Chapter 4

Implementation and Analysis of Grid Platforms

4.1 Implementation of Grid Environments

In the course of our research for grid implementation we experiment with available implementations. This chapter reviews the implementations and presents the observations of execution of sample applications on these grid platforms. We first present the Globus Toolkit [53] for grid implementation. sections explain the architecture, components and video streaming example on Globus. In the next section we give implementation and observation details for Java Parallel Processing Framework(JPPF) [65] for an application of dense matrix multiplication on top of it.

4.2 Globus Implementation

This chapter presents some of the main components of the Globus Toolkit 2.2, which provides:

- Single sign-on, authorization, and authentication
- Job submission
- Resource monitoring, searching, and allocation
- Data movement Also, the Globus Toolkit provides a set of tools for application programming (APIs) and system development kits (SDKs).
4.2.1 Three pyramids

Globus Toolkit has three pyramids as shown in Figure 4.1 of support built on top of a security infrastructure, as illustrated in They are:

- Resource management
- Data management
- Information services

All of these pyramids are built on top of the underlying Grid Security Infrastructure (GSI). This provides security functions, including single/mutual authentication, confidential communication, authorization, and delegation.

- **Resource management**: The resource management pyramid provides support for:
  - Resource allocation
– Submitting jobs
– Remotely running executable files and receiving results
– Managing job status and progress

Globus Toolkit does not have its own job scheduler to find available resources and automatically send jobs to suitable machines. Instead, it provides the tools and interfaces needed to implement schedulers and is often used with third-party schedulers.

- **Information services** The information services pyramid provides support for collecting information in the grid and for querying this information, based on the Lightweight Directory Access Protocol (LDAP).

- **Data management** The data management pyramid provides support to transfer files among machines in the grid and for the management of these transfers

### Open Standards

Globus Toolkit is an open standard software developed and blueprinted by the Globus Project. In addition to the Globus Project, the Global Grid Forum has contributed a collection of references and standards. GGF lays the implementation standards for Grid products and is accepted world over.

#### 4.2.2 Components of Globus Toolkit

For each pyramid previously presented, Globus provides a component to implement resource management, data management, and information services, as illustrated in Figure 4.2 The components are:

- **GRAM/GASS** The primary components of the resource management pyramid are the Grid Resource Allocation Manager (GRAM) and the Global Access to Secondary Storage (GASS).

- **MDS (GRIS/GIIS)**

  Based on the Lightweight Directory Access Protocol (LDAP), the Grid Resource Information Service (GRIS) and Grid Index Information Service (GIIS) components can be configured in a hierarchy to collect the information and distribute it. These two services
Figure 4.2: The system overview of Globus Toolkit

are called the Monitoring and Discovery Service (MDS). The information collected can be static information about the machines as well as dynamic information showing current CPU or disk activity. A rich set of information providers is included with the Toolkit and the Globus users can add their own. The information provides an interface with the GRIS, which reports this information to a hierarchy of GIIS servers in the grid. The LDAP query language is used to retrieve the desired information.

- GridFTP GridFTP is a key component for the secure and high-performance data transfer. The Globus Replica Catalog and Management is used to register and manage both complete and partial copies of data sets. These three pyramids are modularized and can function in isolation; however, together, they complement each other.

- GSI All of the above components are built on top of the underlying Grid Security Infrastructure (GSI). This provides security functions including single/mutual authentication, confidential communication, authorization, and delegation.

### 4.2.3 Grid Security Infrastructure (GSI)

GSI provides elements for secure authentication and communication in a grid. The infrastructure is based on the SSL protocol (Secure Socket Layer), public key encryption, and x.509 certificates. For a single sign-on, Globus add some extensions on GSI. It is based on the Generic Security Service API, which is a standard API promoted by the Internet Engineering Task Force (IETF). These are the main functions implemented by GSI:
4.2.4 Grid Resource Allocation Manager (GRAM)

GRAM is the module that provides the remote execution and status management of the execution. When a job is submitted by a client, the request is sent to the remote host and handled by the gatekeeper daemon located in the remote host. Then the gatekeeper creates a job manager to start and monitor the job. When the job is finished, the job manager sends the status information back to the client and terminates. Figure 4.3 depicts the conceptual view about GRAM. It contains the following elements:

- globusrun: The globusrun command submits and manages remote jobs and is used by almost all GRAM client tools. This command provides the following functions:

  - Request of job submission to remote machines: Job submission uses security functions (such as GSS-API) to check mutual authentication between clients and servers, and also to verify the rights to submit the job.

  - Transfer the executable files and the resulting job-submission output files: The globusrun command can get the standard output of job results from remote machines. It uses GASS to provide the secure file transfer between grid machines.
• Resource Specification Language (RSL):

RSL is the language used by the clients to submit a job. All job submission requests are described in RSL, including the executable file and condition on which it must be executed. You can specify, for example, the amount of memory needed to execute a job in a remote machine.

• The gatekeeper daemon:

The gatekeeper daemon builds the secure communication between clients and servers. The gatekeeper daemon is similar to inetd daemon in terms of functionality. However, gatekeeper provides a secure communication. It communicates with the GRAM client (globusrun) and authenticates the right to submit jobs. After authentication, gatekeeper forks and creates a job manager delegating the authority to communicate with clients.

• The job manager:

Job manager is created by the gatekeeper daemon as part of the job requesting process. It provides the interfaces that control the allocation of each local resource manager, such as a job scheduler like PBS, LSF, or LoadLeveler. The job manager functions are:

- Parse the resource language Breaks down the RSL scripts.
- Allocate job requests to the local resource managers The local resource manager is usually a job scheduler like PBS, LSF, or LoadLeveler. The resource manager interface is written in the Perl language, which easily allows you to create a new job manager to the local resource manager, if necessary.
- Send callbacks to clients, if necessary
- Receive the status and cancel requests from clients
- Send output results to clients using GASS, if requested

• Global Access to Secondary Storage (GASS): GRAM uses GASS for providing the mechanism to transfer the output file from servers to clients. Some APIs are provided under the GSI protocol to furnish secure transfers. This mechanism is used by the globusrun command, gatekeeper, and job manager.

• Dynamically-Updated Request Online Coallocator (DUROC) By using the DUROC mechanism, users are able to submit jobs to different job managers at different hosts or to dif-
different job managers at the same host (see Figure 4.4). The RSL script that contains the
DUROC syntax is parsed at the GRAM client and allocated to different job managers.
The grammar and attributes of RSL and DUROC are explained in Resource Specification
Language (RSL).

**Monitoring and Discovery Service (MDS)**

MDS provides access to static and dynamic information of resources. Basically, it contains the
following components:

- **Grid Resource Information Service (GRIS):** GRIS is the repository of local resource in-
formation derived from information providers. GRIS is able to register its information
with a GIIS, but GRIS itself does not receive registration requests. The local information
maintained by GRIS is updated when requested, and cached for a period of time known
as the time-to-live (TTL). If no request for the information is received by GRIS, the in-
formation will time out and be deleted. If a later request for the information is received,
GRIS will call the relevant information provider(s) to retrieve the latest information.

- **Grid Index Information Service (GIIS):** GIIS is the repository that contains indexes of
resource information registered by the GRIS and other GIISs. It can be seen as a grid
wide information server. GIIS has a hierarchical mechanism, like DNS, and each GIIS
has its own name. This means client users can specify the name of a GIIS node to search
for information.

- **Information Provider IP:** The information providers translate the properties and status of
local resources to the format defined in the schema and configuration files. In order to add

Figure 4.4: The overview DUROC
your own resource to be used by MDS, you must create specific information providers to translate the properties and status to GRIS.

- **MDS client**: The MDS client is based on the LDAP client command, ldapsearch. A search for a resource information that you want in your grid environment is initially performed by the MDS client.

- **Hierarchical MDS**: The MDS hierarchy mechanism is similar to the one used in DNS. GRIS and GIIS, at lower layers of the hierarchy, register with the GIIS at upper layers. Clients can query the GIIS for any information about resources that build a grid environment.

Figure 4.5 represents the conceptual view interconnection of the MDS components. As illustrated, the resource information is obtained by the information provider and it is passed to GRIS. GRIS registers its local information with the GIIS, which also registers with another GIIS, and so on. MDS clients can get the resource information directly from GRIS (for local resources) and/or a GIIS (for grid-wide resources). The MDS uses LDAP, which provides the decentralized maintenance of resource information.

- **Resource information**: Resource information contains the objects managed by MDS,
which represent components resources (static and dynamic) as follows:

- Infrastructure components For example, name of the job manager or name of the running job
- Computer resources For example, network interface, IP address, or memory size

### 4.2.5 GridFTP

GridFTP provides a secure and reliable data transfer among grid nodes. The word GridFTP can referred to a protocol, a server, or a set of tools. GridFTP protocol GridFTP is a protocol intended to be used in all data transfers on the grid. It is based on FTP, but extends the standard protocol with facilities such as multistreamed transfer, auto-tuning, and Globus based security. This protocol is still in draft level, so for more information, please refer to the following Web site (you must have Adobe Acrobat Reader to view the document): As the GridFTP protocol is still not completely defined, Globus Toolkit does not support the entire set of the protocol features currently presented. A set of GridFTP tools is distributed by Globus as additional packages. Globus Project has selected some features and extensions defined already in IETF RFCs and added a few additional features to meet requirements from current data grid projects.

**GridFTP server and client** Globus Toolkit provides the GridFTP server and GridFTP client, which are implemented by the in.ftpd daemon and by the globus-url-copy command, respectively. They support most of the features defined on the GridFTP protocol. The GridFTP server and client support two types of file transfer: standard and third-party. The standard file transfer is where a client sends the local file to the remote machine, which runs the FTP server. An overview is shown in Figure 4.2. Third-party file transfer is where there is a large file in remote storage and the client wants to copy it to another remote server, as illustrated in Figure 4.5.

**GridFTP tools** Globus Toolkit provides a set of tools to support GridFTP type of data transfers. The gsi-ncftp package is one of the tools used to communicate with the GridFTP Server. This package is available at the following site: The GASS API package is also part of the GridFTP tools. It is used by the GRAM to transfer the output file from servers to clients.
4.2.6 API and SDK:

Two other components are available to help develop Globus related grid applications are APIs and Developers toolkit.

Globus Toolkit APIs are basically implemented in the C language. Developer’s kit Globus provides a rapid development kit known as Commodity Grid(CoG), which supports the following technologies:

- Java
- Python
- Web services
- CORBA
- Java Server Pages
- Perl
- Matlab

4.3 The Globus Environment-Setup

4.3.1 Grid Environment

Figure introduces a conceptual grid environment presented after a Globus Toolkit installation. In this environment, there are three servers: demoCA, A and host B. demoCA is the simple Certificate Authority. Host A and host B are the grid nodes. The users' names are different on host A and host B, but they share the same grid user ID, which is known as the Distinguished Name.

4.3.2 GRAM

Here we present two advanced configurations, adding a new job manager and adding and enabling a GRAM reporter. Adding a job manager The Globus Toolkit provides a job manager, in addition to the default fork job manager, to help you expand the job management capability. The job manager supports job submissions to some third-party vendors' job schedulers, such as PBS, LSF, Condor, and LoadLeveler. The job manager is created by the gatekeeper to satisfy
every request submitted to the gatekeeper. By default, the job manager is the process that executes the job request from the gatekeeper as a forked child process. As presented in Figure 4.7, the job manager for the PBS job scheduler has been chosen. PBS is installed on host B and host C and it contains three daemons:

- pbs-mom: Execution daemon
- pbs-sched: Scheduler daemon
- pbs-server: Management daemon

The pbs-mom daemon is needed on each system where jobs are expected to run (on host B and host C). The pbs-sched and pbs-server daemons are needed on the system where the job manager and job queues exist (on host B only).

When a client (on host A) submits a PBS job to a gatekeeper (on host B), the job manager for PBS runs and then PBS sends the job to executed in another machine (host C), where Globus Toolkit is not installed.

The GRAM reporter provides status information for the job manager. The GRAM reporter for the forked job manager is contained in the server bundle. This section describes the activation of GRAM reporter for the default job manager and for the PBS job manager. The GRAM Reporter publishes the job manager status information into MDS. To enable it, you need to install the globus-gram-reporter package, plus one job manager-specific setup package.

### 4.3.3 MDS

MDS is based on OpenLDAP, allows to created your own configuration of hierarchical GIIS. For more informations regarding MDS and LDAP customization. Configuration of hierarchi-
cal GIIS MDS supports a hierarchical structure for GIIS similar to the Domain Name Servers hierarchy. In Figure 4.7, we present an overview of a hierarchical GIIS. The boxes represent the configuration files and the ovals represent the programs and daemons that update the resource information.

4.3.4 Application Execution

As shown in Figure 4.9, the video conversion application shows a potentially practical use of the grid as well as to give a feel for the problems that may be encountered with data intensive applications. The implementation takes a home video tape and converts it to a Video CD that can be played on DVD players supporting this format. Commercial programs exist for this application, but they use a single system for the conversion. The compression of the raw captured video data into an MPEG-1 or MPEG-2 data stream takes an enormous amount of time, which increases with higher quality conversions. Depending on the quality level of the video capture, the data required for a typical one hour tape can create over 10 GB of video data, which needs to be compressed to approximately 650 MB to fit on a Video CD. The compression stage is CPU intensive, since it matches all parts of adjacent video frames looking for similar sub-pictures, and then creates an MPEG data stream encoding the frames. The MPEG data stream is an instruction stream on how to reconstruct the video frames from prior (and sometimes future) video frames and compressed data that represent differences in portions of the images. The audio compression also is performed. Application writes the data in a standard format to a CD-R or writable DVD disk and is then played on DVD players or other media players supporting that
format. At higher quality levels, more data capture takes place and we require enhanced algorithms, which consume more time. The compression process takes a day or more, depending on the quality level and the speed of the system being used. For commercial DVD quality, conversions are typically done by a service company that has developed higher quality conversion algorithms. Such conversions may take weeks.

Hence, grid technology is ideal for improving the process of video conversion. Our application uses a Linux system to capture the video from a FireWire enabled camcorder or VCR that is used to play the video tape to be converted. Then we split captured video file into a number of smaller segments. Globus sends these segments to Linux grid systems for compression. After the reassembling of compressed segments, CD is written in the VCD format. This process can be seen in Figure. Currently, the compression programs run only on Linux x86 systems. The process can be modified to handle more hours of video, and use different levels of quality with different media formats. However, we limit the video for 1 hour, which fits on one CD written in the VCD format. Sending many gigabytes of data from one computer to another takes a considerable amount of time, even with a 100 Mb Ethernet connection. To reduce required time, we can store file on a system other than the one running the conversion for that segment. When a file is copied into a distributed file system. This can cause yet another large transfer of the data to get it to the system doing the conversion. Thus, for grid programs processing large amounts of data, it is imperative to understand the network topology and to have the means of keeping

Figure 4.8: Overview of GIIS structure and configuration files
the data near the processing node that is using it.

To improve this grid application, we can install more network cards in the grid machines that split the video data. These machines would be arranged in a hierarchical network topology. With these, it is possible to split the captured video into fewer pieces initially, and have other grid machines split the files further and distribute them to several machines in parallel. Another optimization would be to split the data while it is being captured.

There are four main parts to the installation of the video conversion demo:

- Globus Toolkit
- Capture software
- Conversion software
- VideoCD creation software

Capture Software There are several packages that are needed to capture a video using a FireWire card and a FireWire video source. Libraw1394 is used to access the IEEE 1394 bus directly. The libdv libraries, also known as the Quasar DV codec, are used to decode video in the DV format. DV video is defined by the IEC61384 and SMPTE314M standards and is most commonly used in digital camcorders. The libavc1394 libraries are needed for utilities to be able to control IEEE-1394 devices. The dvgrab package is used to capture DV input into AVI files. Video capture process is shown in Figure 4.10.

Video Conversion Software we require a package that must be installed on each of the server machines to perform the video conversion which consists of the mjpeeg tools. These mjpeeg tools contains the commands necessary to do the mpeg compression. The lav2wav command is used to extract the audio from an captured AVI file. The flow of commands for the video conversion is illustrated in Figure 4.11. We install the mjpeeg tools on all of the servers.

VideoCD Creation Software The final two packages are used to create the VideoCD:

- vcdimager
- cdrdao

as shown in Figure 4.12, The vcdimager command converts a video in MPEG format to a VCD image, in a BIN/CUE format suitable for use by the cdrdao CD-R writing program.
Figure 4.9: Video Streaming Application on grid

Figure 4.10: Video Capture Process
Once all of the above steps complete and the necessary scripts have been written, we start conversion process. The DV Camera is connected to the capture machine via FireWire and manually started. As soon as the video playback starts, the main module starts the process which in turn starts the video capture process and all of the other conversion steps. This application creates four video segments, each one minute long, and sends them to each of the four sever machines for conversion. The resulting VCD can be played in DVD players that support the VCD format. The videocap.mpg file can also be played directly using a media player on a personal computer.

To improve the performance of the video conversion application we incorporate certain changes in processing. One of these changes is to begin the video file transfer during the capture, as shown figure. Since dvgrab creates multiple video files while capturing the video, this improvement can be easily scripted. Once the first video is completed, it is staged to the remote machine and the conversion can begin. This process repeats to capture each individual AVI file. Furthermore, using MDS, we locate a machine on the grid that is not very busy and send the video file to it. In this way, we use a variable number of machines.
Other enhancements is to automatically generate a second VCD when the video is longer than 60 minutes. Modification in MDS locates only x86 Linux machines that have the mjpeg-tools installed, so that these procedures could be used in a larger grid. Figure 4.13 shows the parallel data transfer operations.

4.4 JPPF Grid Implementation

A Java Parallel Processing Framework (JPPF) [65] grid is composed of many distributed components interacting with each other, often in different environments. JPPF is platform independent and works on most of the environments. Sun has proposed this framework as java grid which can be used to simulate parallel execution of application. We have experimented with JPPF and present the finding in this section. JPPF enables applications with large processing power requirements to be run on any number of computers, in order to dramatically reduce their processing time. This is done by splitting an application into smaller parts that can be executed simultaneously on different machines.
Salient Features of JPPF

- A JPPF grid can be up and running in minutes
- dynamically scalable on-demand
- connectors with leading J2EE application servers
- connector with GigaSpaces eXtreme Application Platform
- easy programming model
- fine-grained monitoring and administration
- fault-tolerance and self-repair capabilities
- exceptional level of service and reliability
- fully documented samples, using JPPF on real-life problems flexible open-source licensing with Apache 2.0

4.4.1 Architecture

In the diagram, we can see the framework has a 3-tiered architecture, made of 3 distinct layers:

- Client layer: This provides an API and communication tools to use the framework to submit tasks, to execute in parallel.

- service layer: This Responsible for the communication between with the clients and the nodes, along with the management of the execution queue, the load-balancing and recovery features, and the dynamic loading of both framework and application classes onto the appropriate nodes. ”

- execution layer: These are the nodes. They execute individual tasks, return the execution results, and dynamically request, from the JPPF driver, the code they need to execute the client tasks.

Flow and class loader JPPF can be extended to include many servers, communicating together in a peer-to-peer topology. In this topology, each server is seen by its peers as a node, and sees its peers as clients. There are a number of major advantages to this design:
Figure 4.14: The JPPF Grid Architecture
Figure 4.15: The JPF Class Loader
Figure 4.16: The JPPF Flow of Execution
• It enables a greater scalability of the JPPF grid, by allowing the plugging-in of additional servers dynamically. This way, a server can delegate a part of its load to other servers.

• No matter how many servers are present, nodes and clients communicate with them in the exact same way.

• Peer server connections benefit from the same failover and recovery features available to nodes and clients

4.4.2 Configuration of JPPF

JPPF works in most environments, the default behavior may not be appropriate or adapted to some situations. Much of the behavior in JPPF components can thus be modified, fine-tuned or sometimes even disabled, via numerous configuration properties. These properties apply to many mechanisms and behaviors in JPPF, including: network communication management and monitoring performance, load-balancing, failover and recovery. Any configuration property has a default value that is used when the property is not specified, and which should work in most environments. In practice, this means that JPPF can work without any explicitly specified configuration at all. Layers of JPPF, architecture of Class Loader and flow of execution is shown in Figure 4.14, 4.15 and 4.16 respectively.

Before reviewing the details of each configuration property, it is useful to have the big picture of what we are configuring exactly. In a few words, a JPPF grid is made of clients (instances of your applications), servers and nodes. An application submits jobs to a server, and a server distributes the tasks in each job to its attached nodes. We can see here that we have a single client communicating with one server, to which a single node is attached. In practice,
there will be many nodes attached to the server, and many clients will be able to communicate with the server concurrently. It is also possible to link servers together, forming a peer-to-peer network of JPPF servers, allowing servers to delegate a part of their workload to other servers. We could effectively build a much more complex JPPF network as shown in Figure 4.17.

The role of the configuration will be essentially to determine where each component can find the others, and how their interactions will be processed.

Basic network configuration The server network communication mechanism uses TCP/IP. To do its basic work of receiving jobs and dispatching them for execution, Three TCP ports are required: one port is reserved for the distributed class loader, and is also used by clients, nodes and other servers another port is reserved for the communication with the clients, to receive job requests and send the results the third port is reserved to communication with the nodes, and is used to dispatch jobs and tasks to the nodes, and receive the results of their execution. It is also used to delegate jobs to other servers.

Load-balancing The distribution of the tasks to the nodes is performed by the JPPF driver. This work is actually the main factor of the observed performance of the framework. It consists essentially in determining how many tasks will go to each node for execution, out of a set of
tasks sent by the client application. Each set of tasks sent to a node is called a "bundle", and the role of the load balancing (or task scheduling) algorithm is to optimize the performance by adjusting the number of task sent to each node. JPPF has 4 predefined load-balancing algorithms to compute the distribution of tasks to the nodes, each with its own configuration parameters.

- **Manual**: each bundle has a fixed number of tasks, meaning that each will receive at most this number of tasks
- **Autotuned**: adaptive heuristic algorithm based on the Monte Carlo algorithm
- **Proportional**: an adaptive deterministic algorithm based on the contribution of each node to the overall mean task execution time
- **RI**: adaptive algorithm based on an artificial intelligence technique called reinforcement learning

Each algorithm uses its own set of parameters, which define together a strategy for the algorithm, It is also called a performance profile. Strategy has a name that serves to identify a group of parameters and their values, using the following pattern:

```
jppf.load.balancing.strategy = <profile_name> strategy. <profile_name> . <parameter_1> = <value_1> ...strategy. <profile_name> . <parameter_n> = <value_n>
```

### 4.4.3 Implementation

We have executed sample matrix multiplication application on top of JPPF. This experimentation is performed with varying number of clients. The size of matrix is also increased so as observe the performance of algorithm.

### 4.5 Observations

With the point of view to select the platform for experimentation, Globus is complex in structure and will lead to practical implementation difficulty. Globus uses condor as the scheduler. Here we find possibility of deviation of focus from algorithms to implementation of scheduler. The implementation of Video Streaming Application has given valuable inputs for our research.

JPPF is simple to implement but it does not provide possibility to add new algorithm apart from its offered selection. It provides attractive UIs and has proper infrastructure for job scheduling. As the objective is to put a newly designed scheduling algorithm for implementation, we find our algorithm difficult to integrate with JPPF.