RPVM Torque Tutorial

Introduction

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

Week1

PVM 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 PVM 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 RPVM program - called "week1" - and describe how it works and how to run it under the Torque Scheduler.

Part One: The Files

rpvmweek1.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 rpvmtest):

tar xvf rpvmweek1.tar

The files, summary

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

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

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

Essentially, the R programmer needs to include two libraries when writing a PVM program. The first, predictably, is the rpvm 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 makePVMcluster 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 PVM is harnessed once we issue a PVM 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 shown in red.

```
rm(list=ls())

# R code:  parallel version of OLS monte carlo code

library(snow)
library(rpvm)

nclus=6
```

```
cl <-makePVMcluster(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(cl,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(cl)
```

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/PVM 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 script needs to start PVM before it can start R. PVM is started by issuing the PVM command in the following format:

pvm <hosts>

Next, the script outputs the nodes we are using with the following command:

conf

Finally, the script quits out of PVM with the following command:

quit

Now that PVM is ready, the script is able to start R, load and run the program, and quit upon completion with the following commands:

/usr/bin/R --no-save <<EOF
source('week1.R')
q()

The shell script works as a kind of super-executable. It automatically determines which nodes are online, then starts PVM. It runs the R program afterwards, and terminates cleanly - all with one command on the user's part. Finally, since a shell script is required in order for the user to utilize Torque, it is strongly suggested that users make use of a shell script for even small programs.

An example shell script, called run.sh, 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).

This is run.sh

show the working nodes Torque reserved for you
cd $PBS_O_WORKDIR

echo "My machine will use the following nodes:"
echo "---"

#cat ${PBS_NODEFILE}

#echo "--"

run pvm on the reserved nodes
pvm $PBS_NODEFILE <<EOF
conf
quit
EOF
echo "--"

start R
run the RPVM program on R (You need only change the filename)
quit R after you have done the job
the execution result will be printed into the file run.sh.oXXX (XXX is job #)
the error information will be printed into the file run.sh.eXXX if the R program
terminates abnormally
/usr/bin/R --no-save <<EOF
source('week1.R')
q()
EOF

shut down PVM
we don't need to shut down PVM, because when the job finished and the system
release the nodes, the PVM will be terminated automatically.
#pvm <<EOF
#halt
#EOF

delete all the previous standard output and standard error information (default)
comment the following line if you want to keep the previous output data and error
info
rm ~/rpvmtest/run.sh.*

IMPORTANT - Note the following line in the script above:

/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. 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 ~/rpvmtest/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. For this reason, users are encouraged to always use Torque for parallel programming with PVM.

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

qsub –l nodes=4,walltime=1:00:00 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 (found in file run.sh.oXX where XX is your torque job number):

```
> source('week1.R')
2 19998   0.9996805 1.000416
20000  iterations with sample sizes of  2000  took  8.116 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

It is 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!