

# INTRODUCTION TO GRID COMPUTING

by  
Phuong Nguyen Duc

Technical report (**DRAFT**)  
Department of Computer Science  
University of Basel  
Research group  
Prof. Dr. Helmar Burkhart

Basel, Switzerland, November 2005



## Abstract

The goal of this technical report is to study current efforts in Grid Computing. This is an interesting and promising technology. However, it is not so clear what exactly a Grid is. There are many terms such as Computational Grids, Data Grids, Science Grids, Learning Grids, Semantic Grids, Knowledge Grids, Bio Grids, Cluster Grids, Campus Grids, Commodity Grids, and so on. Ian Foster wrote in [1]

If by deploying a scheduler on my local area network I create a “Cluster Grid” then doesn’t my Network File System deployment over that same network provide me with a “Storage Grid?” Indeed, isn’t my workstation, coupling as it does processor, memory, disk, and network card, a “PC Grid?” Is there any computer system that isn’t a Grid?

The reason of the confusion is that Grid computing is still a developing technology. While many researchers share a common vision of the Grid, there are several Grid’s definitions. It is, therefore, a problem for many who want to learn and explore the potential power of Grid technology. However, attempts to define Grid precisely may excluding implementations that many would consider to be grids [1], [2]. Researcher even refine their own Grid definition over time. For examples, in 1998, Foster wrote “A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.” [1]. Two year later, he provided an updated definition “Grid computing is concerned with coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.” [3]

In this report, we will not try to give yet another Grid definition. Instead, we start with the Grid vision, what the users expect from a Grid and discuss in details what kind of components are needed in order to realize the vision. We discuss in brief other related technologies in distributed computing such as P2P, Client/Server, and the World Wide Web.

In addition, a Grid infrastructure, the Globus Toolkit [4] is introduced. The Toolkit is widely used in many Grid research projects. A very simple “Grid Service” written in Java is shown. Although, the example can not demonstrate many benefits of using Grid Service, it gives the reader a first glance at how a Grid Service implementation would look like.

## Table of Contents

Part 1 Introduction to Grid computing .....	4
1.1 Grid vision .....	4
1.2 Grid components .....	5
1.2.1 Portal .....	7
1.2.2 Information Service .....	8
1.2.3 Broker .....	8
1.2.4 Security .....	8
1.2.5 Scheduler.....	8
1.2.6 Resource management .....	9
1.2.7 Data management .....	9
1.2.8 Other .....	9
1.3 Grid classification.....	10
1.3.1 Grid uses.....	10
1.3.2 Grid characteristics .....	11
1.4 OGSA and OGSI.....	12
1.5 Relationships with other distributed system technologies .....	14
1.5.1 WWW .....	14
1.5.2 P2P.....	14
1.5.3 Middleware.....	14
1.5.4 Parallel computing .....	14
1.6 Summary .....	15
Part 2 Globus Toolkit.....	16
2.1 Introduction.....	16
2.2 Globus components .....	17
2.2.1 Information Service .....	17
2.2.2 Security .....	17
2.2.3 Resource management .....	18
2.2.4 Data management .....	18
2.2.5 Software that integrated with Globus.....	18
Appendix A Introduction to programming with Globus 3.2 using Java.....	19

## Part 1

### Introduction to Grid computing

#### 1.1 Grid vision

Many researchers agree that a Grid should be seen as a single virtual computer, as described in [5]:

“An user will have access to a **virtual computer** that is reliable and adaptable to the user’s needs. This virtual computer will consist of many **diverse computing resources**, but these individual resources will **not be visible** to the user, just as the consumer of electric power is unaware of how his electricity is being generated.”

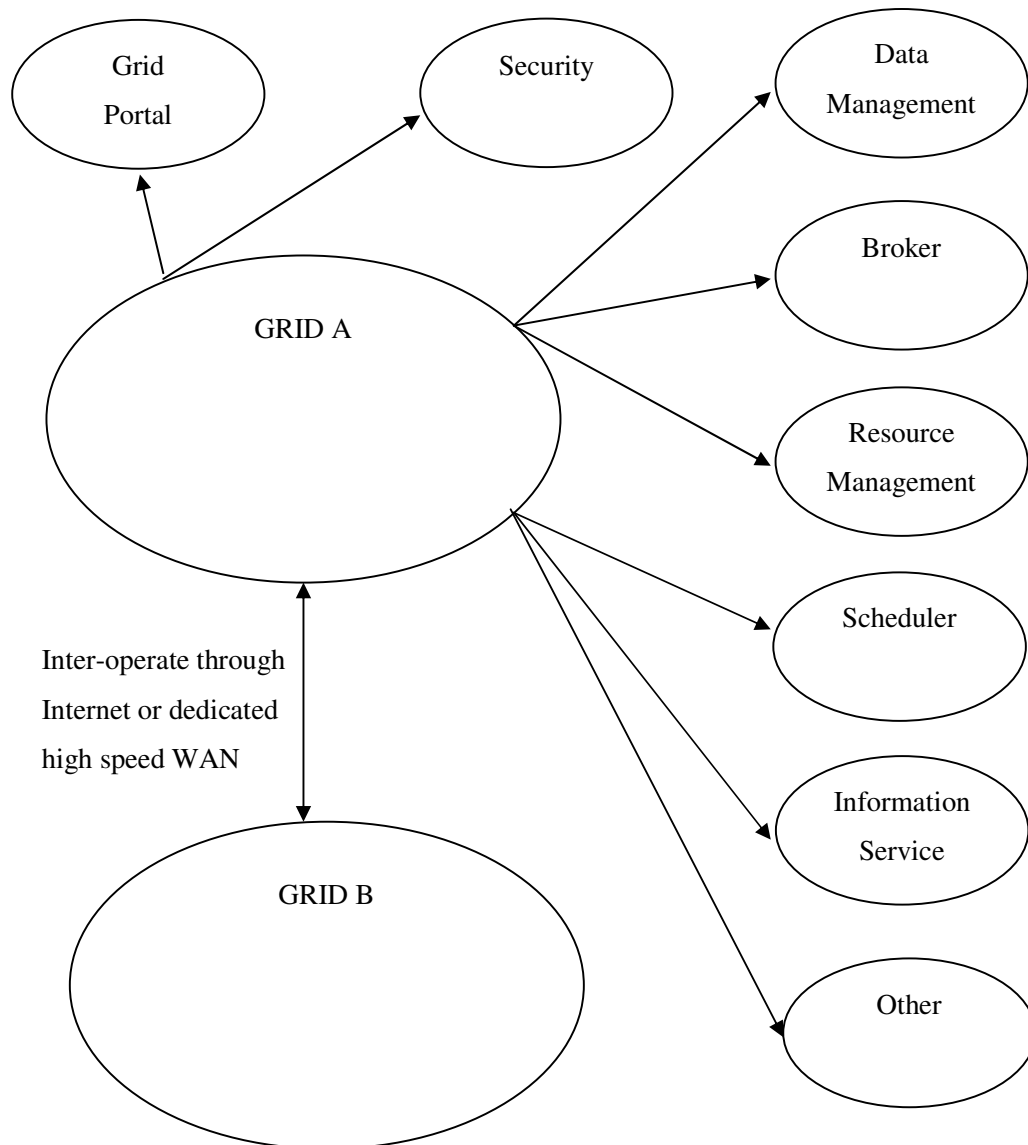
A computer is composed of hardware and software components. In Grid context, the term *resources* refers to anything that can be shared on a distributed network. Resources on a grid, for instance, can be CPUs, hard disks, memory, sensors, network bandwidth as well as operating systems, software, code libraries, and so on. Up to now, it is quite difficult to build such virtual computers that software developers can use as easy as plugging a TV into the electric socket, especially in heterogenous environment (both hardwares and software aspect). Let’s consider a detail example from [6]:

“Given an MPI code, for example, we would like to be able to just run “make”, and some command to run the job, “grid-run”. To do this today, the first step is to determine what software may need to be installed on the machine(s), installing and then checking it. The security infrastructure needs to be in place as well, which means making sure all the security credentials are setup, not only certificates, keys, etc., but accounts as well. The next step generally involves trying to get information about the available resources – which means determining where to ask, what you can ask, and what the answer means. After determining where to run a job, it must then be submitted (which may include staging files, setting up the proper environment, etc), run (preferably in a way that the progress can be monitored or adapted), and then cleaned up afterwards (including data movement and analysis). None of these are straightforward, single commands for a user to just run. **All of the pieces to do this exist today in one form or another. What doesn’t yet exist is the entire service to do this in a simple and straightforward manner.**”

From the example above, we can point out at least 7 basic components that needed in order to simplify these tasks. These components are discussed in detail in the next section.

## 1.2 Grid components

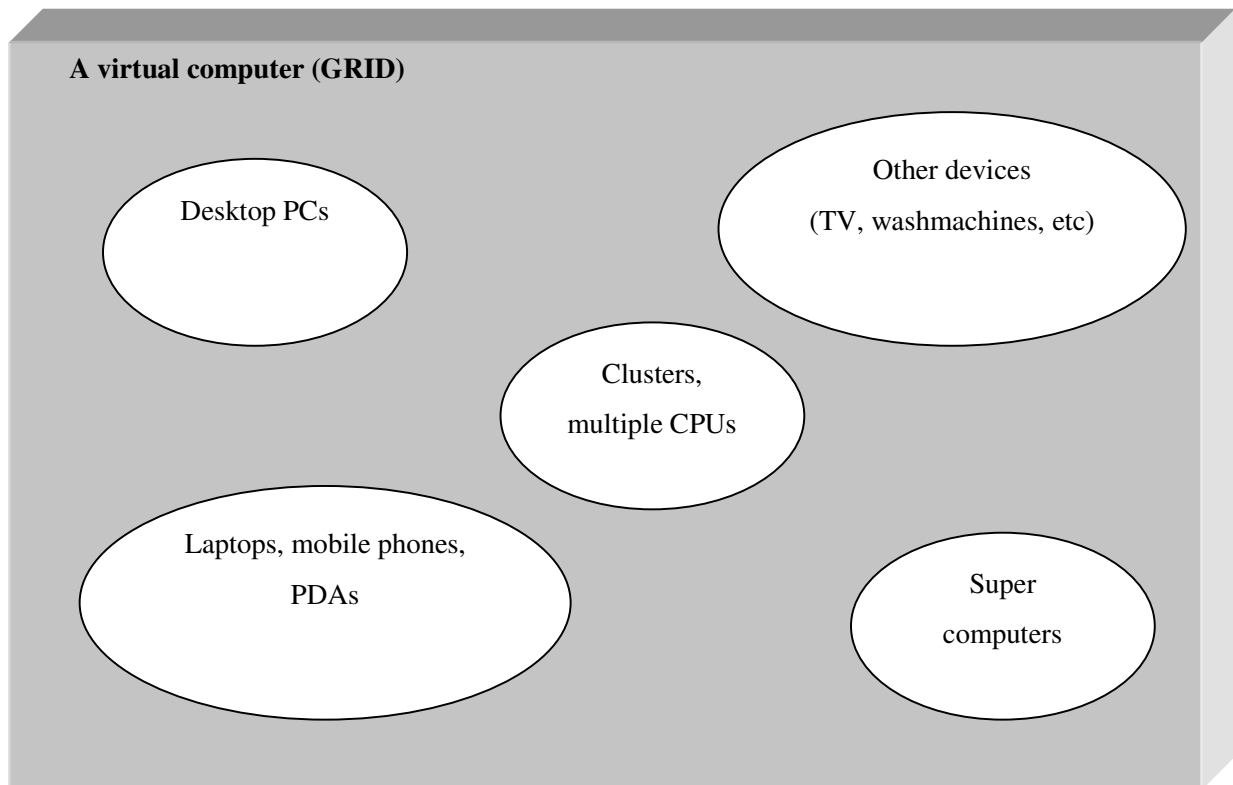
- |                        |                    |                |              |
|------------------------|--------------------|----------------|--------------|
| 1. Information Service | 2. Broker          | 3. Security    | 4. Scheduler |
| 5. Resource management | 6. Data management | 7. Grid portal | 8. Other     |



**Figure 1: Grid software components**

Figure 1 shows the basic components of a Grid. Depending on the application requirements, some of these components may or may not be required in a grid implementation.

Figure 2 shows the hardware components of a Grid. Not only super computers or high performance computers are in Grid but also PCs, laptop or even low power devices such as PDA, and mobile phones.



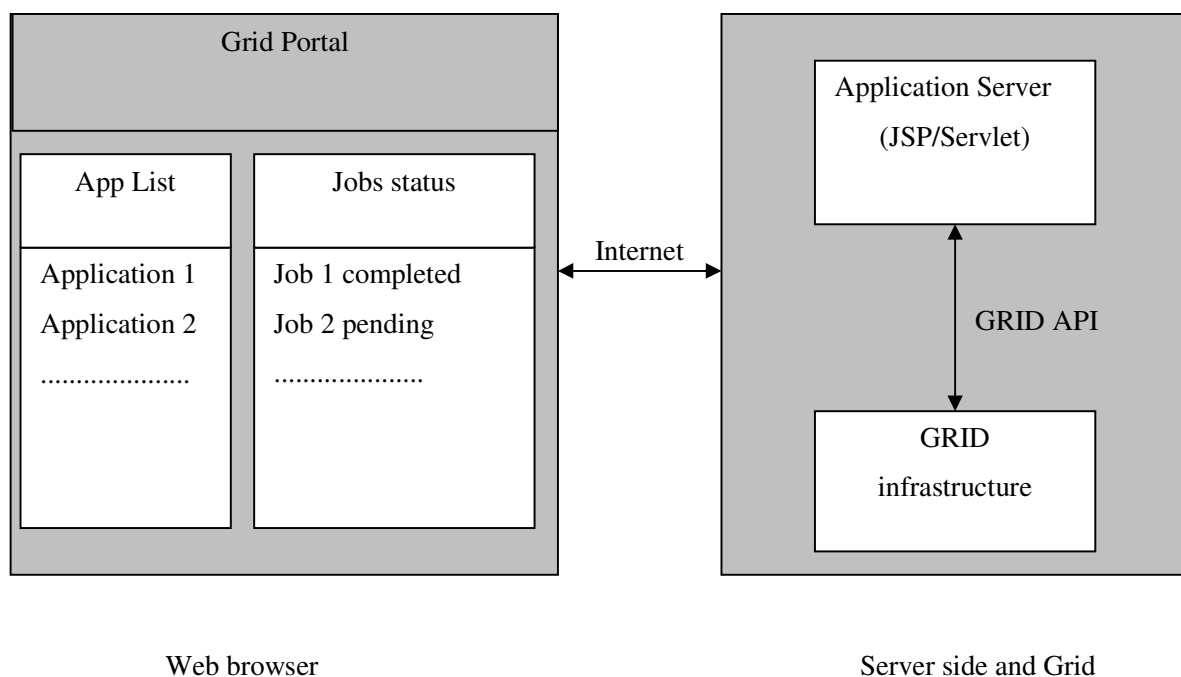
**Figure 2 Grid hardware components**

Depending on the application requirements, some of these components discussed in this section may or may not be required in a grid implementation.

### 1.2.1 Portal

A grid portal provides the interface for a user to launch applications that will utilize the resources and services provided by the grid without the need to know the complex architecture of the Grid. Examples include Grid Portal Development Kit [7], GridPort [8].

Figure 3 shows a possible implementation of a Grid portal. The users access Grid resources through web browser. Several features are provided such as browsing the list of applications on the Grid, submit jobs and monitor the job's status, etc. On the server side, the server interacts with the Grid by calling Grid APIs. Theoretically, any programming language can be used. In this example, JSP and Servlet are selected.



**Figure 3 Grid Portal**

### 1.2.2 Information Service

The Information Service component provides information about the available resources, their capacity, and current utilization. There are often repositories of local resource information. Grid-wide information service can be implemented by central server or in a decentralized way by communicating between local repository servers. The Globus Toolkit (see Part II for detail) provides implementation of this component.

### 1.2.3 Broker

The basic role of a broker is to provide match-making services between a service requester (jobs submitted for execution) and a service provider (available resources on the Grid). A request specify what are required to execute a job such as CPU power, minimum memory, disk space, etc. The broker parse these information and try to find matching resources. The broker use the Information Service to get resource information and then make decisions on resource assignments. Once some resources are selected, the information is passed to the scheduler and then to the resource manager to execute the job. If no scheduler is used, the information is passed directly to the resource manager. Examples of grid resource broker are UNICORE Resource Broker [9], Grid Service Broker [10].

### 1.2.4 Security

Security is an important component in grid computing. The security component provides security infrastructure for: single sign-on authentication, authorization, secure communication (integrity, confidentiality), delegation, etc. Generally, Grid resources are shared by many organizations and they have therefore different security policies. The grid security mechanism should be able to deal with this issue. The users should not care where the resources come from, but the Grid must satisfy all the security requirements such as confidentiality, integrity, etc. However, the application designer should take into account what happens to the data after it has arrived at its destination. For instance, if sensitive data is passed to a resource to be processed by a job and is written to the local disk in a non-encrypted format, other users or applications may have access to that data. This is a big challenge when the number of nodes are often very large and each resource may have different security policy that need to complied with. Furthermore, access to grid resources should be transparent to user. Even the job can be delegate to many intermediate nodes before it actually run on a suitable resource, only a single sign-on authentication method should be required. A common, widely-used grid security mechanisms is the Grid Security Infrastructure [3].

### 1.2.5 Scheduler

Once the resources have been identified, the next step is to schedule jobs to run on them. For a small set of nodes and users, or when jobs are to be executed with no interdependencies and all jobs are treated equally, a scheduler may not be required. However, in reality, there are a lot of jobs that should run concurrently. In addition, some jobs have higher priority than other and some jobs require

long runtime. Then a job scheduler should be used to coordinate the execution of the jobs. Jobs should be scheduled appropriately to satisfy load balancing as well as high throughput computing (maximize the amount of resources accessible to users). Example of schedulers are: Condor [11], Sun Grid Engine [12], OpenPBS [13], LoadLeveler [14]. These software provide not only scheduling functions but also act as broker and resource manager. In a grid there could be different levels of schedulers. For instance, a cluster has its own scheduler to manage the jobs to be done within the cluster. A higher level scheduler (meta scheduler, Grid scheduler) might be used to schedule tasks to be distributed to several clusters, for example, the Community Scheduler Framework [15], and the Maui Cluster Scheduler [16].

### 1.2.6 Resource management

The responsibility of resource management is to run the jobs, monitor their status, and return outputs when jobs are done. The resource manager may consult the broker about resource assignments and launch the jobs with the appropriate resources. Moreover, it must authenticate the user and check if he is allowed to access the resources before launching the job. The schedulers listed above (see Section 1.2.5) combine the functionality of resource manager, information service and resource broker. For example, the Condor system [11] has 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 provides information service. 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.

### 1.2.7 Data management

If any data (including application modules) must be moved or made accessible to the nodes where a job executes, then there needs to be a secure and reliable method for moving files and data to various nodes within the grid. Many applications require the efficient management and transfer of terabytes or petabytes of data in wide-area network. The data management component is concerned with accessing shared storage, moving data between nodes securely, reliably, and efficiently.

### 1.2.8 Other

**Inter-process communications:** An application may split itself into a large number of sub-jobs. Each of these sub-jobs is a separate job in the grid. However, the application may implement an algorithm that requires that the sub-jobs communicate some information among them. The sub-jobs need to be able to locate other specific sub-jobs, establish a communications connection with them, and send the appropriate data. The open standard Message Passing Interface (MPI) [17] is often used for this kind of communication. MPICH-G2 [18] is an implementation of MPI optimized for running on grids. It combines easy secure job startup, performance, data conversion, and multi-protocol communication. However, when communicating over wide-area networks, applications may encounter network congestion that severely impacts the performance of the application. There are many possible solutions for inter-process communication, of which MPICH-G2 is just one.

## 1.3 Grid classification

There are many Grid projects that build software tools, applications, testbeds. These precursors of Grid are used for specific cases and can not be seen as a “virtual computer” that adaptable to user’s needs in general case. However, they are the building blocks that will be used to fulfill the Grid’s vision in the future. Some literatures consider Grid as a distributed system that used to tackle complex scientific or commercial problems. This view is correct but not complete. Grid can be used also in many other kinds of applications. The current Grid’s implementations can be classified according to their uses and characteristic [19], [20].

### 1.3.1 Grid uses

1. Computational intensive support (High performance computing): allows applications to get enough computational resources in order to reduce the runtime of jobs that can not be solved or take very long time to run on a single system. This can also be achieved by Parallel or Cluster systems. The different is that instead of using highly expensive resources (parallel, multiple processors) or required of homogeneous computers (in case of Cluster), a grid implementation can make use of heterogeneous, cheap resources that are already available. However, there are challenges for the scalability of protocols and algorithms with a large number of nodes, latency-tolerant to achieving high levels of performance. Typical applications are weather forecast, simulations (car-crash, physical and chemical experiments).

2. Data intensive applications (increase data storage capability): There are applications that generate terabytes of data per day, for example LHC@home [21], DataGrid [22]. The total generated data would exceed the capacity of any centralized storage system.

3. Optimize use of resources(High throughput computing): CPUs are often in idle. Giving tasks to unused processor cycles is called CPU **scavenging**. Disk storage, when aggregate from millions users can provide a huge virtual storage system. Examples of these applications are: Monte Carlo simulations, BOINC [23] code-based projects (Seti@home; Folding@home; LHC@home, etc). Another example of high throughput computing is the aggregate of network bandwidth. If a user needs to increase his total bandwidth to the Internet (for example a Web mining search engine), the tasks can be split among grid machines that have independent Internet connection. Within an organization where the computers may shared the Internet connection, there would not very much gain in bandwidth.

4. Collaborative works across multiple virtual organizations: enabling and enhancing human to human interactions. Such applications also have characteristics of other application described above, as they enable the sharing of computational and storage resources. The challenging issues of these applications are real time requirements imposed by human perceptual capabilities and the rich variety of interactions [20]

### 1.3.2 Grid characteristics

1. Large scale: the number of resources in a grid can be range from just a few to millions.
2. Geographical distribution: resources may be located at distant places.
3. Heterogeneity: hardwares and softwares resources can be varied: different devices (super computers, PC, PDA, mobile phone, etc), different operating systems, etc.
4. Resource sharing: users and organization contribute their resources to Grid.
5. Multiple administrations: each virtual organization can establish different security policies under which their resources can be accessed and used.
6. Resource coordination: coordinate use of resources to form an aggregate power.
7. Transparent access: a grid is seen as a single virtual computer.
8. Predictable performance: A Grid should ensure several QoS requirements (e.g.: run time, availability of resources, etc). Even though, it is very difficult to achieve this character in a heterogenous environment and the lack of central control. Several fault-tolerance techniques can be used to solve this problem such as resubmit jobs, re-allocation when some resources are down during the job execution, redundant job execution (multiple copies of job can be run on different machines), etc.
9. Consistent access: standard services, protocol and interfaces to access resource from different grid's implementation.

## 1.4 OGSA and OGSF

From the previous sections, we have seen several basic components of a grid. In fact, many researchers are going to build such components. To maximize the global efforts, we have to be sure that those components can be combined to create useful systems. This point is discussed in [24] “*Key to the realization of **Grid vision** is standardization, so that the diverse components that make up a modern computing environment can be discovered, accessed, allocated, monitored, accounted for, billed for, etc., and in general managed as a **single virtual system**—even when provided by different vendors and/or operated by different organizations*”.

The Global Grid Forum (GGF) [25] is established to address this issue. It is the community of users, developers, and vendors leading the global standardization effort for grid computing. Currently, the GGF community consists of thousands of individuals in industry and research, representing over 400 organizations in more than 50 countries. The GGF mission are:

- Define grid specifications that lead to broadly adopted standards and interoperable software
- Building an international community for the exchange of ideas, experiences, requirements and best practices.

Two important documents about Grid architecture from GGF are OGSA and OGSF specifications.

### OGSA (Open Grid Service Architecture)

The purpose of OGSA is to address inter-operability between grids that may have been built using different underlying toolkits. Just as the Web need the standard HTTP protocol to work, so we must agree on standard interfaces and protocols that will be use for inter-operability between grids. OGSA is a sevice-oriented architecture. It defines the core services, their interfaces, and the semantics, behavior and interaction of these services. However, the software architecture driving the implementation of the internals of these services is not addressed in OGSA. In other words, OGSA does not specify the implementation programming model, programming language, or excution environment. The services that defined by OGSA are :

- Execution Management Services,
- Data Services,
- Resource Management Service,
- Security Services,
- Self-managed Service,
- Information Services.

Detail information about these services can be found in [24].

### **OGSI (Open Grid Service Infrastructure)**

According to the Global Grid Forum[26], a Grid service instance is “a service that conforms to a set of conventions, expressed as Web Service Definition Language (WSDL) interfaces, extensions, and behaviors, for such purposes as lifetime management, discovery of characteristics, and notification. Grid services provide for the controlled management of the distributed and often long-lived state that is commonly required in sophisticated distributed applications.”

Whereas the OGSA defines the semantics of fundamental Grid Services at a high-level of abstraction, the OGSI defines Grid services at a lower level. It specifies how clients create, discover, and interact with a Grid service instance. That is:

- How Grid service instances are named and referenced;
- The base, common interfaces (and associated behaviors) that all Grid services implement;
- The additional (optional) interfaces and behaviors associated with factories and service groups.

However, OGSI does not address how Grid services are created, managed, and destroyed within any particular hosting environment. Thus, services that conform to OGSI specification are not necessarily portable to various hosting environments, but any client program that follows the conventions can invoke any Grid service instance conforming to this specification (of course, subject to policy and compatible protocol bindings).

OGSI version 1.0 defines a component model that extends WSDL and XML Schema definition to incorporate the concepts of

- Stateful Web services,
- Extension of Web services interfaces,
- Asynchronous notification of state change,
- References to instances of services,
- Collections of service instances, and
- Service state data that augments the constraint capabilities of XML Schema definition.

## **1.5 Relationships with other distributed system technologies**

### **1.5.1 WWW**

The World Wide Web is a global information space. It works very well for information exchange, linking related information together but does not focus on the coordinated use of resources at multiple sites. In addition, the Web lacks several features that often required in Grid environment, such as single sign-on, delegation, or secure, reliable, and efficient large data movement .

### **1.5.2 P2P**

P2P networks are often characterized by a large number of peers (millions) that collaborate equally, dynamically to share data (text, audio, video, etc). Data are often replicated in a P2P system. A common P2P model is decentralized systems where nodes join and leave in a dynamic and ad hoc manner and there is no central repository. Examples of this type are Gnutella [27], Freenet [28], P-Grid[29], Chord [30], Farsite [31], Oceanstore [32]. Decentralized P2P systems trade-off short response time and expensive servers for simplicity and scalability. Only a few P2P systems have central repository, for instance Napster [33]. The disadvantages of a centralized system are the problem of scalability and single point of failure. However, these systems have faster search times and guaranteed to find all results if exists. There are also some hybrid approaches in which there is a central repositories for each small set of peers (Kazaa [34]). File sharing without access control and the freedom for a peer to act as client or server or both in P2P systems introduces security issues, for example giving wrong answer, distributing malicious code.

Contrarily, in a Grid, the sharing resources that we are concerned with are not only files but also computational resources, data storage, software, networks, and other resources. Furthermore, this resources are highly controlled by local policies.

### **1.5.3 Middleware**

Middleware technologies such as CORBA, RMI, DCOM provide remote invocation mechanisms that simplify the implementation of distributed applications by hiding the complexity network communication. However, they require centralize administration and are often used within a single organization and homogenous environment. The form of interaction is client-server, rather than the coordination of multiple resources.

### **1.5.4 Parallel computing**

The thousands of processors in a Grid provide a significant computational power. However, this does not imply that traditional parallel high-performance computers are obsolete. Many problems require low latencies and high communication bandwidths. Such requirements are, up to date, still difficult to archieve in a Grid with heterogenous hardwares and softwares components.

## 1.6 Summary

Grid computing support many features that other technologies also provide. This causes confusion to classify some systems. Systems that have client server model, sometimes, also are seen as Grids. For examples, the BOINC system [23]. It has servers that distributed jobs to millions of clients, monitor (check for errors, fake data) and integrate the results. Whereas the authors of BOINC do not consider their system as Grid [35], some projects that derived from the BOINC code-base are considered as Grid in some literatures [20].

The OGSI (see Section 1.4) define the term “Grid Services”. Grid services are Web services that conform to a specific set of conventions defined by OGSI. Many researcher would ask “Is it just a change of name from Web services to Grid services or are there any fundamental differences?”.

In summary, a Grid is a hardware and software infrastructure that expose to users as a single virtual computer. So in a Grid, there can be diverse type of hardwares (supercomputer, desktop PC, PDA, etc), diverse types of softwares (different OS, middlewares such as RMI, CORBA, Web services) but all of these complex, heterogenous resources are expose to the users simply as Grid services, and they can access them simply by using a set of standard protocols and interfaces.

An important point that has to be emphasized here is that Grid computing is a complement rather than compete with other distributed computing technologies [3]. For example, HPC systems can use Grid technology to achieve computational resource sharing across institutional boundaries easily.

At the moment, it is still quite complex to build a Grid. Expertise of toolkits, middlewares, etc are required. In the future, these requirements should diminish considerably in order for Grid technology to be widely accepted and deployed. The users should only concentrate on how to solve a problem rather than on how to map a solution onto available Grid implementations. There are several efforts to archieve this goal. The Globus Toolkit is one of these. It provides several basic components which developers can use to build a useful Grid system. The toolkit is described in brief in the next part.

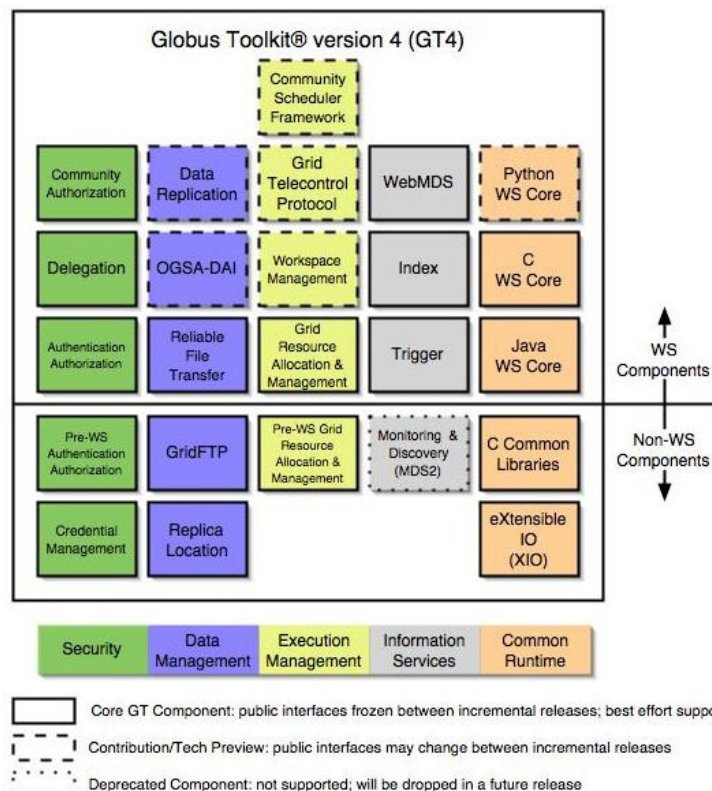
## Part 2

### Globus Toolkit

This section discuss in brief the Globus Toolkit, a Grid Toolkit that have been used in many Grid research projects. The main part of this section is related to the widely used, stable release 3.2 (7/2004). However, the recent stable release GT 4.0 (4/2005) is also mentioned because of some major changes. Particularly, GT4 uses pure Web Services instead of OGSI 1.0 Grid Services.

#### 2.1 Introduction

The Globus Toolkit is an open source toolkit used for building Grid systems and applications. It is being developed by the Globus Alliance [4] and many others all over the world. It composed of a set of services and software libraries that support Grids applications. The toolkit addresses issues of security, information discovery, resource management, data management, communication, fault detection, and portability.



**Figure 4 Globus Toolkit version 4. Picture is taken from [4]**

## 2.2 Globus components

The most important components in Globus are: Information Service, Security, Resource management (Execution Management), and Data management.

### 2.2.1 Information Service

The Monitoring and Discovery System (MDS) in Globus is the information services component. This component is used to discover the existence of resources and obtain their properties, for example, its configuration, current load, usage policy, etc.

### 2.2.2 Security

Grid computing introduces new challenging security issues. In a distributed environment with virtual organizations, diverse local security policy, it is difficult and time-consuming to manually implement secure Grid applications. However, the Globus Toolkit helps a lot to reduce these complexity by providing built-in blocks. The Grid Security Infrastructure (GSI) of Globus provide the mechanisms for authentication, authorization, and secure communication between elements in the grid. Specifically, GSI can be used for:

- Secure communication (authenticate, authorize and confidential) between elements of a Grid.
- Security across organizational boundaries, thus prohibiting a centrally-managed security system.
- Single sign-on for users of the Grid, including delegation of credentials for computations that involve multiple resources and/or sites. A Grid job often require that several Grid resources be used. There are sometimes agents requesting services on behalf of a user. Users should be able to authenticate just once and then have access to multiple resources without further user intervention. The need to re-enter the user's passphrase can be avoided by creating a proxy.

GSI uses standard security mechanisms such as public key cryptography, SSL, X.509 certificate . It is easy to install and configure GSI to work with existing tools.

**Mutual authentication:** When two nodes in the Grid want to communicate, each node authenticates with one another based on their digital certificate.

**Secure communication:** After the two nodes can be successfully authenticated, a unique session key is exchange to encrypt all session data using asymmetric encryption. Globus use standard Web Service security protocols. It allows the use of current and future Web service tools and software.

There are also an improve version SSH, called GSI-SSH: When using SSH, if the user is connecting to a new host or to a host whose host key has changed, they have no way of verifying the host key to be genuine, leaving themselves open to a man-in-the-middle attack. This problem is handled as part of the GSI-SSH protocol by hashing the sshd host key, signing the result with the GSI

host certificate on the sshd host and sending this to the client. With this information, the client now can verify that a host key belongs to the host it is connecting to and detect an attacker in the middle.

GSI component provides security functionality to other components such as Job management, data management.

### 2.2.3 Resource management

The Grid Resource Allocation and Management (GRAM) component of Globus provide interface for job submission and control. GRAM reduces the number of mechanisms required for using remote resources. Local systems may use variety of schedulers (e.g: Condor, SGE), batch systems (e.g: PBS) but users only need to use GRAM to request and use these resources.

GRAM uses RSL (Resource specification language) for job submission. All requests are described in an RSL string (XML-based) that includes information about executable file, its parameters, stdin, stdout and so on. Then GRAM parse this information and convert them into the local resource manager commands (e.g: Condor job submission file).

The secure communication between nodes is done by the gatekeeper. The gatekeeper receives the job request, authenticate the user (using GSI) before launching the job. In addition, GRAM send the job status and output to cliens, if requested.

### 2.2.4 Data management

**GridFTP** extends the traditional FTP with additional features:

- Authentication using GSI
- Allows a third party to transfer files between two servers
- Partial file transfer: transfer only a portion of a file
- Parallel data transfer (using multiple TCP streams) to improve the aggregate bandwidth
- Reliable data transfer that includes fault recovery methods
- Automatic negotiation of TCP buffer sizes both for large files and large sets of small files.

Other Globus primary tools for data management are RFT (Reliable File Transfer) service for managing multiple transfers and Replica Location Service(RLS) for maintaining location information for replicated files.

### 2.2.5 Software that integrated with Globus

From section 1.2 we see that there are some features that are not provided by Globus: Scheduler, Portal and Broker. However, several projects have been established to implements these components. For instance, [9] [10] provide resource broker that can be integrated with Globus. There are a number of schedulers that are integrated with Globus [11][12]. Also, several Grid Portal [7], [8] are available.

## Appendix A

### Introduction to programming with Globus 3.2 using Java

A very simple example written in Java is shown here. The example is based on the Globus programming Tutorial [36]. This section give the readers a first glance about how a “Grid service” look like. For detail on Grid programming with Globus, please refer to [36].

**To write and deploy a simple Grid service, there are 5 steps:**

1. Define the sevice’s interface: using GWSDL (Grid Web Service Description Language, an extension of WSDL)
2. Implement the service: Theoretically, grid service can be written in any languages. In this example, we are using Java.
3. Define the deployment parameters: using WSDD (Web Service Deployment Descriptor)
4. Compile everything and generate GAR file (Grid Archive).
5. Deploy the service.

#### 1. Step 1: Defining the interface in GWSDL

There are two options:

- + Writing the GWSDL manually
- + Generating WSDL from a Java interface

```
public interface Math
```

```
{
```

```
    public void add(int a);
```

```
}
```

```
<gwsdl:portType name="MathPortType" extends="ogsi:GridService">
```

```
    <operation name="add">
```

```
        <input message="tns:AddInputMessage"/>
```

```
        <output message="tns:AddOutputMessage"/>
```

```
        <fault name="Fault" message="ogsi:FaultMessage"/>
```

```
    </operation>
```

```
</gwsdl:portType>
```

**Note:** using `<gwsdl:portType>` instead of `<wsdl:portType>`

## 2. Step 2: Implementing the service in Java

The stubs classes are generated from the GWSDL file.

```
import org.globus.ogsa.impl.ogsi.GridServiceImpl;
import org.globus.progtutorial.stubs.MathService.MathPortType;
import java.rmi.RemoteException;

public class MathImpl extends GridServiceImpl implements MathPortType{
private int value = 0;
public void add(int a) throws RemoteException
{
    value = value + a;
}
} //end class MathImpl
```

### 3. Step 3: Configuring the deployment in WSDD

Now we make the service interface (GWSDL) and the service implementation available through a Grid Service-enabled web server.

```
<?xml version="1.0"?>
<deployment name="defaultServerConfig" xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="progtutorial/core/first/MathService" provider="Handler" style="wrapped">
    <parameter name="name" value="MathService"/>
    <parameter name="className"
value="org.globus.progtutorial.stubs.MathService.MathPortType"/>
    <parameter name="baseClassName"
value="org.globus.progtutorial.services.core.first.impl.MathImpl"/>
    <parameter name="schemaPath"
value="schema/progtutorial/MathService/Math_service.wsdl"/>
    ...
  </service>
</deployment>
```

### 4. Step 4: Compiling

This step can be done easily by using Ant to automate the following tasks:

- + Converting the GWSDL into WSDL
- + Creating the stub classes from the WSDL
- + Compiling the stub classes
- + Compiling the service implementation
- + Organizing all the files into a specific directory structure.

### 5. Step 5: Deploying the service

Copies the files (WSDL, compiled stubs, compiled implementation, WSDD) into several required locations in the Goobus Toolkit directory tree. For detail, see [36].

## 6. A simple client

```
package org.globus.progtutorial.clients.MathService;
import org.globus.progtutorial.stubs.MathService.service.MathServiceGridLocator;
import org.globus.progtutorial.stubs.MathService.MathPortType;
import java.net.URL;
public class Client
{
    public static void main(String[] args)
    {
        try
        {
            // Get command-line arguments
            URL GSH = new java.net.URL(args[0]);
            int a = Integer.parseInt(args[1]);
            // Get a reference to the MathService instance
            MathServiceGridLocator mathServiceLocator = new MathServiceGridLocator();
            MathPortType math = mathServiceLocator.getMathServicePort(GSH);
            // Call remote method 'add'
            math.add(a);
            System.out.println("Added " + a);
            // Get current value through remote method 'getValue'
            int value = math.getValue();
            System.out.println("Current value: " + value);
        }catch(Exception e){
            System.out.println("ERROR!"); e.printStackTrace();
        }
    }
} //end main
} //end class Client
```

## 7. Run the example

1. In the server side: **globus-start-container**

If the service is correct deployed, we will see something similar to:

<http://localhost:8080/ogsa/services/>

2. Start the client: after setting the CLASSPATH

**java org.globus.progtutorial.clients.MathService.Client \**

**http://127.0.0.1:8080/ogsa/services/progtutorial/core/first/MathService 5**

If all goes well, you should see the following:

Added 5

Current value: 5

Run the command again and you should see:

Added 5

Current value: 10

There are many other interesting topics in Grid programming with Globus, such as: Notifications, Service Data (structured collection of information about Grid Service, used for service discovery), Callback, Security, Reliable data transfer, etc.

## 8. Summary

The simple example doesn't show much the power of Grid. It even look very much like writing a normal Web Service. The different is that when we deploy the Grid service using Globus, it is easy to enable security, index search (information service), notification, stateful service, and so on. One important point is that Grid technology is still developing so there are still much works for programmers. But in the future, writing Grid service should be much easier.

## Bibliography

- [1]. Ian Foster. "What is the Grid? A Three point checklist". July 22, 2002: Vol. 1 No. 6.
- [2]. Links to debates on Foster article about Grid definition: Available online at <http://users.cs.cf.ac.uk/J.P.Giddy/debate.html> (Last visited on 03/11/2005)
- [3]. Ian Foster, Carl Kesselman, and Steve Tuecke, "The Anatomy of the Grid: Enabling Scalable virtual Organizations", Intl Journal Supercomputer Applications, 2001.
- [4]. Globus Toolkit: <http://globus.org/toolkit/>. (Last visited on 03/11/2005)
- [5]. Bart Jacob, IBM DeveloperWorks, "Grid computing: What are the key components?"
- [6]. J.M. Schopf and B. Nitzberg, "Grids: Top Ten Questions", Scientific Programming, special issue on Grid Computing, 10(2):103 - 111, August 2002.
- [7]. Grid Portal Development Kit (GPDK): Available online at <http://doesciencegrid.org/projects/GPDK/> (Last visited on 03/11/2005)
- [8]. Grid Portal Toolkit: <http://gridport.net/index.cgi/> (Last visited on 03/11/2005)
- [9]. UNICORE Resource Broker: <http://uombroker.sourceforge.net/> (Last visited on 03/11/2005)
- [10]. Grid Service Broker: <http://www.gridbus.org/broker/> (Last visited on 03/11/2005)
- [11]. Condor project: <http://www.cs.wisc.edu/condor/> (Last visited on 03/11/2005)
- [12]. Sun Grid Engine: <http://gridengine.sunsource.net/> (Last visited on 03/11/2005)
- [13]. Portable Batch system: <http://www.openpbs.org/> (Last visited on 03/11/2005)
- [14]. IBM LoadLeveler: (Last visited on 03/11/2005)  
<http://www-128.ibm.com/developerworks/linux/library/l-halinux3/?ca=dgr-lnxw41HAMPLP3>
- [15]. Community Scheduler Framework (CSF): <http://sourceforge.net/projects/gcsf/>  
(Last visited on 03/11/2005)
- [16]. Maui Cluster Scheduler: <http://www.clusterresources.com/products/maui/>  
(Last visited on 03/11/2005)
- [17]. Message Passing Interface: <http://www-unix.mcs.anl.gov/mpi/> (Last visited on 03/11/2005)
- [18]. MPICH-G2, a grid-enabled implementation of the MPI v1.1 standard:  
<http://www.hpclab.niu.edu/mpi/> (Last visited on 03/11/2005)
- [19]. Miguel L. Bote-Lorenzo, Yannis A. Dimitriadis, Eduardo Gómez-Sánchez, "Grid Characteristics and Uses: A Grid Definition." European Across Grids Conference 2003: 291-298
- [20]. Ian Foster and Carl Kesselman, Computational Grids, Chapter 2 of "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann Publishers, 1999. ISBN: 1558604758

- [21]. LHC@Home: <http://lhcatome.cern.ch/> (Last visited on 03/11/2005)
- [22]. DataGrid: <http://eu-datagrid.web.cern.ch/eu-datagrid/> (Last visited on 03/11/2005)
- [23]. BOINC project, Berkeley Open Infrastructure for Network Computing:  
<http://boinc.berkeley.edu> (Last visited on 03/11/2005)
- [24]. OGSA specification: The Open Grid Services Architecture, Version 1.0, 29/01/2005  
<http://www.ggf.org/documents/GFD.30.pdf> (Last visited on 03/11/2005)
- [25]. The Global Grid Forum: <http://www.ggf.org/index.php> (Last visited on 03/11/2005)
- [26]. Open Grid Services Infrastructure specification, Version 1.0, 27/06/2003  
<http://www.ggf.org/documents/GFD.15.pdf> (Last visited on 03/11/2005)
- [27]. Gnutella, P2P system: <http://www.gnutella.com> (Last visited on 03/11/2005)
- [28]. Freenet, P2P system: <http://freenet.sourceforge.net/> (Last visited on 03/11/2005)
- [29]. P-Grid, P2P system: <http://www.p-grid.org/> (Last visited on 03/11/2005)
- [30]. The Cord project, P2P system: [www.pdos.lcs.mit.edu/chord](http://www.pdos.lcs.mit.edu/chord) (Last visited on 03/11/2005)
- [31]. Farsite, a serverless, distributed file system that does not assume mutual trust:  
<http://research.microsoft.com/sn/Farsite/> (Last visited on 03/11/2005)
- [32]. OceanStore, a global persistent data store:  
<http://oceanstore.cs.berkeley.edu/> (Last visited on 03/11/2005)
- [33]. Napster, P2P system: <http://www.napster.com/> (Last visited on 03/11/2005)
- [34]. Kazaa, P2P system: <http://www.kazaa.com/us/index.htm> (Last visited on 03/11/2005)
- [35]. David P. Anderson, "BOINC: A system for Public resource computing and storage", 5th IEEE/ACM International Workshop on Grid Computing, November 8, 2004, Pittsburgh, USA
- [36]. Borja Sotomayor, "The Globus Toolkit 4 Programmer's Tutorial"  
<http://gdp.globus.org/gt4-tutorial/> (Last visited on 03/11/2005)
- [37]. I. Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration.", 2002.
- [38]. CERN GridCafé: <http://gridcafe.web.cern.ch/gridcafe/> (Last visited on 03/11/2005)
- [39]. IBM redbook, "Introduction to Grid Computing with Globus"  
<http://www.redbooks.ibm.com/abstracts/sg246895.html?Open> (Last visited on 03/11/2005)
- [40]. IBM redbook, "Globus Toolkit 3.0 Quick Start"  
<http://www.redbooks.ibm.com/abstracts/redp3697.html?Open> (Last visited on 03/11/2005)