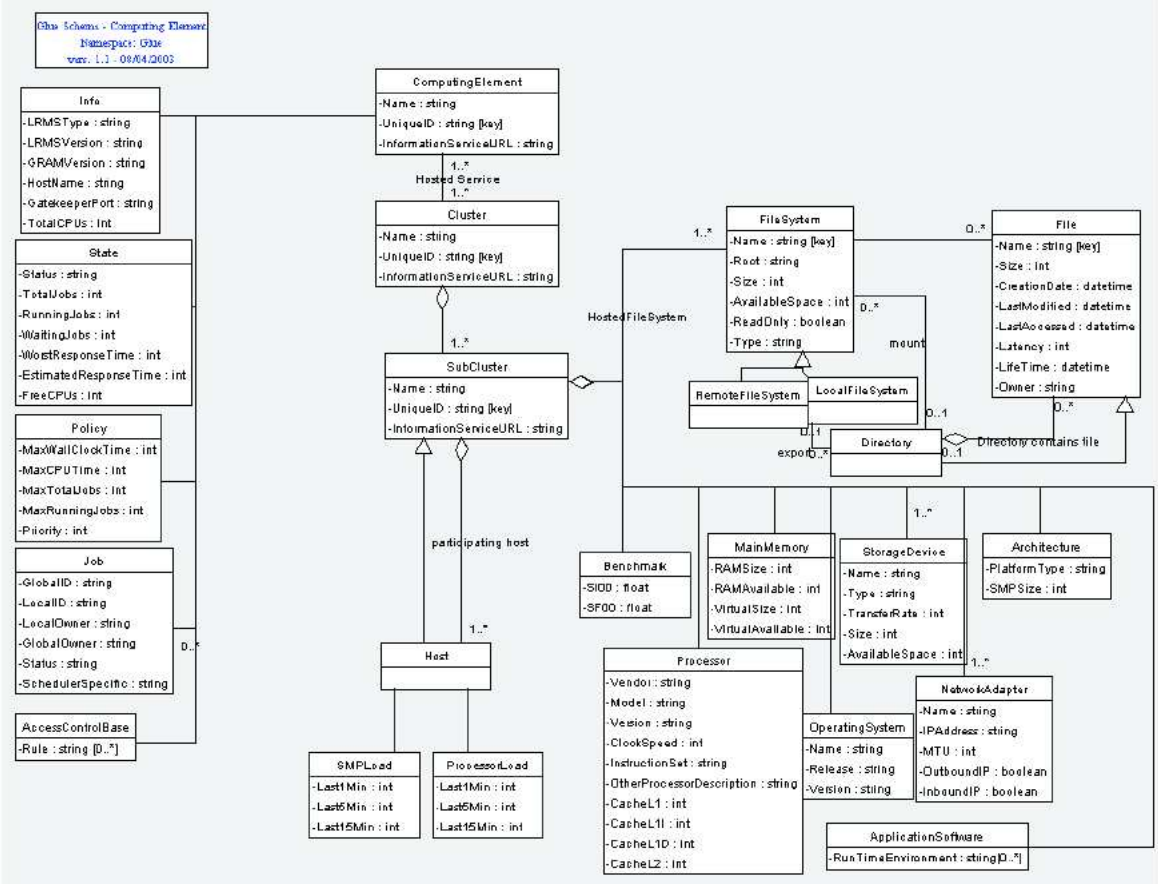


FIGURE 4.5: GLUE Schema: computing element.

Computer that are to do calculation, and computers that are deal to store data. We don't have just a computer, but computing elements. We need to describe computing element and storage elements. Each computer data element can be a single CPU or a multiprocessor machine or a cluster so they started adding network adapters, outbound IP, inbound IP.

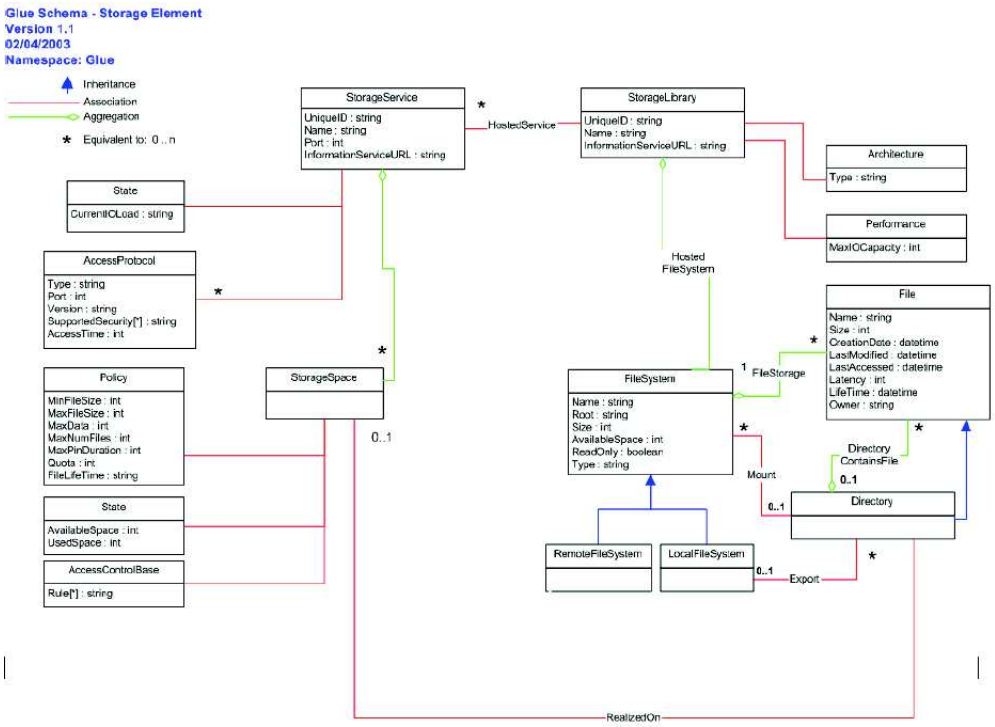


FIGURE 4.6: GLUE Schema: computing element.

5

Grid Resource Management

Gianmarco Saba
gianmarco.saba@gmail.com

5.1 Resource Management on HPC Resources

Which are the resources used in HPC? Basically clusters of computers or supercomputers. A cluster is a set of independent commodity computer hardware connected with an high speed network to manage the work. A supercomputer is a huge bunch of processors and they are tipically managed (at least a subset) in a shared way. They[1] can have very large scale: *Earth Simulator*[2], *Jaguar*[3]. What does manage resource mean ? Basically when we want to manage these HPC resources we want to manage processors and memory to send jobs submitted by a potentially large number of users and we then want to manage this kind of jobs, we need to have on the resources something installed to perform some management work to garantee that the system is working more or less well. This means that those Resource Management systems need to include at least three components. The first one is a mechanism, tool, laboratories to configure this machine and its behaviour, the submitted jobs we will have at runtime. The second part is another tool distributed in the implementation that is able to collect information about the status of jobs and the status of machines to give a potential snapshot as good as possible of the current state machine in order to take scheduling decisions to run the job. The last part is the actual low-level working stuff that is the job management services that are responsible to take the job from

[1] `http://www.top500.org/`
[2] `http://www.jamstec.go.jp/es/en/index.html`
[3] `http://www.nccs.gov/computing-resources/jaguar/`

the user and allocate, distribute, run these jobs on some machines that must be selected according to some intelligence inside this resource management system. Basically there is no standard at all for this kind of resource manager. Every different kind of cluster or supercomputer can have installed completely different resource management systems and the most import that are commercially available are the PDS (*Portable Batch System*[4]), LSF (*Load Sharing Facilities*[5]), NQS(*Network Query System*[6]), LoadLeveler[7], Condor[8].

5.2 HPC Management Architecture in General

In the general architecture of a resource management system, we have a set of resources, in this case a set of high performance computers, a set of clusters, that are basically processing executables. Every computing node has on top of it a local resource manager that is in charge of managing the local resource and to collect information about the status of the resource (busy, available, etc.). All these local resource managers are in contact with another computer that is the master computer responsible to control the general status of the single resources and to accept the jobs from the users. As you can notice, this is quite similar to the grid architecture saw previously, but we must consider that a computational cluster from the grid point of view is seen as a resource so we will see at the fabric layer. We can have two clusters in the same grid with completely different local resource management systems: one for example running *PDS* and another running *Condor*. How to have them communicating? We need some middleware to have an homogeneous interface to access these cluster resource management systems.

5.3 Computational Job

Who is going to provide this kind of information? This kind of information is provided by the user. The user must be in some way an expert in describing the job; the tipical descriptions for the computational jobs in every cluster resource management system with completely different syntaxes and maybe semantics are really low level. They are expressing just these attributes: the family of the CPU, the number of the nodes and their speed, the memory size, the I/O capabilities. Another point that is important is that we can specify, for example, the software libraries needed to be installed on the resources because, for example, we are dispatching Java jobs that require some parsing facilities or some high performance mathematics that is installed into the resources (*ScaLAPACK*[9], *eiPack*[10]s). Another issue is the licensing. If, for example, I am using some operating system with propetary software, I have a bunch of resources, a bunch of licenses, I must be sure that I have a license available on every machine where I want to exploit the propetary software.You can, in general, not just specify constraints (if you don't give me this particular resource, I am not going to run the job on your machine) but, in a cluster, if several solutions are available, I can provide a kind of preference specification like providing a scoring formula that say that

[4] `http://www.cs.cmu.edu/~mseltzer/talks/pbsqueue.pdf`
[5] `http://www.platform.com/workload-management/high-performance-computing/lp`
[6] `http://www.eecis.udel.edu/documentation/nqs.html`
[7] `http://www.mhpcc.edu/training/workshop/loadleveler/MAIN.html`
[8] `http://www.cs.wisc.edu/condor/`
[9] `http://www.netlib.org/scalapack/`
[10] `http://people.iq.harvard.edu/~olau/software/eiPack.html`

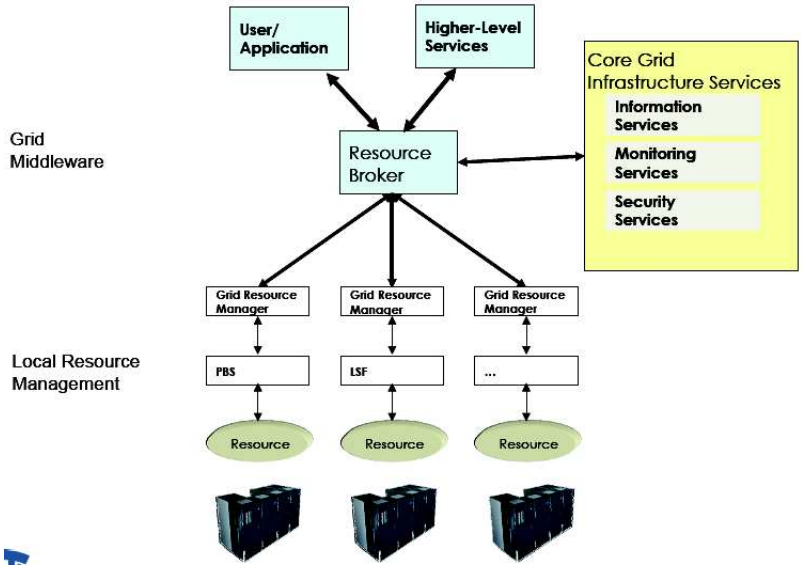


FIGURE 5.1: Grid Resource Management.

in the case we have more then one solution to run the job, select the one maximizing or minimizing a particular cost function specified through preferences. For example I prefer that all the CPUs are inside the same supercomputer or the CPUs are in different computers but connected to the same rack.

We must describe all this stuff using a particular syntax and format but we have no standard at all. PSD can express some things in a way, LoadLeveler in another way and is not easy to have they communicating with each other.

5.4 Transition to the Grid

Until now we spoke only about clusters; let's move to grids. We have this scenario which is really complex but when we speak about grids the scenario is more complex than the previous one because we have different levels of complexity. The first one is the fact that we have more kind of resources. In grids we can have a broader meaning for the term job: just consider the tipical high performance computing job is a computational one (which is a data intensive one). Another problem for grids is that resources are distributed and may belong to different administrative domains and they are coming and going with no control at all. In a grid you can completely loose the control. All these problems must hence be addressed and solved.

5.5 Scope of Grids

The main goal of grids was to be able to run the same applications which were running on HPC clusters but in a distributed way such that the grid reseambles like a huge cluster and is managed with solutions close to the cluster resource management system. One of the most common scenarios is the one where the grid is composed of supercomputers. Just consider

this example: here in Pisa we have a supercomputer, a cluster, and in the computer science department there is another cluster. All clusters are not enough to run an application because we have a very CPU intensive application that require a lot of nodes. The solution can be buy/rent extra clusters that are very expensive.

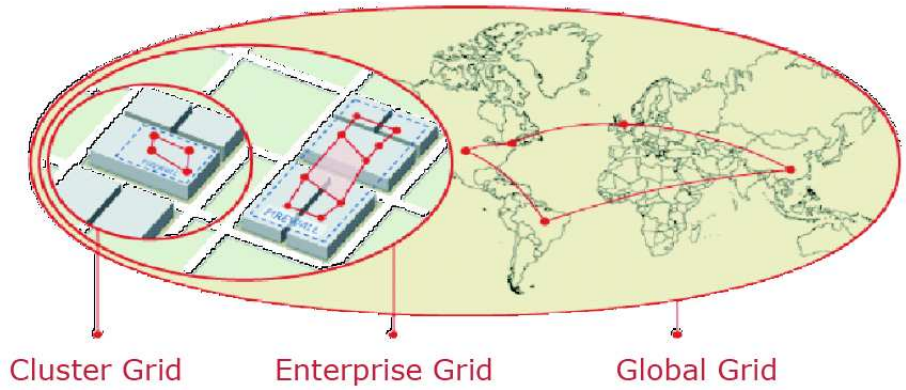


FIGURE 5.2: Scope of grids.

If the entire grid is hosted inside the same hosting building we have no security problems but if we have an enterprise grid with a connection with a remote cluster we want to collect all the remote resources which can have different administrators but in some way some security mechanisms are required at institution level. The same is true for companies like IBM who have huge supercomputers in different buildings and they want to collect them in order to provide a so called *enterprise grid*. We have in this way geographically distributed resources that have their own set of routines but we have a set of commons rules that are shared by all the resources. Then we move to the upper level called *global grid* that is a connection of computers around the world with the most heterogeneous set of specifications and is almost impossible to have this kind of grid. I can have at least two kind of grids that are at the global view. The SETI@Home[11] grid is spread all around the world. The High Energy physics grid [12] is spread among departments of each university in the world are connected to share and process the LHC data. They are running models to identify particles and performing other important tasks.

5.6 Implications of Grid Resource Management

What are the implications of this distribution of resources on the resource management in grid? The most important are:

- *Security.* The king of problems is security, often the root of all the other problems.

[11] `http://setiathome.berkeley.edu/`
[12] `https://igi.cnaf.infn.it/communities/hep`, `http://grid.uchicago.edu/`, `http://heprc.phys.uvic.ca/`

This is because we have a complete heterogeneous set of access policies to access resources, we have a completely heterogeneous set of mechanisms to authenticate, authorize and taking into account of who is doing what, who is going to pay to use my resources. How do we express the access to a particular resource?

- *Lack of global information.* We can just say: even if we have a common interface and a common syntax to describe the jobs and access clusters, clusters start from the point that they have a snapshot of the state of the resources. In grids it is impossible. We have in the latter case a huge round-trip time to collect data and it is changing very often. In a cluster is just a matter that inside the same High Performance Network Infrastructure I have to connect 10, 100, 1000 job status information. On a global grid the same amount of machines connected by faulty links, modems or ADSLs we cannot garantee this service.
- *The resources are quite heterogeneous.* Of cource, while a cluster or a supercomputer is a set of general homogeneous resources and each computer is the same from the architectural point of view, with the same software installed, in grids is very difficult to collect all this information.

5.7 Grid Resource Management

At which level of the original architecture are we going to put new protocols to have a grid resource management architecture? After this point is clear that at this level we have to work on two levels at least: the *resource level* because we need to give at each single resource an interface to expose to the upper layers and then we need something at the *collective layer* that must be able to cop with all these distributed interfaces along the world and in some way manage huge sets of resources and the relative huge set of users. What do we have at resource layer? At resource layer we can have a cluster, a desktop computer, a supercomputer. These computers are the basic unit of work for the resource management so in each one we must provide a common standard infrastructure in such a way that the resource is accessible in the same way from remote locations. In Globus this is implemented by the protocol that is called GRAM (*Grid Resource Allocation Manager*)then, once we have this GRAM infrastructure, we must move to the next level that is the *global resource management system* at *collective layer* where we must provide a protocol or a set of protocols to manage all the local resource management system filtered by the GRAM protocol within a distributed organization like environment. At this layer we must provide the additional mechanism to manage the complexity of the jobs and of the infrastructure in the grids. For example we need to provide a mechanism where users can submit job that will be distributed to the next resources. We need a mechanism, in some way, to discover resources available in such a way that when a resource appears we can automatically take its existence into account and use this resource to schedule jobs. We need mechanisms to schedule jobs on these resources.

Globus is actually not providing any kind of collective layer protocol or mechanism to manage resources but we have several solutions at this level using the Globus and, in general, the Resource Layer protocols to implement some work. We have for example services called *meta schedulers* or *resource brokers* that are in charge of doing exactly this: manage a large set of resources. The new architecture of grids is going to be composed by computers, supercomputers and clusters. Everyone is a resource and is locally managed by a local resource management system. All these heterogeneous schedulers are filtered by the grid resource manager that is GRAM in the Globus case, then we have that the resource broker

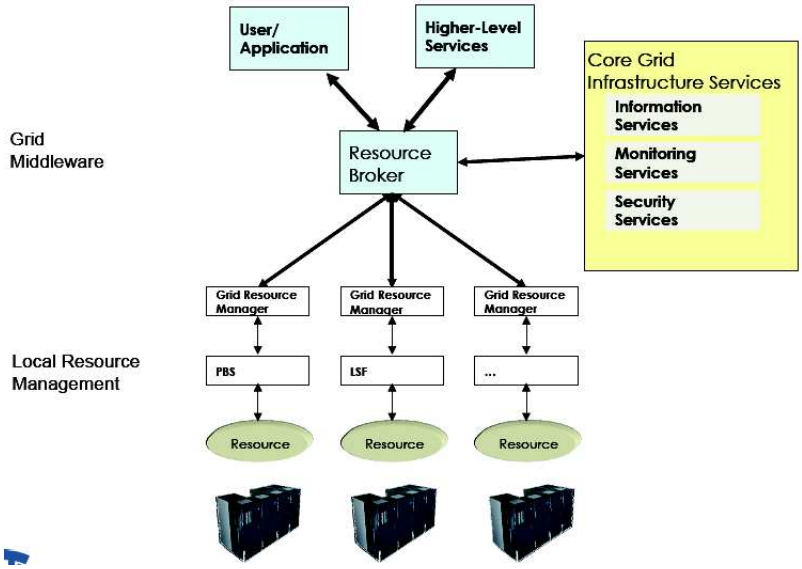


FIGURE 5.3: Grid Resource Management.

can use the address services, if available, on a grid system to work better. So, for example, if we need to take into account the status of the resources are we going to really implement mechanisms or can we use something provided by the other protocols or services? In this case we can think to use MDS.

5.8 Some Definitions

- A *job* is something that must be run on a resource as part of a job request.
- A *job request*, on the converse, is a message containing the request of the execution and the specification of a job on a remote resource. A job request is containing the description of which kind of resource we must run, which kind of process downloaded from this particiular point in the grid and writing data in this particular directory and using this particular directory for the files and so on. Tipically we can have even when and where, how and what to create and how to execute the process beacuse remember that a job can be a single process or a parallel application we are going to run on a cluster, so they could be more than one job, they could be more then one task created on the local resources.
- In Globus Toolkit 2 the GRAM service is called *gatekeeper*. It is a name for a general service that is the one speaking and giving the common layer at the resource layer to dispatch jobs. What is the gatekeeper doing?

5.9 Gatekeeping (GT2)

A gatekeep has a chain of responsabilities. At the end of the chain of responsabilities it will instanciate what is called the job manager that is the local process in charge of execute and

monitor the required job. It is a kind of container that is in charge of forking a new process, loading the process and the data in memory and publishing on the MDS the status of the job for dektop resources. For cluster resources the job manager will be an instance of the local resource management system responsible for submitting the job to a particular queue or a set of queues of the cluster and in charge of meaning from the mechanisms of the cluster the status of the jobs and publishing this information, if required, in the grid information service. The gatekeeper is, from a very low-level point of view, a deamon process always running in background that is on each computational resource. It is just listening for new job requests. When a new job request arrives it always execute a sequence of operations: first of all *mutual authentication*: when the authentication is correct it means that the job is arriving from a good resource with the ID proxy certificate of the user and I am providing him information about who I am and the user accepts my credential. What do I need? I need to map this proxy credential to the local user, then, with the rights of this local user, the gatekeeper creates a job manager process and after this process is created we load on this process all the descriptions of the run time environment of the job expressed in the job request and then it gives to the job manager an independent life. The gatekeeper then starts listening again for new requests. Let's see an example:

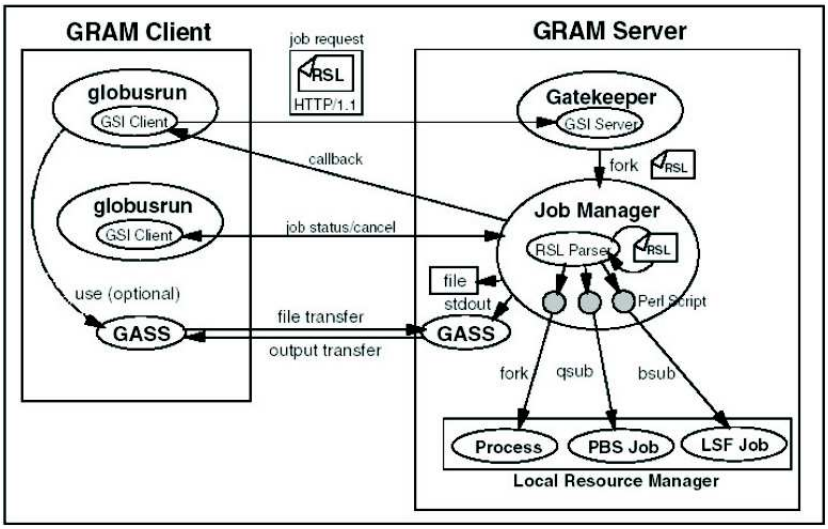


FIGURE 5.4: GRAM components.

The above is the tipical view of the functioning of the protocol. Inside the gatekeeper we have the server responsible for security. *globusrun* is going to send to the gatekeeper a job request. In GT2 the job request was expressed with a sintax named RSL (*Resource Specification Language*) that was sent on top of HTTP and was a standard description of every job. The gatekeeper is going to mutual authenticate with the client and then with fork a job manager was created. To the job manager a RSL description was passed. Inside there was a parser that was able to obtain all the information to create the job from the RSL description. This means that for every process that was specified in the request, a new local smaller manager is created always inside the job manager for every single process that

is responsible to translate the process execution request into the terms of the local resource manager. This means that this is responsible for desktop resource to write forks, execs, etc. In RSL is specified that the output must be stored in a local file. The job manager is in charge of redirecting the output from these processes to the local files or could be specified that the output must be streamed back to the client. This straming is implemented via another protocol that is the protocol to manage data in the grid (the GASS protocol). For example, in the picture above we are transferring output files in a streamed mode during the execution. Again, from the client point of view, the client should be able to cancel the job whenever replies; so the job manager is responsible to provide the information about the status of the job to the MDS but to the client as well because the client is always here and if the clients wants to cancel the job, the job manager must be listening for this kind of request to readily cancel the job or to readily notify the client of job status changes if the client, of course, is not going to use MDS to have this information.

```
'&(executable=/bin/ls)
  (directory=/tmp)
  (arguments=-l)
  (environment=(COLUMNS 40))'

'&(executable=/bin/ls)
  (count=3)(project=GlobusTeach)
  (mAxCpUtImE=10)(max_Memory=30)'

'&(executable=https://barolo.cnuce.cnr.it::65092/bin/ls)'

'&(executable=/bin/ls)
  (stdout=$(HOME)/ls.out)
  (stderr=$(HOME)/ls.err)'
```

FIGURE 5.5: RSL Examples.

5.10 Job Status

A job can flow through a different set of states with respect to a simple process. Here we have a job that can be *unsubmitted*. When a job is submitted, it can be *queued* (waiting for a resource), it can be *run* (there is a job manager controlling the work) or can be in a *staging* situation that means that before the job is actually started, the job manager needs to download from somewhere on the grid the data needed to run the job, for example the input data of the job. From staging we can fail, we can be queued or we can become active. When we are active we can finish with success, we can finish with failure or we can require to stage out at the end of the computation the data produced back to the client. In this case we can go to the stage out state and again stage out can run complete successfully or for some reasons the download back to the client goes wrong and then the complete job is considered failed because at the end of the day we need to produce the data.

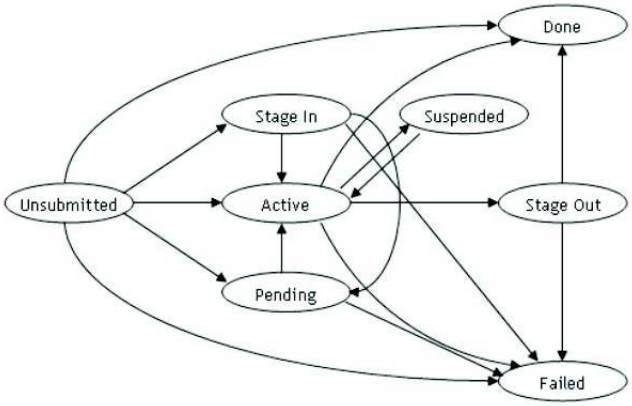


FIGURE 5.6: Job Status.

5.11 Problem: Job Submission Descriptions Differ

The RSL was developed keeping in mind desktop machines but when things changed people from OGF[13] started working at a new specification that is the JSDL (*Job Specification Description Language*[14]) that is the job submission description layer own language in such a way that we have a standard that is good for desktop machines and cluster resource managers (RSF ,PPS, LoadLeveler, GridEngine, Condor). This is one of the most successful specifications in the OGF and is basically an XML file. What is particularly good in the standard is that we have a set of translation tables that are able to translate a JSDL file (a job request) in a local job request for PDS or LSF schedures. The job attribute categories include:

- **Job Identity Attributes**: ID, owner, group, project, type, etc.
- **Job Resource Attributes**: hardware, software, including applications, Web and Grid Services, etc.
- **Job Environment Attributes**: environment variables, argument lists, etc.
- **Job Data Attributes**: databases, files, data formats, and staging, replication, caching, and disk requirements, etc.
- **Job Scheduling Attributes**: start and end times, duration, immediate dependencies etc.
- **Job Security Attributes**: authentication, authorization, data encryption, etc.

5.12 What do we have at the collective layer?

At collective layer we have custom solutions. The GRAM protocol provides a standard interface to access local resources. At collective layer:

[13] `http://www.gridforum.org/`
[14] `http://www.gridforum.org/documents/GFD.56.pdf`

- Resource brokers;
- Metaschedules.

5.13 The Metascheduler Example

The following example is drawn from Condor. When Globus Project started its activity, Condor was adapted in some way in order to work with Globus and one of the most successfull additions to Condor was the *Condor Metascheduler*. The name of such scheduler is DAMan[15] (*(Directed Acyclic Graph Manager)*). DAGMan allows you to specify the *dependencies* between your Condor-G jobs, so it can *manage* them automatically for you (e.g., Dont run job B until job A has completed successfully.). A DAG is defined by a *.dag* file, listing each of its nodes and their dependencies: each node will run the Condor-G job

```
# diamond.dag
Job A a.sub
Job B b.sub
Job C c.sub
Job D d.sub
Parent A Child B C
Parent B C Child D
```

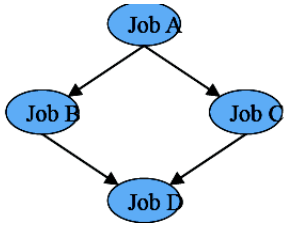


FIGURE 5.7: Dependencies graph.

specified by its accompanying *Condor submit file*.

[15] `http://www.cs.wisc.edu/condor/dagman/`

6

Grid Data Management

Gianmarco Saba
gianmarco.saba@gmail.com

6.1 Data Management Services

Which are the core data services of data management available in Globus? We have two families of protocols: protocols to access and transfer data and protocols to manage replicas of the data. Data access and transfer mean that we want to move data between resources in a standard way. Which kind of data? Small files: for example standard output, standard error, configuration files. This means that a simple protocol like HTTP is enough. HTTP is generally enough in order to let the partners speaking about the resources, so we just need an HTTP implementation simple enough for the scope of the Grid Data Management. This is implemented by GASS. When we have huge amounts of data we need a high performance protocol to move this data. Such protocol proposed, implemented and provided by the Globus community was the GridFTP that is an extension of the FTP protocol with some added features in order to provide reliability and high performances during file transfer or big files.

The second family is the family to manage replicas of this data: if I have more than one copy of a file that is a huge one, how can I guarantee the coherence of this file once it is modified? If I have multiple copies which one can I use in order to perform a download? This is a set of two main protocols and services:

- **Replica Catalog**: Service to keep updated information on sets of replicated data;
- **Replica Management**: Service to create and manage sets of replicated data

6.2 GASS: Global Access to Secondary Storage

GASS is a simple service strictly integrated in GRAM. It is a mechanism to access files from remote locations and is basically used for two things:

- to download executables from remote sites;
- for move standard input, output and error to/from remote sites in a streamed way. Stramed means: don't download the standard output, just at the end of the execution or during the execution, sustain the traffic to send information back

in order to check the status. It can be used with configuration files as well, just consider it as HTTP.

The components use, as usual, APIs to access the files that are the OS specific *open* and *close* and the read and write operations are automatically managed by the GRAM (GASS) because you read and write just when you want to download the files or when you want to stream back the standard input, output and error. We have just open/close but read and write are not exposed to the users. We have JSDL extensions to the language to specify uniform resource specifiers to express where to upload or download the files and some simple tools to cache remote data. For example, if an executable is used three or five times, I am loading it just one time and then the content is cached in memory and all the processes can access this copy in cache. Here we have an example:

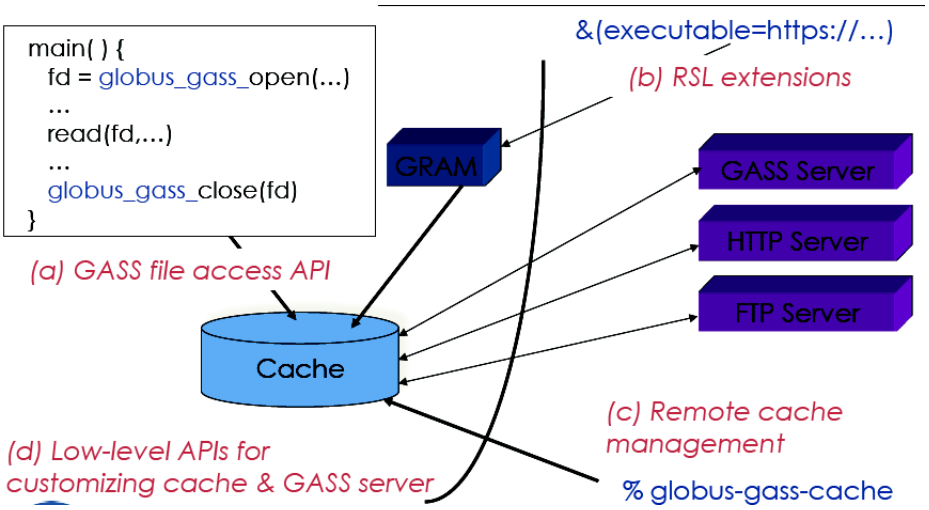


FIGURE 6.1: GASS Architecture.

In the picture we can notice the APIs that we can use. If, for example, I use a program where instead of open I have Globus GASS open/read/close, I can use the file using these APIs or I can specify files that can be staged here. This specification is going to be parsed by the GRAM so both of them access a file that is locally cached in the remote site. In this cache, the GASS middleware is responsible to download or upload the files required in the applications or specified in the RSL specification using a family of protocols that are HTTP, FTP, GASS. This means that I must have from the local site a server providing the file which can be a simple GASS server but I can download these data from HTTP servers and FTP servers as well. Then I have a set of low level mechanisms to manage the data inside the local cache.

We already discussed about the integration of the GRAM and here we have a general picture of how it works:

6.3 FTP

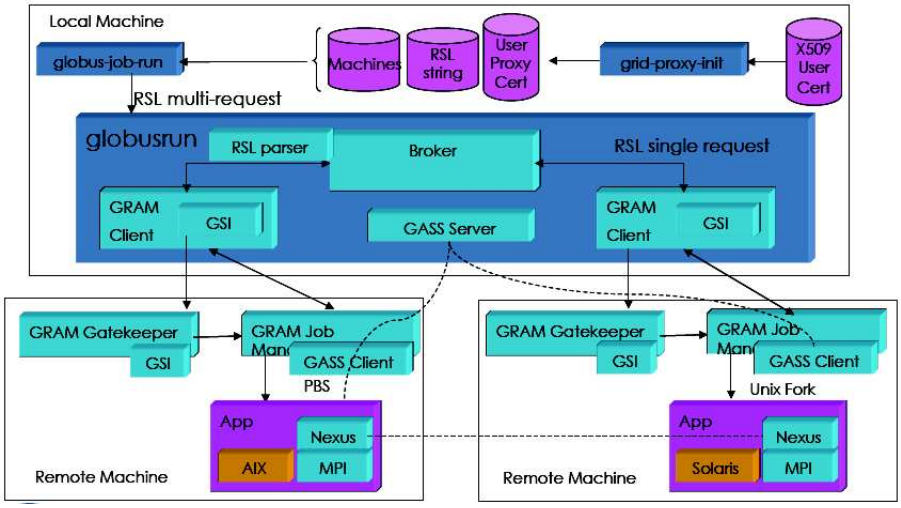


FIGURE 6.2: Globus Components in Action.

File Transfer Protocol is a simple protocol to transfer files that is able to transfer files in two ways because it is implemented using two sockets: we have a *control socket*, a control channel that connects two protocol interfaces and a *data socket*, a data channel that is responsible to tranfer the 0s and 1s of the data. On the control channel, all the FTP commands flow. For example, if I want to download from an FTP server something, I open a protocol interface on my site and I connect to the IP of the server, I indicate that I want to use this channel and how we are going to do the setup in order to perform the download. When we agree on every mechanism to connect all two data transfer ports together, we are ready to download the file.

These two socket implementations allows for a end-to-end transfer: this means that a client can control using two control channel between a server A anda server B because the two server are sharing the channel and the two server control channel are connected to the relative control channel in the client.

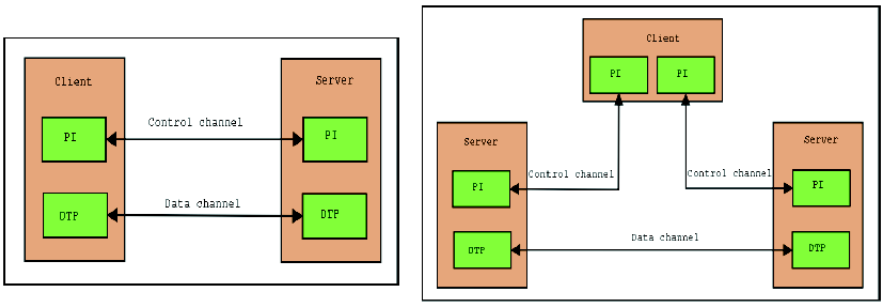


FIGURE 6.3: FTP in a nutshell.

What are the additions of GridFTP to this transfer? They chose FTP because it was very simple to modify so they implemented two additional features to provide parallel transfers instead of having just a single socket to transfer data, we open another socket in parallel and we can have *striped* transfer in the sense that we can open multiple data highways not just in one local machine but in more than one. We can have portions of a file transferred in parallel.

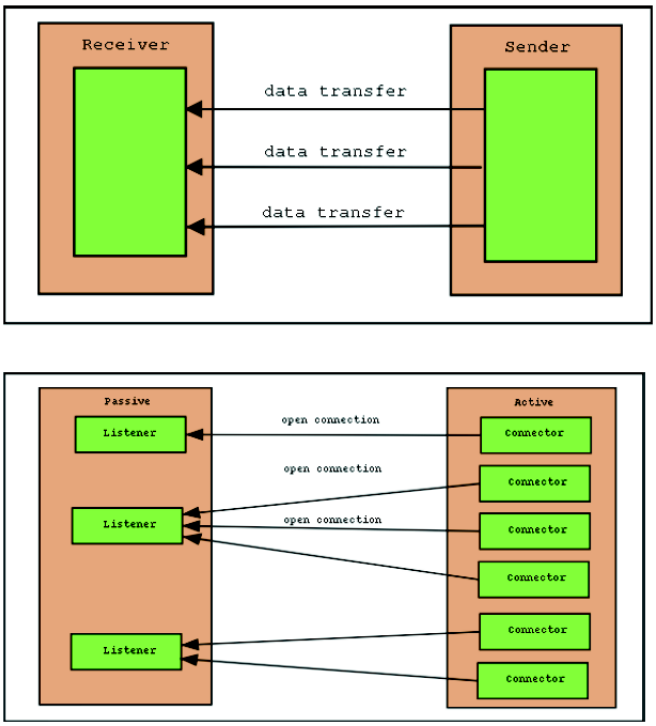


FIGURE 6.4: GridFTP Add-ons.

Why? Because at the end of the day, I am receiving the data and I have a limited bandwidth. With a single connection I can saturate this bandwidth. What is the added value of having more than one transfer? If I have two transfers and one transfer is unsing one half of the bandwidth and another transfer is using the other half, at the end the performance are identical? But this is not the way it works because FTP is built on top of TCP and TCP does not use all the bandwidth you have at disposal. It is going to give at each socket only a small portion. TCP says: ok, I'll give you a small part and some space for additional connections. Often with TCP we have glitches on the network, we are losing data and so we have to resend it back. This means that if I use just one flow if I have losses along the path, at the end if you go to calculate the data transfer rate is very low. If I have more then one, these losses will be again present but in some way they are not completely wasted because other flows that are flowing in different physical links can not incur in these losses and can send something.

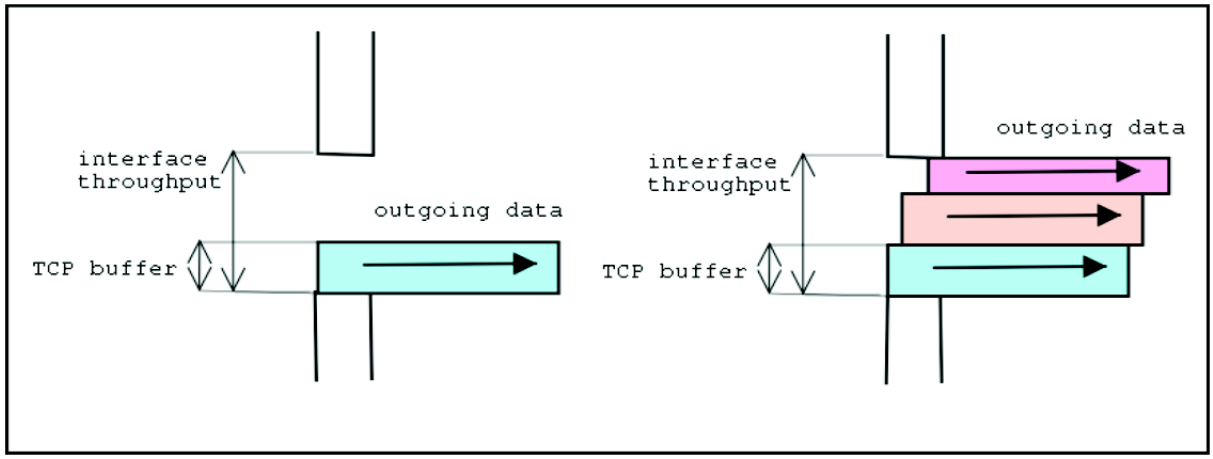


FIGURE 6.5: Network throughput maximization (TCP window).

6.4 Data Management

Data can be *present* in different sites. Data can be *replicated* in different locations. How to *manage* the copies of data on the Grid? How to leverage the copies of data?

Low level: Replica Catalog

- A catalog represents files, collections and locations
- A collection is represented by a logical file
- A replica (partial or complete) of a collection is represented by a physical file
- Given a logical filename, how to obtain the relevant physical filename?
  - Physical File Name (PFN): host + full path & file name
  - Logical File Name (LFN): logical name unique in the Grid
  - LFN : PFN = 1 : n

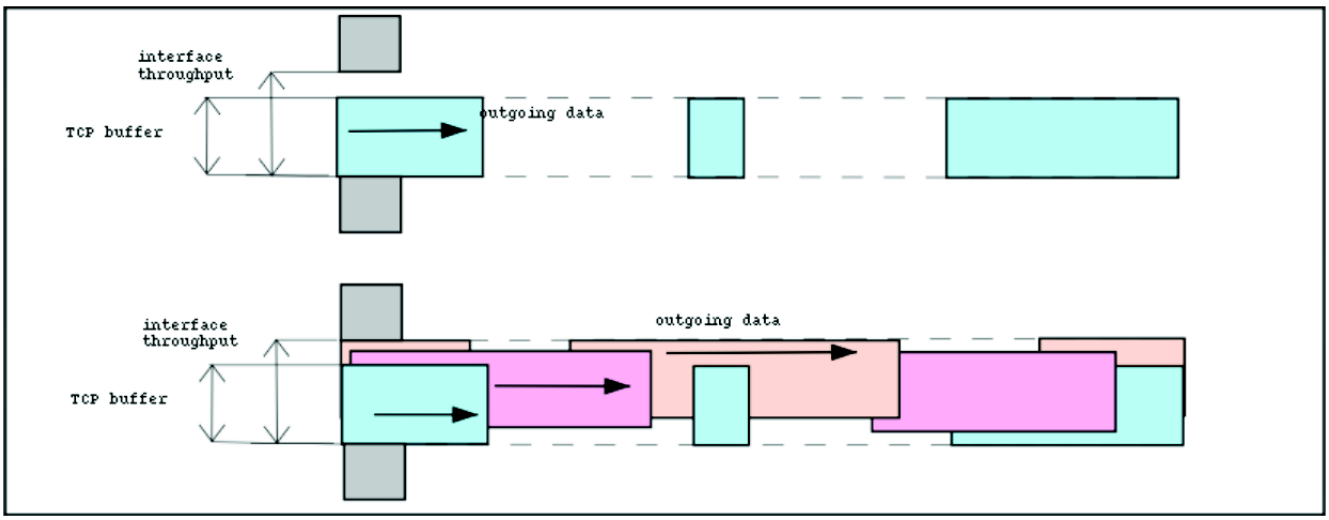


FIGURE 6.6: Network losses minimization (glitch).

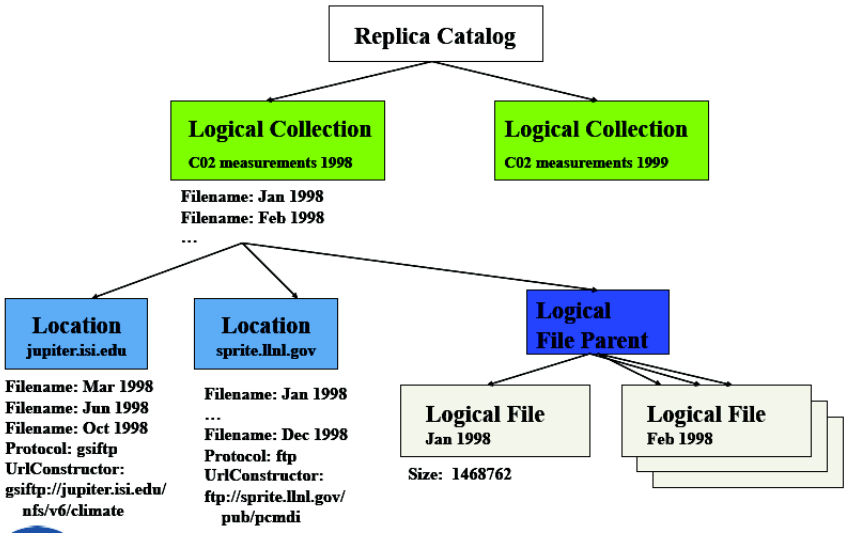


FIGURE 6.7: Replica Catalog.

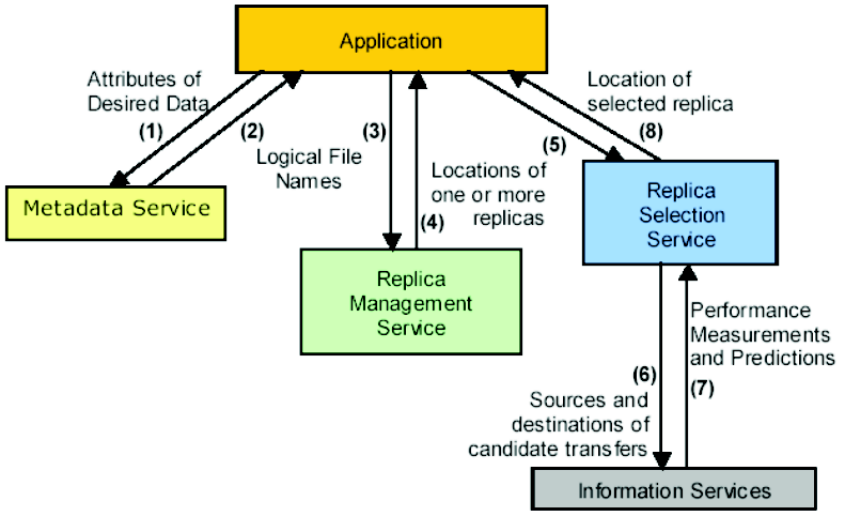


FIGURE 6.8: Replica Management Services.

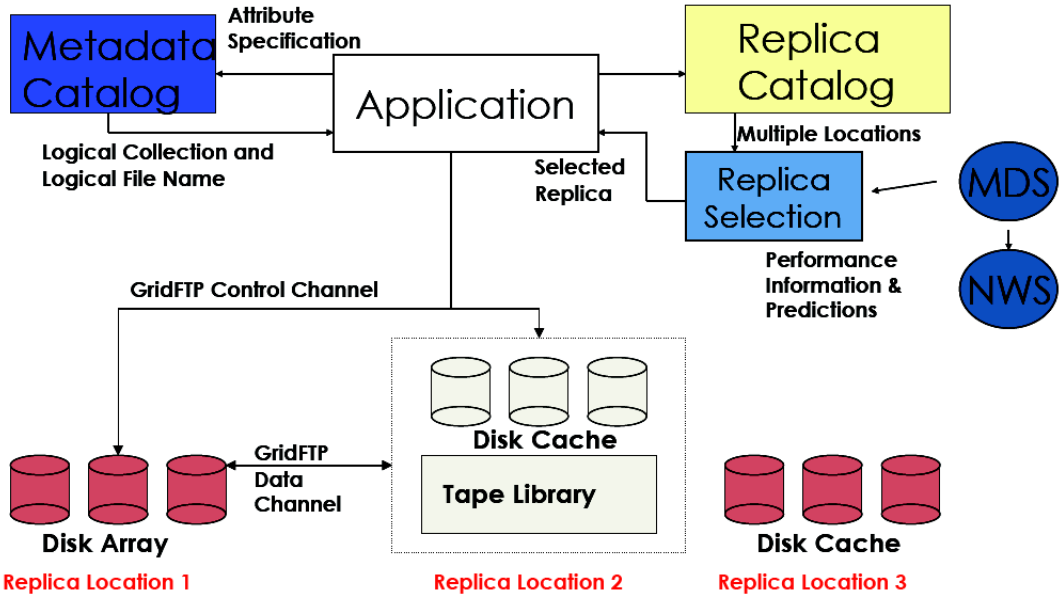


FIGURE 6.9: DataGrid: complete architecture.