Configuring a New Parallel Environment

By templedf on [Mar 23, 2007](https://blogs.oracle.com/templedf/entry/configuring_a_new_parallel_environment)

Since this seems to be a regular topic on the [user mailing list](http://gridengine.sunsource.net/servlets/SummarizeList?listName=users), here's a quick guide to setting up a parallel environment on [Grid Engine](http://gridengine.sunsource.net/):

1. First, create/borrow/steal the startup and shutdown scripts for the parallel environment you're using. You can find MPI and PVM scripts in the $SGE\_ROOT/mpi and $SGE\_ROOT/pvm directories, respectively. If you cannot find scripts for your parallel environment, you'll have to create them. The startup script must prepare the parallel environment for being used. With most MPI implementations, that's just a matter of creating a "machines" file that lists the machines which are to run the parallel job. The shutdown script must clean up after the parallel job's execution. The MPI shutdown script just deletes the "machines" file.
2. Next, you have to tell Grid Engine about your parallel environment. You can do that interactively with*qmon* or *qconf -ap <pe\_name>* or you can write the data to a file and use *qconf -Ap <file\_name>*. For an example of what such a file would look like, see $SGE\_ROOT/mpi/mpi.template or $SGE\_ROOT/pvm/pvm.template.

Let's look at what the parallel environment configuration contains.

|  |
| --- |
| pe\_name template  slots 0  user\_lists NONE  xuser\_lists NONE  start\_proc\_args /bin/true  stop\_proc\_args /bin/true  allocation\_rule $pe\_slots  control\_slaves FALSE  job\_is\_first\_task FALSE  urgency\_slots min |

* + pe\_name - the name by which the parallel environment will be known to Grid Engine
  + slots - the maximum number of job slots that the parallel environment is allowed to occupy at once
  + users\_lists - an ACL specifying the users who are allowed to use the parallel environment. If set to NONE, it means any user can use it
  + xusers\_list - an ACL specifying the users who are **not** allowed to use the parallel environment. Users in both the users\_list and xusers\_list are **not** allowed to use the parallel environment
  + start\_proc\_args - the path to the startup script for the parallel environment followed by any needed arguments. Grid Engine provides some inline variables that you can use as arguments:
    - $pe\_hostfile - the path to a file written by Grid Engine which contains information about how and where the parallel job should be run
    - $host - the host on which the parallel environment is being started
    - $job\_owner - the name of the user who owns the parallel job
    - $job\_id - the id of the parellel job
    - $job\_name - the name of the parallel job
    - $pe - the name of the parallel environment
    - $pe\_slots - the number of job slots assigned to the job
    - $queue - the name of the queue in which the parallel job is running

The value of this setting is the command that will be run to start the parallel environment for every parallel job.

* + stop\_proc\_args - the path to the shutdown script for the parallel environment followed by any needed arguments. The same inline variables are available as with start\_proc\_args.
  + allocation\_rule - this setting controls how job slots are assigned to hosts. It can have four possible values:
    - a number - if set to a number, Grid Engine will assign that many slots to the parallel job on each host until the assigned number of job slots is met. Setting this attribute to 1, for example, would mean that the job gets a single job slot on each host where it is assigned. Grid Engine will not assign the job more job slots than the number of assigned hosts multiplied by this attribute's value.
    - $fill\_up - use all of the job slots on a given host before moving to the next host
    - $round\_robin - select one slot from each host in a round-robin fashion until all job slots are assigned. This setting can result in more than one job slot per host.
    - $pe\_slots - place all the job slots on a single machine. Grid Engine will only schedule such a job to a machine that can host the maximum number of slots requested by the job. (See below.)
  + control\_slaves - this setting tells Grid Engine whether the parallel environment integration is "tight" or "loose". See your parallel environment's documentation for more details.
  + job\_is\_first\_task - this setting tells Grid Engine whether the first task of the parallel job is actually a job task or whether it's just there to kick off the rest of the jobs. This setting is also determined by your parallel environment integration.
  + urgency\_slots - this setting affect how resource requests affect job priority for parallel jobs. The values can be "min," "max," "avg," or a number. For more information about resource-based job priorities, see this [white paper](http://www.sun.com/blueprints/1005/819-4325.html)

For more information about these settings, see the [sge\_pe](http://gridengine.sunsource.net/nonav/source/browse/~checkout~/gridengine/doc/htmlman/htmlman5/sge_pe.html) man page.

1. The next step is to enable your parallel environment for the queues where it should be available. You can add the parallel environment to a queue interactively with *qmon* or *qconf -mq <queue>* or in a single action with *qconf -aattr queue pe\_list <pe\_name> <queue>*.
2. Now you're ready to test your parallel environment. Run *qsub -pe <pe\_name> <slots>*. Aside from the usual output and error files (<job\_name>.o<job\_id> and <job\_name>.e<job\_id>, respectively), you should also look for the parallel environment startup output and error files, <job\_name>.po<job\_id> and <job\_name>.pe<job\_id>.

That's all there is to it! Just to make sure we're clear on everything, let's do an example. Let's create a parallel environment that starts up an [RMI](http://java.sun.com/javase/6/docs/technotes/guides/rmi/index.html) registry and stores the port number in a file so that the job can find it.

First thing we have to do is write the startup and shudown scripts for the RMI parallel environment. Here's what they look like:

|  |
| --- |
| **rmi\_startup.sh** |
| #!/bin/sh  # $TMPDIR and $JOB\_ID are set by Grid Engine automatically  # Borrowed from $SGE\_ROOT/mpi/startmpi.sh  PeHostfile2MachineFile()  {  cat $1 | while read line; do  host=`echo $line|cut -f1 -d" "|cut -f1 -d"."`  nslots=`echo $line|cut -f2 -d" "`  i=1  while [ $i -le $nslots ]; do  echo $host  i=`expr $i + 1`  done  done  }  # get arguments  pe\_hostfile=$1  # ensure pe\_hostfile is readable  if [ ! -r $pe\_hostfile ]; then  echo "$me: can't read $pe\_hostfile" >&2  exit 1  fi  # create machines file  machines="$TMPDIR/machines"  PeHostfile2MachineFile $pe\_hostfile >> $machines  # We use ports 40000-40999  port=`expr \\( $JOB\_ID % 1000 \\) + 40000`  # Start the registry  /usr/java/bin/rmiregistry $port &  # Save the registry's PID so that we can stop it later  echo $! > $TMPDIR/pid  # Save the port number so the job can find it  echo $port > $TMPDIR/port |
| **rmi\_shutdown.sh** |
| #!/bin/sh  # $TMPDIR is set by Grid Engine automatically  # Get the registry's PID  pid=`cat $TMPDIR/pid`  # Kill the registry  kill $pid  # Clean up the files the startup script created  rm $TMPDIR/pid  rm $TMPDIR/port  rm $TMPDIR/machines |

Next thing we have to do is add our parallel environment to Grid Engine. First we create a file, say /tmp/rmi\_pe, with the following contents:

|  |
| --- |
| pe\_name rmi  slots 4  user\_lists NONE  xuser\_lists NONE  start\_proc\_args /home/dant/rmi\_startup.sh $pe\_hostfile  stop\_proc\_args /home/dant/rmi\_shutdown.sh  allocation\_rule $round\_robin  control\_slaves TRUE  job\_is\_first\_task FALSE  urgency\_slots min |

Note that control\_slaves is true and job\_is\_first\_task is false. Because we're writing the integration scripts, the choice is somewhat arbitrary, but it affects how the job scripts must be written, as we'll see below. It also affect whether the qmaster is able to keep accounting records on the slave tasks. If control\_slaves is false, the qmaster is have no records of how much resources the slaves tasks consumed.

Now we add the parallel environment with *qconf -Ap /tmp/rmi\_pe*. We could have skipped a step by running *qconf -ap rmi* and entering the data in the editor that comes up, but they way we've done it here is scriptable.

The next step is to add our parallel environment to our queue with *qconf -aattr queue pe\_list rmi all.q*. Again, we could have run *qconf -mq all.q* and edited the pe\_list attribute in the editor, but the way we've done it is scriptable.

Last thing to do is test out our parellel environmemt. First we need a job script:

|  |
| --- |
| #!/bin/sh  #$ -S /bin/sh  port=`cat $TMPDIR/port`  qrsh=$SGE\_ROOT/bin/$ARC/qrsh  cat $TMPDIR/machines | while read host; do  $qrsh -inherit $host /usr/bin/java -cp ~/rmi.jar RMIApp $port &  done |

Let's look at this job script for a moment. The first thing to notice is the use of *qrsh -inherit*. The -inherit switch is specifically for kicking off slave tasks. It requires that the target host name be supplied. In order to get the target host name, we read the machines file that the startup script generated from the one Grid Engine supplied.

The second thing to notice is how ugly the use of *qrsh -inherit* is. RMI is not really a parallel environment. It's a communications framework. It doesn't do the work of kicking off remote processes for you. So, instead, we have to do it ourselves in the job script. With a true parallel environment, like any of the MPI flavors, the framework also takes care of starting the remote processes, often through rsh. In the MPI scripts included with Grid Engine, an rsh wrapper script is included, which transparently replaces calls to rsh with calls to *qrsh -inherit*. By using that wrapper script, the parallel environment's calls to rsh can be rerouted through the grid via qrsh without having to modify the parallel environment itself to work with Grid Engine.

The last thing to notice is how this script correlates to the control\_slaves and job\_is\_first\_task attributes of the parallel environment configuration. Let's start with first\_job\_is\_task. In our configuration, we set it to false. That means that this master job script is not counted as a job and does no real work. That is why our script doesn't do anything but kick off sub-tasks. If job\_is\_first\_task had been true, our job script would be expected to run one of the RMIApp instances itself.

Now let's talk about the control\_slaves attribute. If control\_slaves is true, we are allowed to use *qrsh -inherit* to kick off our sub-tasks. The qmaster will not, however, allow us to kick off more subtasks than the number of slots we've been assigned (minus 1 if job\_is\_first\_task is true). The advantage of using *qrsh -inherit* is that the sub-tasks are tracked by Grid Engine like regular jobs. If control\_slaves is false, we have to use some mechanism external to Grid Engine, such as rsh or ssh, to kick off our sub-tasks, meaning that Grid Engine cannot track them and is actually fully unaware of them. That's why job\_is\_first\_task is meaningless when control\_slaves is false.

In order to test our job we need a Java application called RMIApp. As that's outside the scope of the example, let's just pretend we have a parallel Java application that uses the RMI registry for intra-process communication. To submit our job we use *qsub -pe rmi 2-4 rmi\_job.sh*. The *-pe rmi 2-4* argument tells the qmaster that we're using the rmi parallel environment and we want 4 job slots assigned to our job, but we will accept as few as 2. Because our job script starts a sub-task for every entry in to host file, it will start the right number of sub-tasks, no matter how many slots we are assigned. Had we written the job script to start exactly two sub-tasks, we would have to use *-pe rmi 2* so that we could be sure we got exactly two job slots.

While the job is running, run *qstat -f*. You'll see output something like this:

|  |
| --- |
| % qstat -f  queuename qtype used/tot. load\_avg arch states  ----------------------------------------------------------------------------  all.q@ultra20 BIP 4/10 0.08 sol-amd64  253 0.55500 rmi\_job.sh dant r 03/23/2007 11:46:51 4 |

From this output we can see that the job has been scheduled and has been assigned four job slots. Those four job slots only account for the four sub-tasks. The master job itself is not counted because the job\_is\_first\_task attribute is false.

After our job completes, if we look in our home directory (which is where Grid Engine will put the output files since we didn't tell it otherwise), we will find four new files: rmi\_job.sh.e253, rmi\_job.sh.o253, rmi\_job.sh.pe253, and rmi\_job.sh.o253, assuming, of course, that our job was number 253. The \\*.o253 and \\*.e253 files should be familiar. They're the output and error streams from the job script. The \\*.po253 and \\*.pe253 files are new. They're the output and error streams from the parallel environment startup and shutdown scripts.

So, there you have it. A complete, top-to-bottom example of creating, configuring, and running a parallel environment.