The Gridbus Broker Manual

Srikumar Venugopal
Grid Computing and **D**istributed **S**ystems (GRIDS) Laboratory,
Department of Computer Science and Software Engineering,
The University of Melbourne, Australia
Email:srikumar@cs.mu.oz.au

November 8, 2004

1 Introduction

1.1 What is the Gridbus Grid Broker

The Gridbus Grid Broker is a software component which matches users' job requirements to the available resources and schedules the job to fulfill those requirements. The Broker therefore, takes care of resource discovery, job scheduling, execution, monitoring and gathering of output. It provides an abstraction to users particularly those who are not familiar with the complexity of Grid environements.

2 Installing and Configuration

2.1 Requirements

On the Broker side:

- Java Virtual Machine 1.4.x
- Valid certificate (if using Globus machines)

Additionally, all ports above 32000 must be open so that the jobs can connect back to the broker.
On the Grid nodes side: Must run one of:

- Globus 2.4.x
- AlchemiManager 0.8.0
- Unicore Gateway 4.1.0 (Experimental support within Broker)

In case of Globus, the user must have valid credentials to submit a job to Globus, i.e., the user's certificate-id must be mapped to an account on the node.

2.2 Installing

Linux

- Unzip the archive
  `$ tar -zxvf gridbusbroker.1.2.tgz`
- In the `gridbus/bin` directory, change `classpath.rc` and `gridbus-broker` script to reflect your setup. (Change the DB_HOME variable to point to your broker directory)
- Set your JAVA_HOME variable to your Java installation. Additionally, it is recommended that you put the "gridbus-broker" command in your PATH by doing (for Bash shell)
  `$ export PATH=$PATH:<path-to-broker-directory>/bin`

2.3 Running the Broker

Here we will describe how to run the broker through the command line. There are 2 inputs to the broker: the job description which is encoded within a *plan file* and the list of hosts to run the jobs on which is given within a *gatekeeper* file. We will describe the plan file and gatekeeper files in more detail in the next section.

Running the Broker for the first time
All the instructions assume a Linux or a similar Unix environment. The sequence for Windows is similar. Also they assume that currently you are in the top-level directory of the distribution, i.e, in the `gridbus/` directory

- Change to the `examples/simple` directory within the distribution
- Modify the gatekeeper file to include hosts that you have access to
- In case of Globus resources, initialize proxy
  `$ grid-proxy-init`
- Type the following at the prompt
  `$ gridbus-broker --plan=example.pln --gate=gatekeeper`
  You should see lines of log output scrolling by. The broker finishes off with a "`Broker exiting..`" when its done.
- In the directory, you will see output files named `j1.out`,`j2.out`,etc. Each file should contain a "HELLO WORLD" message. If there are 10 output files, the broker has been able to successfully execute all the jobs.

3 Working with the Broker

In this section, we will cover the various ways in which we can work with the broker.

3.1 About the input

What are plan files?
The input to the broker follows the parameter-sweep execution model in which the same application is run for different values of input parameters often expressed as ranges. This model is similar to that followed by systems such as Nimrod/G,Enfuzion and AppLeS PST. Parametric executions are suitable for applications such as data analysis in which a program has to be run multiple times over different datasets. The Gridbus broker has extended this model to data grids, that is, one can specify remote files as input parameters.
The Gridbus broker has adopted the Nimrod/G plan file format as the basis for its input description. Below is the plan file from the example in the Installation section

```
parameter st integer range from 1 to 10 step 1;
task main
        node:execute echo "HELLO WORLD! $st" > $jobname.out
        copy node:$jobname.out $jobname.out
endtask
```

As seen above, a plan file is a text file consisting of *parameters* which specify value ranges and *tasks* consisting of commands. There are as many jobs as there are unique sets formed by cross-product of parameter values, i.e., if there are 2 parameters each having 10 values then there will be 10x10=100 jobs. A job therefore is defined by a unique set of parameter values and the task specified within the plan file.
The syntax of the Nimrod/G plan file can be found online at `http://www.csse.monash.edu.au/cluster/enFuzion`. However, the Gridbus broker does not support all the declarations that have been defined in the Nimrod/G plan file format. Notably, only the main task is supported at the present. Some other declarations such as random and compute ranges are also not supported. However, the broker has introduced some declaratives of its own such as:

- **Gridfile parameter** The Gridfile parameter allows the user to specify remote files as input parameters. The syntax of this parameter reads as:

  `parameter <name of parameter> Gridfile <file_location>, <file_location>,..`

  Following is an example of how this parameter can be specified:

  `parameter INFILE Gridfile lfn:/users/winton/fsimddks/fsimdata*.mdst`

  Here, the URI (Uniform Resource Identifier) of a Globus Replica Catalog location is passed to the broker as an input. Note the location holds multiple files. The broker internally resolves this to the actual number of files present. Also, while scheduling the jobs for execution, the broker takes care to see that the jobs are scheduled close to where the files are physically located. This means that the overhead of transferring large data files is significantly reduced. More details about the scheduling can be found in the technical report on the broker available at `http://gridbus.org/papers/gridbusbroker.pdf`.

  Currently, the broker supports only the Globus Replica Catalog. However, support for the SDSC SRB(Storage Resource Broker) is close to being finalised while support for other URI such as http://, ftp:// and file:// are in the offing.

- **gcopy command** The gcopy command allows the user to copy a single file to or from a defined remote host which is not the node on which the job is to be executed (for that facility, look at the

copy command). This command is useful when all the jobs required a single data file which is however hosted on a remote server or when the job outputs have to be piped to a defined location on a remote storage host. The syntax of this parameter is:

`gcopy <hostname>:<file_location\filename> filename` or

`gcopy filename <hostname>:<file_location\filename>`

An example of its usage is:

`gcopy belle.cs.mu.oz.au:input.dat input.dat`

Internally gcopy resolves to a GSIFTP URI so the file has to be available through GSIFTP and the appropriate permissions have to be in place. Support for SRB is being worked upon.

- **mcopy command** While the copy command allows only a single file to be copied over, the mcopy command copies multiple files at the same time. The syntax of the mcopy command is similar to the copy command except that it supports wildcards. For example:

  `mcopy input* node:input/`

  The destination(either remote or local) must be a directory.

The XML input
The broker internally converts the text plan file to an XML file. This XML file is available in the same directory as the plan file when the broker exits execution.(Look out for example.xml when the example shown in the Installation section is run) The schema for this XML document can be found in `xml/planfile_Schema.xsd` within the broker distribution. The schema is extensible and can be used to introduce new parameter types if necessary. Future versions of the broker may contain more functionality within the XML than can be made available through the plan file. Therefore, we encourage users to work with the XML file for developing applications on top of the broker.
It is simple to extend the schema for your own purposes. The XML is parsed using a DOM parser and a reflection mechanism so the resolution of the XML document to jobs is independent of the schema itself. Introducing a new parameter involves writing a resolver and sticking to the naming scheme. Interested users can look through the source (package `org.gridbus.broker.plan`)for details on implementation.
Note: The broker can be provided with an XML input file by passing it through the –plan command line option. For example, you could try executing the sample plan file, referred in the installation section, by giving `--plan=example.xml` as the command line option. To create the XML out of a plan file, you can use the `gridbus-plan2xml` script in `gridbus/bin` as:
`$ gridbus-plan2xml <plan_file_name> <xml_file_name>`

3.2 The Gatekeeper file

The gatekeeper file is passed to the broker by the `--gate=` option on the command line or by setting the `GATEKEEPER_FILE` property within DB.properties. This file contains the list of compute nodes to which the user has acess for executing jobs. A sample gatekeeper file is shown below:

```
globus  belle.cs.mu.oz.au 2
alchemi  horowitz.cs.mu.oz.au http://horowitz/alchemi.
crossplatformmanager/crossplatformmanager.asmx
unicore  testgrid.unicorepro.com:4001 3
```

The first column is the middleware name. Presently, 3 middleware systems are supported Globus 2.4.x(”globus”), Alchemi 0.8.0(”alchemi”) and UNICORE 4.1.0(”unicore”). For Alchemi and Globus, the second column is the hostname of the resource.In case of UNICORE, it is the hostname of the gateway and the port number. The third column can be used to specify an optional cost of using the resource for Globus and UNICORE and it is the URI of the web-service in case of Alchemi(Note: the remote resource must be an Alchemi manager and must be running the webservice)
If the first column were to be omitted altogether, the broker will assume it is a Globus resource. The broker probes every resource to see whether it is alive and able to execute jobs. For Globus resources, it also probes the GRIS (Grid resource Information Service) for details about the resource. If the broker is not able to gather information successfully, it considers the remote resource as unavailable.
It is also possible to avoid using the gatekeeper file in situations where the resources are available through an online index such as a Virtual Organisation index, GIIS(Grid Index Information Service) or a GMD(Grid Market Directory). In such cases, a (simple) program has to be written to add these resources to the broker. Future versions of the broker will include this feature.

3.3 Programming the Broker

While the examples shown here assume a command line environment, the Broker’s capabilities are not limited to such a configuration. Other applications such as web portals have been built on top of the Broker by using its APIs(Application Programming Interfaces). The Broker was also designed to be independent of the input model. Therefore, it is possible to port other programming and scheduling models to the Broker. Some of this work has been described in the tech reports available at
`http://www.gridbus.org/broker`
Since a job consists of tasks and tasks themselves consist of commands, its is possible to build a job from its substituents and add it to the broker independent of the input plan files. Similarly, a compute node can be added in the same way. An example of the possible uses of APIs is given in `examples\prog\GridTask.java`. Interested developers are encouraged to look at this code and the APIs.

4 Conclusion

This manual attempts to explain the design of the Gridbus Broker, how to installand configure it and how to create programs around it. While some of the sections have not been detailed, notably the programming section, it is hoped that this provides enough material for a first exploration. The Gridbus Broker team would be very happy to answer any queries that you may have regarding the Broker. Relevant contact information is given below.
Contact: Dr. Rajkumar Buyya(raj@cs.mu.oz.au), Srikumar Venugopal(srikumar@cs.mu.oz.au), Krishna Nadiminti(kna@unimelb.edu.au)