

O2 for Orchestra Users

HMS Research Computing

Fall 2017



Welcome to O2!

- HMS Research Computing's second High-Performance Compute cluster to enhance the compute capacity available to HMS Researchers
- Homogeneous environment of newer, faster cores with high memory allocation to facilitate multi-core and parallelized workflows
- SLURM scheduler to efficiently dispatch jobs



O2 Tech Specs



- 5000 homogeneous cores
- Dell Xeon E5-2683 processors: 2.1GHz Broadwell chips
- 16 x 2 (32) cores per node
- 256GB RAM per node (8GB/core)
- Login/load balancer 5 VM (8 cores/16GB memory)
- InfiniBand connectivity between nodes available
- CentOS 7

Coming Soon to O2...

- Orchestra fastest compute nodes “ottavinos” re-provisioned for O2
- Orchestra High-Memory nodes “saxophone” re-provisioned for O2
- Orchestra GPU K80/M40 resources re-provisioned for O2



Ottavino
2000 cores
28 x 224GB



Saxophone
24 cores
12 X 1TB



GPU
24 cores

Storage on O2

- Mounts all Orchestra storage
- Isilons: /home, /n/groups, /n/data1, /n/data2
- research.files: transfer partition, /n/files
- Lustre: /n/scratch2
- GPFS: /n/no_backup



Storage Mount Points

- /home, /n/data1, /n/data2 same
- /groups = /n/groups
- /files= /n/files on transfer partition (ask us)
- /n/no_backup
- /n/scratch2



slurm

- slurm is an open-source scheduler
- Rich, user-contributed community of questions and answers
- Simpler partition structure and management of fairshare (partition, user, group)
- Optimization for MPI and GPU
- Better resource allocation
- Advanced job accounting



Login/transfer

- Login Nodes (5 Virtual Machines)
\$ ssh eCommons@o2.hms.harvard.edu
- With X11:
\$ ssh -XY eCommons@o2.hms.harvard.edu
- Transfer Server (sFTP/port 22, 10 nodes):
transfer.rc.hms.harvard.edu

.bashrc

- Orchestra and O2 both source the same ~/.bashrc
- Perl local::lib from Orchestra will not work on O2
- Module files are not synonymous between systems, and module loads will break
- Modify .bashrc as follows:

```
if [[ ! -z $SLURM_CONF ]]
then
    # put O2 specific commands here
else
    # put Orchestra specific commands here
fi
```

Interactive Job:

- Interactive session: up to 20 cores, 12 hours
\$ srun -p interactive -t 0-12:00 --pty /bin/bash
- Can add cores (up to 20)
-n
- Can add total memory (default 1G, max 250G):
--mem 8G

Jobs: bsub -> sbatch

- All in one line: `--wrap="command here"` #not recommended
- `sbatch -p partition -t 0-1:00 --wrap="sh script.sh"`
`$ sbatch -p short -t 0-1:00 --wrap="cp file.txt .."`
- Complete shell script #recommended
- `$ sbatch completeSlurmJob.run`

```
#!/bin/bash
```

```
#SBATCH -p short
```

```
#SBATCH -t 0-1:00
```

```
cp file.txt ..
```

Partitions (queues): -q -> -p

Partition	Priority	Max Runtime	Max Cores	Limits
short	10	12 hours	20	
medium	8	5 days	20	
long	6	30 days	20	
interactive	14	12 hours	20	2 job limit
priority	14	30 days	20	2 job limit
mpi	12	5 days	640	20 core min
transfer		30 days	4	

Walltime: -W -> -t

- -t days-hours:minutes
- -t hours:minutes:seconds
- Need to specify how long you estimate your job will run for
- Aim for 125% over
- Subject to maximum per partition
- Excessive runlimits (like partition max) take longer to dispatch, and affect fairshare

CPU: -n -> -n

- -n X to designate CPU: max 20
- -N X to constrain all cores to X nodes
- CPU time: wall time (-t) * (-n) CPUs used
- Unable to use CPU not requested (no overefficient jobs): cgroups constraint

Memory: -Rusage -> --mem

- Only 1G is allocated by default
- --mem XG #total memory over all cores
- --mem-per-cpu XG #total memory per CPU requested, use for MPI
- No unit request (G) defaults to Megabytes
 - 8G ~= 8000

Job Construction

`#!/bin/bash`

`#SBATCH -p #partition`

`#SBATCH -t 0-01:00 #time days-hr:min`

`#SBATCH -n X #number of cores`

`#SBATCH -N 1 #confine cores to 1 node, default`

`#SBATCH --mem=XG #memory per job (all cores), in GB`

`#SBATCH -o %j.out #out file`

`#SBATCH -e %j.err #error file`

`#SBATCH --mail-type=BEGIN/END/FAIL/ALL`

`#SBATCH --mail-user=mfk8@med.harvard.edu`

Output/Error Files

- Can add jobid to filename with %j
- Sample:
 - e %j.err
 - o %j.out
- SLURM by default creates this outfile:
slurm-jobid.out
- Additional Flags
 - %a job array id
 - %A master array job id
 - %N node name
 - %u user id



Mail

- Mail is not auto-generated upon completion/failure
- #SBATCH --mail-type= NONE, BEGIN, END, FAIL, REQUEUE, ALL
- #SBATCH [--mail-user=mfk8@med.harvard.edu](#)
- Not recommended, not a verbose output like LSF
- Use sacct instead

Job Master Allocations/Job Steps

- “sbatch” or “salloc” will create a master allocation of shared resources
- To run job steps or multiple commands in parallel with resources from the master allocation, use srun

```
#!/bin/bash
#SBATCH -n 4
#SBATCH --mem 32000
srun -n 1 --mem=8000 COMMAND1 &
srun -n 1 --mem=8000 COMMAND2 &
srun -n 1 --mem=4000 COMMAND3 &
srun -n 1 --mem 12000 COMMAND4 &
wait #necessary
```

starts 4 independent, single-threaded parallel tasks with memory constraints

```
#!/bin/bash
#SBATCH -c 4
#SBATCH --mem 32000

srun -c 2 COMMAND1 &
srun -c 2 COMMAND2

wait #necessary
```

starts 2 independent, multicores tasks

Job Arrays

- `sbatch --array=1-30 submit.sh`
- `#SBATCH --array=1-30`
- `slurm` creates:
- `$SLURM_JOB_ID` #jobid of each job in array %j
- `$SLURM_ARRAY_JOB_ID` #jobid of entire array %A
- `${SLURM_ARRAY_TASK_ID}` #index of job %a

Example:

Files named: File1.txt File2.txt File3.txt ...File30.txt

Execution is:

`myscript.sh File${SLURM_ARRAY_TASKID}.txt`

Job Dependencies

- `sbatch --dependency=`
- `after:jobid #(asynchronous)`
- `afterany:jobid #after exit or done`
- `afterok:jobid #success, exit code 0`
- `afternotok:jobid #failure`
- `singleton #after jobs with same name have terminated`
- `--kill-on-invalid-dep=<yes|no> #kill on unmet dependency (on by default)`

Command Line Arguments

- slurm scripts can take command line arguments after them, unlike bsub
- Reference as \$1, \$2 etc
- sbatch submit.run 25 output.txt

```
#!/bin/bash
```

```
#SBATCH -p short
```

```
#SBATCH -t 0-1:00
```

```
python myscript.py $1 $2
```

X11 on O2

- To visualize or initiate plot devices, an X11 device must be active
- Mac: Xquartz installed and running
- Windows: Xming installed and running
- Login: `ssh -XY`
- SSH keys:

Host `login.rc.hms.harvard.edu`

ForwardX11 `yes`

ForwardX11Trusted `yes`

- To sbatch jobs add: `--x11=batch`
- To interactives, srun add: `--x11`



Fairshare -> Priority

- Dynamically assigned
- Factors contributing:
- Age, Fairshare, Partition, QOS, Nice
- Fairshare: 0-1 scale



Job Dispatch Order in O2

Sum of

- AGE = how long a job has been pending
- FAIRSHARE = fairshare of the user
- PARTITION = priority of the partition used
- QOS = quality of services, normally not used
- NICE = additional custom priority, normally not used

Notes:

- Jobs submitted to the partitions *interactive* and *priority* are evaluated before jobs in any other partitions, independently of the priority values.
- A lower priority job can be dispatched before an higher priority job if it does not impact the expect start time of the higher priority job (*job backfilling*)

Job Monitoring

- `$ queue -u eCommons -t RUNNING/ PENDING`
- `$ queue -u eCommons -p partition`
- `$ queue -u eCommons --start`
- Detailed job info:
`$ scontrol show jobid <jobid>`
- Completed job statistics:
`$ sacct -j <jobid> --format=JobID,JobName,MaxRSS,Elapsed`

Job information: sacct

- Advanced job accounting options

- `$ saact -j jobid`

Options:

`--name`

`-r/--partition`

`-s/--state`

`-o/--format`

- `$ sacct -u eCommons`

- `$ sacct --helpformat #get available accounting features`

sacct: basic information

- `sacct --helpformat` displays fields
- RC Recommends:

```
sacct -u eCommons --  
format=jobid,Partition,state%22,MaxRSS,MaxVMSize,ReqTRES%20,Start%20  
,End%20,Timelimit%14,exitcode --units=G
```

- Gives jobid, partition, end state, Max Memory, Max Virtual Memory, Requested Cores/Mem, Start, Stop, Timelimit, Exit Code in GB format

saact: state

- BF BOOT_FAIL
- CA CANCELLED
- CD COMPLETED
- CF CONFIGURING
- CG COMPLETING
- DL DEADLINE
- F FAILED
- NF NODE_FAIL
- PD PENDING
- PR PREEMPTED
- R RUNNING
- RS RESIZING
- S SUSPENDED
- TO TIMEOUT

Cancelling/Pausing Jobs

- `$ scancel <jobid>`
- `$ scancel -t PENDING`
- `$ scancel --name JOBNAME`
- `$ scancel jobid_[indices] #array indices`
- `$ scontrol hold <jobid> #pause`
- `$ scontrol release <jobid> #resume`
- `$ scontrol requeue <jobid> #cancel and rerun`

MPI on O2

- Message Passing Interface
- Distribute work over multiple nodes, allowing for the utilization of more cores
- openMPI-2.0.1 compiled against GCC 6.2.0
- MATLAB, Python, R, Perl, Java, C++, Fortran implementations
- Needs wrapper function “mpirun” to dispatch to compute nodes with SLURM
- Run in “mpi” partition –p mpi after being added to partition
- Core cap: 640 processors, 5 day runtime



LMOD: Software Modules

- LMOD system adds directory paths of software into \$PATH variable, and resolves software dependencies and conflicts
- Most software compiled against gcc-6.2.0: load first
- \$ module load gcc/6.2.0 #note, no directories anymore
- \$ module spider #software currently available
- \$ module load software/version #load software, no directories
- \$ module unload software/version #unload
- \$ module purge #dump all modules
- \$ module help <software> #displays run info

Programming Languages

- Python: load module (2.7.12, conda2, 3.6.0)
 - use virtualenv to maintain packages (pip/easy install)
- R: load module (3.2.5, 3.3.3, 3.4.1)
 - Setup O2-specific personal R library, .Renviron (install.packages/biocLite)
- Perl: load module (5.24.0)
 - Setup O2-specific local::lib (cpan/cpanm) in .bashrc
- MATLAB: load module (2016b, 2017a)
 - Setup cluster profile, only use on 1 HPC at a time

For more direction

- <http://hmsrc.me/O2docs>
- <http://rc.hms.harvard.edu>
- RC Office Hours: Wed 1-3p Gordon Hall 500
- rchelp@hms.harvard.edu

