

Parallel Job Tutorial

Overview

Scope

This tutorial will show you how to acquire multiple hosts from the cluster and start the environment necessary to run LAM/MPI or PVM based applications. It does not go into parallel programming topics- i.e. how to write, architect, debug, or optimize applications.

What is a Parallel Job

Parallel jobs run the gamut from so-called "embarrassingly parallel" jobs- where discrete tasks have little-to-no communication or dependence between each other and "tightly parallel" jobs in which tasks are highly-interdependent, sharing memory and delegating tasks to each other.

These jobs run within a parallel machine- software daemons and libraries that enable tasks to exchange information. Many methods exist to create a parallel machine- but for this tutorial, we focus on "Parallel Virtual Machine" or PVM and the "Local Area Multicomputer" implementation of the Message Passing Interface- LAM/MPI. These are the most commonly used environments. If you have need of another, please let us know.

The Parallel Execution Environment

In discussions about the parallel environment, some terms are necessary:

MOM

The *Machine Oriented Mini-server* is the daemon (running on all the nodes in this cluster) responsible for execution and monitoring of jobs.

Sisterhood

All of the MOMs allocated to a single task.

Mother Superior (MS)

The MOM which, in multi-node jobs, is the "master" for all the other MOMs allocated to this job. It is with this MOM that all other nodes communicate and it is this MOM that communicates with the Torque server.

Sisters

The other (non-MS) MOMs in a sisterhood

Thus, some things to keep in mind for your parallel job:

- When your job starts, each node is set up to allow you passwordless SSH access between all nodes in the sisterhood.
- Each node has a temporary directory created (of the same name as the temporary directory on the mother superior, the value of the environment variable `$(TMPDIR)`).
- However, the `TMPDIR` environment variable is only available on the MS node- you will have to export that to any processes on the sister nodes.
- If you are staging files (either in or out) this staging is done to (from) the mother superior *only*.

PVM

About PVM

From the introduction at the PVM website <http://www.csm.ornl.gov/pvm/>

PVM (Parallel Virtual Machine) is a software package that permits a heterogeneous collection of Unix and/or Windows computers hooked together by a network to be used as a single large parallel computer. Thus large computational problems can be solved more cost effectively by using the aggregate power and memory of many computers. The software is very portable. The source, which is available free thru netlib, has been compiled on everything from laptops to CRAYs.

PVM works by starting up a PVM daemon on each machine in the sisterhood which is the "pipe" through which messages and data are passed to other PVM tasks.

Setting Up Your Account

To use PVM, you'll need to have two environment variables set:

PVM_ROOT

Where the PVM executables and libraries are found: On the GP cluster, this path is
/usr/share/pvm3

PVM_RSH

The remote shell program to be used by PVM when starting the remote PVM daemons. For this cluster, we use secure shell- /usr/bin/ssh.

The environment created during login will automatically add these to your environment. It is important to note that the PVM startup process does not invoke |login(1)| on the other nodes in the cluster. These sessions are non-interactive, non-login shells. Each shell handles it differently- where possible (tcsh, for example) the cluster hosts have been set up to set up the environment automatically for these environments.

Your shell may not use global environment files for non-login environments (bash is such a shell). In these cases, you will need to source the PVM environment files (/etc/profile.d/pvm.csh] for "C" shells and /etc/profile.d/pvm.csh for Bourne-family shells) in the appropriate login file (like .bashrc for bash).

You can verify this by submitting a multinode job that checks the environment on each node in the sisterhood. The script "pvm-check.qsub" in the PVM directory of the parallel examples tar file does just that:

```
hoppy:(mrg) >cat pvm-check.qsub
# Request two processors on two nodes
# and 10 minutes of run-time
#PBS -l nodes=2:ppn=2,walltime=00:10:00
#
# Use the bourne-again shell
#PBS -S /bin/bash
#
```

```

# Check the environment on the remote node to
# make sure that the PVM variables are being set
#
for NODE in `sort ${PBS_NODEFILE}|uniq` ;
do
    echo "Checking PVM environment on ${NODE}"
    #
    ssh $NODE "/bin/echo \"PVM_ROOT: \${PVM_ROOT}\"; /bin/echo
    \"PVM_RSH: \${PVM_RSH}\"; "
    #
done
exit 0

hoppy:(mrg) >

```

When run through the cluster, we can see that my environment isn't set when SSH runs those commands on the sisters:

```

hoppy:(mrg) >cat pvm-check.qsub.o872
PROLOGUE(872.hoppy): Prologue starting 20 Mar 2006 15:02:56(PST)
PROLOGUE(872.hoppy): jobid:872.hoppy, user:mrg, group:mrg
PROLOGUE(872.hoppy): MOM: kpz6394, sisters: KPZ6394 KPZ6443
PROLOGUE(872.hoppy): Configuring KPZ6394 .... OK
PROLOGUE(872.hoppy): Configuring KPZ6443 .... OK
PROLOGUE(872.hoppy): Prologue Complete
Checking PVM environment on KPZ6394
PVM_ROOT:
PVM_RSH:
Checking PVM environment on KPZ6443
PVM_ROOT:
PVM_RSH:
EPILOGUE(872.hoppy): Epilogue starting 20 Mar 2006 15:02:57(PST)
EPILOGUE(872.hoppy): jobid:872.hoppy, user:mrg, group:mrg
EPILOGUE(872.hoppy): Cleaning up KPZ6394 .... OK
EPILOGUE(872.hoppy): Cleaning up KPZ6443 .... OK
EPILOGUE(872.hoppy): Epilogue Complete

```

Adding a couple lines to my .bashrc fixes this:

```

hoppy:(mrg) >cat >> ~/.bashrc <<EREH
> PVM_ROOT=/usr/share/pvm3
> PVM_RSH=/usr/bin/ssh
> EREH
hoppy:(mrg) >qsub pvm-check.qsub
873.hoppy
hoppy:(mrg) >cat pvm-check.qsub.o873
PROLOGUE(873.hoppy): Prologue starting 20 Mar 2006 15:06:10(PST)
PROLOGUE(873.hoppy): jobid:873.hoppy, user:mrg, group:mrg
PROLOGUE(873.hoppy): MOM: kpz6394, sisters: KPZ6394 KPZ6443
PROLOGUE(873.hoppy): Configuring KPZ6394 .... OK
PROLOGUE(873.hoppy): Configuring KPZ6443 .... OK
PROLOGUE(873.hoppy): Prologue Complete
Checking PVM environment on KPZ6394

```

```

PVM_ROOT: /usr/share/pvm3
PVM_RSH: /usr/bin/ssh
Checking PVM environment on KPZ6443
PVM_ROOT: /usr/share/pvm3
PVM_RSH: /usr/bin/ssh
EPILOGUE(873.hoppy): Epilogue starting 20 Mar 2006 15:06:11(PST)
EPILOGUE(873.hoppy): jobid:873.hoppy, user:mrg, group:mrg
EPILOGUE(873.hoppy): Cleaning up KPZ6394 .... OK
EPILOGUE(873.hoppy): Cleaning up KPZ6443 .... OK
EPILOGUE(873.hoppy): Epilogue Complete
hoppy:(mrg) >

```

Starting the Machine

Now that we are reasonably confident we can start the machine, let's make a job to do just that. PVM is controlled via an interactive client, which makes scripting this all that much more difficult. The command `|pvm|` will start the session- this command has many subcommands (see the manpage for pvm). For creating the machine, we'll use the hosts file created by torque (PBS_NODEFILE) as the sole argument to pvm- subsequent commands we'll use some odd shell constructs.

```

hoppy:(mrg) >cat pvm-start.qsub
# Request two processors on two nodes
# and 10 minutes of run-time
#PBS -l nodes=2:ppn=2,walltime=00:10:00
#
# Use the bourne-again shell
#PBS -S /bin/bash
#
echo "My machine will have the following nodes:"
echo "-----"
cat ${PBS_NODEFILE}
echo "-----"
#
# Start the PVM machine- commands need to be sent
# to the PVM console's STDIN, hence the redirect
pvm $PBS_NODEFILE <<EREH
conf
EREH
#
pvm <<EREH
halt
EREH
exit 0

hoppy:(mrg) >qsub pvm-start.qsub
883.hoppy
hoppy:(mrg) >cat pvm-start.qsub.o883
PROLOGUE(883.hoppy): Prologue starting 20 Mar 2006 15:48:48(PST)
PROLOGUE(883.hoppy): jobid:883.hoppy, user:mrg, group:mrg
PROLOGUE(883.hoppy): MOM: kpz6394, sisters: KPZ6394 KPZ6443
PROLOGUE(883.hoppy): Configuring KPZ6394 .... OK
PROLOGUE(883.hoppy): Configuring KPZ6443 .... OK
PROLOGUE(883.hoppy): Prologue Complete

```

My machine will have the following nodes:

```
-----  
KPZ6394  
KPZ6394  
KPZ6443  
KPZ6443  
-----
```

```
pvm> conf
```

```
2 hosts, 1 data format
```

HOST	DTID	ARCH	SPEED	DSIG
kpz6394	40000	LINUXI386	1000	0x00408841
KPZ6443	80000	LINUXI386	1000	0x00408841

```
pvm> quit
```

```
Console: exit handler called
```

```
pvmd still running.
```

```
pvmd already running.
```

```
pvm> EPILOGUE(883.hoppy): Epilogue starting 20 Mar 2006 15:48:49(PST)
```

```
EPILOGUE(883.hoppy): jobid:883.hoppy, user:mrg, group:mrg
```

```
EPILOGUE(883.hoppy): Cleaning up KPZ6394 .... OK
```

```
EPILOGUE(883.hoppy): Cleaning up KPZ6443 .... OK
```

```
EPILOGUE(883.hoppy): Epilogue Complete
```

Stopping the Machine

Torque attempts to kill any process you have running when your job script completes. However, there have been instances where this leads to incomplete clean-up of PVM files, which, in turn, leads to future problems starting machines.

Thus, best practice is to clean up your PVM machine prior to your script's exit. This is done with the PVM "halt" command, piped through to the PVM console tool as in the script above:

```
pvm <<EREH  
halt  
EREH
```

This results in a message being output to STDERR, but this message can be safely ignored.

A Basic PVM Program

The PVM examples contain a simple "hello,world" type program. I have compiled it and included the binary (and sources) in the parallel examples tar file for us to use as an example of a "real" PVM program.

```

hoppy:(mrg) >pwd
/home/mrg/parallelx/PVM
hoppy:(mrg) >cp hello hello_other ~/pvm3/bin/LINUXI386
hoppy:(mrg) >ls ~/pvm3/bin/LINUXI386/hello*
/home/mrg/pvm3/bin/LINUXI386/hello
/home/mrg/pvm3/bin/LINUXI386/hello_other
hoppy:(mrg) >cat hello.qsub
#
# This runs the "hello" example on two
# nodes.
#
# Make sure you've copied hello and hello_other
# to $HOME/pvm3/bin/LINUXI386
#
# Request two processors on two nodes
# and 10 minutes of run-time
#PBS -l nodes=2:ppn=2,walltime=00:10:00
#
# Use the bourne-again shell
#PBS -S /bin/bash
#
echo "My machine will have the following nodes:"
echo "-----"
cat ${PBS_NODEFILE}
echo "-----"
#
# Start the PVM machine- commands need to be sent
# to the PVM console's STDIN, hence the redirect
pvm $PBS_NODEFILE <<EREH
conf
EREH
# Now that we have a running machine, run
# "hello":
${HOME}/pvm3/bin/LINUXI386/hello
#
# Now shut down the machine
pvm <<EREH
halt
EREH
#
exit 0

hoppy:(mrg) >qsub hello.qsub
889.hoppy
hoppy:(mrg) >more hello.qsub.o889
PROLOGUE(889.hoppy): Prologue starting 21 Mar 2006 11:10:53(PST)
PROLOGUE(889.hoppy): jobid:889.hoppy, user:mrg, group:mrg
PROLOGUE(889.hoppy): MOM: kpz6369, sisters: KPZ6369 KPZ6401
PROLOGUE(889.hoppy): Configuring KPZ6369 .... OK
PROLOGUE(889.hoppy): Configuring KPZ6401 .... OK
PROLOGUE(889.hoppy): Prologue Complete
My machine will have the following nodes:
-----
KPZ6369

```

```

KPZ6369
KPZ6401
KPZ6401
-----
pvm> conf
2 hosts, 1 data format
          HOST      DTID      ARCH      SPEED      DSIG
          kpz6369   40000  LINUXI386   1000  0x00408841
          KPZ6401   80000  LINUXI386   1000  0x00408841
pvm> quit

Console: exit handler called
pvmd still running.
i'm t40002
from t80001: hello, world from kpz6401
pvmd already running.
pvm> EPILOGUE(889.hoppy): Epilogue starting 21 Mar 2006 11:10:55(PST)
EPILOGUE(889.hoppy): jobid:889.hoppy, user:mrg, group:mrg
EPILOGUE(889.hoppy): Cleaning up KPZ6369 .... OK
EPILOGUE(889.hoppy): Cleaning up KPZ6401 .... OK
EPILOGUE(889.hoppy): Epilogue Complete

```

The output from the two "hello" programs is:

```

i'm t40002
from t80001: hello, world from kpz6401

```

The numbers (t40002 and t80001) are the *task IDs* for each of the tasks (programs) spawned. The second task includes output from the other host in the sisterhood.

References

- The PVM website <http://www.csm.ornl.gov/pvm/>

LAM/MPI

About LAM/MPI

From the NCSA tutorial <<http://webct.ncsa.uiuc.edu:8900/public/MPI/>> on MPI:

... the Message Passing Interface (MPI), [is] a standard library of subroutines (Fortran) or function calls (C) that can be used to implement a message passing program. MPI allows the coordination of a program running as multiple processes in a distributed memory environment, yet is flexible enough to also be used in a shared memory system. MPI programs can be used and compiled on a wide variety of parallel computers such as the IBM SP2, the Silicon Graphics Origin 2000, or a cluster of workstations (homogenous or heterogeneous) over a network.

The implementation currently installed is the "Local Area Minicomputer" implementation maintained by staff and students at Indiana University.

NOTE: LAM/MPI is being deprecated in favor of "Open-MPI"- though the LAM team will be supporting this implentation for a while. We will be switching implementations at some point- let us know if you have a need for a particular MPI version.

LAM/MPI follows a similar approach to passing messages- daemon processes are started on each node you wish to use. These daemons start binaries at the request of a master and handle communication amongst all of the MPI processes. When the job is done, the daemon exits.

Running the Machine

The first thing that needs to happen is that the machine must be started- or "booted"- with a file containing a list of the hosts that are to be included in the machine. This is typically done with lamboot:

```
lamboot hostlist.txt
```

The list of hosts (called a "boot schema") is, at its simplest, a text file with a simple list of hosts. Once the machine's been started, mpirun is used to start MPI programs on this virtual machine:

```
mpirun my.mpi.prog.exe
```

When you're done with the machine, you must stop it to prevent future issues with starting new machines. The "lamhalt" command should stop all the LAM daemons. If it should fail for some reason, the "lamwipe" command can be used- however, it requires that the boot schema be passed to it:

```
lamwipe hostlist.txt
```

However, LAM/MPI provides a command that neatly combines all those steps- "mpiexec"- which boots the machine, runs the command, and stops the machine once complete:

```
mpiexec -machinefile <bhost> <command>
```

Many other arguments are available to control the MPI machine- refer to the man-page for mpiexec for more details.

A Sample MPI Job

In the MPI examples directory of the parallel examples tar-file you will see "hello.mpi" (an application compiled for LAM/MPI) and "hello.mpi.qsub":

```
hoppy:(mrg) >cat hello.mpi.qsub
# Request two processors on two nodes
# and 10 minutes of run-time
#PBS -l nodes=2:ppn=2,walltime=00:10:00
#
# Use the bourne-again shell
#PBS -S /bin/bash
#
#PBS -j oe
#
# print "hello, world" on stdout on each node
# using our MPI program
#
# ${PBS_NODEFILE} points to a file containing
# a newline-delimited list of hosts allocated
# to this job (the "sisterhood").
#
cd $PBS_O_WORKDIR
echo "starting MPI with ${PBS_NODEFILE} in `pwd`"
echo "-----"
mpiexec -v -machinefile ${PBS_NODEFILE} hello.mpi
echo "-----"
exit 0
```

```
hoppy:(mrg) >qsub hello.mpi.qsub
917.hoppy
hoppy:(mrg) >cat hello.mpi.qsub.o917
PROLOGUE(917.hoppy): Prologue starting 23 Mar 2006 15:04:29(PST)
PROLOGUE(917.hoppy): jobid:917.hoppy, user:mrg, group:mrg
PROLOGUE(917.hoppy): MOM: kpz6369, sisters: KPZ6369 KPZ6401
PROLOGUE(917.hoppy): Configuring KPZ6369 .... OK
PROLOGUE(917.hoppy): Configuring KPZ6401 .... OK
PROLOGUE(917.hoppy): Prologue Complete
starting MPI with /var/spool/torque/aux//917.hoppy in
/home/mrg/parallellex/MPI
-----
mpiexec: Booting lam..
n-1<30468> ssi:boot:base:linear: booting n0 (KPZ6369)
n-1<30468> ssi:boot:base:linear: booting n1 (KPZ6401)
n-1<30468> ssi:boot:base:linear: finished
```

LAM 7.1.1/MPI 2 C++/ROMIO - Indiana University

```
mpiexec: Lamboot Complete
mpiexec: Launching MPI programs
30480 hello.mpi running on n0 (o)
30481 hello.mpi running on n0 (o)
6987 hello.mpi running on n1
6988 hello.mpi running on n1
```

```
Hello World! I am 0 of 4
Hello World! I am 2 of 4
Hello World! I am 1 of 4
Hello World! I am 3 of 4
mpiexec: MPI program execution over..
mpiexec: Performing Lamhalt

LAM 7.1.1/MPI 2 C++/ROMIO - Indiana University

Shutting down LAM
hreq: received HALT_ACK from n1 (KPZ6401.FHCRC.ORG)
hreq: received HALT_ACK from n0 (kpz6369)
LAM halted
mpiexec: Lamhalt complete
-----
EPILOGUE(917.hoppy): Epilogue starting 23 Mar 2006 15:04:31(PST)
EPILOGUE(917.hoppy): jobid:917.hoppy, user:mrg, group:mrg
EPILOGUE(917.hoppy): Cleaning up KPZ6369 .... OK
EPILOGUE(917.hoppy): Cleaning up KPZ6401 .... OK
EPILOGUE(917.hoppy): Epilogue Complete
```

In this example, you see the use of the "mpiexec" command to start the machine. Use of the "-v" flag shows the individual steps along the way.