VACCUserDocumentation

From UVMInfoTech

This contains documentation for users of *bluemoon*, the VACC's HPC cluster. For general information including hardware and system adminstration, go back to VACC.

Contents

How do I get an account?

Fill out the application form (https://www.uvm.edu/~vacc/application/) . *Bluemoon* is netid-integrated, so you'll use your UVM Netid to get access. You will be added to the *vaccusr* group which will allow you to access the cluster.

Usage policies have not been written yet, but since the VACC's mission is to support research at UVM, it would help if you would give us a summary of what your research area is, your funding sources, and your need for computing support. It would also help if you could tell us what your specific requirements are (e.g. approximate length of time access is needed, number of jobs, number of CPUs, amount of memory and disk space consumed by a typical job.)

Logging in

There are two machines that are accessible to VACC users, `bluemoon-user1.uvm.edu` and `bluemoon-user2.uvm.edu`. Use ssh to get access to the cluster.

Caveats

While we consider research computing central to the University's mission, *bluemoon* is not considered a *business-critical* service. Its architecture has been chosen for performance, as most HPC clusters are, rather than resiliency or reliability. This means that if a component of the cluster fails, we may not respond in the same timeliness that we would if, say, www.uvm.edu were down. If a node or two were to fail, it will probably be several days before it is repaired, since it is presumed the cluster will continue to operate. In the event of multiple failures of critical systems across campus, we will attend to *business-critical* services before we get to fixing problems with *bluemoon*.

There is some small investment in backups for /gpfs1. As of this writing, we back up to a DR location nightly. However, files in /gpfs2 ("scratch") are NOT backed up. --Jtl 19:26, 1 June 2011 (UTC)

Operating system

Bluemoon is currently running RedHat Enterprise Linux 5 (64-bit), with OSCAR (http://oscar.openclustergroup.org/) as the cluster management software.

As such it comes installed with the GNU compilers (gcc, f77) and various libraries.

Libraries

Interface	Implementation	Located in	To use	Notes
MPI	mpich2-mx-gcc453-1.3.1..10	`/gpfs1/arch/x86_64/mpich2-mx-gcc453-1.3.1..10/`	`switcher mpi = mpich2-mx-gcc453-1.3.1..10`	Requires Myrinet, uses GCC 4.5.3
MPI	lam-7.1.4	`/opt/lam-7.1.4`	`switcher mpi = lam7.1.4`	
MPI	mpich-ch_p4-gcc-1.2.7	`/opt/mpich-ch_p4-gcc-1.2.7/`	`switcher mpi = mpich-ch_p4-gcc-1.2.7`	
MPI	mpich-mx-pgi-1.2.7..5	`/gpfs1/arch/x86_64/mpich-mx-pgi-1.2.7..5`	`switcher mpi = mpich-mx-pgi-1.2.7..5`	Requires Myrinet
BLAS, LAPACK	ACML 3.1.0	`/gpfs1/arch/x86_64/acml3.1.0/`	See `/gpfs1/arch/x86_64/acml3.1.0/Doc`	
BLAS, LAPACK	ACML 4.2.0	`/gpfs1/arch/x86_64/acml4.2.0/`	See `/gpfs1/arch/x86_64/acml4.2.0/Doc`	
FFTW	FFTW 3.1	`/gpfs1/arch/x86_64/`	See FFTW documentation (http://www.fftw.org/#documentation)	
FFTW	FFTW 2.1	`/gpfs1/arch/x86_64/fftw-2-pgi/`	See FFTW documentation (http://www.fftw.org/#documentation)	Built with PGI Compilers
GSL	GSL 1.9	`/gpfs1/arch/x86_64/`	See GSL documentation (http://www.gnu.org/software/gsl/)	
CUDA	NVidia CUDA 4.0	`/gpfs1/arch/x86_64/cuda`		Requires a GPU node

Third party software

Vendor	Software	Version	Located in	To use
Fluent, LLC	Fluent CFD suite	6.2.16	`/gpfs1/arch/x86_64/fluent`	`module load fluent`
Portland Group, Inc	PGI compiler suite (`pgcc, pgf77, pgf95, pghpf,` etc)	7.2-5	`/gpfs1/arch/x86_64/pgi`	`module load pgi`
Mathworks	MATLAB	R2006a	`/gpfs1/arch/matlab72`	Already in $PATH

Submitting jobs

We are currently using TORQUE (PBS) (http://www.clusterresources.com/pages/products/torque-resource-manager.php) version 1.2 which is provided with OSCAR. It acts as an arbitrator when multiple people are using the cluster simultaneously, so that jobs do not contend for processors and memory (as they do in normal timesharing systems.)

Single processor jobs

These are pretty easy, and if you have lots of them, PBS will farm them out to the different processors that are available, so you can queue up many jobs and they will be started as soon as processors become available. However, keep in mind that there is some overhead in scheduling jobs, so if you have many jobs (hundreds or thousands) that each take less than 10 seconds or so to run, you should think about aggregating them.

For each job you want to run, create a job script. In this case we'll call it `myjob.script`:

```
# This job needs 1 compute node with 1 processor per node.
#PBS -l nodes=1:ppn=1
# It should be allowed to run for up to 1 hour.
#PBS -l walltime=01:00:00
# Name of job.
#PBS -N myjob
# Join STDERR TO STDOUT.  (omit this if you want separate STDOUT AND STDERR)
#PBS -j oe
# Send me mail on job start, job end and if job aborts
#PBS -M kapoodle@uvm.edu
#PBS -m bea

cd $HOME/myjob
echo "This is myjob running on " `hostname`
myprogram -foo 1 -bar 2 -baz 3
```

Submit the job by running: `qsub myjob.script`

If you are thinking about submitting so many single-processor jobs that you are about to write a for loop, you should probably aggregate them.

Jobs that require more memory

As above, but if your job requires more than about 1.5gb of memory, you should specify it, so that your job is run on a machine with enough memory. Note that PBS/qsub treats "gb" as 2^30 bytes, not 10^9. Use the `pmem` and `pvmem` resource to specify on the -l line. `pmem` specifies physical memory needed per process and `pvmem` specifies virtual memory needed per process (physical + disk paging space, which should only be slightly larger than pmem.) See the man page for `pbs_resources` for more info.

```
# This job is a single process that needs 8gb of physical memory and 9gb of virtual memory
```

```
# (it can push 1GB into swap/paging space.)
#PBS -l nodes=1:ppn=1,pmem=8gb,pvmem=9gb
# It should be allowed to run for up to 1 hour.
#PBS -l walltime=01:00:00
# Name of job.
#PBS -N mybigmemjob
# Join STDERR TO STDOUT.  (omit this if you want separate STDOUT AND STDERR)
#PBS -j oe
# Send me mail on job start, job end and if job aborts
#PBS -M kapoodle@uvm.edu
#PBS -m bea
```

Presently we have 1 machine with 32GB of memory (shared1), and 1 machine with 128GB of memory (shared2) for running these kinds of jobs.

Note that the `pmem` resource is defined as *physical (workingset) memory per process*. So, if you are running an OpenMP job with 4 processes (`nodes=1:ppn=4`), specifying `pmem=4gb` will reserve 16gb of physical memory for your job. The same goes for `pvmem`, which is *virtual memory per process*.

The same applies for MPI jobs. Since the default is pmem=800mb,vmem=1gb (allowing 2 processes to fit on a machine with 2GB memory), if you need more than that you will need to specify it.

You should, in general, request slightly more virtual memory than physical memory. About 110% of your physical memory is a good place to start. However, keep in mind that virtual memory used in excess of physical memory is usually paged out to local disk, which will hurt the performance of your job significantly.

MPI jobs

For jobs that take advantage of multiple processors, TORQUE will help communicate to your program which nodes it should run on. You just need to make use of the right variables.

LAM

TORQUE provides a number of environment variables to your running jobs. One of these is the **PBS_NODEFILE** variable, which points to a local file which lists the nodes your job can run on. You can take advantage of this by following this scriptfile example:

```
# this job needs 8 nodes, 16 processors total
#PBS -l nodes=8:ppn=2
# job should be allowed to run for up to 12 hours
#PBS -l walltime=12:00:00
#PBS -N myjob
#PBS -j oe
#PBS -M kapoodle@uvm.edu
#PBS -m bea

echo "This is myjob being started on" `hostname`
cd ~/myjob
mpiexec -machinefile $PBS_NODEFILE ./myprogram
```

Start the job with `qsub` *`scriptfile`* as usual.

MPICH

Note that to use MPICH, you will need to use `switcher` to choose the correct environment. See Libraries.

Also note that the MPICH-GM libraries make use of Myrinet for high performance. You will need to make sure that you request the `myri` resource for your job so that it doesn't end up on a node without a Myrinet connection. The MPICH-ch_p4 libraries use regular ethernet and can run on any node.

The syntax for `mpiexec` when using MPICH is slightly different than when using LAM. Please refer to the man pages.

Note: **Do not use mpirun** when using MPICH. It does not work well with TORQUE (PBS).

Example scriptfile:

```
# this job needs 8 nodes, 16 processors total
#PBS -l nodes=8:ppn=2
# it needs to run for 4 hours
#PBS -l walltime=04:00:00
#PBS -N myjob
#PBS -j oe
#PBS -M kapoodle@uvm.edu
#PBS -m bea

echo "This is myjob being started on" `hostname`
cd ~/myjob
mpiexec ./myprogram
```

Interactive jobs

Interactive jobs are discouraged, because a compute node gets tied up simply waiting for your keystrokes. But in a parallel debugging scenario, there may be no other way to do what you want.

`qsub -I` will start an interactive job and give you a shell on the first assigned node.

MATLAB jobs

You may get enhanced performance and some freedom from MATLAB licensing restrictions if you use the MATLAB compiler.

The MATLAB Distributed Computing Engine allows distributed and parallel jobs to be submitted from within the MATLAB interpreter itself. Jobs are submitted to TORQUE (PBS) and are scheduled following the same policy as other jobs. There is a separate page documenting this.

Limitations

Walltime limit

The default queue has a walltime limit of 30 hours, so you can specify up to that long when submitting a job:

```
qsub -l walltime=30:00:00
```

At this time, there are no queues with a longer walltime limit. The walltime limit is enforced in order to allow the queue to make progress, and to require users to periodically save state. In the event that one of the nodes a job is running on crashes, it should not lose more than 30 hours worth of work.

If, for whatever reason, it is too difficult to checkpoint your job within the 30 hour walltime, you might try VACC Checkpointing manually with blcr.

1 week queue

If you are willing to wait a long time to get a node, there are a small number of nodes set aside for very long runs of up to 1 week. These are in a separate queue, called l1wkq. To submit a job to this queue, add `-q l1wkq`, like in the following example:

```
qsub -q l1wkq -l walltime=4:00:00:00 myjob
```

This example will submit "myjob" to run for up to 4 days.

If you can possibly checkpoint and request shorter walltimes, you are better off doing so. Your job will start faster, and you run less risk of a node crashing and wasting days of compute effort. While some compute nodes will run for months on end, others regularly fail and are considered fungible.

Using the scheduler and monitoring jobs - Maui

Maui is the scheduler we are currently using. Here are some links to documentation for it:

- Maui users guide (http://www.clusterresources.com/products/maui/docs/mauiusers.shtml)
- Maui admin guide (http://www.clusterresources.com/products/maui/docs/mauiadmin.shtml)

Here are some brief examples of common commands:

showq

`showq` will show what the currently queued and running jobs are. It will also show the remaining walltime allocated to each job.

```
[jtl@bluemoon-user1 home]$ showq
ACTIVE JOBS--------------------
JOBNAME            USERNAME      STATE  PROC   REMAINING            STARTTIME

47709                   jtl    Running    24     4:28:46  Fri Aug 11 08:59:51
47714                   jtl    Running    24     8:11:12  Fri Aug 11 12:42:17
47715              kapoodle    Running    10  1:10:52:40  Fri Aug 11 13:23:45
47694              kapoodle    Running    32  1:23:11:19  Thu Aug 10 01:42:24
47713              kapoodle    Running    16  6:23:17:39  Fri Aug 11 13:48:44

    5 Active Jobs     106 of  114 Processors Active (92.98%)
                       53 of   54 Nodes Active      (98.15%)

IDLE JOBS----------------------
JOBNAME            USERNAME      STATE  PROC     WCLIMIT            QUEUETIME


0 Idle Jobs

BLOCKED JOBS----------------
JOBNAME            USERNAME      STATE  PROC     WCLIMIT            QUEUETIME


Total Jobs: 5   Active Jobs: 5   Idle Jobs: 0   Blocked Jobs: 0
```

checkjob

When you want to know "why the heck isn't my job running yet"?

```
[jtl@bluemoon-user1 maui]$ checkjob 47717

checking job 47717

State: Idle
Creds:  user:jtl  group:bin  class:workq  qos:DEFAULT
WallTime: 00:00:00 of 00:30:00
SubmitTime: Fri Aug 11 14:43:12
  (Time Queued  Total: 00:01:01  Eligible: 00:01:01)
...
```

Here you can see what the current state of the job is (Idle, or waiting), that it's used no walltime (hasn't started yet), when it was submitted, and how long it's been sitting in a queue.

```
...
```

```
Total Tasks: 128

Req[0]  TaskCount: 128  Partition: ALL
...
```

In Maui-speak, "tasks" are equivalent to the number of processors your job needs.

```
...
Reservation '47717' (2:23:04:31 -> 2:23:34:31  Duration: 00:30:00)
PE:  128.00  StartPriority:  1
job cannot run in partition DEFAULT (insufficient idle procs available: 8 < 128)
```

Here, we see that the job can't run because there are not enough processors available yet, but it has been granted a reservation. Jobs that are waiting to run and are near the top of the queue (in the first 3 spots) typically are granted an automatic reservation by the scheduler. You can see here that the job is currently scheduled to run in 2 days, 23 hours... (Ouch!) I may reconsider whether I really need 128 processors to run my job.

showstart

If you just want to know when your job is likely to start, `showstart` is the command you want.

```
[jtl@bluemoon-user1 maui]$ showstart 47717
job 47717 requires 96 procs for 00:30:00
Earliest start in       6:22:53:31 on Fri Aug 18 13:48:44
Earliest completion in  6:23:23:31 on Fri Aug 18 14:18:44
Best Partition: DEFAULT
```

This is the "earliest start" based upon all of the jobs that are currently running, and those that have reservations, running all the way to their maximum requested walltime. Jobs that are ahead of your job in the queue but do NOT have a reservation are not included in this estimate, so if there are many jobs ahead of yours, your actual start time is probably quite a bit later.

Unless your job has a reservation, there's no guarantee that it will start at the displayed time. As of the time of this writing (15:37, 22 July 2008 (EDT)) jobs tend to run to their maximum walltime. To check if your job has a reservation, use `checkjob`.

showbf

If you want your job to run immediately, you can use `showbf` to see what processors are currently available (unused) and for how long:

```
[jtl@bluemoon-user1 maui]$ showbf
backfill window (user: 'jtl' group: 'etsstaff' partition: ALL) Fri Aug 18 14:28:54

   34 procs available for 10:25:30
   26 procs available for  7:00:19
    1 proc available with no timelimit
```

Ordinarily, if I submit a job now that uses 34 processors for up to 10 hours, it would run immediately.

However, during the upgrade in February 2008, partitions were added between the "old" myrinet-2000 nodes, the "new" myrinet10g nodes, and the shared memory nodes, because running a single multiprocessor job, particularly an MPI job, across these different nodes would either be counterproductive or fail completely, because they don't share a high-speed interconnect. So plain old "showbf" isn't very useful any longer.

To see what the backfill available is on any partition, I use the `-p ANY` flag.

```
bash-3.1$ showbf -p ANY
backfill window (user: 'jtl' group: 'etsstaff' partition: ANY) Sat Feb 16 09:26:42
```

```
 56 procs available for       2:09:24
 44 procs available for      18:38:30
 42 procs available with no timelimit

partition DEFAULT:
  2 procs available with no timelimit

partition myri2g:
 27 procs available with no timelimit

partition myri10g:
 27 procs available for       2:09:24
 15 procs available for      18:38:30
 13 procs available with no timelimit
```

So, I could submit a single job using up to 27 processors, and it should run immediately (in myri2g if more than 2:09, in myri10g if less.)

If I have a large-memory job, I use the -M flag. For example, If I need 12 gigs of memory per process, I'd do something like this:

```
[jtl@bluemoon-user1 ~]$ showbf -M 12000
backfill window (user: 'jtl' group: 'bin' partition: ALL) Mon Sep 11 11:49:28

  2 procs available with no timelimit
```

So I can run 2 processes with 12 GB per process.

If I want to see what processors are available that use Myrinet (the 'myri' feature):

```
[root@bluemoon-mgmt1 etc]# showbf -f myri
backfill window (user: 'root' group: 'root' partition: ALL) Wed Oct 11 10:06:58

 58 procs available with no timelimit
```

Monitoring jobs - PBS/Torque

These commands are provided by Torque, the resource manager:

qstat

`qstat` will print out the currently queued, running, and recently exited jobs.

```
[jtl@bluemoon-user1 ~]$ qstat
Job id           Name             User             Time Use S Queue
---------------- ---------------- ---------------- -------- - -----
37.bluemoon-mgmt hpl-2proc        jtl              00:00:02 R exec1
38.bluemoon-mgmt ...procs-realbig jtl              00:00:00 R exec1
39.bluemoon-mgmt ...procs-realbig jtl              00:00:00 R exec1
```

You can see from the above that there are 3 jobs in the queue, and all three are running (**S=R**).

`qstat -r` will give slightly more detail (but only for jobs that are currently running):

```
bluemoon-mgmt1.cluster:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
37.bluemoon-mgm jtl      exec1    hpl-2proc    3823   1  --    --     --  R 01:54
39.bluemoon-mgm jtl      exec1    hpl-32proc    --   16  --    --     --  R 01:44
```

This shows the number of nodes assigned to each job, and the elapsed runtime (wall clock time, not CPU time.)

To get more detailed job status, use `qstat -f`. This will give details such as the nodes your job is currently running on, the environment variables set, the amount of resources used so far, or if your job has not started yet, the reason why.

For more information on TORQUE/PBS

Run `man` for the following commands:

- `qsub` - submitting jobs
- `qstat` - getting status of jobs
- `pbsnodes` - getting status of compute nodes
- `qdel` - cancelling or deleteing jobs
- `qalter` - altering a queued job, e.g. execution time, start time, holds, joins, etc

Other useful man pages:

- `pbs_resources` - for information on the different types of resources available - memory, specific hosts, architectures, etc

If this documentation needs more detail or to be corrected, let me know (Jtl), put a note on the Talk page or edit this doc yourself! It's a Wiki after all...
