Intermediate Usage of
Shared Computing Cluster (SCC)
Topics for Today

- Introduction
- The Cluster
- Running Jobs
- Monitoring Jobs
- Resource Bottlenecks
- Workflow and Job Management
- Bonus Material

Notes:
- Not Hands-on
- Overview of concepts
- Discuss common challenges
- Please ask questions
- Time at the end for personal help
Research Computing Services
Research Computing Services (RCS)

A group within Information Services & Technology at Boston University provides computing, storage, and visualization resources and services to support research that has specialized or highly intensive computation, storage, bandwidth, or graphics requirements.

Three Primary Services:

- Research Computation
- Research Visualization
- Research Consulting and Training
Me

- Research Facilitator and Administrator
- Background in biomedical engineering, technology, and bioinformatics
- RCS has offices on both CRC and BUMC
- Contact: help@scc.bu.edu
You

- Who just came from the Intro to SCC tutorial?
- Who has an account on SCC?
- Who has used SCC for more than 1 month?
  - 6 months?
  - 1 year?
- Who has used other compute clusters?
The Shared Computing Cluster
Shared Computing Cluster

- **Shared** - Transparent multi-user and multi-tasking environment
  - Also, buy-in

- **Computing** - Heterogeneous environment for
  - Interactive jobs
  - Single processor and parallel jobs
  - Graphics jobs

- **Cluster** - Many connected computers
  - Connected via fast local area network
  - Jobs scheduler coordinates work loads.
Massachusetts Green High Performance Computing Center
Shared Computing Cluster

Server Cabinets

Rear View

- Ethernet Switch
- Infiniband Switch
- Compute Chassis

Compute Nodes
SCC Architecture

Login Nodes
SCC1
SCC2
SCC3
SCC4

Public Network

Compute Nodes

Private Network

File Storage
6PB

VPN only

900 nodes
19,000 CPUs
300 GPUs

File Storage

900 nodes
19,000 CPUs
300 GPUs
SCC Resources

- Processors: Intel and AMD
- CPU Architecture: nehalem, sandybridge, ivybridge, bulldozer, haswell, broadwell, skylake, cascadelake knl, epyc
- Ethernet connection: 1 or 10 Gbps
- Infiniband: EDR, FDR, QDR (or none)
- GPUs: NVIDIA Tesla M2050, M2070, K40m, P100, V100
- Number of cores: 8, 12, 16, 20, 24, 28, 32, 36, 64 / node
- Memory: 24GB – 1TB / node
- Scratch Disk: 244GB – 1TB /node

http://www.bu.edu/tech/support/research/computing-resources/tech-summary/
Running Jobs

Interactive, Interactive Graphics, Batch
The Login Nodes

Login nodes are designed for light work:

- Text editing
- Light debugging
- Program compilation
- File transfer

Anything else should be done on a compute node with a “Job”
Running Jobs: Types of Jobs

Interactive Job
- Interactive shell, run GUI applications, code debugging, benchmarking of serial and parallel code performance.

Interactive Graphics Job
- Interactive shell with GPU and hardware acceleration for software with advanced graphics.

Non-Interactive “Batch” Job
- Controlled script or binary execution.
Interactive Jobs

“qrsh” - Request from the queue (q) a remote (r) shell (sh)

- Interactive shell
- GUI applications
- Code debugging
- Benchmarking
Interactive Jobs

Multiple Interactive Modes!

- Mostly do the same thing.
- We usually teach **qrsh**
  - Doesn’t require X11
- You have options.

<table>
<thead>
<tr>
<th></th>
<th>qsh</th>
<th>qlogin / qrsh</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-forwarding is required</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td>Session is opened in a separate window</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td>Allows X-Forwarding</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Current environment variables passed to session</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td>Batch-system environment variables ($NSLOTS, etc.)</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Interactive Jobs

Request appropriate resources for the interactive job:

- Some software (like Matlab, Stata) might use multiple cores.
- Make sure to request enough time if you need more than 12 hours.
- Make sure to request enough memory (4GB min is often not enough).

More on this later.
Interactive Graphics (VirtualGL) Jobs

“qvgl” - Request from the queue (q) a VirtualGL (vgl) session

- Combines an Interactive Job with a dedicated GPU resources and VNC.
- Provides **OpenGL for full 3D graphics**
  - fMRI Applications
  - Molecular Modeling
- Provides **GPU hardware acceleration** for visualization

VirtualGL offering is a very limited resource.
Most applications (Matlab, RStudio, QGIS, etc) do **NOT** need Virtual GL to use a GUI.
Interactive Graphics (VirtualGL) Jobs

- Request qvgl job
- Forward port for VNC from compute node to scc1
- Open VNC and connect
- If OpenGL is needed, precede app with “vglrun”
- Delete job when finished

```
[cjahnke@scc ~]$ qvgl -P project
Waiting for immediate job to be scheduled.

Your immediate job 5278659 has been successfully scheduled.

on your local workstation run:
    ssh cjahnke@scc-ha2.bu.edu -L 7000:localhost:5901

then execute vncviewer to connect to localhost port 7000
    vncviewer localhost:7000

your VirtualGL gpu display is: :0.0

to run an OpenGL application on a hardware GPU use vglrun command
for example:
    vglrun glxinfo

to quit the job please run:
    qdel 5278659
```
Non-Interactive “Batch” Jobs

“qsub” - To the queue (q) submit (sub) a script or program

- Script or program
- Blind execution

[cjahnke@scc1 ~]$ qsub test.qsub
Your job 9253374 ("test") has been submitted
[cjahnke@scc1 ~]$
Non-Interactive “Batch” Jobs

- **Using a Script**
  - Submit script to scheduler
  - After completion, we get an output file.
  - Example:
    
    ```bash
    [cjahnke@scc1 ~]$ qsub test.qsub
    Your job 9253374 ("test") has been submitted
    [cjahnke@scc1 ~]$
    [cjahnke@scc1 ~]$ ls
    test.qsub  test.o9253374  results
    ```

- **Using a Binary**
  - Submit binary “cal”
  - Example:
    
    ```bash
    [cjahnke@scc1 ~]$ qsub -b y cal -y
    Your job 542359 ("cal") has been submitted
    [cjahnke@scc1 ~]$
    [cjahnke@scc1 ~]$ ls
    cal.e542359  cal.o542359
    ```
Non-Interactive “Batch” Scripts

Script Interpreter

Scheduler Directives

Task Commands

#!/bin/bash -l
#$ -P rcs
#$ -N test
#$ -j y
#$ -m bae

# Load Modules
module load R

# Run the program
Rscript my_R_process.R
sleep 10

Execute login shell (for proper interpretation of the module commands)
## Scheduler Options - General Directives

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-P project_name</code></td>
<td>Project to which this jobs is to be assigned. <strong>Mandatory</strong> for all users associated with any BUMC project.</td>
</tr>
<tr>
<td><code>-N job_name</code></td>
<td>Specifies the job name. The default is the script or command name.</td>
</tr>
<tr>
<td><code>-o outputfile</code></td>
<td>File name for the stdout output of the job.</td>
</tr>
<tr>
<td><code>-e errfile</code></td>
<td>File name for the stderr output of the job.</td>
</tr>
<tr>
<td><code>-j y</code></td>
<td>Merge the error and output stream files into a single file.</td>
</tr>
<tr>
<td>`-m b</td>
<td>e</td>
</tr>
<tr>
<td><code>-M user_email</code></td>
<td>Overwrites the default email address used to send the job report.</td>
</tr>
<tr>
<td><code>-V</code></td>
<td>All current environment variables should be exported to the batch job.</td>
</tr>
<tr>
<td><code>-v env=value</code></td>
<td>Set the runtime environment variable <code>env</code> to <code>value</code>.</td>
</tr>
<tr>
<td><code>-hold_jid job_list</code></td>
<td>Setup job dependency list. <code>job_list</code> is a comma separated list of job ids and/or job names which must complete before this job can run. See <a href="#">Advanced Batch System Usage</a> for more information.</td>
</tr>
</tbody>
</table>
SCC General limits

Default:
- Login nodes are limited to 15 min. of CPU time
- Default Compute Job: 1 CPU core, 4GB RAM, 12 hours
- Maximum number of processors - 1000

Upper Limits:
- 1 processor job (batch or interactive) 720 hours
- OMP job (16 processors or less) 720 hours
- MPI job (multi-node job) 120 hours
- GPU job 48 hours
- Interactive Graphics job (VirtualGL) 48 hours

These will come up again.
## Scheduler Options - Runtime Directives

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-l h_rt=hh:mm:ss</td>
<td>Hard runtime limit in <em>hh:mm:ss</em> format. The default is 12 hours.</td>
</tr>
</tbody>
</table>

Probably the most used option on SCC.
Scheduler Options - CPU Directives

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-l cpu_arch=ARCH</td>
<td>Select a processor architecture (haswell, broadwell, etc). See Technical Summary for all available choices.</td>
</tr>
<tr>
<td>-l cpu_type=TYPE</td>
<td>Select a processor type (E5-2670, E5-2680, X5570, X5650, X5670, X5675). See Technical Summary for all available choices.</td>
</tr>
</tbody>
</table>
### Scheduler Options - Memory Directives

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-l mem_total =#G</code></td>
<td>Request a node that has at least this amount of memory. Current possible choices include 94G, 125G, 252G 504G.</td>
</tr>
<tr>
<td><code>-l mem_per_core =#G</code></td>
<td>Request a node that has at least these amount of memory per core.</td>
</tr>
</tbody>
</table>

`mem_per_core` is very useful for OMP shared memory jobs.
## Scheduler Options - GPU Directives

### Directives to request SCC resources

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-l gpus=G/C</td>
<td>G is the number of GPUs per node. C is the number of CPUs per node. G/C should be expressed as a decimal number (e.g., 1.5). In the case of a repeating decimal like one-third, truncate it to .333.</td>
</tr>
<tr>
<td>-l gpu_type=GPUMODEL</td>
<td>Current choices for <code>GPUMODEL</code> are M2050, M2070, K40m, P100. (will be specific, not better than)</td>
</tr>
<tr>
<td>-l gpu_memory=#G</td>
<td><code>#G</code> represents the minimum amount of memory required per GPU. The M2070 has 6GB, K40m and several of the P100 have 12GB of memory. Some of the P100 GPUs and the V100 GPUs have 16 GB of memory.</td>
</tr>
<tr>
<td>-l gpu_c=GPUCAPABILITY</td>
<td>GPU compute capability. M2070 NVIDIA cards have a compute capability of 2.0. K40m cards have 3.5 compute capability. The <strong>P100 is 6.0</strong> and the <strong>V100 is 7.0</strong>. Some GPU-enabled software (like the popular Tensorflow machine learning program) have restrictions on the compute capability they support and require 3.5 or higher.</td>
</tr>
</tbody>
</table>
Scheduler Options - Misc Directives

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-l eth_speed=N</td>
<td>Ethernet speed (1 or 10 Gbps).</td>
</tr>
<tr>
<td>-l scratch_free=#G</td>
<td>Request a node that has at least this amount of available disc space in scratch. Note that the amount changes!</td>
</tr>
</tbody>
</table>

Rarely used.
## Scheduler Options - Parallel Directives

### Directives to request SCC resources

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-pe omp N</code></td>
<td>Request multiple slots for Shared Memory applications (OpenMP, pthread). This option can also be used to reserve larger amount of memory for the application. <em>N</em> can vary from 1 to 16.</td>
</tr>
<tr>
<td><code>-pe mpi_#_tasks_per_node N</code></td>
<td>Select multiple nodes for MPI job. Number of tasks can be 4, 8, 12 or 16 and <em>N</em> must be a multiple of this value. See <a href="#">Advanced Batch System Usage</a> for more information.</td>
</tr>
</tbody>
</table>

### Shared Memory (OpenMP, pthread)
- “*N*” = any number 1 - whole node
- “-pe omp 16 -l excl=TRUE” option for whole node

### MPI
- `mpi_4_tasks_per_node N`
- `mpi_8_tasks_per_node N`
- `mpi_12_tasks_per_node N`
- `mpi_16_tasks_per_node N`
### Directives to request SCC resources

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-l h_rt=hh:mm:ss</code></td>
<td>Hard runtime limit in <code>hh:mm:ss</code> format. The default is 12 hours.</td>
</tr>
<tr>
<td><code>-l mem_total=#G</code></td>
<td>Request a node that has at least this amount of memory. Current possible choices include 94G, 125G, 252G, 504G.</td>
</tr>
<tr>
<td><code>-l mem_per_core=#G</code></td>
<td>Request a node that has at least these amount of memory per core.</td>
</tr>
<tr>
<td><code>-l cpu_arch=ARCH</code></td>
<td>Select a processor architecture (sandybridge, nehalem, etc). See <a href="#">Technical Summary</a> for all available choices.</td>
</tr>
<tr>
<td><code>-l cpu_type=TYPE</code></td>
<td>Select a processor type (E5-2670, E5-2680, X5570, X5650, X5670, X5675). See <a href="#">Technical Summary</a> for all available choices.</td>
</tr>
<tr>
<td><code>-l gpus=G/C</code></td>
<td>Requests a node with GPU. <code>G/C</code> specifies the number of GPUs per each CPU requested and should be expressed as a decimal number. See <a href="#">Advanced Batch System Usage</a> for more information.</td>
</tr>
<tr>
<td><code>-l gpu_type=GPUMODEL</code></td>
<td>Current choices for GPUMODEL are M2050, M2070 and K40m. (will be specific, not better than)</td>
</tr>
<tr>
<td><code>-l gpu_memory=#G</code></td>
<td><code>#G</code> represents the minimum amount of memory required per GPU. The M2050 has 3GB and the M2070 has 6GB.</td>
</tr>
<tr>
<td><code>-l gpu_c=GPUCAPABILITY</code></td>
<td>GPU compute capability. On shared nodes GPUs have a compute capability of 2.0. The buy-in nodes with K40m cards have 3.5 compute capability.</td>
</tr>
<tr>
<td><code>-l eth_speed=N</code></td>
<td>Ethernet speed (1 or 10 Gbps).</td>
</tr>
<tr>
<td><code>-l scratch_free=#G</code></td>
<td>Request a node that has at least this amount of available disc space in scratch. Note that the amount changes!</td>
</tr>
<tr>
<td><code>-pe omp N</code></td>
<td>Request multiple slots for Shared Memory applications (OpenMP, pthread). This option can also be used to reserve larger amount of memory for the application. <code>N</code> can vary from 1 to 16.</td>
</tr>
<tr>
<td><code>-pe mpi_#_tasks_per_node N</code></td>
<td>Select multiple nodes for MPI job. Number of tasks can be 4, 8, 12 or 16 and <code>N</code> must be a multiple of this value. See <a href="#">Advanced Batch System Usage</a> for more information.</td>
</tr>
</tbody>
</table>
Monitoring a Job
Monitoring: Running Jobs

- "qstat" - Show the status of Grid Engine jobs and queues

```
[cjahnke@scc1 ~]$ qstat -u cjahnke
job-ID   prior name   user     state submit/start at       queue                 slots ja-task-ID
--------- ----------- -------- -------------- ---------------------- ----------------------
336431    0.10059 phy.24 cjahnke  r  09/03/2016 18:02:32  l@scc-ka4.scc.bu.edu  16
336432    0.10059 phy.25 cjahnke  qw 09/03/2016 18:00:00                        16
```
Monitoring: Running Jobs In-Depth

- “-s {p|r|s|...}” -- Prints only jobs in the specified state

```
[cjahnke@scc1 ~]$ qstat -s r -u cjahnke
```

```
job-ID  prior   name    user     state  submit/start at      queue                 slots ja-task-ID
------------------------------------------------------------------------------------------------
336431  0.10059  phy.24  cjahnke  r      09/03/2016 18:02:32  l@scc-ka4.scc.bu.edu  16
```

- “-r” -- Prints extended information about the resource requirements

```
[cjahnke@scc1 ~]$ qstat -r -u cjahnke
```

```
job-ID  prior   name    user     state  submit/start at      queue                 slots ja-task-ID
------------------------------------------------------------------------------------------------
336431  0.10059  phy.24  cjahnke  r      09/03/2016 18:02:32  l@scc-ka4.scc.bu.edu  16
                                              Full jobname: phy.24.ortho1.pbs
                                              Master Queue: linga@scc-ka4.scc.bu.edu
                                              Requested PE: omp16 16
                                              Granted PE: omp16 16
                                              Hard Resources: h_rt=2588400 (0.000000)
                                                              mem_free=8g (0.000000)
                                              job=1 (default)
                                              Soft Resources: buyin=TRUE
```
[cjahnke@scc4 ~]$ qstat -j 336431

==============================================================================
job_number:                 336431
exec_file:                  job_scripts/336431
submission_time:            Sat Sep  3 18:02:22 2016
owner:                      cjahnke
uid:                        157672
group:                      scv
gid:                        2630
sge_o_home:                 /usr3/bustaff/cjahnke
sge_o_log_name:             cjahnke
sge_o_path:                 /share/apps/6.0/cufflinks/2.2.0/bin:/share/pkg/r/3.1.1/install/bin:/...:/...
sge_o_shell:                /bin/bash
sge_o_workdir:              /projectnb/scv/test
sge_o_host:                 scc4
account:                    sge
cwd:                        /projectnb/scv/test
merge:                      y
hard resource_list:         h_rt=2588400,mem_free=8g
soft resource_list:         buyin=TRUE
mail_options:               be
mail_list:                  cjahnke@bu.edu
notify:                     FALSE
job_name:                   phy.24.ortho1.pbs
stdout_path_list:           NONE:NONE:phy.24.ortho1.pbs.qlog
jobshare:                   0
shell_list:                 NONE:/bin/bash
eenv_list:                  PATH=/share/apps/6.0/cufflinks/2.2.0/bin:/share/pkg/r/3.1.1/install/bin:/...:/.../
script_file:                phy.24.ortho1.pbs
parallel environment:       omp16 range: 16
project:                    scv
usage 1:                    cpu=174:11:55:06, mem=67369810.71167 GBs, io=0.05496, vmem=4.584G, maxvmem=4.584G
scheduling info:            (Collecting of scheduler job information is turned off)
Monitoring: Running Jobs In-Depth

Can look at your processes directly on the compute-node

1. Login to the compute node
   ```
   [scc1 ~]$ ssh scc-ca1
   ```

2. Run top command
   ```
   [scc-ca1 ~]$ top -u <userid>
   ```

3. Exit from the compute node
   ```
   [scc-ca1 ~]$ exit
   ```
Monitoring: Completed Jobs

- **qacct** - query the accounting system
  - Usually, job specific
  - Can summarize information

```bash
cjahnke@scc1 ~]$ qacct -j 9253374
==================================================================
qname        linga
hostname     scc-ka4.scc.bu.edu
group        sibs
owner        cjahnke
project      sibs
department   defaultdepartment
jobname      test
jobnumber    9253374
taskid       undefined
account      sge
priority     0
qsub_time    Wed Jun 29 12:35:21 2019
start_time   Wed Jun 29 12:35:37 2019
end_time     Wed Jun 29 12:35:47 2019
granted_pe   NONE
slots        1
failed       0
exit_status  0
ru_wallclock 10
...
cpu          0.126
mem          0.000
io           0.000
iow          0.000
maxvmem      13.953M
arid         undefined
```
Monitoring: Completed Jobs In-depth

- **User Summary**

  ![User Summary](image)

- **Project Summary**

  ![Project Summary](image)

- **Time/Date**

  ![Time/Date](image)
Monitoring: Accounting Tool

- A “Service Unit” (SU) is a normalized measure of CPU time usage.
  - Used for project management, allocation, and accounting.
  - Some processors are faster than others.
  - There is no monetary charge.
- Use `acctool` to get the information about SU (service units) usage:

```bash
[cjahnke@scc4 ~]$ acctool -b 9/01/16 y
Hostname               Time         Jobs
shared              3190.72          235
linga                896.07          203
jcvf                538.73          122
TOTAL               5866.51          607

Project      SU Balance
scv            874.0952
charges      37538.8022
linga_admin    2398.3860
fhspl        41304.6475
```

```bash
[cjahnke@scc4 ~]$ acctool -p scv -b 9/01/16 y
Hostname               Time         Jobs
shared              2596.14           42
jcvf                337.75           55
TOTAL               3562.13          128

Project      SU Balance
scv            874.0952
```

```bash
[cjahnke@scc1 ~]$ acctool -host shared -b 9/01/16 y
Hostname               Time         Jobs
shared              3190.72          235
```

(acctool)
Monitoring: Email Notifications

The system can send email notifications

- Use "-m" qsub option
  - b = when job begins,
  - a = if job aborts
  - e = when job ends

For example:

```bash
qsub -P project -m bae spline.qsub
```

To: cjahnke@bu.edu
From: nobody@scc.bu.edu
Subject: Job 7883980 (spline) Complete

```
User           = cjahnke
Queue          = p@scc-pi2.scc.bu.edu
Host           = scc-pi2.scc.bu.edu
Start Time     = 08/29/2016 13:18:02
End Time       = 08/29/2016 13:58:59
User Time      = 01:05:07
System Time    = 00:03:24
Wallclock Time = 00:40:57
CPU            = 01:08:31
Max vmem       = 6.692G
Exit Status    = 0
```
Delete Jobs

- “qdel” - Delete a job from the queue

[cjahnke@scc1 ~]$ qdel -j 12345
cjahnke has deleted job 12345

If you have a lot of jobs -- delete them all by user

[cjahnke@scc1 ~]$ qdel -u cjahnke
cjahnke has deleted job 12345
cjahnke has deleted job 12346
cjahnke has deleted job 12347
cjahnke has deleted job 12348
cjahnke has deleted job 12349
Bottlenecks
Bottleneck: Time
SCC General limits

Default:
- Login nodes: 15 min
- Compute Job: 12 hours

Upper Limits:
- 1 Processor job (batch or interactive) 720 hours
- OMP job (16 processors or less) 720 hours
- MPI job (multi-node job) 120 hours
- GPU job 48 hours
- Interactive Graphics job (VirtualGL) 48 hours
Bottleneck: Time

Policies regarding time usage on SCC protect the system (and you).

- Keep high occupancy nodes running
  - Compute time limited to 15 minutes on login nodes.
  - Prevents a single user (or several users combined) from bogging down a system that many other people are using for administrative tasks.

- Prevent runaway processes or endless loops
  - Default runtime of jobs limited to 12 hours.
**Example:** You need to run a compute process for >15 min

```
[cjahnke@scc4 ~]$ ./yourprogram
Killed.
```

**Solution:** Interactive Job

```
[cjahnke@scc4 ~]$ qrsh
[cjahnke@scc-pi2 ~]$ ./yourprogram
```

---

To: cjohnke@bu.edu
From: nobody@scc.bu.edu
Subject: Message from the process reaper on SCC1

The following process, running on SCC1, has been terminated because it exceeded the limits for interactive use. **An interactive process is killed if its total CPU time is greater than 15 minutes and greater than 25% of its lifetime.** Processes which may exceed these limits should be submitted through the batch system.


```
COMMAND STATE   PID  PPID  TIME RATE(%) SIZE RSS START TIME
MATLAB          S    12768 12759 17    100 8427 5562 01/14 12:27:34
```

Please email help@scc.bu.edu for assistance.
**Example:**
You need to run a job for >12 hours.

**Solution:**
Increase the “Hard Runtime Limit” with the “\-l h\_rt=HH:MM:SS” qsub option.

```bash
$ qsub -P proj \-l h\_rt=24:00:00 script.qsub
```

---

To: cjahnke@bu.edu
From: nobody@scc.bu.edu
Subject: Job 9022506 (myJob) Aborted

Job 3828407 (myJob) Aborted

<table>
<thead>
<tr>
<th>Exit Status</th>
<th>= 137</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal</td>
<td>= KILL</td>
</tr>
<tr>
<td>User</td>
<td>= cjahnke</td>
</tr>
<tr>
<td>Queue</td>
<td>= <a href="mailto:b@scc-he1.scc.bu.edu">b@scc-he1.scc.bu.edu</a></td>
</tr>
<tr>
<td>Host</td>
<td>= scc-he1.scc.bu.edu</td>
</tr>
<tr>
<td>Start Time</td>
<td>= 01/06/2018 00:32:16</td>
</tr>
<tr>
<td>End Time</td>
<td>= 01/06/2018 12:32:17</td>
</tr>
<tr>
<td>CPU</td>
<td>= 11:56:32</td>
</tr>
<tr>
<td>Max vmem</td>
<td>= 379.809M</td>
</tr>
</tbody>
</table>

failed assumedly after job because:
job 3828407 died through signal KILL(9)
Bottleneck: Time - Compute Job

Dear Admins,

I submitted a job and it takes longer than I expected. Is it possible to extend the time limit?

-- Advanced User

Unfortunately, not. Once running, the job parameters cannot be modified.
Bottleneck: CPU
Bottleneck: CPU Optimization

Writing code from scratch?  
→ **Optimize it!**

- There are best-practices and techniques for every language.
- There are also some specifics in running the code on the cluster.
- Do this before parallelizing your code!
  - Parallelized bad code is still bad code.
- Discuss with us at help@scc.bu.edu

Are you compiling your code?

Modern CPUs can handle complex instructions, but you need to use non-default compilers.

**Compiler Options and Versions:**
- GCC  
  - 5.5.0, 7.4.0, 8.1.0
- PGI  
  - 18.4, 18.10
- Intel  
  - 2018, 2019
Bottleneck: CPU Optimization (program debug)

Integrated Development Environments (IDE)

- codeblocks
- geany
- Eclipse

Debuggers:

- gdb
- ddd
- TotalView
- OpenSpeedShop
Bottleneck: CPU Parallelization

Parallelization

- **OpenMP**: Single node using *multiple processes*
  - Common with scripts when the user only wants a single job.

- **OpenMP**: Single node *threading a single process*
  - Commonly built into applications.

- **OpenMPI**: Multi-node, many CPU, shared memory processing
  - Very powerful computation, not used much on BUMC.

- **Tasks/Arrays**
  - We will discuss this later
Bottleneck: CPU Parallelization

Single node with multiple processes run simultaneously.

Example: User only wants to manage one job, but wants to run a script on 4 files.

```
#!/bin/bash -l
#$ -pe omp 4
module load python
python script.py file1 &
python script.py file2 &
python script.py file3 &
python script.py file4 &
wait
```

On compute node: 4 processes running simultaneously

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>38681</td>
<td>cjahnke</td>
<td>20</td>
<td>0</td>
<td>12.4g</td>
<td>12g</td>
<td>1708</td>
<td>100.0</td>
<td>9.8</td>
<td>1:06.26</td>
<td>python</td>
</tr>
<tr>
<td>38682</td>
<td>cjahnke</td>
<td>20</td>
<td>0</td>
<td>12.3g</td>
<td>12g</td>
<td>1708</td>
<td>100.0</td>
<td>9.7</td>
<td>1:06.26</td>
<td>python</td>
</tr>
<tr>
<td>38683</td>
<td>cjahnke</td>
<td>20</td>
<td>0</td>
<td>12.2g</td>
<td>12g</td>
<td>1708</td>
<td>100.0</td>
<td>9.6</td>
<td>1:06.27</td>
<td>python</td>
</tr>
<tr>
<td>38684</td>
<td>cjahnke</td>
<td>20</td>
<td>0</td>
<td>11.6g</td>
<td>12g</td>
<td>1708</td>
<td>100.0</td>
<td>8.4</td>
<td>1:06.27</td>
<td>python</td>
</tr>
<tr>
<td>46777</td>
<td>cjahnke</td>
<td>20</td>
<td>0</td>
<td>13404</td>
<td>1528</td>
<td>948</td>
<td>0.3</td>
<td>0.0</td>
<td>0:00.03</td>
<td>top</td>
</tr>
<tr>
<td>46696</td>
<td>cjahnke</td>
<td>20</td>
<td>0</td>
<td>88256</td>
<td>1812</td>
<td>896</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>sshd</td>
</tr>
<tr>
<td>46697</td>
<td>cjahnke</td>
<td>20</td>
<td>0</td>
<td>9680</td>
<td>1820</td>
<td>1360</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>bash</td>
</tr>
</tbody>
</table>
Bottleneck: CPU Parallelization

Background processes with distinct threads are not the best way to do this.

● Could also be done in python itself
  ○ “import subprocess” - spawn new processes
  ○ “from joblib import parallel” - runs functions in parallel
  ○ “from multiprocessing import Pool” -- pools processing

● Also in R
  ○ “library(parallel)” - Basic parallel package
  ○ “library(snowfall)” - Easier cluster computing (based on “snow” library)
  ○ Go to R tutorials next week.

● And most languages

http://scv.bu.edu/examples/SCC/par/python/par.py
Bottleneck: CPU Parallelization

OpenMP: Single Process Threading

- Many applications have this built in -- Look for a “threads” option

```
#!/bin/bash -l
#$ -pe omp 2
module load bowtie2
bowtie2 --threads 2 -x <bt2-idx>
```

On compute node: A single process using 200% CPU

```
PID USER     PR  NI   VIRT   RES  SHR S  %CPU %MEM    TIME+  COMMAND
38681 cjahnke  20   0  12.4g   12g 1708 R 200.0  9.8  1:06.26  bowtie2
46777 cjahnke  20   0  13404  1528  948 R   0.3  0.0  0:00.03  top
46696 cjahnke  20   0  88256  1812  896 S   0.0  0.0  0:00.00  sshd
46697 cjahnke  20   0  9680  1820 1360 S   0.0  0.0  0:00.00  bash
```
Bottleneck: CPU Parallelization

**STAR Aligner Compute Time by Thread Count**

- Compute Time (minutes) vs. Cores/Threads Used
- As the number of cores/threads increases, compute time decreases significantly.

**Bowtie Compute Time by Thread Count**

- Compute Time (minutes) vs. Cores/Threads Used
- Similar trend as STAR Aligner, showing efficiency gain with increased parallelism.
Bottleneck: CPU Parallelization

Some applications parallelize automatically (use all cores on node).

- Examples: Matlab, stata-mp
- This is bad behavior and your job will be killed
  - Instruct them not to (if possible), request appropriate cores, or request the whole node.

Matlab single core:

```
matlab -nodisplay -singleCompThread -r "n=4, rand(n), exit"
```

MATLAB Parallel Computing Toolbox

```
# Request 4 cores:
#$ -pe omp 4
matlab -nodisplay -r "matlabpool open 4, s=0; parfor i=1:n, s=s+i; end, matlabpool close, s, exit"
```
Bottleneck: CPU Parallelization

The $NSLOTS Variable

- When using "-pe omp #" the $NSLOTS variable is set equal the "#".
- This allows you to change the number of threads and not edit the script

```bash
#$ -pe omp 4
bowtie2 --threads $NSLOTS -x <bt2-idx>
```

$NSLOTS=4
Bottleneck: CPU Parallelization

- Copy Simple Examples

  The examples could be found on-line:
  
  - http://scv.bu.edu/examples/SCC/

Copy examples to the current directory:

```
scc1 % cp /project/scv/examples/SCC/depend .
scc1 % cp /project/scv/examples/SCC/many .
scc1 % cp /project/scv/examples/SCC/par .
```
Bottleneck: CPU -- Process Reaper

Use more slots than requested?

- Process Reaper will kill it to preserve other jobs running on that node.

If you have email notifications enabled, you will receive a notice that the job was aborted.

- Note that the job ran for 9 minutes and the CPU ran for 22 min.

You will also receive an explanation email.

To: cjahnke@bu.edu
From: nobody@scc.bu.edu
Subject: Job 9022506 (myJob) Aborted

Job 2885976 (rnaseq.ngs) Aborted
Exit Status = 137
Signal = KILL
User = cjahnke
Queue = b@scc-hc2.scc.bu.edu
Host = scc-hc2.scc.bu.edu
Start Time = 02/01/2016 15:51:07
End Time = 02/01/2016 16:00:01
CPU = 00:22:03
Max vmem = 1.026G
failed assumedly after job because:
job 2885976 died through signal KILL(9)

20 min
10 min
The following batch job, running on SCC-GB11, has been terminated because it was using 5.5 processors but was allocated only 1. Please resubmit the job using an appropriate PE specification.


job 461082.1: owner: cjahnke pe: none type: "Qsh interactive" slots: 1
  sge_gid: 1000791 job_pid: 8447
cputime: 42 min. rate: 548.39% starttime: 09/14 11:57:17

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>STATE</th>
<th>PID</th>
<th>PPID</th>
<th>TIME(min.)</th>
<th>RATE(%)</th>
<th>SIZE</th>
<th>RSS</th>
<th>START TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSF_process.x64</td>
<td>R</td>
<td>8483</td>
<td>8473</td>
<td>4</td>
<td>268</td>
<td>68</td>
<td>13</td>
<td>09/14 11:58:54</td>
</tr>
<tr>
<td>TSF_process.x64</td>
<td>R</td>
<td>8482</td>
<td>8473</td>
<td>4</td>
<td>174</td>
<td>68</td>
<td>13</td>
<td>09/14 11:58:54</td>
</tr>
<tr>
<td>TSF_process.x64</td>
<td>R</td>
<td>8481</td>
<td>8473</td>
<td>4</td>
<td>68</td>
<td>68</td>
<td>13</td>
<td>09/14 11:58:54</td>
</tr>
<tr>
<td>xterm</td>
<td>S</td>
<td>8447</td>
<td>8446</td>
<td>0</td>
<td>0</td>
<td>53</td>
<td>3</td>
<td>09/14 11:57:17</td>
</tr>
</tbody>
</table>

Please email help@scc.bu.edu for assistance.
Bottleneck: Memory
Memory Optimization

First things first, optimize memory usage in your code

- Many languages allow operations on vectors/matrices
- Pre-allocate arrays before accessing or writing
  - Especially within loops.
- Reuse variables when possible
- Delete variables that are not needed.
- Access elements within your code according to the storage pattern in this language
  (FORTRAN, MATLAB, R – columns; C, C++ – rows)

Some of this can be tricky, our applications team is happy to assist. Email help@scc.bu.edu
Memory Limits

The memory on each node in the SCC is shared by all of the jobs on that node.

$$\text{Memory}_{\text{slot}} = \frac{\text{Memory}_{\text{total}}}{N_{\text{cores}}}$$

Using too much memory

- Will slow down your job
- Could cause your job to fail (process is killed automatically)
- Could bring down the node (causing yours and others job to fail)
Memory Availability

SCC is Heterogenous!

<table>
<thead>
<tr>
<th>Shared Nodes</th>
<th>Medical Campus Only</th>
<th>Buy-In Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 cores 24 GB RAM</td>
<td>64 cores 256 GB RAM</td>
<td>16 cores 64 GB RAM</td>
</tr>
<tr>
<td>12 cores 48 GB RAM</td>
<td>64 cores 512 GB RAM</td>
<td>12 cores 96 GB RAM</td>
</tr>
<tr>
<td>20 cores 128 GB RAM</td>
<td>64 cores 512 GB RAM</td>
<td>28 cores 256 GB RAM</td>
</tr>
<tr>
<td>16 cores 128 GB RAM</td>
<td></td>
<td>20 cores 256 GB RAM</td>
</tr>
<tr>
<td>28 cores 256 GB RAM</td>
<td></td>
<td>28 cores 512 GB RAM</td>
</tr>
<tr>
<td>8 cores 96 GB RAM</td>
<td></td>
<td>28 cores 512 GB RAM</td>
</tr>
<tr>
<td>16 cores 256 GB RAM</td>
<td></td>
<td>16 cores 1024 GB RAM</td>
</tr>
<tr>
<td>28 cores 512 GB RAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36 cores 1024 GB RAM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

http://www.bu.edu/tech/support/research/computing-resources/tech-summary/
## Memory Requests

### Requesting Node Resources

<table>
<thead>
<tr>
<th>Job Resource Requirements</th>
<th>qsub Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Partial Node</strong></td>
<td></td>
</tr>
<tr>
<td>(\leq 12 \text{ GB})</td>
<td>Request 4 cores.</td>
</tr>
<tr>
<td>(\leq 24 \text{ GB})</td>
<td>Request 8 cores.</td>
</tr>
<tr>
<td>(\leq 64 \text{ GB})</td>
<td>Request 8 cores on a machine with at least 128 GB of RAM.</td>
</tr>
<tr>
<td><strong>Whole Node</strong></td>
<td></td>
</tr>
<tr>
<td>(\leq 128 \text{ GB})</td>
<td>Request a whole node with 16 cores and at least 128 GB of RAM.</td>
</tr>
<tr>
<td>(\leq 192 \text{ GB})</td>
<td>Request a whole node with 28 cores and at least 192 GB of RAM.</td>
</tr>
<tr>
<td>(\leq 256 \text{ GB})</td>
<td>Request a whole node with 16 cores and at least 256 GB of RAM.</td>
</tr>
<tr>
<td>(\leq 384 \text{ GB})</td>
<td>Request a whole node with 28 cores and at least 384 GB of RAM.</td>
</tr>
<tr>
<td>(\leq 512 \text{ GB})</td>
<td>Request a whole node with 28 cores and at least 512 GB of RAM.</td>
</tr>
<tr>
<td>(\leq 1 \text{ TB})</td>
<td>Request a whole node with 36 cores and at least 1 TB of RAM.</td>
</tr>
</tbody>
</table>
Memory Requests

**Example:** You’ve profiled your job and expect it to use 12 GB of RAM

**Solution:**

1. Request single slot on node with >12GB/slot   
   (e.g. an 16 core/256GB node = 16GB/slot)   
   
   ```bash
   #$ -l mem_per_core=12G
   ```

2. Request multiple slots on a more common node   
   (e.g. 2 slots on a 16 core/128GB node = 16GB/2slots)   
   
   ```bash
   #$ -pe omp 4
   #$ -l mem_per_core=4G
   ```
Bottleneck: Disk
Bottleneck: Disk Space

The most common disk bottleneck is a full directory.

- Depending on the program, the error can be relatively cryptic
- Check both your **home directory** and **project space**.

```
[cjahnke@scc1 ~]$ quota -s
Home Directory Usage and Quota:
Name        GB   quota   limit  grace |  files  quota  limit in_doubt  grace
Cjahnke     7.37  10.0    11.0   none  |     39144  200000  200000      40   none

[cjahnke@scc1 ~]$ pquota rcs
project space
------------------------  -------  -----------------  --------  --------
/project/rcs             50       1638400            21.00    687
/projectnb/rcs           1050    33554432           2.01     1454
```
Bottleneck: Disk Optimization

SCC has a large distributed file system shared to compute nodes on a high speed network, but transactions on disk can be slow (compared to other transactions).

Optimization:

- Reduce transactions
- Use a local disk (/scratch)
- Use RAM instead
Bottleneck: Disk -- Optimize Read/Write

- Reduce the number of I/O to the home directory/project space (if possible)

- Group small I/O transactions into blocks.
  - Don’t: open file, read line, close file, process, open file, write line, close file.
  - Do: open, read whole file, process, write.

- Optimize the seek pattern to reduce the amount of time waiting for disk seeks.
  - Sequential vs. Random

- If possible, read and write numerical data in a binary format.
Bottleneck: Disk -- Use /scratch

If you can’t do that or it’s not enough

- Avoid the network entirely!
- Utilize local /scratch space
- The $TMPDIR environment variable refers to a job specific directory in scratch space. This directory is deleted at the end of the job.
- Scratch files are kept for 30 days, with no guarantees.

```bash
#!/bin/bash -l

# copy data to scratch
cp /project/proj/file $TMPDIR/

cd $TMPDIR

module load bowtie2
bowtie2 -x $TMPDIR/file fastq1 fastq2

# Copy results back to project space
cp $TMPDIR/resultsfile /project/proj/dir
```

https://www.bu.edu/tech/support/research/system-usage/running-jobs/resources-jobs/#scratch
Bottleneck: Network
Bottleneck: Network

Request a node with 10Gbps network connectivity

- Not a common bottleneck
- All new nodes (since 2015) are 10GbE
- Useful if you know that you will be moving a lot of data
  - Great for moving lots of big files within the cluster
  - I'll talk about a better way to do this if you are downloading data from external sources
Job Management and Workflows
Decide where to run jobs

Typically, job requirements dictate the resources needed, but you have options.

**Example**: Your job needs 4 cores, 40 GB RAM and will take 20 hours.

<table>
<thead>
<tr>
<th>Literal needs</th>
<th>Run on 128GB nodes</th>
<th>Make it 2 Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>#$ -pe omp 4</td>
<td>#$ -pe omp 8</td>
<td>#$ -pe omp 4</td>
</tr>
<tr>
<td>#$ -mem_per_core=10</td>
<td>#$ -mem_per_core=5</td>
<td>#$ -mem_per_core=5</td>
</tr>
<tr>
<td>#$ -l h_rt=20:00:00</td>
<td>#$ -l h_rt=20:00:00</td>
<td>#$ -l h_rt=10:00:00</td>
</tr>
</tbody>
</table>

Pro:
- Simple

Con:
- 256GB node less common
- Buy-in nodes limit 12 hr

Pro:
- Use common 128GB node

Con:
- Wastes some CPU
- Still exceeds 12 hour limit

Pro:
- 128 GB = common node
- <12 hr = use shared node

Con:
- Not always possible
How to decide what to request

Information about our resources

- "qconf -sc" - Show the resources that can be requested
- "qhost" - Show the status of hosts, queues and jobs
- "qselect" - Show the nodes that support specified options
- See cheat sheet.
- If your job is complicated → email [help@scc.bu.edu](mailto:help@scc.bu.edu)
Command Line Arguments

Submit a job with extra arguments.

Your `script.qsub` could contain:

```bash
#!/bin/bash -l
echo $1
echo $2
echo $3
Rscript my_R_program.R $4
```

The output would be:

```
1
two
3 word string
```

Very useful for using a generic script on multiple files or parameters -- but wait until you see “tasks”!
Job Dependency

**Pipeline:** Some jobs may be required to run in a specific order

Example: or this application, the job dependency can be controlled using "-hold_jid" option:

```
qsub -N job1 script1
qsub -N job2 -hold_jid job1 script2
qsub -N job3 -hold_jid job2 script3
```

**Post-Processing:** A job might need to wait until a group of jobs have completed.

In this example, “lastJob” won’t start until job1, job2, and job3 have completed.

```
qsub -N job1 script1
qsub -N job2 script2
qsub -N job3 script3
qsub -N lastJob -hold_jid "job*" script4
```
Job Arrays / Tasks

Let’s say you have some “embarrassingly parallel” code

- Simulations - want 1000 runs, using different seed
- Chromosome Analysis - same analysis, different chromosome files
- Large File - divide and conquer / scatter-gather / map-reduce

Array Jobs (qsub option “-t”)

- One “array job” has many related “tasks”.
- Each task runs the same job script, but is has a unique ID to work with.
- Task is placed on the cluster independently (different nodes).
Job Arrays / Task Variables

Submit a 10 task array job, numbered 1 to 10:

```
sccl % qsub -t 1-10 <my_script>
```

Your `<my_script>` could contain:

```
#!/bin/bash -l
Rscript my_R_program.R $SGE_TASK_ID
```

Which would run `my_R_program.R` 10 times, each one using a number from 1 to 10.

Batch environment variables:

- SGE_TASK_FIRST=1
- SGE_TASK_STEPSIZE=1
- SGE_TASK_LAST=3
- SGE_TASK_ID=2
Bonus Material

VNC, Data transfer, Buy-In, Hadoop
VNC - Remote Desktop

VNC (Virtual Network Computing) is a graphical desktop sharing system that allows users to remotely work on another computer. It works by transmitting the keyboard and mouse events from your local machine to the remote machine.

- Graphics compression
- Persistent Sessions

Allows responsive graphics interaction
Disconnect and reconnect later

http://www.bu.edu/tech/support/research/system-usage/getting-started/remote-desktop-vnc/
From your local system, forward the port

```
[local ~]$ ssh user@scc1.bu.edu
[cjahnke@scc4 ~]$ vncpasswd
Password:
Verify:
```

From your local system, open the VNC session

```
[local ~]$ ssh user@scc4.bu.edu -L 7000:localhost:5901
```

Password from “vncpasswd”

```
[cjahnke@scc4 ~]$ vncstart
=================================================================
*** Your VNC server is now running! ***
  VNC desktop number: 1
  VNC Port number: 5901
=================================================================
To connect via VNC client:
1. On your local machine execute the following:
   
   ssh cjahnke@scc4.bu.edu -L XXXX:localhost:5901

   where XXXX - some number greater than 1023.
   You will be prompted to enter your SCC password.

2. Start your local VNC Client application and
   enter the following address in VNC server field:
   
   localhost:XXXX

   where XXXX is the number you selected in step 1.
   When prompted, use your VNC password.

To terminate VNC server, execute command (in your scc1
terminal window):

    vncserver -kill :1

=================================================================
[cjahnke@scc4 ~]$
[cjhke@sc-c-lite ~]$ export PATH=/ad/eng/bin/:$PATH
[cjhke@sc-c-lite ~]$ comsol51-research
Transfer Node: scc-globus.bu.edu

- High-bandwidth node for data transfer to and from the SCC
- Has 10 Gbps Ethernet connection to internet
- Designed for Globus Connect Service
  - Can set up endpoints on project spaces.
- Supports other protocols
  - Aspera, Globus, GridFTP, AWS, you name it

https://www.bu.edu/tech/support/research/system-usage/getting-started/globus-online/
Service Models – Shared and Buy-In

**Buy-In:** purchased by individual faculty or research groups through the Buy-In program with priority access for the purchaser.

35%

**Shared:** Centrally funded by BU and university-wide grants. Resources are free to the entire BU Research Computing community.

65%

- **Shared**
- **Buy-In**
SCC Compute Nodes

Buy-in nodes:

- All buy-in nodes have a hard limit of 12 hours for non-member jobs. The time limit for group member jobs is set by the PI of the group.

- About 60% of all nodes are buy-in nodes. Setting time limit for a job larger than 12 hours automatically excludes all buy-in nodes from the available resources;

- All nodes in a buy-in queue do not accept new non-member jobs if a project member submitted a job or running a job anywhere on the cluster.
Hadoop

- Pilot Cluster
- Must request access
- Limited support
- Bioinformatics software soon

If you know what Hadoop is, you might be a good test user.
Support Links and Email

- RCS Website:  http://rcs.bu.edu
- RCS Software:  http://rcs.bu.edu/software/
- RCS Examples:  http://rcs.bu.edu/examples/
- RCS Tutorials:  http://rcs.bu.edu/tutorials/

Please contact us at  help@scc.bu.edu  if you have any problem or question
SCC OnDemand
SCC OnDemand

Based on an NSF-funded open source project “Open OnDemand”, developed by the Ohio Supercomputing Center (OSC) and fully customized for the BU Shared Computing Cluster (SCC). Provides cluster access entirely through a web browser.

Provides:

- Easy file management
- Command-line shell access
- Graphical desktop environments and desktop applications
- Web-server based applications (e.g. RStudio, Jupyter, Tensorboard)
**Existing SCC Account**

1. Open a web browser
2. Navigate to [http://scc-ondemand.bu.edu](http://scc-ondemand.bu.edu)
3. Log in with your BU Kerberos Credentials

**Temporary Tutorial Account**

1. Open a web browser
3. Log in with Tutorial Account
http://scc-ondemand.bu.edu
Questions?

Research Computing Services Website
http://rcs.bu.edu

RCS Tutorial Evaluation
http://rcs.bu.edu/eval