

RMPI Torque Tutorial

Introduction

The example shown here demonstrates the use of the Torque Scheduler for the purpose of running an RMPI program. Knowledge of R is assumed. Having read the Basic Torque Tutorial prior to this one is also highly recommended.

Week1

MPI is a powerful foundation for your R programs which can enable you to write code that will run on not just one computer, but on multiple computers. By splitting your code up into more than one process, you will be able to do more work in a shorter amount of time. The Aries Cluster (which has OpenMPI installed) has this foundation in place and is the ideal place for running very complex or long-running programs.

The purpose of this tutorial is to show you a basic RMPI program — called “week1” — and describe how it works and how to run it under the Torque Scheduler.

Part One: The Files

rmpiweek1.tar

All the files needed to try out this example are available as a .tar file on the Star Lab website under the help heading. Under Linux, these files can be extracted with the following command (this will create a directory called rmpitest and change to that dir):

```
wget
http://www.rochester.edu/college/psc/thestarlab/help/mars/rmpiweek1.tar
tar xvf rmpiweek1.tar; cd rmpitest
```

The files, summary

week1.R - The R program itself which runs multiple processes. Uses the RMPI R package (and also snow).

run.sh - A shell script which is designed to run with week1.R under the Torque Scheduler. Run.sh is responsible for reserving nodes, starting R, and running the R program.

Output Files, summary

run.sh.oXX - This file is generated by Torque and shows the output information, if any. XX represents the torque job number. May be used for troubleshooting.

run.sh.eXX - This file is generated by Torque and shows the error information, if any. XX represents the torque job number. May be used for troubleshooting.

Part Two: Discussion of MPI in R

Before proceeding, a description of MPI's basic operation is in order. An R/MPI program progresses through several stages before completion. In short, the stages can be summarized as follows:

Initialize snow and Rmpi packages
Initialize the cluster (ie with a call to makeMPIcluster)
Do work and execute functions on all nodes by using snow commands
Terminate the mpi environment (with the StopCluster command)

Essentially, the R programmer needs to include two libraries when writing an MPI program. The first, predictably, is the Rmpi library. The second library is called snow. Snow provides a number of higher level functions that make parallel programming in R much easier.

Once these libraries have been included, the cluster needs to be initialized from within R. This can be done by use of the makeMPIcluster command as shown in our example program. Note that in our example, we have chosen to specify six nodes.

Once the environment is ready, real work can begin. R commands can be issued as normal, but the real power of MPI is harnessed once we issue an MPI command - such as the ClusterCall command. A function, when specified as a parameter of ClusterCall, will execute on every node in our cluster. (There are many other functions of this nature provided through the snow package.)

When all parallel work has been completed, the cluster needs to be de-initialized with the stopCluster command.

In full, here is our example program, week1.R. The functions described above are in red.

```
rm(list=ls())

# R code: parallel version of OLS monte carlo code
library(snow)
library(Rmpi)

nclus=6

cl <-makeMPIcluster(nclus)

#alter either n or mc to affect run time
n=2000
mc=20000
```

```

mcperclus=round(mc/nclus)

x=matrix(runif(n),n,1)
x=cbind(1,x)

estimatebetas=function(x,n,nmc){

e=matrix(rnorm(n*nmc),n,nmc)
y=1+rep(x[,2],nmc)+e
solve(t(x)%*%x)%*%t(x)%*%y

}

ptim=proc.time()[3]

b=clusterCall(c1,estimatebetas,x=x,n=n,nmc=mcperclus)

b=cbind(b[[1]],b[[2]],b[[3]],b[[4]],b[[5]],b[[6]])

tim=proc.time()[3]-ptim

cat(dim(b)," ",apply(b,1,mean),"\n")

cat(mc," iterations with sample sizes of ",n," took ",tim,"seconds.\n")

stopCluster(c1)

```

Regarding the following line:

```
b=cbind(b[[1]],b[[2]],b[[3]],b[[4]],b[[5]],b[[6]])
```

It is worth mentioning that the parameters given here are related to the value of `nclus` earlier in the program. For example, if `nclus` had been 10, we would have to substitute the `cbind` command with the following:

```
b=cbind(b[[1]],b[[2]],b[[3]],b[[4]],b[[5]],b[[6]],b[[7]],b[[8]],b[[9]],
b[[10]])
```

For more information on snow commands, the following reference may be of use:

<http://www.sfu.ca/~sblay/R/snow.html#>

Part Three: Creating the Shell Script

There is a special method which must be used for running an R/MPI program - simply running the program from within R is not enough! That is because we need to specify additional details about the cluster at runtime and also we need to specify the number of processes we need. When choosing a number of processes, normally it is a good idea to

choose no more than double the number of computers in the cluster. For a cluster with ten machines in it, the user would want to avoid choosing more than, say, 20. Performance will suffer if this guideline is not followed.

The `mpirun` command is used to start R under the MPI environment. This command is included in the `run.sh` shell script along with the other commands that need to be executed. The format of the `mpirun` command is as follows:

```
mpirun -np <number of processes> -hostfile <hostfile> <executable>
```

where `np` is the number of processes needed and `hostfile` is a textfile which contains a list of the nodes available on the cluster.

Although it is theoretically possible to write the `hostfile` by hand as a text file, this should never be done. It is safer to have this done automatically by use of a shell script (as shown in `run.sh` below). This is because any error at all in the `hostfile` will cause the RMPI program to crash (for example, in the event that the `hostfile` refers to a node that is offline).

The shell script works as a sort of super-executable. Not only does the shell script issue the `mpirun` command as described above, but it can also be made to generate the required host file (`cluster_nodes`) on the fly.

An example that works with the `week1.R` program is shown below. (Note that this shell script is designed to work only with Torque - see Part Four for instructions on how to submit it to the scheduler).

```
#!/bin/bash

# show the working nodes Torque reserved for you
cd $PBS_O_WORKDIR

echo "My machine will have the following nodes:"
echo "-----"
cat ${PBS_NODEFILE}
echo "-----"

# run the R program on the reserved nodes with MPI command
# run the week1.R source once R has been started
# quit R (mpi.quit()) after the job is finished
# the execution result of your R program will be printed into
run.sh.oXXX
# the error info will be printed into run.sh.eXXX if the R program
terminates abnormally
mpirun -np 1 -hostfile $PBS_NODEFILE /usr/bin/R --no-save <<EOF
source('week1.R')
mpi.quit()
EOF
```

```
# delete the previous standard output and standard error information
(defalut)
# comment out the following line if you want to keep previous output
data and error info
rm ~/rmpitest/run.sh.*
```

IMPORTANT - Note the following line in the script above:

```
mpirun -np 1 -hostfile $PBS_NODEFILE /usr/bin/R --no-save <<EOF
```

In particular, /usr/bin/R is included because it is necessary to include the full path to R in order to successfully execute the run.sh script. Using simply “R” for example would cause an error to occur. Additionally, the -np option should always specify the value “1” in order to avoid an error. Basically, when designing a new shell script, the only lines the user may need to change are the following:

```
source('week1.R')
```

and possibly

```
rm ~/rmpitest/run.sh.*
```

The user can instead include the name of a different R source file as required and/or can specify a different path for run.sh.* if the program is being run in a different directory.

Finally, it is worth mentioning again that this shell script will only run under Torque (see Part Four for details on how to submit it to Torque).

Part Four: Submission of the Torque Script

Important Note: *The user must be logged into the ariessrv in order to submit a job successfully to Torque.*

In order to ensure job runs in the fastest manner possible, users are encouraged to use the Torque Scheduler. Torque constantly monitors the nodes on the cluster and is able to track which nodes have the most resources free and which nodes are overburdened. Because MPI is very unforgiving when it comes to manually created “mpihosts” files, it is ***strongly*** suggested that the user run every RMPI program under Torque.

Torque accepts a shell script (.sh file) which describes the job we want to run. We can use the shell script we created in Part Three.

Run your MPI job with the following command (in this example, four nodes are requested):

```
qsub -l nodes=4,walltime=1:00:00 ~/rmpitest/run.sh
```

Once the command has been issued, you will be given the job number. The `qstat` command can be issued at any time after submission in order to see where your job is in the queue. *The `qstat -a` command is another useful command - `qstat -a` will allow you to see the status of other users' jobs and can be used before you make your own submission to Torque. In particular, the NDS column (node number) can be of use when you are deciding how many nodes to request. For example, if all but four nodes are already occupied, you may choose to run your job using four nodes or fewer. (See the Basic Torque Tutorial for more details on how to track your job's progress, etc).*

Once the job has finished, two new files will have been created. The `run.sh.oXX` file will contain the program's output and `run.sh.eXX` will provide troubleshooting information in the event that there was a problem with the execution. If the program executes successfully, `run.sh.eXXX` will be empty.

A successful run will provide output similar to the following:

```
6 slaves are spawned successfully. 0 failed. 6 processes are spawned
in the program.
```

```
2 19998 0.9998407 0.9998238
```

```
20000 iterations with sample sizes of 2000 took 7.701 seconds.
```

OPTIONAL - TRY THIS: In a separate terminal window run the command prior to submitting your job:

```
mosmon -d
```

By running Mosix at the same time you are running your job you can actually watch the activity on the cluster as it happens in real time. This command can also be very helpful when you are deciding how many nodes to use when submitting your job.

Part Five: Final Considerations

As mentioned earlier, MPI is very unforgiving unless it is run under Torque. As a result, users need to run all RMPI programs under the Torque scheduler as described in this document.

It is also worth noting that when a user occupies more nodes, the program will, in theory, run faster. However, in practice, because Torque will not start your job until the requested number of nodes has become available, the same user may actually end up having to wait longer for the program to begin. Because of this trade-off, it is strongly suggested that users request only a moderate number of nodes when submitting a script. Not only will your program have the best chance to start running sooner, but some nodes

will remain free for other users as well. Please be a good Star Lab citizen and do your part!