

GridSAM Tutorial.

Using GridSAM to submit simple jobs.

Introduction

The aim of this tutorial is to describe how GridSAM works and how it can be used in research. Anyone with a basic knowledge of Linux should be able to complete this tutorial. The tutorial is intended for researchers who would like to learn more about deploying a custom GridSAM application in a grid environment. Other related parties, such as systems administrators and technical managers, can also benefit from the tutorial.

What is GridSAM?

GridSAM performs job submission. It takes a *job* and submits it for computation to a *resource*. The *job* could be anything that requires computation – in this tutorial we use the example of an algorithm that discovers prime numbers. The *resource* could be the user's own computer (as used in this tutorial), another computer, a cluster of computers (via Condor, PBS or SGE) or national Grid resources (such as the National Grid Service in the UK).

To operate, GridSAM requires some supporting software (a Web Service container, Axis, security, etc.). To make life easy, in this tutorial we will use GridSAM through OMII-UK's Campus Grid Toolkit (CGT). CGT provides you with GridSAM and all the software that GridSAM needs in one easy-to-install package. Another benefit of CGT is that it is provided with all the example files needed in this tutorial.

How is this Tutorial structured?

The GridSAM tutorial will show you how to run a command-line-based client: from generating results locally, through to running the same command line in a controlled manner on GridSAM. The tutorial will provide a coherent end-to-end description, rather than a full technical reference.

All files needed in this tutorial are installed when you install Campus Grid Toolkit, as described below. If you would like to know what files are used and what each file does, see the table at the end of this tutorial.

To make a distinction between text, code and screen output, in this tutorial filenames and directory names are written in `courier`. Commands are written in `courier` and surrounded by a box:

```
Which java
```

Screen output is written in `courier` and highlighted as shown below:

```
1
2
3
5
```

Support

For more information about OMII-UK or for help with this tutorial, visit the OMII-UK website (www.omii.ac.uk) or contact support@omii.ac.uk.

For more information about GridSAM and Campus Grid Toolkit, and for downloads, visit the OMII-UK software overview (www.omii.ac.uk/wiki/SoftwareOverview).

Where to start?

Install a Campus Grid Toolkit Server – it's easy!

Before you can start the tutorial, you will need to install Campus Grid Toolkit (CGT). This is very easy and is explained in detail in the CGT guide:

- Download: www.omii.ac.uk/wiki/CGT
- CGT guide: www.omii.ac.uk/docs/CGT/CampusGridToolkit.pdf

During installation of CGT, you will be given a few choices for customisation. You should choose the default settings, namely:

1. A *direct gridsam* interface
2. Execution to occur via the fork connector

Once you have installed your CGT Server, install a client on the same machine. Installation of the client is performed by visiting the URL for your CGT server in a web browser and following the on-screen instructions (this process is described in detail in the CGT guide).

The Primes Tutorial

Primes is a simple Java program for finding prime numbers within a defined range. It was chosen as a simple problem which can easily be adapted, and easily deployed in a Grid environment.

All of the example files needed for this tutorial are installed with the Campus Grid Toolkit 1.1.3 client. The files are stored in the `gridsam/primes` directory.

1. Running Primes locally

Open the folder in which the CGT client was installed (for example: `/home/user/campus-grid-toolkit-client/gridsam/primes`) although the exact location will depend on your system and where you chose to install CGT) and change directory to `gridsam/primes`.

One of the files in the directory is called `Primes.java`, this is the basic code that will be used in this tutorial. Feel free to examine `Primes.java` to see how it works.

Before the java code can be run, it must be compiled. Do this by entering the command:

```
javac Primes.java
```

If the compiler ran successfully, your `gridsam/primes` directory should now include a `Primes.class` file.

We can get an idea about how Primes works by running it locally. Do this by entering the command:

```
java Primes 0 20
```

Your terminal should list all of the prime numbers up to 20. Your terminal window should now look something like this:

```
javac Primes.java
java Primes 0 20

1
2
```

```
3
5
7
11
13
17
19
```

2. Running Primes with GridSAM

Now that we've executed Primes locally, we can run the code through GridSAM. We will do this by running a few command lines – keeping it simple at first. Then we'll move onto a more real-world example, using some Perl to help manage the running of several parallel jobs.

When GridSAM runs the job it works through four steps:

1. Stage any input data and/or programs to the system
2. Submit a JSDL file, which describes how GridSAM should run the job run on the system
3. Wait for the job to complete
4. Retrieve any result files

In this trivial example, we will *stage* the inputs and retrieve the outputs via the `/tmp` directory on the local filesystem.

Examine `primes-0to99.jsdl`

Look in the `gridsam/primes` directory for the `primes-0to99.jsdl` file. This file describes how to run a simple job. As implied by the file name, it finds the prime numbers between 0 and 99.

Open the `primes-0to99.jsdl` file and examine it. Of particular interest are the following features:

- `<Executable>` which calls the `primes.sh` script
- `<Argument>` which provides the upper and lower limits through which prime numbers will be found
- `<DataStaging>` sections, containing `<Source>` features which describe where the prime numbers will be generated or `<Target>` features where the results will be written

Run `primes-0to99.jsdl`

Before the job can be executed, you will need to edit the `primes.sh` file. This file is simply a wrapper of a java command line. To operate it, it needs to know the location of your java runtime. To find out where java is on your system, enter the command:

```
which java
```

Now edit `primes.sh` and add the path reported by the `which` command.

To run the job, two files must be staged: `Primes.class` and `primes.sh`. They are staged using the `gridsam-submit` command line application, which submits the job to the server using the JSDL file. Once this has been done, a unique job ID is returned and stored in the `running-jobs` file (where `localhost` is the fully qualified name of your computer).

```
cp primes.sh Primes.class /tmp
```

```
../bin/gridsam-submit -s http://localhost:18080/gridsam/services/gridsam
?wsdl \ -j primes-0to99.jsdl > running-jobs
```

To see whether the job has been completed, you can use the `gridsam-status` application. Run this a few times in quick succession to see the details of the job as the job progresses. The output of `gridsam-status` is quite verbose, so it is advisable to use a suitably large terminal window with scroll bars, or use a pager like `less` or `more`.

```
../bin/gridsam-status -s http://localhost:18080/gridsam/services/gridsam
?wsdl \ -file running-jobs
```

Once the job has completed, the results in the `results.txt` file in the `/tmp` directory should look like the following:

```
1
2
3
5
7
11
...
```

3. Running Multiple Primes Jobs

Next, we will run two jobs and have them both submitted at the same time. This will better demonstrate the advantages of submitting multiple jobs which can be run in parallel.

We will stage the files in and out via `/tmp`. You will note a second JSDL file, `primes-100to199.jsdl`. This file, as the name suggests, describes a job which finds primes between 100 and 199. Examine the differences between this file and the first (`primes-0to99.jsdl`), and you will find that only the arguments and the output file are different.

Submit the two jobs using `gridsam-submit` commands, as before. The unique job IDs that are generated can be sent to the same file, which allows the status of both running jobs to be established at one time using `gridsam-status` command.

```
../bin/gridsam-submit -s http://localhost:18080/gridsam/services/gridsam
?wsdl \ -j primes-0to99.jsdl > running-jobs
```

```
../bin/gridsam-submit -s http://localhost:18080/gridsam/services/gridsam
?wsdl \ -j primes-100to199.jsdl >> running-jobs
```

```
../bin/gridsam-status -s http://localhost:18080/gridsam/services/gridsam
?wsdl \ -file running-jobs
```

Once the status of both jobs has reached done, examine `/tmp/results.txt` and `/tmp/results2.txt`. These files should contain the result sets for both jobs.

Given the verbosity of `gridsam-status`, if you are only interested in the current state of the jobs, you can easily filter out the rest of the lines using `grep`:

```
../bin/gridsam-status -s http://localhost:18080/gridsam/services/gridsam
?wsdl \ -file running-jobs | grep "Job Progress"
```

```

Job Progress 'urn:gridsam:4028808120334c7601203815039c07e4': pending ->
staging-in -> staged-in -> active -> active-running -> executed ->
staging-out -> staged-out -> done
Job Progress 'urn:gridsam:4028808120334c76012038151a5207e7': pending ->
staging-in -> staged-in -> active -> active-running -> executed ->
staging-out -> staged-out -> done
  
```

Managing job submission with a simple custom client

The management of job submission in the way shown above lends itself to automation. To this end, we will use a Perl script to submit several jobs and then automatically retrieve the results as the jobs complete. The Perl script `primes_file.pl` takes a range of integers, splits it into a number of equal sub-ranges and then submits each sub-range as a separate job.

Examine the `primes_file.pl` file. The section shown below indicates the range in which to search for prime numbers, and how many jobs the problem should be chopped into. In this case, 0 to 20000 in 10 jobs, i.e. 0 to 1999, 2000 to 3999, etc.

```

## Parameters ##
my $range_min = 0;
my $range_max = 20000;
my $number_of_jobs = 10;
  
```

The next noteworthy section is used to create a number of JSDL files: one for each job. The Perl uses a template for this (which is abbreviated below).

```

foreach my $job (1..$number_of_jobs)
{
    open(JSDL, "+> ${job}.jsdl");
    print JSDL <<EOT;
<JobDefinition xmlns="http://schemas.ggf.org/jSDL/2005/11/jSDL">
...
        <Executable>./primes.sh</Executable>
        <Argument>${jobs{"$job.min"}}</Argument>
        <Argument>${jobs{"$job.max"}}</Argument>
...
</JobDefinition>
EOT
    close(JSDL);
}
  
```

Once the JSDL has been prepared, the call to `run_jobs` is made. This is a library routine defined in `gridam_webdav_util.pm`. Feel free to read its definition in this file.

```

my %job_ids = run_jobs(@jsdls);
  
```

The next section loops, checking for job completion until all jobs have finished.

```

while(scalar keys %job_ids)
{
...
    open(STATUS, "../bin/gridsam-status -s
        '$remote_server_url/gridsam/services/gridsam?wsdl' -file
        '$jobs_list_file'|");
...
}
  
```

```
}

```

Finally, the results are summarised. Rather than listing each prime number found, the number of primes found in each range is given.

```
# summarise the results;
print "Results\n";
foreach my $job (1..$number_of_jobs)
{
  open(RESULTS, "results/$job.txt") or next;
  my @results = <RESULTS>;
  printf("%05d - %05d : %s primes found\n", $jobs{"$job.min"},
        $jobs{"$job.max"}, scalar(@results));
}

```

To try this example, simply run:

```
./primes_files.pl

```

The output will be something similar to the following:

```
Submitting jobs... jobs submitted.
Checking for job completion.
Job 1 has run successfully.
Jobs outstanding: 9. Jobs completed: 1. Jobs failed: 0
Jobs outstanding: 9. Jobs completed: 1. Jobs failed: 0
Jobs outstanding: 9. Jobs completed: 1. Jobs failed: 0
Job 7 has run successfully.
Job 5 has run successfully.
...
Jobs outstanding: 0. Jobs completed: 10. Jobs failed: 0
Results
00000 - 01999 : 304 primes found
02000 - 03999 : 247 primes found
04000 - 05999 : 233 primes found
06000 - 07999 : 224 primes found
08000 - 09999 : 222 primes found
10000 - 11999 : 209 primes found
12000 - 13999 : 214 primes found
14000 - 15999 : 210 primes found
16000 - 17999 : 202 primes found
18000 - 19999 : 198 primes found

```

You can get a better idea of how `primes_file.pl` works by editing the file and re-running it. Try changing the number of jobs or the range over which prime numbers are found tested.

Using WebDAV for staging files

In the simple case above, we used staged inputs and outputs via a shared filesystem (`/tmp`). We could also use a network based share, such as NFS, or stage files through the WebDAV service installed on Tomcat. The `primes_webdav.pl` file found in the `gridsam/primes` directory can be used in this way. It transfers the inputs to the WebDAV server and retrieves the result files from it.

The `primes_webdav.pl` file and the `primes_file.pl` file are very similar. The few differences between the files can be shown by entering the following command:

```
diff primes_file.pl prime_webdav.pl
```

The most important difference is the initial creation of a staging area. This supplies a workspace for your jobs which is separate from other concurrently running jobs. There is also a `destroy` command, which is used to clean up the staging area once the results have been retrieved.

Also note that file references within the JDSL template are different, they are all written in terms of `webdav://` rather than `file://`. The variables `webdav_user`, `webdav_password`, etc. are initialised by the library routine `read_server_details`. This routine can be found in the `gridsam_webdav_util.pm` file.

To run the webdav version of the primes tutorial, enter the following command:

```
./primes_webdav.pl
```

Again, try different parameters by editing the ranges and number of jobs in the parameters section of the `gridsam_webdav_util.pm` file. To gain extra insight into the workings of the client, you could also try setting the output level to debug. To do this, open `gridsam_webdav_util.pm` and edit it by setting:

```
our $DEBUG = 1;
```

Then re-run the `primes_webdav.pl` file.

When the client is staging files – either via network-based filesystem or WebDAV – it no longer needs to be on the same machine as the server. So you could now install the CGT client along with the primes example files and run them in the same manner.

Further information

For more information about GridSAM visit the OMII-UK website: www.omii.ac.uk/wiki/GridSAM.

Wondering what some of the files in the 'primes' directory on your client are?

File name	Description
<code>Primes.java</code>	A simple Java program for finding prime numbers between two limits. This is the source file for <code>Primes.class</code> .
<code>Primes.class</code>	Used as a simple example for computational jobs. It searches for prime numbers between two limits, e.g. <code>java Primes 10 20</code> .
<code>primes-0to99.jsdl</code>	A JSDL file that describes a GridSAM job to search for prime numbers between 0 and 99.
<code>primes-100to199.jsdl</code>	A JSDL file that describes a GridSAM job to search for prime numbers between 100 and 199.
<code>primes.sh</code>	A simple wrapper script for running <code>Primes.class</code> on a remote system. It also contains a path to the java run-time, so you may have to update this file to point to your system's java.
<code>primes_file.pl</code>	A Perl file which manages submission of a set of jobs, monitors for their completion and collates the results.
<code>primes_webdav.pl</code>	A Perl file much like <code>primes_file.pl</code> , however the file staging is performed via a local WebDAV server rather than via the filesystem directly.
<code>gridsam_webdav_util.pm</code>	A Perl utility library containing useful routines for managing GridSAM and WebDAV interactions from Perl scripts.
<code>results/</code>	The directory where the prime number results files will end up, when either <code>primes_file.pl</code> or <code>primes_webdav.pl</code> are used. These can be safely removed after completion of the tutorial.
<code>running-jobs</code>	This file contains the list of currently running job IDs. It is only valid for the current run of <code>primes_file.pl</code> or <code>primes_webdav.pl</code> and can be safely removed after completion of the tutorial.