Hyak Job Scheduler

Overview

Please refer to the following pages for information on both the ikt.hyak (hyak classic) and mox.hyak (hyak next gen) scheduler:

Mox_scheduler
Mox_mpi

Information below is obsolete

The information below is obsolete, and for historical purposes only. Please ignore. For information on the current Hyak scheduler, please see the links at the top of this page.

All tasks on ikt.hyak are run by the job manager `qsub`. The jobs can be run either "right away" or as backfill jobs later. By default, your jobs will run immediately (in practice during a next few minutes) but only on the nodes owned by your default group (given the nodes are free). If you are a member of more than one group and want your job to execute on the nodes owned by a group other than your default, see the instructions below.

Alternatively, you can run the job as backfill job with access to all nodes. Backfill jobs execute on idles CPUs throughout the cluster. They are subject to immediate preemption by node owner’s jobs or every 4 hours, whichever comes first. See the Backfill Queue Section for a more information about backfill jobs.

To run a job, you have to create a PBS file (Portable Batch System), see the examples below. The file will be run by `qsub` command like

```bash
qsub myjob.pbs
```

for instant run, or

```bash
qsub -q bf myjob.pbs
```

for adding the job in the backfill queue to be run later. You can read more about `qsub` options at the Adaptive Computing webpage.

All examples and templates for job submission scripts use the bash shell. We strongly encourage all users to base their job submission scripts on our examples. C shell users might sometimes experience trouble with paths or variables set in their shell failing to transfer to the bash environment of the job script. Loading the appropriate environment modules within a job submission script can address this issue in cases where the user relies on modules to define her environment.

Important Information

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

- A job should use all the processors (cores) in the node to which it’s assigned (only 1 job can run on each node). e.g. `-l nodes=1:ppn=16, feature=16core`
- The more accurate an estimate you provide for job walltime, the better the scheduler can make effective use of your nodes.
- Specification of accurate memory requirements is critical (e.g. `-l mem=40gb`)
- Be aware of the capabilities of the nodes in your allocation. If you ask for more resources than available in your group, the script will never run. It will sit in the queue indefinitely without any obvious error message.
- Never use msub
- The longer a job is in the queue the higher its priority. The scheduler tries to run the highest priority job first. Placing a hold on a job does not reset its priority.
- Only the 25 least-recently submitted of each user’s jobs waiting in the queue will accumulate scheduler priority (see the #Backfill (bf) Queue section for more detail)
Node local scratch disks (n####:/tmp, n####:/scr, n####:/var/tmp) are cleaned up after the completion of each job. Groups that would like to keep data on the disks in their nodes should save any data into a directory with their group name. If your UNIX group is hyak-mygroup, your directory should be named mygroup.

Limit data sent to standard output and standard error. Redirect or quiet noisy status output from applications.

Do not query with the scheduler with automated scripts more than once per minute.

Submitting Jobs

Jobs are submitted by `qsub` command to be taken care of by the scheduler. In order to submit a job, you have to create a PBS jobscript, essentially a bash script with PBS directives.

PBS Jobscripts

PBS (Portable Batch System) scripts contain three types of directives:

1. instructions for the scheduler
2. setting up the work environment
3. executing your production program

Jobscript file should be executable (use `chmod` like `chmod +x script.pbs`). Otherwise you may get weird errors as shell in the scheduler tries to execute text output from your .bash_profile.

Instructions to the scheduler

Instruction lines start with `#PBS`, for instance

```
#PBS -N test
#PBS -l nodes=2:ppn=16,mem=40gb
```

tells the scheduler to call the program "test" and that it uses 2 nodes with 16 cores each and 40GB memory in total (across all nodes and processors). The options are mostly the same as `qsub` options.

Working Environment

You most likely need to load modules before you can start running your program. You may add lines like

```
module load r_3.2.0
module load icc_14.0.3-ompi_1.8.3
```

to the jobscript. You may also need to do personal setup and cleanup jobs, and print diagnostic information here. This looks like an ordinary bash-script.

Executing Your Code
Finally, executing your programs are also done like in an ordinary bash script. For instance, this line may look like:

```
mpirun --mca mtl mx --mca pml cm --bind-to core --map-by core Rscript >output.txt 2>&1 test.R
```

A simple minimal jobscript may look like this (See further below for the meaning of the options) (Below xyz is your group and abc is your userid):

```
#!/bin/bash
#PBS -N "single_word_logical_job_name"
#PBS -d /gscratch/xyz/abc/mydirectory
#PBS -l nodes=1:ppn=16,feature=16core,mem=40gb,walltime=24:00:00
#PBS -M user@u.washington.edu
#PBS -m abe
module load r_3.2.0

Rscript >output.txt 2>&1 test.R
```

### Useful Submission Options

```
qsub accepts a number of arguments, the most
important of which is perhaps "-l". This allows you to overwrite the requirements
specified in the PBS file (see below). For instance

```
qsub -q bf -l nodes=2:ppn=16,walltime=00:15:00 gridsearch.pbs
```

runs a jobscript named "gridsearch.pbs" as a backfill job and tells the scheduler that it needs 2 nodes with 16 cores each (ppn = Processors Per Node), and the program runs no more than 15 minutes (it will be killed if it does not finish in 15 minutes.)

### Nodes and Cores

- All jobs must specify if they will use 16 core nodes. They will be allocated all the cores on each node even if you really just need one core.

```
qsub -l nodes=1:ppn=16,feature=16core script.pbs
```

- Note that nodes and cores may be pooled: if you ask for two nodes with 16 cores each, you may be allocated 1 node with 32 cores instead. The opposite is not true: if you ask for a single node with 16 cores, you will have to wait until such a node is available, even if many 8-core nodes are idle.

- If you fail to specify PPN, the scheduler assumes you request one processor per node (and most likely pooles these on a single node).

- Using pre-defined host lists, e.g. host n0001 through this is rarely necessary

```
qsub -l nodes=n0001:ppn=16 script.pbs
```
Memory

- Memory can be specified as `mem=` for total memory, or `pmem=` for per-cpu memory:

  ```sh
  qsub -q bf -l nodes=1:ppn=16,mem=32gb
  qsub -q bf -l nodes=1:ppn=16,pmem=2gb
  ```

- specify the same amount of total memory

Features

- Schedule a job using 32 cores (CPUs) on any nodes available with a specific processor type

  ```sh
  qsub -l nodes=2:ppn=16,feature=16core:intel script.pbs
  ```

Groups

- If you are a member of more than one group, you can specify the group's nodes to use by running (you can only specify one group). You must also use `qsub` rather than `msub`:

  ```sh
  qsub -W group_list=hyak-groupname script.pbs
  ```

  `hyak-groupname` is the name of the group associated with the nodes on which you want to run. You can get a list of your group memberships by running the command `groups`.

Other Options

- You can specify a chain of resources requirements separate with ','. Be realistic—if your nodes have 24GB of RAM and you request 24GB of RAM, your job will never run as part of the memory is taken by OS.

  ```sh
  qsub -l nodes=1:ppn=16,feature=16core,mem=40gb,walltime=10:00:00 script.pbs
  ```

Submitting Parallel Jobs
You cannot gain much from running your code on Hyak unless you run it in parallel. There are (at least) two different ways how your code can be made parallel:

- You start a single instance of your program, and later your code takes advantage of the fact that it actually can control many nodes and processor cores. In your pbs script you simply run your code as (R example):

  ```
  Rscript >output.txt myscript.R
  ```

  Now your script is responsible to delegating tasks to the other nodes. Note that the allocated nodes are listed in the file, pointed to by the environmental variable `PBS_NODEFILE`.

- Alternatively, you may start the programs in parallel on all processor cores in a MPI mode by writing in your pbs script

  ```
  mpirun <options> Rscript >output.txt myscript.R
  ```

  Now you have many instances of Rscript running and they must sort it out themselves using MPI methods (such as `Rmpi` package in case of R). Note also that you need a mpi-module (such as `icc_14.0.3-ompi_1.8.3`) loaded to make MPI-tools, including `mpirun`, available.

### Sample Job Scripts

The following are sample pbs-scripts for MPI parallel jobs.

<table>
<thead>
<tr>
<th>COMPILERS</th>
<th>GNU Compilers</th>
<th>Intel Compilers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MPICH2-MX</strong></td>
<td>gnu-mpich.sh **</td>
<td>intel-mpich.sh **</td>
</tr>
<tr>
<td><strong>Open MPI</strong></td>
<td>gnu-openmpi.sh</td>
<td>intel-openmpi.sh</td>
</tr>
<tr>
<td><strong>Intel MPI</strong></td>
<td>gnu-impi.sh</td>
<td>intel-impi.sh</td>
</tr>
<tr>
<td><strong>MPICH3</strong></td>
<td>gnu-mpich3.sh</td>
<td>intel-mpich3.sh</td>
</tr>
</tbody>
</table>

** Unsupported

* Use Intel MPI unless it’s not supported by your application

READ AND UNDERSTAND THE EXAMPLE JOB SCRIPTS. Every line. If your jobs won’t run or crashes or have any problems, review what you are doing in the context of the examples we provide. Asking the Hyak admins to debug your job submission scripts takes time away from their efforts to improve Hyak for everyone.

Other Information

For more information and sample sessions about how to set up MPI jobs, please see the MPICH2-MX, Open MPI, and Hyak Software pages.

### Submitting Serial Jobs

Hyak uses node level scheduling. This means that single threaded tasks need to be bundled together since each job must use all the CPUs on the node to which your job is assigned. Most users should use `parallel-sql` to achieve this. It's a very easy way to run and manage your serial jobs. We provide a link to our vanilla GNU parallel page for posterity.

### Common Solutions for Job Problems

1. Look in `<jobname>.o<jobid>` or `<jobname>.e<jobid>` for error messages. However, this file is created when the program finishes. In order to see the output and errors in real time, you should redirect the output when invoking the program. For instance, in the `jobscript` you may write:

   ```
   Rscript >>output.txt 2>&1 myprog.R
   ```

   ```
   Submitting Serial Jobs

   Hyak uses node level scheduling. This means that single threaded tasks need to be bundled together since each job must use all the CPUs on the node to which your job is assigned. Most users should use `parallel-sql` to achieve this. It's a very easy way to run and manage your serial jobs. We provide a link to our vanilla GNU parallel page for posterity.

   ```
   Common Solutions for Job Problems

   1. Look in `<jobname>.o<jobid>` or `<jobname>.e<jobid>` for error messages. However, this file is created when the program finishes. In order to see the output and errors in real time, you should redirect the output when invoking the program. For instance, in the `jobscript` you may write:

   ```
   Rscript >>output.txt 2>&1 myprog.R
   ```

   ```
   Submitting Serial Jobs

   Hyak uses node level scheduling. This means that single threaded tasks need to be bundled together since each job must use all the CPUs on the node to which your job is assigned. Most users should use `parallel-sql` to achieve this. It's a very easy way to run and manage your serial jobs. We provide a link to our vanilla GNU parallel page for posterity.

   ```
   Common Solutions for Job Problems

   1. Look in `<jobname>.o<jobid>` or `<jobname>.e<jobid>` for error messages. However, this file is created when the program finishes. In order to see the output and errors in real time, you should redirect the output when invoking the program. For instance, in the `jobscript` you may write:

   ```
   Rscript >>output.txt 2>&1 myprog.R
   ```
This tells bash to redirect all the output and error messages to file output.txt. Such redirected output is immediately visible, and the file is not overwritten in case your code is preempted.

2. Ensure you're compiling and running your application with the same environment.
   a. All environment modules you load to compile your code should also be loaded in your job script.
   b. Select the job script from above that meets your requirements, then base your compile environment on that.
   c. Do not load environment modules in your login scripts.
   d. Load only the modules that are not necessary for running your job in your job script.

3. Be sure to use one the example jobs script we provide above: serial, parallel.

4. Your login scripts should not produce any output. Output from login script will be interpreted by the scheduler (and usually gives errors) unless your PBS-jobscript is executable.

5. You should use the default bash shell and reset your login scripts to the defaults.

6. If Intel MPI is not compatible with your code, try Open MPI.

7. Configure your environment to support passwordless ssh logins among nodes, instructions here.

8. Remove the limit for stack usage by adding `ulimit -s unlimited` to your job script.

9. Disable the MX Registration Cache `export MX_RCACHE=0` to your job script.

10. Remove any restart files and start the simulation from the beginning.

11. If you receive, "Job exceeded some resource limit (walltime, mem, etc.). Job was aborted. See Administrator for help," specify a longer walltime.

12. Check to ensure you're not over either home or gscratch disk quota. See the Managing your files page for more detail.

Managing Jobs in Your Queues

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

Job, Queue, and Node Status

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

Job Status

- Check the status of your job
  
  checkjob <job_id>
  qstat -f <job_id>

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

Queue Status

- Show jobs based on group (e.g. hyak-hpc) for different queues. Your group may not have all the queue types listed below. You can determine your group by using the id command.
  
  showq -w qos=hpc
  showq -w qos=hpc-bf
  showq -w qos=hpc-int
  showq -w qos=hpc-gpu

- Show the status of your jobs
  
  showq -w user=<userid>

- Show jobs in the bf queue
  
  showq -w class=bf

- Show all jobs in all queues
  
  showq
- Show extended information for idle (queued) jobs
  
  showq -i
  showq -w qos=hpc -i
  showq -w class=bf -i

- To see what resources are immediately available for a given group (hyak-hpc) and core count on your group's node and in the bf queue.
  
  showbf -q hpc
  showbf -q hpc-bf -f 16core

  This command was created to help users schedule backfill jobs, but it's useful in other contexts, too

- Adaptive Computing provides good documentation for showq

---

**Obsolete**

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

---

**Node Status**

- Show a list of the nodes for a given group (e.g. hpc), the state of those nodes, and the cores available to preemptors (batch queue jobs)
  
  nodestate hpc
  nodestate hpc-int
  nodestate -a hpc

- To get a list of nodes and their hardware configuration for a given group's queue (e.g. hyak-hpc)
  
  mdiagn -t hpc
  mdiagn -t hpc-int

- List all the nodes and some basic configuration information: usage state, processors per node, and memory.
  
  mdiag -n

- To get the attributes for a given node
  
  pbsnodes nXXXX

---

**Manipulating Jobs in Queues**

**Obsolete**

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

---

- To cancel a job
  
  mjobctl -c <job_id>
  qdel <job_id>

- To cancel multiple jobs
  
  mjobctl -c -w user=myuserid
  mjobctl -c "x:<job_id_prefix>*"

  Note that the above command line will kill all jobs starting with <job_id_prefix>. Check the mjobctl docs on the web for more information.

- To hold a job so it doesn't run until you release it
  
  mjobctl -h <job_id>

  Can be useful if you've queued a bunch of jobs then decided you want to rearrange the order in which they execute

- To release a job that you've held or which has annoyingly landed in state Deferred
  
  mjobctl -u <job_id>

  or

  qrls <job_id>

  Your job will switch to state Idle for a minute or so before starting to run (if resources, typically CPUs, are available.)

- Changing Job Priority
  
  mjobctl -w user=username -m userprio=N
Where \( N \) is a number between -1 and -1024. This can be used to lower the priority of some jobs so that others will be run first. Admins can also use this command to prioritize some of their groups' jobs over others.

### Queue Administration

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

You can designate up to five managers for your allocation. This allows the designated users to run `checkjob` and any `mjobctl` commands against jobs in that allocation.

- Check to see if your allocation has any job managers (if your group is hyak-hpc, your allocation name would be hpc)

  ```sh
  mdiag -a <allocation name> | grep Managers
  ```

- Cancel a job or jobs

  ```sh
  mjobctl -c <job_id>
mjobctl -c "x:<job_id_prefix>.*"
  ```

- Check the status of a job

  ```sh
  checkjob <jobid>
  ```

- Other functions

  Check the `mjobctl` docs on the web for more information.

### Interactive Sessions

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

Opening an interactive session is not instantaneous. Resources have to be available and the scheduler has to get them ready for use. However, if resources are available it should not generally take more than five minutes. You must request a node type that's available in your allocation. To see what nodes are available in your allocation, please see Job, Queue and Node Status.

Open an interactive session to a node in the current terminal. The default walltime is 60 minutes.

```sh
qsub -l walltime=2:00:00 -I
```

Open an interactive session to a node with X11 forwarding (you must have X11 forwarding set up to the login servers in order for this to work)

```sh
qsub -l walltime=2:00:00 -V -I
```

Build nodes have access to the Internet and are available to all users. The default walltime is 30 minutes, the max walltime is 8 hours.

```sh
qsub -l walltime=1:00:00 -q build -I
```

This is a handy shell alias for getting an interactive shell on any host with a long wallclock time (10 hours):

```sh
alias shell="qsub -l walltime=10:00:00 -I"
```

Note that if you are running memory-hungry jobs interactively, you may run out of C stack space even if there is still a plenty of free memory on the node. You will see an error like

```
Error: segfault from C stack overflow
Error: C stack usage is too close to the limit
```

In this case you should increase the stack size by `ulimit`, for instance

```sh
ulimit -s 30000
```
Single Command Interactive Sessions

Obsolete
This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

Some users primarily use interactive sessions and prefer to run one command from their workstation to obtain an interactive session on a node. You can do that with a command similar to one of the samples below or a combination.

```
ssh -t hyak.washington.edu /sw/torque4/bin/qsub -l nodes=1:ppn=16 -I
```

- With X11 forwarding
  ```
  ssh -X -t hyak.washington.edu /sw/torque4/bin/qsub -l nodes=1:ppn=16 -I -V
  ```

- With a multiplexed ssh connection
  ```
  ssh -t -S /home/me/hyak-socket-login hyak.washington.edu /sw/torque4/bin/qsub -l nodes=1:ppn=16 -I
  ```

- If your group has interactive nodes
  ```
  ssh -t hyak.washington.edu /sw/torque4/bin/qsub -q int -I
  ```

Other Useful Options and Information

Obsolete
This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

- When a job is submitted, there may be a delay of up to thirty seconds before the job shows up in the output of showq.
- E-mail notifications:
  * Add `-m abe -M user@u.washington.edu` where:
    * `a` mail is sent when the job is aborted by the batch system.
    * `b` mail is sent when the job begins execution.
    * `e` mail is sent when the job terminates.
    * `n` no mail is sent.
  * As a bonus, the job termination email will tell you EXACTLY how much memory your job used so that next time you can specify it more accurately in the resource request. However, the walltime is typically wrong and shows the time since the last preemption.
- Setting the wallclock run time (10 hours)

```
-l walltime=10:00:00 <script>
```

- Commonly used resources

<table>
<thead>
<tr>
<th>resource</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes</td>
<td>node list (n0001,n0002), node range (n0001-n0020), # of nodes</td>
</tr>
<tr>
<td>mem</td>
<td>total memory necessary for job, # followed by units (b, kb, mb, or gb)</td>
</tr>
<tr>
<td>ncpus</td>
<td>1. of cpus</td>
</tr>
<tr>
<td>walltime</td>
<td>maximum wallclock time for job, hhh:mm:ss</td>
</tr>
<tr>
<td>cput</td>
<td>maximum cpu time for job, hhh:mm:ss</td>
</tr>
<tr>
<td>file</td>
<td>available space required on a node's local scratch disk followed by units (kb, mb, or gb)</td>
</tr>
</tbody>
</table>

- Example header for a script

```
#!/bin/sh
#PBS -N "single_word_logical_job_name"
#PBS -d /scratch/working/directory
#PBS -l nodes=1:ppn=16,feature=16core,mem=40gb,walltime=24:00:00
#PBS -M user@u.washington.edu
```
### Queues

**Obsolete**

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

### Build

**Obsolete**

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

Build nodes have access to the Internet and are available to all users. The default walltime is 30 minutes, the max walltime is 8 hours.

```bash
qsub -q build -I -l walltime=3:00:00
```

### Backfill Queue

**Obsolete**

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

Idle CPUs throughout the cluster are available for jobs submitted to the **backfill queue (bf)**. Backfill jobs are subject to immediate **preemption** by node owner's jobs. For this reason jobs which require large numbers of nodes or which require long, uninterrupted stretches of runtime are typically poor backfill candidates. Other guidelines to follow when using the backfill queue:

- **Interactive jobs** `-I` **cannot** be run in the `bf` queue.
- **Always specify exactly the CPU resources and core feature required by your job.** Refer to the other sections on this page for more details.
- **Jobs are preempted (canceled and requeued) at least every 4 hours.** Therefore your jobs must implement some checkpointing scheme (save their progress). Failure to checkpoint ensures your job will simply repeat work every time it is restarted.
- **Use all the cores on a node.** If you're running serial tasks, use `parallel-sql`.
- **Start Small!** Seeing the potential to run on hundreds or thousands of cores "for free" can be tempting, but resist the urge to dive in with both feet on your first try. Instead, start with a single job to see how things work. If that goes well, try two jobs. Build up gradually. If you change your procedures in any way, test things thoroughly before submitting any new jobs. Be a good neighbor.
- **Set the walltime of your job to 260 minutes (and no longer) if you expect your job to run longer than 4 hours.** Setting a very long walltime for your `bf` queue jobs will cause them not to run when they otherwise could since the scheduler does not know the job will be preempted in ~4 hours automatically.
- **Jobs should run for at least an hour.** Do not submit tens of thousands of jobs that run for only a few minutes. These sorts of jobs should be **[bundled together](Hyak Job Scheduler#Submitting Serial Jobs)** into longer running jobs. Submitting many short run time jobs is strain on the job scheduler and it's very inefficient. On average, there's about ninety seconds worth of overhead incurred by the scheduler to process and start your job. When your job only runs for a minute, your job is mostly overhead.
- **Avoid using the node-local scratch disks.** Everything you write to the node-local scratch disks will be deleted when your job is **preempted.** If you've written a lot, then this can be a lengthy process and can delay the node owner's job startup. For Hyak to work we all have to be good neighbors. You can read more about proper use of Hyak storage on the [Managing your Files](Hyak Job Scheduler#Managing your Files) page.
- **Limit output to standard output and standard error.** Since your job's standard output and standard error files must be copied back to the server when your job is preempted, redirect or keep any output made to standard output and standard error to a bare minimum. Standard output and standard error are the files named `<jobname>-o<jobid>` and `<jobname>-e<jobid>` respectively. This output may not exceed 1MB otherwise your job will be automatically canceled. You can redirect the output to a file `(myapp &> /path/to/somefile)` or you can redirect the output to `/dev/null` (`myapp &> /dev/null`) if you don't want it at all.
- **Limit queued and running jobs to 2500 or less.** Large numbers of jobs place a large burden on the job scheduler. If you have a lot of individual tasks to run, use `parallel-sql`.
- **Priority.** Job priority in the `bf` queue is based on the size of your allocation, your allocation's `bf` queue usage level, and in some cases your `bf` queue usage level.

Submitting a job to the `bf` queue:

```bash
qsub -q bf myjob.sh
```
Backfill Wait Queue

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

The backfill wait queue (bfwait) allows a few extra seconds for your application to perform an orderly shutdown when preempted. Your application and your job script must both be written to behave the appropriate way upon receiving a UNIX TERM signal however. One such application is parallel-sql. You can find an example job script on the parallel-sql page.

Interactive Nodes (Interactive queue)

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

Some groups in Hyak have nodes configured to allow multiple jobs on the same node. This is particularly useful if you want to set aside one or more of your group’s nodes to be available for interactive use. Jobs submitted to your standard batch (default) queue will not run on this node. You have to specify the 'int' queue when you want to run jobs on your interactive node(s). You can use all the same options in the 'int' queue as you do in default (batch) queue.

Allocate one core on your interactive node for your use in your interactive session:
```
qsub -q int -I
```

You can also run batch jobs through your interactive node, where # is the number of cores you’d like to use for your job:
```
qsub -l nodes=1:ppn=# -q int pbscript.sh
```

To see if your group has interactive nodes see the node status section of this page.

If you're interested in setting aside one of your nodes for interactive use, please send an e-mail to help@u.washington.edu.

GPU Nodes (GPU queue)

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

There are GPU nodes in Hyak available for use in the Backfill queue. All the rules of the backfill queue listed above apply to the GPU nodes as well. There are 11 nodes with 4 GPUs, 1 node with 1 GPU, and 1 node with 2 GPUs. All the nodes have 12 CPU cores.

CUDA Libraries
```
module load cuda_<version>
```

Submission Instructions for Groups that Own GPU Nodes
```
qsub -l nodes=X:ppn=Y:gpus=Z -q gpu <job script>
qsub -l nodes=X:ppn=Y:gpus=Z -q gpu -I
if using a non-default group
qsub -l nodes=X:ppn=Y:gpus=Z -q gpu -W group_list=hyak-group -I
```

Submission Instructions for Groups that Do Not Own GPU Nodes
```
qsub -l nodes=X:ppn=Y:gpus=Z -q bf <job script>
```

Job Logs

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

Logs of jobs are saved in /sw/joblogs. Job logs are stored in two formats: plain text and XML. You can also search the logs using the jobsearch utility.
-d     list jobs from specific date (format: YYYYMMDD)
-e     show every record for a job (only most recent by default)
-A     search/show records from all days available (only current day by default)
-o     show osg queue jobs
-p     show POSIX time instead of localtime

Reporting

This information is obsolete. Please see the links at the top of this page for information about the current Hyak scheduler (for both Ikt and Mox clusters).

Three reports regarding Hyak usage are available in the /sw/reports directory on Hyak. They are all in csv format. The fields are processor hours, number of jobs, year, month, group name. These reports are updated hourly.

- phoursByQos.csv: Processor hours used by group (QOS) by month. Usage is divided into standard allocation usage and bf queue usage (e.g. esci and esci-bf). This data is complete starting February 2011.
- phoursByGroup.csv: Processor hours used by group by month. This data is complete starting July 2010 and is provided primarily for historical purposes.
- phoursByUser.csv: Processor hours used by user in all allocations by month. This data is complete starting July 2010.

Reports are also produced for each group by user. These reports are located in /sw/reports/byGroup. The reports are all in csv format. These reports include additional information like when the user's account was added and details about their recent Hyak usage. Users' Hyak usage is not tracked separately by group. If a user is in multiple groups, the user's total usage will be displayed in all reports.

If you'd like a list of usage in all allocations by user for a given group, you can use the usersByGroup script (i.e. usersByGroup hyak-group).

You can also refer to the material prepared for the Hyak Governance Board.