GRAM Tutorial - Part II

PPARC Summer School,
NeSC Edinburgh
Guy Warner, NeSC Training Team

This tutorial extends the previous tutorial into the more advanced topics of multiple jobs running simultaneously, the globus GRAM API and using the Message Passing Interface (MPI). The assumption is made that an understanding of job submission and compiling code was acquired from the previous tutorial. Please consult the previous tutorial, which may be found here, if in any doubt.

This tutorial optionally involves some editing of files. To use a graphical editor you must first launch Exceed - double click the "Exceed" shortcut on your desktop. After the splash screen no further screen will appear - this is what is meant to happen. To edit a file use the command

kwrite <filename> &

where <filename> is the name of the file you wish to edit. You can safely ignore any messages that appear in your terminal window. When you exit you may also see a message about "This may end your X session" - just click "Ok".

1. Before starting the tutorial it is necessary to ensure you have a running grid-proxy. Open a putty session onto "pub-234". You can check the state of your proxy by typing

   grid-proxy-info -timeleft

   This shows the amount of time left in your proxy (in seconds). If this is zero or low (anything less than 10800 (3 hours) can be considered as low) then restart your proxy by running the following commands

   grid-proxy-destroy
   grid-proxy-init

2. The simplest form of project to take advantage of the ability to run multiple jobs simultaneously is where the problem to be solved is 'trivially parallel'. Trivially Parallel is when multiple copies of the same job can be run with different input parameters or data. The important characteristic is that once the jobs have started running they don't need to know what the other jobs are doing. When they have finished running, a final job may be run to collate the results. This sort of problem does not require any complicated message passing to a master program (which will be looked at later). The simplest way to store the results is to use a file (or database) which may be analysed later.

   For this tutorial we will look at a (somewhat contrived) simple mathematical problem. The problem to be solved is to integrate, using the trapezium rule (for details on the trapezium rule see http://www.answers.com/topic/trapezium-rule) the function

   $$f(x) = \begin{cases} 1 - x^2 & -1 < x <= 1/3 \\ - x^2 & 1/3 < x < 1 \end{cases}$$

   If you are not familiar with mathematics it is probably sufficient to know that the exact answer is 2/3. The trapezium rule solves this problem numerically and hence a numerical error is expected - this example demonstrates how to reduce this error by using multiple jobs simultaneously. The example uses three perl scripts:

   i. multijob.pl - the master script. This script contains functions that call the appropriate globus commands and captures the output. Due to the numerical nature of the problem it is possible to split the problem into several segments (in the default case [-1, -1/2], [-1/2, 0], [0, 1/2] and [1/2, 1]). Firstly the range [-1,1] is split into several segments (by default 4 - but this can be over-

ridden by a command line parameter to this script). A job to calculate the numerical integral for each segment is launched with the calculated value being stored in a file named "multijobData.X" where X is the segment number. Once all the segment jobs have successfully been run, a final job is submitted that collates the results and outputs the answer.

ii. integrate.pl - the script that carries out the numerical integration. This script accepts the parameters (in this order): starting point, end point and the name of the file to store the output.

iii. sum.pl - the script that collates the results. It accepts the parameters (in this order): base_name - the base name of all the output files (in this example this is always "multijobData") and segment_total - the number of segments the problem was split into.

The code for this example is in the directory "gram2/multijob". Change to this directory with the command

cd ~/gram2/multijob

Before running the multijob script ***upload the perl files to the head node using gsiscp.***

Now run the multijob script with the default number of segments, by using the following command:

./multijob.pl

You should get similar output to the below (job uid's will be different and possibly the number of status checks):

```
Splitting range into 4 equal segments
Stage1
globus-job-submit grid-data.rl.ac.uk integrate.pl -1 -0.5 multijobData.0
globus-job-submit grid-data.rl.ac.uk integrate.pl -0.5 0 multijobData.1
globus-job-submit grid-data.rl.ac.uk integrate.pl 0 0.5 multijobData.2
globus-job-submit grid-data.rl.ac.uk integrate.pl 0.5 1 multijobData.3
globus-job-status https://grid-data.rl.ac.uk:64002/23719/1111587816/
globus-job-clean -f https://grid-data.rl.ac.uk:64002/23719/1111587816/
globus-job-status https://grid-data.rl.ac.uk:64004/23743/1111587817/
globus-job-clean -f https://grid-data.rl.ac.uk:64004/23743/1111587817/
globus-job-status https://grid-data.rl.ac.uk:64001/23707/1111587815/
globus-job-clean -f https://grid-data.rl.ac.uk:64001/23707/1111587815/
globus-job-status https://grid-data.rl.ac.uk:64003/23731/1111587816/
globus-job-clean -f https://grid-data.rl.ac.uk:64003/23731/1111587816/
Stage2
globus-job-submit grid-data.rl.ac.uk sum.pl multijobData 4
globus-job-status https://grid-data.rl.ac.uk:64005/25566/1111587868/
globus-job-get-output https://grid-data.rl.ac.uk:64005/25566/1111587868/
globus-job-clean -f https://grid-data.rl.ac.uk:64005/25566/1111587868/
**********************************
* The answer is 0.6575 *
**********************************
```

3. Now run the job with more segments by using the command

./multijob.pl <n>

where <n>is the number of segments you wish to use. ***Experiment with different values of n*** (below 20 - so as to be considerate of other NGS users). Whilst this example runs each segment very quickly it should be apparent how for larger scale jobs the speed of obtaining results is obtained and hence the accuracy of the results which may be obtained.

4. Optional: Make two copies of the file multijob.pl with the commands

cp multijob.pl multijob_1.pl
cp multijob.pl multijob_2.pl

Edit multijob_1.pl so that it only runs Stage1 by deleting relevant lines. Similarly edit multijob_2.

pl so that it only runs Stage2.For simplicity in this tutorial leave the status checks and job cleanup in the same script as the job submissions (a more realistic scenario would involve these residing in a separate script as well). This demonstrates how with this method of submitting multiple jobs does not rely on a master process and that the results may be collected on a separate occasion.

5. The next topic for this tutorial is a simple example of the globus API. It is beyond the scope of this tutorial to examine the API in depth. This tutorial aims to only show the stages needed to compile code that uses the API. A guide to the functions of the GRAM API may be found at http://www-unix.globus.org/api/c/globus_gram_client/html/index.html.

   Before starting this section change to the directory in your account containing files needed for this section of the tutorial:

   cd ~/gram2/api

   Any program that needs to be compiled against an API needs to find the relevant header files (files containing the definitions of the functions), the libaries of the functions and any other relevant definitions. The Globus toolkit provides a method of generating a file containing all the necessary definitions. This file may be used from within a Makefile (a file used by the "make" program to help automate the process of building an application - very similar to the "ant" program for java code) to compile the code. Since globus is a tool available across the spectrum of unix derivatives it needs to be able to handle different variations of compiler and binary format (called flavor's in globus terminology). To compile any code you must select an appropriate for which a run time library (rtl) and development version (dev) are available. Availability of flavors and different configurations can also depend on which parts of the globus API the code being developed will use. For this tutorial one part only is being used - globus_gram_client. To find the available flavors use the command

   $GPT_LOCATION/sbin/gpt-query globus_gram_client

   You should get the below output

   ```
   2 packages were found in /opt/globus that matched your query:

   packages found that matched your query
      globus_gram_client-gcc32dbg-dev pkg version: 4.1.1
      globus_gram_client-gcc32dbg-rtl pkg version: 4.1.1
   ```

   In this listing the only available flavor is "gcc32dbg"

6. Having now chosen a suitable flavor a file can be created with all the necessary definitions by using the command:

   globus-makefile-header --flavor gcc32dbg globus_gram_client > globus_header

   If you examine the file "globus_header" with the command

   cat globus_header

   the advantage of having an automated system for generating this configuration is apparent.
7. Having generated the necessary configuration it can be included in a Makefile and the code compiled. Examine the "Makefile" and you will that "globus_header" is included and the commands that are used to compile your code. Compile the command by typing

   make

   and then run it with the command

   ./apidemo

   . This code simply tests that the queue "grid-data.rl.ac.uk/jobmanager-pbs"can be contacted (a ping type message is sent).

8. If you examine the code you will see that it is necessary to initialize the part of the globus api code

being used:

globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);

and similarly to deactivate it. If you miss this out of your code it will still compile, but you will encounter errors at run time. ***Modify the code and introduce an error into the name of the queue being tested.*** If make complains about being already up to date use the command

make clean

before repeated the make command.

9. For the final part of this tutorial an example of running an mpi job is introduced. Projects that are not trivially parallel and where run time communication between the parallel components is required typically make use of mpi. The Message Passing Interface (mpi) library provides the tools for passing these messages between parallel components running on different nodes. The implementation of mpi being used in this tutorial is "mpich-gm". Full details on this implementation may be found at http://www.myri.com/scs/index.html (note that an understanding of the implementation details is not required for this tutorial).

   Before starting this example change to the directory in your accounts that contains the mpi example code with the command

   cd ~/gram2/mpi

   and then ***upload (using gsiscp) the file "mpi_ex1.c" to grid-data.rl.ac.uk***

10. As with the fortran example in the previous tutorial, to compile code for mpi requires access to libraries that are not available to you locally. Open a new gsissh connection onto grid-data.rl.ac.uk. You may find it easiest to work with two Putty connections to pub-234, one for running commands on pub-234 and one for running commands on grid-data.rl.ac.uk. ***Load the module "clusteruser".*** This is actually a module that acts a wrapper to the collection of modules needed to run mpi jobs. You can see the list of modules that have been loaded by using one or both of the following commands:

    module list
    module display clusteruser

11. Now compile the example mpi job with the command

    mpicc -w mpi_ex1.c -o mpi_ex1

    If you try to run the program you have just built you will encounter error messages. For the sake of simplicity in this tutorial we will not look at how to run the program from this environment, but concentrate instead on how to run it as a job using the globus commands.

12. From pub-234 run the command

    globus-job-submit grid-data.rl.ac.uk/jobmanager-pbs -np 8 -x '&(jobtype=mpi)(environment=(NGSMODULES clusteruser))' mpi_ex1

    There are two important changes to observe with this command. Firstly, this time 8 processes, which may or may not run on the same node, have been requested for the job to run on (the -np 8 part of the command). Secondly the jobtype has been set to mpi. Once this job has successfully run you should be able to retrieve something similar to the following output

```
Process 0 on host grid-data12.rl.ac.uk broadcasting to all processes
Receiving from all other processes
Received a message from process 4 on grid-data12.rl.ac.uk
Received a message from process 2 on grid-data14.rl.ac.uk
Received a message from process 1 on grid-data13.rl.ac.uk
Received a message from process 5 on grid-data13.rl.ac.uk
Received a message from process 6 on grid-data14.rl.ac.uk
Received a message from process 3 on grid-data15.rl.ac.uk
Received a message from process 7 on grid-data15.rl.ac.uk
Ready
```

13. Optional (requiring a knowledge of C): modify the code so that each of the processes other than process 0 send back an additional piece of information (for example a random number). Several other example mpi examples have been provided in a subfolder of your current folder called "other_examples". You might find comparing to these codes helpful in solving this task. Also try running these programs as jobs (the fortran code will need to use the mpif77 compiler).