

The ARCS Grid – User Documentation

Overview of the Grid	2
The Grid Configuration.....	2
Available Software.....	2
Accessing the Grid.....	3
Accessing the Grid from On Campus.....	3
Transferring data to/from the Grid.....	4
Accessing the Grid from Off Campus	4
Running Jobs on the Grid	5
Setting up to run on the grid	5
Running a Job on the Grid	5
An example using non-interactive Matlab.....	9
Using the Matlab Distributed/Parallel Compute Engine	11
An example using non-interactive JAVA.....	21
An example using non-interactive R.....	23
Deleting a job from the grid.....	24
Running a job with email notification	24
Batch Queueing multiple jobs with different datasets	24
Using PVM on the Grid	25
How PVM is run on the grid.....	25
Setting up for PVM.....	25
Compiling a testing a PVM example.....	26
Running the Grid Engine PVM Example	26
Running my own PVM job	29
Using MPI on the Grid.....	30
Array Jobs	33
Other Grid Queues	35
Compiling my own software (C, C++, Fortran)	36
Grid Etiquette.....	37
Further Information about the Grid.....	38

Overview of the Grid

The Grid Configuration

The current grid configuration consists of a grid head node that maintains the queues, a submit node on which jobs are compiled/test run/submitted to the grid and a number of execute nodes on which your program is executed. The execute nodes are configured identically and there is no way to tell which node will be selected by the grid. The hardware configuration of the grid is as follows:

- Grid Head: 64 bit Sun Solaris 10 x86 (1x Opteron 1GB ram)
- Submit Node: 64 bit Redhat AS4 Linux (2x Opteron 4GB ram)
- 18 Execute Nodes: 64 bit Redhat AS4 Linux (2x Opteron 4GB ram)

The grid is suited to non-interactive jobs where the data is read in from a file and results are written to an output file. Interactive jobs are usually run one at a time and may have to wait until a free grid node is available. There are other non-grid nodes available for interactive jobs and compiling/developing software.

Before using the grid you will need to contact John.Brattan@newcastle.edu.au the Manager of ARCS and discuss with him your requirements to see if the grid is able to run your software.

Available Software

Most of the standard Linux Redhat software is installed in /usr/bin or /usr/local/bin

Other packages are in /grid/local/**ARCH**/bin (where **ARCH** is linux-32bit, linux-64bit, solaris10-x86)

Current software on the grid includes:

- java 1.5 64bit /usr/java/jdk1.5.0_05/bin
- java 1.6 64bit /usr/java/jdk1.6.0_02/bin
- cfx 10 & 11 64bit /grid/local/cfx/
- g++,gcc,g77 3.4.6 64bit /usr/bin
- matlab 17 64bit /usr/local/matlab/matlab2009b/bin
- Molpro 64bit /grid/local/linux-64bit/bin
- R 2.51 64bit /grid/local/linux-64bit/R/R-2.5.1/bin
- Eman 64bit /grid/local/linux-64bit/EMAN
- Gaussian 64bit /grid/local/linux-64bit/g03
- MPICH2 64bit /grid/local/linux-64bit/mpich2
- Vasp 4.6 64bit /grid/local/linux-64bit/vasp

The Commercial Intel Fortran and C++ compilers and the Intel Maths Library are also available upon request. Some of the software has limited licenses (and you need to be a member of a group to run it). Please contact arcsgrid@newcastle.edu.au if you need to use it. If additional software is required then it can be installed on the grid.

Accessing the Grid

Accessing the Grid from On Campus

The grid submit node runs the Linux Operating system (Redhat 4 at the time of writing). The grid does not run local X-windows sessions so if you are not already on a X-windows capable desktop (eg Linux or MacOS X) then you will need to set up VNC if you need a graphical session.

If you just need a UNIX prompt then you can connect via SSH using:

- Linux – ssh from the command line (eg ssh username@ares.uni.newcastle.edu.au)
- MacOS X – ssh from the command line (eg ssh username@ares.uni.newcastle.edu.au)
- Windows XP – ssh using putty SSH
 - <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>
 - choose putty.exe from the above web site
 - Run it and select 'ssh' and type in ares.uni.newcastle.edu.au as the host name
 - The default settings can be saved
 - Click on 'Open' to make a connection

The grid firewall settings are as follows:

- Telnet is not supported and is blocked by the grid firewall.
- FTP is also blocked by the grid firewall. You can scp to ares.uni.newcastle.edu.au
- Off campus SSH is blocked by the campus firewall. See notes in the next section in order to connect to the grid from off campus

Transferring data to/from the Grid

All student directories (U: drive) are now on the campus SAN and can be accessed via the windows share \\student\home . Staff home directory (U: drive) are also on the campus SAN and can be access by the windows share \\staff\home

Your UNIX home directory on the grid (/home/username) can be accessed outside the grid by using:

- Linux – smbmount \\home\username
- MacOS X – Map a windows drive \\home\username
- Windows XP – Map a network drive \\home\username (You will need to connect up as UNCLE\username and use your campus proxy password.)
- You can also scp user@jumpgate.newcastle.edu.au and transfer your files using your campus username and proxy password.

NOTE: The files in your home directory /home/username are only accessible by you. If your job uses temporary directories eg /tmp for scratch space please check that correct permissions/umasks are set for these temporary files. Other users may have access to these temporary data files if you (or your software) have changed the system default permissions. Please make sure that your job removes these temporary files when it finishes.

Accessing the Grid from Off Campus

The grid submit node has firewall settings that do not allow it to be accessed off campus.

If you need to access the grid/data from off campus then you will need to ssh/scp to jumpgate.newcastle.edu.au and login there first. Then ssh a second time to the grid submit node (ares.uni.newcastle.edu.au).

Running Jobs on the Grid

Setting up to run on the grid

- To run a job on the grid you must login to the grid submit node (ares.newcastle.edu.au).

```
% ssh username@ares.newcastle.edu.au (using your web proxy password)
```

- Your home directory will be /home/username and is accessible to all nodes of the grid
- Most of the standard linux software is installed in /usr/local/
- Other packages are installed in /grid/local/*ARCH*/bin (where *ARCH* is linux-32bit, linux-64bit, solaris10-x86)

Edit your ~/.user_profile to include this directory in your PATH

```
% export PATH:/grid/local/.../bin:$PATH
```

- If you get output from the ‘qhost’ command then you already have the grid settings and do not need to do the following step.

```
% cp /grid/sge6/default/common/settings.sh ~/.grid_settings.sh (note filename begins with a dot)
```

- Copy the grid settings file into your home directory and incorporate it into your .user_profile by using a text editor and adding these files into end of the file.

```
if [ -d /grid/sge6 ] ;  
then  
source ~/.grid_settings.sh  
fi
```

Running a Job on the Grid

- Make sure that all of your grid paths are set up first

```
% source .grid_settings.sh (don't need to do this if already set up in your .user_profile)
```

- Test out that you can see the grid

```
% qhost
```

```
HOSTNAME      ARCH      NCPU  LOAD  MEMTOT  MEMUSE  SWAPTO  SWAPUS  
-----  
global        -        - - - - - - -
```

```

arcsgrid01      lx24-amd64    2 0.00  3.9G 336.8M 16.0G  0.0
arcsgrid02      lx24-amd64    2 1.00  3.9G  1.1G 16.0G 88.0K
arcsgrid03      lx24-amd64    2 1.00  3.9G  1.3G 16.0G  0.0
arcsgrid04      lx24-amd64    2 0.05  3.9G 243.0M 16.0G 88.0K
arcsgrid05      lx24-amd64    2 0.00  3.9G 133.4M 16.0G  0.0
arcsgrid06      lx24-amd64    2 0.00  3.9G 295.0M 16.0G 88.0K
arcsgrid07      lx24-amd64    2 1.02  3.9G  1.2G 16.0G  0.0
..... (several more hosts)

```

If this command does not work then you don't have all of the grid paths set up correctly.

- All jobs submitted to the grid must be shell scripts. The shell script must automatically set up all environment variables, paths. It must also execute your program and be able to determine the correct input and output datasets. It should not require user input.
- All grid job scripts are run in your **home directory** (/home/username) as yourself even if the executable program is not located in your home directory. This means that you will need to explicitly specify all file names from "/home/username/.....". You will need to make sure that multiple concurrent jobs running do not write the same data files otherwise your data will be corrupted.
- Copy a simple grid example script (there are other examples in the directory too)

```
% cp /grid/sge6/examples/jobs/simple.sh ~/simple.sh
```

- Now look at the script to see what it does

```
% cat simple.sh
```

```

#!/bin/sh
#
# (c) 2004 Sun Microsystems, Inc. Use is subject to license terms.

# This is a simple example of a SGE batch script

# request Bourne shell as shell for job
#$ -S /bin/sh

#
# print date and time
date
# Sleep for 20 seconds
sleep 20
# print date and time again
date

```

- Notice that it prints the time then sleeps and prints the time again - run the script manually to check that it works correctly

```
% ./simple.sh
```

```
Fri Jun 9 10:06:35 EST 2006
```

```
Fri Jun 9 10:06:55 EST 2006
```

- Before running new jobs on the grid it is best to do some trial runs on the Unix prompt on ares to check that everything is running as expected.
- Submit your job to the grid. You will be given a job number and your job (shell script) will be copied to any free grid node and be executed there.

```
% qsub simple.sh
```

Your job 149 ("simple.sh") has been submitted.

- Check the status of the grid queue

```
% qstat
```

job-ID	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
149	0.00000	simple.sh	ajs365	qw	06/09/2006 11:06:27		1	

- A status of 'qw' is (queued and waiting) – check it again after a few seconds

```
% qstat
```

job-ID	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
149	0.55500	simple.sh	ajs365	r	06/09/2006 11:06:35	all.q@arcsgrid03.newcastle.edu	1	

- The job now has a status of 'r' (running) and is on arcsgrid03
- Check the queue again and see if your job has completed

```
% qstat
```

(no output will be displayed when the job is finished)

NOTE: Sometimes there are dozens of jobs running on the grid. To see just your own jobs use

```
% qstat -u username (using your UNIX account name)
```

- Check for the output of the job

```
% ls -l simple.sh.*
```

```
-rw-r--r-- 1 ajs365 LDAPstaff 58 Jun 9 11:06 simple.sh.o149  
-rw-r--r-- 1 ajs365 LDAPstaff 0 Jun 9 11:06 simple.sh.e149
```

- scriptname.oXXXX - is the standard output of the job numbered XXXX
- scriptname.eXXXX – is the standard error of the job numbered XXXX

- Look at the output file. It should be similar to when the job was run manually on the unix prompt

```
% cat simple.sh.o149
```

```
Fri Jun 9 11:06:35 EST 2006
```

```
Fri Jun 9 11:06:55 EST 2006
```

- If there are any errors (or the program terminated for some reason) they will be logged in the error file.

An example using non-interactive Matlab

- Create a matlab file (example1.m) that we are going to run

Note the exit line at the end is VERY important otherwise matlab will never exit.

```
-----example1.m-----  
% Matlab example1.m program  
  
clear all;  
  
for i = 1:2  
    for j = 1:2  
        x(i,j) = i*j^2;  
    end  
end  
  
z = x^2;  
Z = z'  
exit  
-----example1.m-----
```

- Create a shell script called matlab.sh which will run the file. Note: this assumes that example1.m is in your home directory.

```
#!/bin/sh  
/usr/local/matlab/matlab17/bin/matlab -nodisplay -r example1  
exit
```

- Set the script so that it will execute

```
% chmod 700 matlab.sh
```

- Run it manually on the Unix prompt (on ares) to check that it works

```
% ./matlab.sh
```

Warning:

MATLAB is starting without a display, using internal event queue.
You will not be able to display graphics on the screen.

< M A T L A B >
Copyright 1984-2005 The MathWorks, Inc.
Version 7.1.0.183 (R14) Service Pack 3
August 02, 2005

To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

Z =

```
9 18
36 72
```

- Once the program has correctly run we can now submit the shell script to the grid engine

```
% qsub matlab.sh
```

Your job 150 ("matlab.sh") has been submitted.

- Check that it is queued

```
% qstat
```

job-ID	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
150	0.00000	matlab.sh	ajs365	qw	06/09/2006 11:35:57		1	

- Check that it is running

```
% qstat
```

job-ID	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
150	0.55500	matlab.sh	ajs365	r	06/09/2006 11:36:20	all.q@arcsgrid01.newcastle.edu	1	

- Standard output and standard error files were produced

```
% ls -l matlab.sh.*
```

```
-rw-r--r-- 1 ajs365 LDAPstaff 501 Jun 9 11:36 matlab.sh.o150
-rw-r--r-- 1 ajs365 LDAPstaff  0 Jun 9 11:36 matlab.sh.e150
```

Using the Matlab Distributed/Parallel Compute Engine

Recent versions of Matlab have a distributed compute Engine which allows matlab computations to be performed on a remote machine. This engine has been named from 'Distributed' to 'Parallel' in Matlab 2008a onwards as it can also run locally on multi-cored computers.

There are two ways this toolbox can be used:

- 1) (distributed job) Submit one or more single cpu matlab jobs from within matlab and it runs on remote machines when they become available. This is Similar to the previous matlab command line job.
- 2) (parallel job) Submit a job from inside matlab that uses multiple machines concurrently. This runs the matlab Compute Engine on multiple grid nodes at the same time and the nodes talk to each other in parallel across the network. Your code must be written to communicate between the nodes.

The setup of the grid is as follows. Each grid node 'arcsgridXX' has a copy of the Matlab 2007b distributed compute engine installed on it. The user never runs this version of matlab interactively. The compute engine is started on each node by the grid software. The grid submit node (ares) has a full copy of Matlab 2007b from which users can submit their jobs.

Example Distributed Job

- On ares.newcastle.edu.au run matlab

```
# /usr/local/matlab/matlab2007b/bin/matlab
```

- Check to see if your matlab path is set up to access the toolbox

```
>> which sgeSubmitFcn
```

- It should return

```
/usr/local/matlab/matlab2007b/toolbox/distcomp/sgeSubmitFcn.m
```

- If not then type the following command to rebuild the toolbox cache. The type the which command again

```
>> rehash toolboxcache
```

- Type the following commands to configure the scheduler to talk to the Grid Software and to tell matlab that your files are in your home directory which is commonly shared between all grid nodes (otherwise matlab needs to internally copy your code and data to each node and copy the results back).

```
>> sched = findResource('scheduler', 'type', 'generic');
>> set(sched,'DataLocation','/home/ajs365');           (replace with your home directory)
>> set(sched,'HasSharedFilesystem',true);
>> set(sched,'ClusterMatlabRoot','/grid/local/linux-64bit/matlab17p');
>> set(sched,'SubmitFcn','@sgeSubmitFcn');
```

- Now create a job that consists of three tasks that could be run on 3 different compute nodes. The tasks in this example are just summing numbers and do not communicate between each other.

```
>> j = createJob(sched);
>> createTask(j, @sum, 1, {[1 1]});
>> createTask(j, @sum, 1, {[2 2]});
>> createTask(j, @sum, 1, {[3 3]});
```

- Now submit the job from inside matlab.

```
>> submit(j);
```

- Matlab gives the following output as the jobs are submitted via qsub to the grid software

Submitting task 1

Job output will be written to: /home/ajs365/Job1_Task1.out
 QSUB output: Your job 21359 ("Job1.1") has been submitted

Submitting task 2

Job output will be written to: /home/ajs365/Job1_Task2.out
 QSUB output: Your job 21360 ("Job1.2") has been submitted

Submitting task 3

Job output will be written to: /home/ajs365/Job1_Task3.out
 QSUB output: Your job 21361 ("Job1.3") has been submitted

What is happening underneath?

Matlab is creating code and data files for each task and then using qsub on the unix prompt to submit each task as an individual grid engine job.

In your home directory you will find some files created by matlab for this job.

```
-rw----- 1 ajs365 unother 2193 Aug 8 13:28 Job1.in.mat
-rw----- 1 ajs365 unother   7 Aug 8 13:28 Job1.state.mat
-rw----- 1 ajs365 unother 183 Aug 8 13:28 Job1.out.mat
-rw----- 1 ajs365 unother 181 Aug 8 13:28 Job1.common.mat
-rw----- 1 ajs365 unother 564 Aug 8 13:28 matlab_metadata.mat
drwx----- 2 ajs365 unother 512 Aug 8 13:28 Job1
```

There will also be a folder created called 'Job1' that contains all of the information for the tasks.

```
Job1: ls -l
```

```

-rw----- 1 ajs365 unother 564 Aug 8 13:28 matlab_metadata.mat
-rw----- 1 ajs365 unother 267 Aug 8 13:28 Task1.common.mat
-rw----- 1 ajs365 unother 583 Aug 8 13:28 Task1.in.mat
-rw----- 1 ajs365 unother 546 Aug 8 13:28 Task1.out.mat
-rw----- 1 ajs365 unother 8 Aug 8 13:28 Task1.state.mat
-rw----- 1 ajs365 unother 267 Aug 8 13:28 Task2.common.mat
-rw----- 1 ajs365 unother 582 Aug 8 13:28 Task2.in.mat
-rw----- 1 ajs365 unother 546 Aug 8 13:28 Task2.out.mat
-rw----- 1 ajs365 unother 8 Aug 8 13:28 Task2.state.mat
-rw----- 1 ajs365 unother 267 Aug 8 13:28 Task3.common.mat
-rw----- 1 ajs365 unother 584 Aug 8 13:28 Task3.in.mat
-rw----- 1 ajs365 unother 546 Aug 8 13:28 Task3.out.mat
-rw----- 1 ajs365 unother 8 Aug 8 13:28 Task3.state.mat

```

- Looking at the grid engine side of things three grid engine jobs have been submitted (one for each matlab task)

```
ares~: qstat -u ajs365
```

job-ID	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
21359	0.55250	Job1.1	ajs365	qw	08/08/2008 13:28:50		1	
21360	0.55167	Job1.2	ajs365	qw	08/08/2008 13:28:50		1	
21361	0.55125	Job1.3	ajs365	qw	08/08/2008 13:28:51		1	

Getting your results

If you are not already running matlab then login to ares.newcastle.edu.au

```
# /usr/local/matlab/matlab2007b/bin/matlab
```

- Search for the scheduler and tell it where its jobs directory is located

```
>> sched = findResource('scheduler', 'type', 'generic')
>> set(sched,'DataLocation','/home/ajs365')           (this directory holds the state of jobs)
```

- Now find the status of your matlab jobs that the scheduler knows about

```
>> j= findJob(sched)
```

```
j =
```

```
Job ID 1 Information
```

```
=====
```

```

      UserName : ajs365
      State    : queued
      SubmitTime : Fri Aug 08 13:28:48 EST 2008
      StartTime :
      Running Duration :

```

- Data Dependencies

```
FileDependencies : {}  
PathDependencies : {}
```

- Associated Task(s)

```
Number Pending : 3  
Number Running : 0  
Number Finished : 0  
TaskID of errors :
```

- Now look at the tasks in the jobs

```
>> get(j,'Tasks')
```

```
ans =
```

```
Tasks: 3 by 1  
=====
```

Task ID	State	End Time	Function Name	Error
1	pending		@sum	
2	pending		@sum	
3	pending		@sum	

- If the tasks have not finished we can tell our matlab to wait until the tasks in the job have run

```
>> waitForState(j,'finished')
```

- Then we can collect the results from all tasks in the job

```
>> results = getAllOutputArguments(j)
```

```
results =
```

```
[2]  
[4]  
[6]
```

NOTE: Because the distributed compute Engine software only allows eight concurrent licenses it is best to run serial matlab jobs of one task using the UNIX command line method of matlab which is described in the 'non-interactive matlab' section of this document.

Example Parallel Matlab Job

- The setup for the parallel job is similar to the distributed job except that a Parallel Submit Function is specified and the workers are run on different computers concurrently.

```
>> sched = findResource('scheduler', 'type', 'generic');
>> set(sched, 'DataLocation', '/home/ajs365');
>> set(sched, 'HasSharedFilesystem', true);
>> set(sched, 'ClusterMatlabRoot', '/grid/local/linux-64bit/matlab17p');
>> set(sched, 'ParallelSubmitFcn', @sgeParallelSubmitFcn);
```

- Now we must create the parallel job. In this example we are specifying that exactly four worker processes are used.

```
>> pjob = createParallelJob(sched);
>> set(pjob, 'MaximumNumberOfWorkers', 4);
>> set(pjob, 'MinimumNumberOfWorkers', 4);
>> createTask(pjob, 'rand', 1, {4});
```

- Now we submit this job

```
>> submit(pjob);
```

Job output will be written to: /home/ajs365/Job11.mpiexec.out
QSUB output: Your job 21429 ("Job11") has been submitted

What is happening underneath?

- If you do a qstat you will see that a four way job has been submitted to the grid software. The job starts up the matlab parallel environment on the grid which ‘reserves’ four grid nodes and then passes the list of nodes to matlab which then runs the workers which use SMPD.

```
ares~: qstat -u ajs365
job-ID prior name user state submit/start at queue slots ja-task-ID
-----
21429 0.60125 Job11 ajs365 r 08/12/2008 08:22:52 all.q@arcsgrid13.newcastle.edu 4
```

- The job output file is as follows

```
# cat /home/ajs365/Job11.mpiexec.out
```

```
Starting SMPD on arcsgrid13
```

```
arcsgrid16
```

```
arcsgrid03
```

```
arcsgrid17 ...
```

```
ssh arcsgrid13 "/grid/local/linux-64bit/matlab17p/bin/mw_smpd" -s -phrase MATLAB -port 21429
```

```
ssh arcsgrid16 "/grid/local/linux-64bit/matlab17p/bin/mw_smpd" -s -phrase MATLAB -port 21429
```

```
ssh arcsgrid03 "/grid/local/linux-64bit/matlab17p/bin/mw_smpd" -s -phrase MATLAB -port 21429
```

```
ssh arcsgrid17 "/grid/local/linux-64bit/matlab17p/bin/mw_smpd" -s -phrase MATLAB -port 21429
```

All SMPDs launched

```
"/grid/local/linux-64bit/matlab17p/bin/mw_mpiexec" -phrase MATLAB -port 21429 -l  
-n 4 -machinefile /tmp/21429.1.all.q/machines -genvlist MDCE_DECODE_FUNCTION,MD  
CE_STORAGE_LOCATION,MDCE_STORAGE_CONSTRUCTOR,MDCE_JOB_LOCATION  
"/grid/local/linux-64bit/matlab17p/bin/worker" -parallel
```

- The job then executes and then the matlab workers are shut down as shown in the output file:

```
Stopping SMPD on arcsgrid13 arcsgrid16 arcsgrid03 arcsgrid17 ...  
ssh arcsgrid13 "/grid/local/linux-64bit/matlab17p/bin/mw_smpd" -shutdown -phrase  
MATLAB -port 21429  
ssh arcsgrid16 "/grid/local/linux-64bit/matlab17p/bin/mw_smpd" -shutdown -phrase  
MATLAB -port 21429  
ssh arcsgrid03 "/grid/local/linux-64bit/matlab17p/bin/mw_smpd" -shutdown -phrase  
MATLAB -port 21429  
ssh arcsgrid17 "/grid/local/linux-64bit/matlab17p/bin/mw_smpd" -shutdown -phrase  
MATLAB -port 21429  
Exiting with code: 0
```

- From inside matlab we can check the status of the job

```
>> findTask(pjob)
```

```
ans =
```

```
Tasks: 4 by 1
```

```
=====
```

Task ID	State	End Time	Function Name	Error
1	finished	Aug 12 08:23:09	rand	
2	finished	Aug 12 08:23:09	rand	
3	finished	Aug 12 08:23:09	rand	
4	finished	Aug 12 08:23:09	rand	

If the job is still executing or queued to run we can tell matlab to wait until the job is finished (or we can logout and come back later on).

```
>> waitForState(pjob)
```

When the job is finished we can now collect the results of the four tasks.

```
>> results = getAllOutputArguments(pjob)
```

```
results =
```

```
[4x4 double]
```

```
[4x4 double]
```

```
[4x4 double]
```

```
[4x4 double]
```

```
>> disp(results{1})
    0.5488    0.4237    0.9637    0.5680
    0.7152    0.6459    0.3834    0.9256
    0.6028    0.4376    0.7917    0.0710
    0.5449    0.8918    0.5289    0.0871
```

```
>> disp(results{2})
    0.9173    0.4612    0.2155    0.4621
    0.6839    0.1562    0.4978    0.9846
    0.8661    0.4626    0.2904    0.9587
    0.4809    0.8009    0.9071    0.5795
```

```
>> disp(results{3})
    0.2951    0.7010    0.9143    0.7375
    0.0990    0.3821    0.2740    0.5407
    0.3277    0.9602    0.6484    0.6348
    0.6902    0.7780    0.2781    0.0948
```

```
>> disp(results{4})
    0.3527    0.5647    0.0360    0.8565
    0.9411    0.0864    0.4363    0.6978
    0.3007    0.9689    0.7699    0.7676
    0.4783    0.4288    0.3194    0.3136
```

```
>> celldisp(results) (can also be used to display all results)
```

Parallel Message Passing Example

Usually with parallel jobs there is a master job that sends/receives the data from the worker jobs. This example shows how to pass messages between jobs.

Useful variables:

```
X = numlabs; (the number of labs operating on the current job)
X = labindex ; (which lab number am I - index 1 being the master job)
```

Useful message passing functions ('lab' is equivalent to a worker process):

```
labProbe() (see if any data is available but don't read it)
labBroadcast() (send the data from worker X to all of the lab)
labSend() (send data to a particular lab)
labReceive() (receive data from any or a particular lab)
labBarrier; (blocks all labs until every worker has reached this point)
```

Further information on these functions and example code can be found at www.mathworks.com

- For this example create the following file called 'colsum.m' in your home directory. This is the code that will be executed by all of the workers. As your home directory is accessible across all grid nodes there is no need to use the FileDependencies value of the job (which lists the files that need to be copied to each node before the job is run).

```
function total_sum = colsum
if labindex == 1
    % Send magic square to other labs
    A = labBroadcast(1,magic(numlabs))
else
    % Receive broadcast on other labs
    A = labBroadcast(1)
end

% Calculate sum of column identified by labindex for this lab
column_sum = sum(A(:,labindex))

% Calculate total sum by combining column sum from all labs
total_sum = gplus(column_sum)
```

This function is executed on all worker nodes. Worker #1 (master) broadcasts a 4x4 magic square to each worker. The other three workers read in the data from worker #1. All four workers then sum up their particular column of the matrix. The gplus function returns the addition of column_sum across all workers. The total_sum is the value returned from the worker.

```
>> sched = findResource('scheduler', 'type', 'generic')
>> set(sched,'DataLocation','/home/ajs365')
>> set(sched,'HasSharedFilesystem',true)
>> set(sched,'ClusterMatlabRoot','/grid/local/linux-64bit/matlab17p')
>> set(sched,'ParallelSubmitFcn','@sgeParallelSubmitFcn')
>> t = createTask(pjob, @colsum, 1, {})
```

t =

Task ID 1 from Job ID 12 Information

=====

```
State : pending
Function : @colsum
StartTime :
Running Duration :
```

- Task Result Properties

```
ErrorIdentifier :
ErrorMessage :
```

```
>> submit(pjob)
```

Job output will be written to: /home/ajs365/Job12.mpiexec.out
 QSUB output: Your job 21430 ("Job12") has been submitted

```
>> findTask(pjob)
```

```
ans =
```

```
Tasks: 4 by 1
```

```
=====
```

Task ID	State	End Time	Function Name	Error
1	finished	Aug 12 09:06:55	@colsum	
2	finished	Aug 12 09:06:56	@colsum	
3	finished	Aug 12 09:06:56	@colsum	
4	finished	Aug 12 09:06:56	@colsum	

- The results show that each of the workers got the same answer in summing the matrix.

```
>> results = getAllOutputArguments(pjob)
```

```
results =
```

```
[136]  
[136]  
[136]  
[136]
```

Cleaning up after Matlab jobs

Matlab keeps a record of the jobs/tasks so they need to be removed after you have collected your results. This can be done by finding the job object from the scheduler object and then removing all finished jobs.

```
>> sched
```

```
sched =
```

```
GENERIC Scheduler Information
```

```
=====
```

Type	: generic
ClusterOsType	: unix
DataLocation	: /home/ajs365
HasSharedFilesystem	: true

```
- Assigned Jobs
```

Number Pending : 3
Number Queued : 4
Number Running : 0
Number Finished : 3

- Scheduler Specific Properties

ClusterMatlabRoot : /grid/local/linux-64bit/matlab17p
ParallelSubmitFcn : @sgeParallelSubmitFcn
SubmitFcn : []

- This tells us that there are jobs pending, jobs queued and 3 jobs that have finished.

```
>> finished_jobs = findJob(sched, 'State', 'finished');  
>> destroy(finished_jobs); (removes jobs from the scheduler)  
>> clear(finished_jobs); (deletes the object from matlab)
```

- When we display the sched object again we find that record of the finished jobs have been removed

- Assigned Jobs

Number Pending : 3
Number Queued : 4
Number Running : 0
Number Finished : 0

An example using non-interactive JAVA

Java is installed in the grid in /usr/java/jdk1.5.0_05/bin or /usr/java/jdk1.6.0_02/bin so you will to change your scripts to point to this directory

- Login to the grid submit node ares.newcastle.edu.au and put the java files in your home directory

```
# cat hello.java
class hello{
    public static void main (String args[]) {
        System.out.print("Hello World \n");
    }
}
```

- Compile the java program using the java on the grid

```
# /usr/java/jdk1.5.0_05/bin/javac hello.java
```

- Do a test run of the java program – at least check that it starts

```
# /usr/java/jdk1.5.0_05/bin/java hello
```

Hello World

- Create a UNIX shell script that will set up all of the java variables and run the program

```
# cat javajob.sh
#!/bin/sh
/usr/java/jdk1.5.0_05/bin/java hello
exit
```

- Make the script executable

```
# chmod 700 javajob.sh
```

- Submit the java job to the grid queue

```
# qsub javajob.sh
Your job 3597 ("javajob.sh") has been submitted
```

- Check the queue to see if the job is running

```
# qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-----
3597 0.00000 javajob.sh ajs365 qw 03/30/2007 14:49:02 1
```

- When the job is finished two files (STDOUT & STDERR) are created in your home directory

```
-rw-r--r-- 1 ajs365 LDAPstaff 13 Mar 30 14:49 javajob.sh.o3597  
-rw-r--r-- 1 ajs365 LDAPstaff  0 Mar 30 14:49 javajob.sh.e3597
```

- Check that the output is ok

```
# cat javajob.sh.o3597
```

```
Hello World
```

An example using non-interactive R

- Firstly create an example R script in your UNIX home directory.

```
# cat example.R
x <- pi
floor(x)
ceiling(x)
```

Note: A more complicated script would require reading in from an input file, processing the data and then writing to a unique output file. It is best to let the simulation write to a uniquely named output file rather than letting the grid software capture ‘standard output’ in the grid job log file because it makes tracking down failed jobs easier.

- Create a UNIX shell script that will run the R job. All paths and environment variables must be specified as this script is copied to the grid node to run from a temporary area.
- This script runs example.R in batch mode and stores the output in a file called ‘output.txt’. The R workspace is not saved at the end of the run.

```
# cat runR.sh
```

```
#!/bin/sh
/grid/local/linux-64bit/R/R-2.5.1/bin/R CMD BATCH --no-save $HOME/example.R output.txt
exit 0
```

- Make the script executable

```
# chmod 700 runR.sh
```

- Submit the job

```
# qsub runR.sh
```

Your job 13721 ("runR.sh") has been submitted

- Check the queue

```
# qstat
```

```
13721 0.55500 runR.sh ajs365 qw 05/22/2008 12:07:29 1
```

- The following files will be created in your home directory

```
-rw-r--r-- 1 ajs365 LDAPstaff 810 May 22 12:09 output.txt
-rw-r--r-- 1 ajs365 LDAPstaff 0 May 22 12:09 runR.sh.e13721
-rw-r--r-- 1 ajs365 LDAPstaff 0 May 22 12:09 runR.sh.o13721
```

- Note that the grid log output (o) and error (e) files are zero bytes long. All output from R has been redirected to ‘output.txt’.

Deleting a job from the grid

- If a job has an error or been in incorrectly queued it may be deleted using the qdel command

```
% qstat
```

```
job-ID prior name user state submit/start at queue slots ja-task-ID
-----
151 0.55500 matlab.sh ajs365 Eqw 06/09/2006 11:35:57
```

```
% qdel 151
```

Running a job with email notification

- Several jobs can be submitted using the qsub command. The grid engine software will submit them as free nodes become available. It is possible to get the grid software to send an email notification when the job is started/finished or aborted.

(Haven't worked out the correct command line yet as it only sends email 50% of the time ☹)

Batch Queueing multiple jobs with different datasets

- The qsub command can take additional arguments and pass them to your shell script. If your shell script can handle arguments then you should be able to do the following on the unix prompt without waiting for the previous job to finish.

```
% qsub myprocess.sh infile1 outfile1
% qsub myprocess.sh infile2 outfile2
% qsub myprocess.sh infile3 outfile3
% qsub myprocess.sh infile4 outfile4
```

Note: In this case there could be up to four instances of “myprocess.sh” running concurrently so you will need to make sure that all temporary files are unique and will not be overwritten by other instances. The input and output files are assumed to be located in your home directory (unless you specify the full path on the command line).

Using PVM on the Grid

How PVM is run on the grid

The grid is configured to have a parallel environment. At present the size of the parallel environment is a maximum of 8 slots. This allows you to have a job running concurrently on up to 8 cpus. If there are not enough available slots on the grid then your job will wait in the queue until enough are available. The number of slots you require for your PVM job is specified when you submit your job. You can also set an upper limit of slots so that at submit time your PVM job will use 'up to' that number of slots if they are available at the time.

Setting up for PVM

- After logging in to the grid submit node (ares) you will need to set up these variables in the .userrc file in your home directory:

```
export PVM_ROOT=/usr/share/pvm3
export PVM_ARCH=LINUX86_64
export PVM_RSH=/usr/bin/ssh
export PATH=$PATH:$PVM_ROOT/lib
```

NOTE: The ARCH variable for PVM is LINUX86_64 but the ARCH variable for the grid is lx24-
amd64

- PVM runs jobs on remote nodes via RSH so you will need to check that RSH works you're your account to the grid nodes. The master node for the PVM job can be any grid node so you must be able to connect between any two nodes in the grid – check – ENV variable override
- You will need to edit your .rhosts file add in the following hostnames:

```
ares.newcastle.edu.au
arcsgrid.newcastle.edu.au
arcsgrid01.newcastle.edu.au
....
arcsgrid14.newcastle.edu.au
```

- You will need to set the permissions on the .rhosts file so that rsh will not complain.

```
% chmod o-rwx .rhosts
```

- Check that you can rsh from ares to arcsgrid01 and that a password is not required when you login.

```
% rsh arcsgrid01
```

- logout of arcsgrid01

Compiling and testing a PVM example

- Create the directory structure in your own account called

```
~/pvm3/bin/LINUX86_64/
```

- copy the following PVM example source code from /usr/share/pvm3/examples/ into
~/pvm3/bin/LINUX86_64/

```
hello.c and hello_other.c
```

```
# cd ~/pvm3/bin/LINUX86_64/
```

- Compile them with the PVM libraries

```
gcc -o hello hello.c -I/usr/share/pvm3/include -L/usr/share/pvm3/lib/LINUX86_64/ -lpvm3
gcc -o hello_other hello_other.c -I/usr/share/pvm3/include -L/usr/share/pvm3/lib/LINUX86_64/ -lpvm3
```

- Check that the two executables have been created

```
-rwx----- 1 ajs365 LDAPstaff 184860 Aug 31 13:09 hello
-rwx----- 1 ajs365 LDAPstaff 184915 Aug 31 13:09 hello_other
```

- Run the pvm console and test the program

```
% pvm
pvm> spawn -> hello          (-> means redirect output to console)
i'm t40015
from t40016: hello, world from ares
pvm>halt                    (stops all pvm daemons and exits the program)
%
```

- If this works correctly then you should be able to run PVM on the grid

Running the Grid Engine PVM Example

The Sun grid Engine provides a SPMD (single process multiple data) example.

- Copy the example sun grid script into your own home directory

```
cp /grid/sge6/pvm/pvm.sh ~/pvm.sh
```

- The contents of the file is as follows:

```
-----pvm.sh-----
#!/bin/csh -f
#
```

(c) 2004 Sun Microsystems, Inc. Use is subject to license terms.

```
#
# sample pvm job
# this script starts the pvm sample spmd
# note that this job uses the pvm group communication
#
# our name
#$ -N PVM_Job
# pe request
#$ -pe pvm 16,8,4-1
#$ -v SGE_QMASTER_PORT,DISPLAY
#$ -S /bin/csh
# -----
```

```
echo "Got $NSLOTS slots."
```

```
/bin/echo Here I am on a $ARC called `hostname`.
```

```
# spmd requests $NSLOTS on __different__ hosts
$SGE_ROOT/pvm/bin/$ARC/spmd $NSLOTS
-----pvm.sh-----
```

- Some points to note about this script are:
 - Extra arguments to the grid engine are set from within the script
 - The job name
 - The parallel environment name
 - The number of slots we would like (1-4, 8 or 16)
 - \$ARC and \$SGE_ROOT are set up when you submit the job
 - \$NSLOTS is determined at submit time and can be 1 thru 4, 8 or 16.
 - The executable is in the grid engine pvm directory (not your own account)
- Now submit the job

```
% qsub pvm.sh
Your job 1848 ("PVM_Job") has been submitted
```

- The job is now waiting in the queue for 8 slots to become available

```
% qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-----
1848 0.00000 PVM_Job ajs365 qw 09/01/2006 10:45:40 8
```

- The PVM job is now running with master node arcsgrid01 and seven slave nodes

```
% qstat
```

job-ID	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
1848	0.60500	PVM_Job	ajs365	r	09/01/2006 10:45:55	all.q@arcsgrid01.newcastle.edu	8	

- In addition to the 'stdout' and 'stderr' of a normal job. There are an additional two job logs in your home directory. The 'p' files are the output and errors when a parallel environment is used.

```
-rw-r--r-- 1 ajs365 LDAPstaff 797 Sep 1 10:48 PVM_Job.o1848
-rw-r--r-- 1 ajs365 LDAPstaff 26 Sep 1 10:47 PVM_Job.e1848
-rw-r--r-- 1 ajs365 LDAPstaff 283 Sep 1 10:48 PVM_Job.po1848 ←
-rw-r--r-- 1 ajs365 LDAPstaff 0 Sep 1 10:45 PVM_Job.pe1848 ←
```

- The PVM environment output log (PVM_Job.po1848) contains:

```
/grid/sge6/default/spool/arcsgrid01/active_jobs/1848.1/pe_hostfile arcsgrid01.newcastle.edu.au
/usr/share/pvm3
/tmp/pvmtmp029140.0
start_pvm: enrolled to local pvmd
start_pvm: got 8 hosts
/grid/sge6/default/spool/arcsgrid01/active_jobs/1848.1/pe_hostfile arcsgrid01.newcastle.edu.au
```

- There are no PVM errors in (PVM_Job.pe1848)
- The job output log (PVM_Job.o1848) contains the output from the executable:

Warning: no access to tty (Bad file descriptor).

Thus no job control in this shell.

Got 8 slots.

Here I am on a lx24-amd64 called arcsgrid01.

spmd: arcsgrid01: me = 0 mytid = 0x40002

spmd: 0x40000 arcsgrid01 LINUXX86_64 1000

spmd: 0x80000 arcsgrid09.newcastle.edu.au LINUXX86_64 1000

spmd: 0xc0000 arcsgrid03.newcastle.edu.au LINUXX86_64 1000

spmd: 0x100000 arcsgrid06.newcastle.edu.au LINUXX86_64 1000

spmd: 0x140000 arcsgrid05.newcastle.edu.au LINUXX86_64 1000

spmd: 0x180000 arcsgrid04.newcastle.edu.au LINUXX86_64 1000

spmd: 0x1c0000 arcsgrid07.newcastle.edu.au LINUXX86_64 1000

spmd: 0x200000 arcsgrid02.newcastle.edu.au LINUXX86_64 1000

spmd: pvm_spawn("/grid/sge6/pvm/bin/lx24-amd64/spmd", ..)

spmd: src_tid = 0x200001

spmd: dest_tid = 0x40003

spmd: (0) succeed sending token

spmd: (0) got token

spmd: (0) token ring done

- The job error log (PVM_Job.e1848) contains a few spmd debug messages that can be ignored.

Running my own PVM job

- Place your PVM source code in `~/pvm3/bin/LINUX86_64` and compile it with the libraries shown above
- The `pvm.sh` script can be copied and modified to run your job.

Using MPI on the Grid

The implementation of MPI that is installed on the arcs grid is MPICH2. It is installed in /grid/local/linux-64bit/mpich2/bin in is available on all of the grid nodes.

Add this to your \$PATH in your ~/.user_profile

```
export PATH=/grid/local/linux-64bit/mpich2/bin:$PATH
```

Initialise ssh

The MPI environment requires the ability to ssh between grid nodes. From ares make sure that you can ssh across to each of the grid nodes without a password.

Compiling an MPI job (an example found on the internet)

```
# cat mpi-pi.c
=====
#include "mpi.h"
#include <math.h>

int main(argc,argv)
int argc;
char *argv[];
{
    int done = 0, n=50000, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    printf("Numprocs = %d Iterations = %d\n",numprocs,n);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    {
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

        h = 1.0 / (double) n;
        sum = 0.0;
        for (i = myid + 1; i <= n; i += numprocs) {
            x = h * ((double)i - 0.5);
            sum += 4.0 / (1.0 + x*x);
        }
        mypi = h * sum;

        MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
            MPI_COMM_WORLD);
    }
}
```

```

    if (myid == 0)
        printf("pi is approximately %.16f, Error is %.16f\n",
            pi, fabs(pi - PI25DT));
    }
    MPI_Finalize();
    return 0;
}

```

```
# mpicc -o mpi-pi mpi-pi.c
```

Example MPI Grid job

As with most grid jobs the scripts and executable programs are best kept in your home directory.

The parallel MPI environment is not running all of the time. It must be set up on 'n' nodes allocated by the grid (a mpd.hosts file is generated for each job), then your job is executed these nodes, then the MPI environment is shut down.

- The following script reads the lists of hosts given to it from the grid engine, boots MPI then runs the job on all of the hosts then shuts down MPI.

```
# cat mpi-grid.sh
```

```
#!/bin/sh
```

```
cat $PE_HOSTFILE | awk -F. '{print $1}' > mpd.hosts.$JOB_ID
```

```
/grid/local/linux-64bit/mpich2/bin/mpdboot -n $NHOSTS --verbose -f mpd.hosts.$JOB_ID
```

```
/grid/local/linux-64bit/mpich2/bin/mpirun -n $NHOSTS /home/ajs365/mpi-pi
```

```
/grid/local/linux-64bit/mpich2/bin/mpdallexit
```

```
exit
```

- Use the qsub command to submit this job to the mpich2 parallel environment (using 8 nodes)

```
# qsub -clear -pe mpich2 8 -q all.q mpi-grid.sh
```

```
# qstat
```

job-ID	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
10157	0.00000	mpi-grid.s	ajs365	qw	03/06/2008 09:14:01		8	

```
# cat mpi-grid.sh.o10157
```

```
running mpdallexit on arcsgrid09
```

LAUNCHED mpd on arcsgrid09 via
RUNNING: mpd on arcsgrid09
LAUNCHED mpd on arcsgrid03 via arcsgrid09
LAUNCHED mpd on arcsgrid01 via arcsgrid09
LAUNCHED mpd on arcsgrid13 via arcsgrid09
LAUNCHED mpd on arcsgrid08 via arcsgrid09
RUNNING: mpd on arcsgrid01
RUNNING: mpd on arcsgrid03
LAUNCHED mpd on arcsgrid10 via arcsgrid01
LAUNCHED mpd on rhea via arcsgrid01
LAUNCHED mpd on arcsgrid02 via arcsgrid01
RUNNING: mpd on arcsgrid08
RUNNING: mpd on arcsgrid13
RUNNING: mpd on arcsgrid02
RUNNING: mpd on arcsgrid10
RUNNING: mpd on rhea
Numprocs = 8 Iterations = 500000
Numprocs = 8 Iterations = 500000
Numprocs = 8 Iterations = 500000
Numprocs = 8 Iterations = 500000
Numprocs = 8 Iterations = 500000
Numprocs = 8 Iterations = 500000
Numprocs = 8 Iterations = 500000
Numprocs = 8 Iterations = 500000
pi is approximately 3.1415926535901217, Error is 0.0000000000003286

Array Jobs

If the same job needs to be run multiple times then the array job option to qsub can be used. This will submit the same job 'n' times to the queue but also pass to each job a unique numerical id (or seed).

This type of job is best suited to:

- jobs that have multiple datasets that are stored in files indexed numerically
- jobs that have been partitioned into smaller parts and the partitions are uniquely addressable
- jobs that use algorithms that need a seed to start them off

NOTE: As the jobs are considered identical by the grid engine (and can be executing on multiple grid nodes at the same time) your job will need to be able to produce a unique output file and any temporary files must have unique names so that they will not be overwritten.

The unique task identifier is passed to each job in the environment variable SGE_TASK_ID.

- For example submitting 10 identical tasks numbered 1 thru 10 can be done using this command:

```
# qsub -clear -q all.q -t 1-10 myjob.sh
```

- Looking at the queue now shows a 'ja-taskID' of 1-10 which means there are 10 jobs.

```
# qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-----
13618 0.00000 picalc.sh ajs365 qw 05/06/2008 09:09:45 1 1-10:1
```

- The ten jobs are now running with the same grid engine job ID they have a unique ja-task-ID

```
# qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-----
13618 0.55500 picalc.sh ajs365 r 05/06/2008 09:09:56 all.q@arcsgrid02.newcastle.edu 1 10
13618 0.55500 picalc.sh ajs365 r 05/06/2008 09:09:56 all.q@arcsgrid07.newcastle.edu 1 2
13618 0.55500 picalc.sh ajs365 r 05/06/2008 09:09:56 all.q@arcsgrid09.newcastle.edu 1 6
13618 0.55500 picalc.sh ajs365 r 05/06/2008 09:09:56 all.q@arcsgrid10.newcastle.edu 1 4
13618 0.55500 picalc.sh ajs365 r 05/06/2008 09:09:56 all.q@arcsgrid11.newcastle.edu 1 7
13618 0.55500 picalc.sh ajs365 r 05/06/2008 09:09:56 all.q@arcsgrid13.newcastle.edu 1 8
13618 0.55500 picalc.sh ajs365 r 05/06/2008 09:09:56 all.q@arcsgrid15.newcastle.edu 1 9
13618 0.55500 picalc.sh ajs365 r 05/06/2008 09:09:56 all.q@arcsgrid16.newcastle.edu 1 5
13618 0.55500 picalc.sh ajs365 r 05/06/2008 09:09:56 all.q@arcsgrid18.newcastle.edu 1 1
13618 0.55500 picalc.sh ajs365 r 05/06/2008 09:09:56 all.q@ares 1 3
```

The array parameters can also have a step size (ie the array task IDs are incremented by this value). For example submitting 6 identical tasks with array task IDs of 1000,1002,1004,1006,1008,1010 (a step size of 2) can be done using the following command:

```
# qsub -clear -q all.q -t 1000-1010:2 myjob.sh
```

- Looking at the queue now shows a 'ja-taskID' of 1000-1010:2

```
# qstat
```

job-ID	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
13619	0.00000	picalc.sh	ajs365	qw	05/06/2008 09:14:06		1	1000-1010:2

- The six jobs are now running with the same grid engine job ID they have a unique ja-task-ID starting at 1002 (ending at 1010) and having a step size of 2

```
# qstat
```

job-ID	prior	name	user	state	submit/start at	queue	slots	ja-task-ID
13619	0.55500	picalc.sh	ajs365	r	05/06/2008 09:14:11	all.q@arcsgrid01.newcastle.edu	1	1002
13619	0.55500	picalc.sh	ajs365	r	05/06/2008 09:14:11	all.q@arcsgrid05.newcastle.edu	1	1006
13619	0.55500	picalc.sh	ajs365	r	05/06/2008 09:14:11	all.q@arcsgrid08.newcastle.edu	1	1004
13619	0.55500	picalc.sh	ajs365	r	05/06/2008 09:14:11	all.q@arcsgrid12.newcastle.edu	1	1010
13619	0.55500	picalc.sh	ajs365	r	05/06/2008 09:14:11	all.q@arcsgrid14.newcastle.edu	1	1000
13619	0.55500	picalc.sh	ajs365	r	05/06/2008 09:14:11	all.q@arcsgrid17.newcastle.edu	1	1008

There are three other environment variables that are passed to an array job:

- SGE_TASK_FIRST – is the first ID in the array
- SGE_TASK_LAST – is the last ID in the array
- SGE_TASK_STEPSIZE – is the step size

NOTE: Please be careful with array jobs as it is quite easy to submit 10000 jobs to the grid!

Other Grid Queues

The queue name all.q is the default queue. At present there are two other queues available using teaching lab computers. CUP time is available on these computers each night between 10:30pm and 7am and also on weekends and term holiday time (when the labs are locked).

The es409.q queue uses 33 lab PCs that run Solaris 10 x86. Linux binaries will not run on this system. The PCs have 2GB of ram each so the queue is suitable for small jobs that run with java or for programs that only run on Solaris x86.

The queue vm.q is a 'virtual queue' and uses the CPUs of Windows XP lab computers in non teaching times (ie when the lab rooms are physically closed). It consists of computers that have Redhat Linux installations as a guestOS running inside Vmware that is running inside windows XP lab computers.

The Redhat guestOS is set to boot each night at 10:30pm and run until 7:00am each morning when it is shut down. The Redhat guestOS is also set to run continuously for 50 hours each weekend and continuously using University holiday times (when some of the labs are closed).

This queue is best suited to computation jobs that use less than 512MB ram and need to be run multiple times and will finish in under 8 hours (or 50 hours in the case of weekends).

Lab PCs can be turned off or rebooted at any time. The grid engine will detect when a node has gone down and can automatically resubmit the job (using the same jobid) to the queue without any user intervention. Adding "-r yes" will allow jobs to be resubmitted by the grid engine.

```
# qsub -clear -q es409.q -r yes myjob.sh
```

Submitting jobs to multiple queues

A job can be submitted to more than one queue. When it is the jobs turn to run whichever queue has a free slot (cpu) at the time will execute the job.

For example: You are using vm.q over a weekend but the machines are shut down each Monday morning. Submitting to vm.q and all.q will allow any unexecuted jobs to then be run on all.q when vm.q is shut down during daytime teaching.

```
# qsub -clear -q vm.q,all.q myjob.sh
```

NOTE: Any jobs that are running on vm.q when it is shut down are deemed by the grid engine to still be allocated to the machine and will not be requeued until the machine starts up again (the next night). They can be removed by the user using the qdel command.

Compiling my own software (C, C++, Fortran)

There are additional nodes that are not part of the grid that can be used for test-runs, software development and compiling. There are of a similar configuration to the grid nodes. If you have your own source code then put it in your home directory and compile it up there. If you have a Linux product and you will be the only one using it – it will probably be best to install it in your own account. If the product will be run by multiple users and there are enough licenses.

If you have your own software that will be used by a number of users then please contact arcsgrid@newcastle.edu.au with details so that it can be installed on all grid nodes (license permitting).

Once you have software running from the unix prompt then you will need to write a shell script for it. After this is done then it can be run on the grid submit node (ares.newcastle.edu.au).

The machine **rhea.newcastle.edu.au** (quad opteron) can be used for testing and software development.

Compiling C/C++ Code

- Using the GNU Compiler

```
rhea~: gcc hello.c
rhea~: a.out
hello
```

- To run the intel C/C++ on rhea the environment variables must be set up.

```
rhea~: source /opt/intel/cc/10.0.023/bin/iccvars.sh
```

```
rhea~: icc hello.c
rhea~: a.out
hello
```

Compiling Fortran Code

- Using the GNU fortran 77 compiler

```
rhea~: g77 hello.f
rhea~: a.out
Hello World
```

- Using the Intel Fortran 90 compiler

```
rhea~: source /opt/intel/fc/10.1.011/bin/ifortvars.sh
rhea~: ifort helloworld.f90
rhea~: a.out
Hello World
```

Grid Etiquette

The ARCS grid is a shared facility and can have users from any of the faculties within the University. The grid has 40 opteron cores (slots) so this means that if a user submits 40 or more jobs (or an array job) then the whole grid can be used by one user and other users will have to wait until there are less than 40 of the first users jobs left to run before their job will run.

The grid scheduling algorithm is a FIFO queue – that is that if someone submits a job(s) before your job then it will run before your job. If you submit 10 jobs then they will be run in the sequence that they were submitted.

Recently the grid scheduler has been modified so that if you submit a job and you do not have any jobs currently running on the grid then you will go to the top of the pending queue and get the next free slot when it becomes available.

If you run a parallel job that requires 4 slots the job will be queued until four slots become available at the same time. Unfortunately the grid software doesn't reserve you one slot then wait for a second one then a third one etc.

If you have special requirements eg jobs that run for several days or you need to run hundreds of jobs then please contact the grid administrators (contact details on next page) who will see what they can do. There are other grids available using lab computers that are available for overnight/weekend use.

If you do have hundreds of jobs to run you can submit the job(s) at a lower priority so that other users who have single jobs will run at normal priority and will run before your jobs.

- This example shows how to submit 100 jobs at low priority (-200)

```
# qsub -clear -q all.q -t 1-100 -p -200 my_array_job.sh
```

The default job priority is 0 and it can range from (-1023 to 1024) only the grid administrator can increase the priority of a job.

- This example shows how to hold a job with id 23433 that is queued so that it will not run

```
# qalter -h u 23433
```

- To unhold a job with id 23433

```
# qalter -h U 23433
```

Held jobs are given a state of 'h' when shown in the qstat list

Further Information about the Grid

- The grid queuing system documentation can be found by doing

```
% man qsub      or   % man qstat  
% qsub -help    or   % qstat -help
```

Contact arcsgrid@newcastle.edu.au if you have any technical problems using the Grid.

For all other calls please contact the service desk on phone extension 17000 or email to 17000@newcastle.edu.au