Load Sharing Facility (LSF)

Minerva Scientific Computing Environment

https://labs.icahn.mssm.edu/minervalab

Patricia Kovatch
Eugene Fluder, PhD
Hyung Min Cho, PhD
Lili Gai, PhD
Dansha Jiang, PhD
Wayne Westerhold, MS

23 Sep 2020

Mount Sinai

Outline

- LSF introduction and basic/helpful LSF commands
- Dependent job
- Self-scheduler
- Parallel jobs: job arrays, parallel processing and GPUs
- Job checkpoint/restart
- Tips for efficient usage of the queuing system

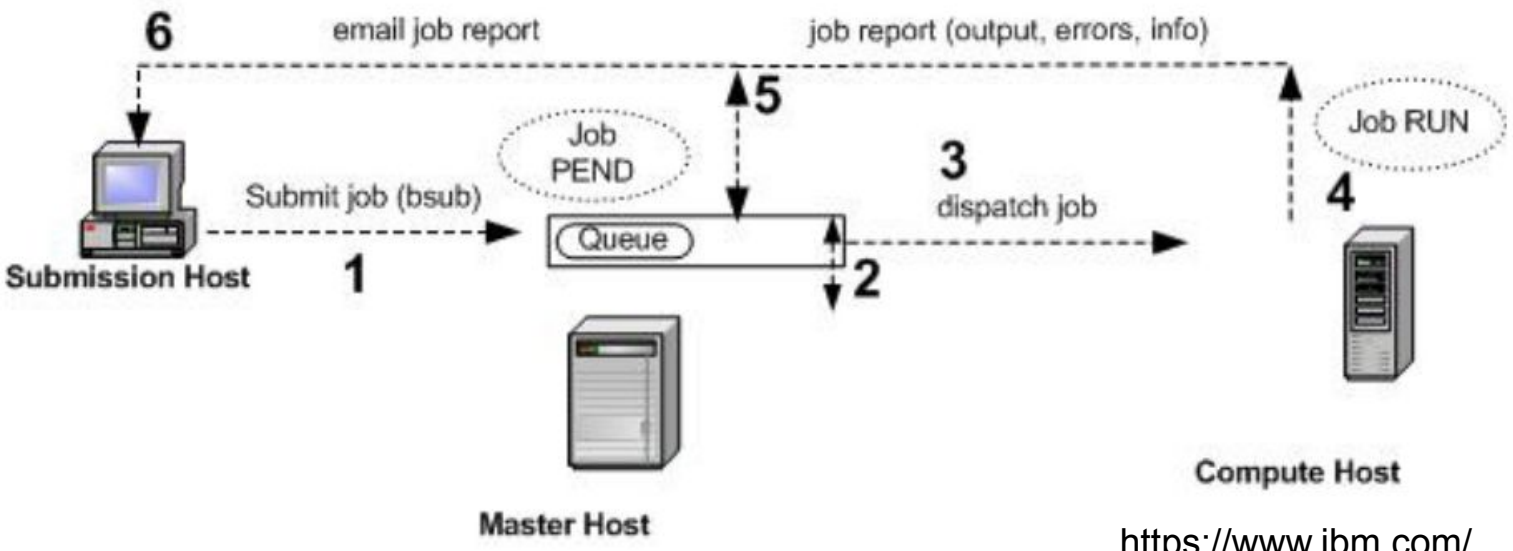
Distributed Resource Management System (DRMS)

- Used to optimize utilization of resources and maximize throughput for high-performance cluster computing systems

- Controls
  - CPU cycles;
  - Memory;
  - Specialty resources

- Widely deployed DRMSs
  - **Platform Load Sharing Facility (LSF)**
  - Portable Batch Systems (PBS)
  - **Simple Linux Utility for Resource Management (Slurm)**
  - others such as IBM Load Leveler and Condor

LSF Job Lifecycle



https://www.ibm.com/

1. submit a job
2. schedule the job
3. dispatch the job
4. run the job
5. return output/record stats
6. send Email to client (disabled on Minerva)

LSF: batch job submission: bsub

Batch jobs submission:

 # simple standard job submission

bsub [options] command

$ bsub -P acc_hpcstaff -q premium-n 1 -W 00:10 echo "Hello World"

 # simple LSF script submission

bsub [options] < *pathToScript*

$ bsub -q premium-n 1 -W 00:10 < helloWorld.lsf

where helloWorld.lsf is:

```
#bsub -P acc_hpcstaff
#bsub -q express
echo "salve mundi"
```

Execution environment:

Shell is your current shell (can change with -L)

Working directory on execution host is same as on submission host

Environment variables are copied over (aliases are not)

...

LSF: batch job submission: bsub

Major options:

- -P accountName - Of the form: acc_projectName

To see the list of accessible project accounts:

```
$ mybalance
User_ID      Project_name      BODE
-------      ---------------   -------
choh07       acc_hpcstaff      Yes
choh07       acc_DGXTrial      No
```

- -q queuename - submission queue
- -W wallClockTime - in form of HH:MM
- -n ncpu - number of cpu's requested (default: 1)
- -R rusage[mem=#MB] - amount of real memory per "-n" in MB
  - max memory per node:160GiB (compute), 326GB (GPU), 1.4TiB (himem)
- -R span[#-n's per physical node]
  - span[ptile=4] - 4 cores per node/host
  - span[hosts=1] - all cores on same node/host
- -R himem - Request high memory node

LSF: bsub major options

- -o Name of output file (concatenated)
- -oo Name of output file (overwrite)
- -e Name of error file (concatenated)
- -eo Name of error file (overwrite)

NOTE: Default output is mailed to the user BUT since we have disabled mail response, it goes into the bit bucket.

If -o(o) is specified but not -e, error is appended to output file

bsub - submit a job to LSF (interactive and batch)

Interactive jobs:

- Set up an interactive environment on compute nodes with **internet access**
- Useful for testing and debugging jobs
- **Interactive GPU** is available for job testing

bsub -XF -P acc_hpcstaff -q interactive -n 1 -W 2:00 -R rusage[mem=3000] -Is /bin/bash

- -q : to specify the queue-name from where to get the nodes
- -Is: Interactive terminal/shell
- -n : to specify the total number of compute cores (job slot) needed
- -R : Resource request specifying in a compute node
- -XF: X11 forwarding
- /bin/bash : the shell to use

```
gail01@li03c03: ~ $ bsub -XF -P acc_hpcstaff -q interactive -n 1 -W 2:00 -R rusage[mem=3000]
-Is /bin/bash
Job <2916837> is submitted to queue <interactive>.
<<ssh X11 forwarding job>>
<<Waiting for dispatch ...>>
<<Starting on lc02a29>>
```

bjobs - status of jobs

- Check your own jobs: **$bjobs**

```
gail01@li03c03: ~ $ bjobs
     JOBID      USER    JOB_NAME   STAT     QUEUE  FROM_HOST   EXEC_HOST   SUBMIT_TIME    START_TIME
TIME_LEFT
    2937044    gail01   myfirstjob  PEND    premium   li03c03     -       Sep 10 14:38      -         -
```

- Check all jobs: **$bjobs -u all**

```
JOBID      USER    JOB_NAME   STAT     QUEUE  FROM_HOST   EXEC_HOST   SUBMIT_TIME    START_TIME
TIME_LEFT
    2845103   beckmn01 *>junkK.432    RUN    premium    regen2    lc02e24  Sep  9 21:19  Sep 10 14:25    23:57 L
    2845113   beckmn01 *>junkK.442    RUN    premium    regen2    lc02e24  Sep  9 21:19  Sep 10 14:26    23:58 L
    2845088   beckmn01 *>junkK.417    RUN    premium    regen2    lc04a10  Sep  9 21:18  Sep 10 14:23    23:55 L
    2845089   beckmn01 *>junkK.418    RUN    premium    regen2    lc04a10  Sep  9 21:18  Sep 10 14:23    23:55 L
    2845090   beckmn01 *>junkK.419    RUN    premium    regen2    lc04a10  Sep  9 21:18  Sep 10 14:23    23:55 L
    2845091   beckmn01 *>junkK.420    RUN    premium    regen2    lc04a10  Sep  9 21:18  Sep 10 14:23    23:55 L
    2845092   beckmn01 *>junkK.421    RUN    premium    regen2    lc04a10  Sep  9 21:18  Sep 10 14:23    23:55 L
    2845093   beckmn01 *>junkK.422    RUN    premium    regen2    lc04a10  Sep  9 21:18  Sep 10 14:23    23:55 L
 ……...
```

- Long format with option -l

LSF Useful Commands

bhosts: Displays hosts and their static and dynamic resources

- List all the compute nodes on Minerva

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
filizm02-3	ok	-	24	1	1	0	0	0
lc01a05	closed	-	48	48	48	0	0	0
lc01a07	closed	-	48	48	16	0	0	32
lc04a19	unavail	-	48	0	0	0	0	0
lg03a01	ok	-	32	0	0	0	0	0
lg03a02	ok	-	32	17	17	0	0	0
lh03c03	closed	-	48	48	48	0	0	0

bhosts: himem, gpu, bode, nonbode (major nodes), interactive

```
gail01@li03c03: ~ $ bhosts himem
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
lh03c01            closed          -     48     48     48      0      0      0
lh03c02            closed          -     48     48     29      0      0     19
lh03c03            closed          -     48     48     26      0      0     22
lh03c04            closed          -     48     48     48      0      0      0
gail01@li03c03: ~ $ bhosts gpu
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
lg03a02            ok              -     32      0      0      0      0      0
lg03a03            closed          -     32     32     32      0      0      0
lg03a04            ok              -     32      1      1      0      0      0
lg03a05            ok              -     32      0      0      0      0      0
lg03a06            ok              -     32      0      0      0      0      0
lg03a07            closed          -     32     32     32      0      0      0
lg03a08            ok              -     32      0      0      0      0      0
lg03a09            ok              -     32     12     12      0      0      0
lg03a10            ok              -     32      0      0      0      0      0
lg03a11            ok              -     32      0      0      0      0      0
lg03a12            unavail         -     32      0      0      0      0      0
gail01@li03c03: ~ $ bhosts bode |head
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
lc01g17            ok              -     48     37     37      0      0      0
lc01g18            closed          -     48     48     48      0      0      0
lc01g19            ok              -     48     37     37      0      0      0
lc01g20            ok              -     48     37     37      0      0      0
lc01g21            ok              -     48     37     37      0      0      0
lc01g22            ok              -     48     17     17      0      0      0
lc01g23            ok              -     48     17     17      0      0      0
```

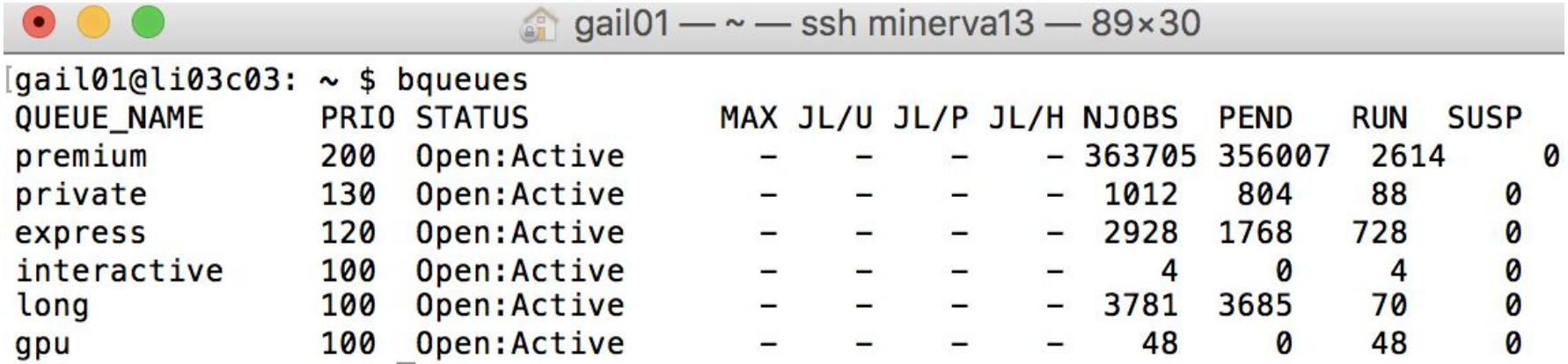
bhosts: himem, gpu, bode, nonbode (major nodes), interactive

```
gail01@li03c03: ~ $ bhosts nonbode | head
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
lc01a05            closed          -     48     48     16      0      0     32
lc01a06            closed          -     48     48     18      0      0     30
lc01a07            closed          -     48     48     16      0      0     32
lc01a08            closed          -     48     48     16      0      0     32
lc01a09            closed          -     48     48     30      0      0     18
lc01a10            closed          -     48     48     12      0      0     36
lc01a11            closed          -     48     48     12      0      0     36
lc01a12            closed          -     48     48     14      0      0     34
lc01a13            closed          -     48     45     13      0      0     32
```

```
gail01@li03c03: ~ $ bhosts interactive
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
lc02a27            ok              -     48      1      1      0      0      0
lc02a28            ok              -     48      0      0      0      0      0
lc02a29            ok              -     48      9      9      0      0      0
lc02a30            ok              -     48     10     10      0      0      0
lg03a01            ok              -     32      0      0      0      0      0
gail01@li03c03: ~ $
```

nonbode and himem are usually quite busy, while bode and interactive are usually open to jobs in minutes; Availability of gpu queue varies from time to time

bqueues: displays information about all the available queues



bqueues -l interactive

QUEUE: interactive
-- For interactive jobs
PARAMETERS/STATISTICS
PRIO NICE STATUS MAX JL/U JL/P JL/H NJOBS PEND RUN SSUSP USUSP RSV
100 0 Open:Active - - - - 4 0 4 0 0 0
Interval for a host to accept two jobs is 0 seconds
DEFAULT LIMITS:
RUNLIMIT
120.0 min
MAXIMUM LIMITS:
RUNLIMIT
720.0 min
…….
USERS: all
HOSTS: interactive/

LSF: Queue structure (bqueues)

Queue structure in Minerva		
Queue	**Wall time limit**	**available resources**
interactive (Dedicated to interactive jobs)	12 hours	4 nodes+1 GPU node
premium	6 days	270 nodes + 4 himem nodes
express	12 hours	270 nodes+ 4 dedicated nodes (may change)
long	2 weeks	4 dedicated (192 cores)
gpu	6 days	44 V100
private	unlimited	private nodes

***default memory : 3000MB / per core**

bmod - modify submission options of pending jobs

bmod takes similar options to bsub

- bmod -R rusage[mem=20000] <jobID>
  - -R replaces **ALL** R fields not just the one you specify
- bmod -q express <jobID>

```
gail01@li03c03: ~ $ bmod -q express 2937044
Parameters of job <2937044> are being changed
```

bpeek - display output of the job produced so far

bpeek <jobID>

```
gail01@li03c03: ~ $ bpeek 2937044
<< output from stdout >>
“Hello Chimera”

<< output from stderr >>
```

bkill - kill jobs in the queue

Lots of ways to get away with murder

bkill <job ID>

Kill by job id
bkill 765814

Kill by job name
bkill -J myjob_1

Kill a bunch of jobs
bkill -J myjob_*

Kill all your jobs
bkill 0

bhist - historical information

```
gail01@li03c03: ~ $ bhist -n 1 -l 2937044

Job <2937044>, Job Name <myfirstjob>, User <gail01>, Project <acc_hpcstaff>, Ap
                     plication <default>, Command <#!/bin/bash;#BSUB -J myfirst
                     job;#BSUB -P acc_hpcstaff  ;#BSUB -q premium;#BSUB -n 1;#B
                     SUB -W  6:00 ;#BSUB -R rusage[mem=4000];#BSUB -o %J.stdout
                     ;#BSUB -eo %J.stderr;#BSUB -L /bin/bash ; module load gcc
                     ;which gcc;echo "Hello Chimera">
Tue Sep 10 14:38:25: Submitted from host <li03c03>, to Queue <premium>, CWD <$H
                     OME>, Output File <%J.stdout>, Error File (overwrite) <%J.
                     stderr>, Re-runnable, Requested Resources <rusage[mem=4000
                     ]>, Login Shell </bin/bash>;

 RUNLIMIT
 360.0 min of li03c03

 MEMLIMIT
    3.9 G
Tue Sep 10 14:38:40: Parameters of Job are changed:
                         Job queue changes to : express;
Tue Sep 10 14:39:36: Dispatched 1 Task(s) on Host(s) <lc02a13>, Allocated 1 Slo
                     t(s) on Host(s) <lc02a13>, Effective RES_REQ <select[((hea
                     lthy=1)) && (type == local)] order[!-slots:-maxslots] rusa
                     ge[mem=4000.00] same[model] affinity[core(1)*1] >;
Tue Sep 10 14:39:37: Starting (Pid 399431);
Tue Sep 10 14:39:39: Running with execution home </hpc/users/gail01>, Execution
                      CWD </hpc/users/gail01>, Execution Pid <399431>;
Tue Sep 10 14:39:41: Done successfully. The CPU time used is 1.5 seconds;
Tue Sep 10 14:39:41: Post job process done successfully;

MEMORY USAGE:
MAX MEM: 9 Mbytes;  AVG MEM: 2 Mbytes

Summary of time in seconds spent in various states by  Tue Sep 10 14:39:41
  PEND     PSUSP    RUN      USUSP    SSUSP    UNKWN    TOTAL
  71       0        5        0        0        0        76
```

Common errors of batch jobs

1. Valid allocation account needed in the submission script

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
   Project acc_project is not valid for user gail01
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Request aborted by esub. Job not submitted.
```

- $mybalance (note BODE eligible)

```
gail01@li03c03: ~ $ mybalance
User_ID       Project_name       BODE
-------       ---------------    ------------
gail01        acc_hpcstaff          Yes
gail01        acc_DGXTrial           No
```

2. Reach memory limit

bhist -n 10 -l 107992756

```
Fri Jul 27 11:07:33: Completed <exit>; TERM_MEMLIMIT: job killed after
reaching LSF memory usage limit;
```

- memory based on one core, with 3000MB as default
- multithreaded applications need to be on the same node, such as STAR, BWA

Wrapper script: LSFqueue module

- We have installed a wrapper script authored by Harm van Bakel, which will make it easier to interact with the LSF job scheduler on Minerva, per user request

  To load them up, $ml LSFqueue

  To get more info on the module, $module help LSFqueue; And a detailed readme file at

  /hpc/packages/minerva-centos7/LSFqueue/1.0/README.txt

Dependent Job

Any job can be dependent on other LSF jobs.
Syntax
bsub -w 'dependency_expression'
usually based on the job states of preceding jobs.

bsub -J myJ < myjob.lsf

bsub -w 'done(myJ)' < dependent.lsf

Self-scheduler

- Submit large numbers of independent serial jobs as a single batch
  - It is mandatory for short batch jobs less than ca. 10 minutes
  - These jobs put heavy load on the LSF server and will be killed

```
#!/bin/bash
#BSUB -q express
#BSUB -W 00:20
#BSUB -n 12
#BSUB -J selfsched
#BSUB -o test01
module load selfsched                    # load the selfsched module
mpirun -np 12 selfsched < test.inp       # 12 cores, with one master process
```

$PrepINP < templ.txt > test.inp (InputForSelfScheduler)

$cat templ.txt

```
1 10000 2  F        ← start,  end,  stride,  fixed field length?
/my/bin/path/Exec_# < my_input_parameters_# > output_#.log
```

$cat test.inp (a series of job command)

```
/my/bin/path/executable < my_input_parameters_1 > output_1.log
/my/bin/path/executable < my_input_parameters_3 > output_3.log
.
/my/bin/path/executable < my_input_parameters_9999 > output_9999.log
```

Job submission script example: selfsched.lsf

```
#!/bin/bash
#BSUB -J myMPIjob                    # Job name
#BSUB -P acc_bsr3101                 # allocation account
#BSUB -q express                     # queue
#BSUB -n 64                          # number of compute cores
#BSUB -R span[ptile=4]               # 4 cores per node
#BSUB -R rusage[mem=4000]            # 256 GB of memory (4 GB per core)
#BSUB -W 00:20                       # walltime (30 min.)
#BSUB -o %J.stdout                   # output log (%J : JobID)
#BSUB -eo %J.stderr                  # error log
#BSUB -L /bin/bash                   # Initialize the execution environment

echo "Job ID                 : $LSB_JOBID"
echo "Job Execution Host     : $LSB_HOSTS"
echo "Job Sub. Directory     : $LS_SUBCWD"

module load python
module load selfsched
mpirun -np 64 selfsched  <  JobMixPrep.inp  > JonMixPrep.out
```

Parallel Job

- **Array job**: Parallel analysis for multiple instances of the same program
  - Execute on multiple data files simultaneously
  - Each instance running independently

- **Distributed memory program**: Message passing between processes (e.g. MPI)
  - Processes execute across multiple CPU cores or nodes

- **Shared memory program** (SMP): multi-threaded execution (e.g. OpenMP)
  - Running across multiple CPU cores on same node

- **GPU programs**: offloading to the device via CUDA

Array Job

- Groups of jobs with the same executable and resource requirements, but different input files.
  - -J “Jobname[index | start-end:increment]”
  - Range of job index is 1~ 10,000
  - LSB_JOBINDEX is set to array index

```
#!/bin/bash
#BSUB -P acc_hpcstaff
#BSUB -n 1
#BSUB -W 02:00
#BSUB -q express
#BSUB -J "jobarraytest[1-10]"
#BSUB -o logs/out.%J.%I
#BSUB -e logs/err.%J.%I
echo “Working on file.$LSB_JOBINDEX”
```

```
gail01@li03c03 $ bsub < myarrayjob.sh
Job <2946012> is submitted to queue <express>.
gail01@li03c03: ~ $ bjobs
     JOBID      USER    JOB_NAME   STAT     QUEUE  FROM_HOST    EXEC_HOST
SUBMIT_TIME    START_TIME  TIME_LEFT
    2946012     gail01 *rraytest[1]   PEND    express    li03c03      -       Sep 10 14:50      -         -
    2946012     gail01 *rraytest[2]   PEND    express    li03c03      -       Sep 10 14:50      -         -
    2946012     gail01 *rraytest[3]   PEND    express    li03c03      -       Sep 10 14:50      -         -
    2946012     gail01 *rraytest[4]   PEND    express    li03c03      -       Sep 10 14:50      -         -
    2946012     gail01 *rraytest[5]   PEND    express    li03c03      -       Sep 10 14:50      -         -
    2946012     gail01 *rraytest[6]   PEND    express    li03c03      -       Sep 10 14:50      -         -
    2946012     gail01 *rraytest[7]   PEND    express    li03c03      -       Sep 10 14:50      -         -
    2946012     gail01 *rraytest[8]   PEND    express    li03c03      -       Sep 10 14:50      -         -
    2946012     gail01 *rraytest[9]   PEND    express    li03c03      -       Sep 10 14:50      -         -
    2946012     gail01 *raytest[10]   PEND    express    li03c03      -       Sep 10 14:50      -         -
```

Message Passing Interface (MPI) Jobs

- This example requests 48 cores and 2 hours in the "express" queue.
  - Those 48 cores can be dispatched across multiple nodes

```
#!/bin/bash
#BSUB -J myjobMPI
#BSUB -P acc_hpcstaff
#BSUB -q express
#BSUB -n 48

#BSUB -W 02:00
#BSUB -o %J.stdout
#BSUB -eo %J.stderr
#BSUB -L /bin/bash

cd $LS_SUBCWD

module load openmpi

mpirun -np 48 /my/bin/executable < my_data.in
```

Multithreaded Jobs - OpenMP

- Multiple CPU cores within one node using shared memory
  - In general, a multithreaded application uses a single process which then spawns multiple threads of execution
  - It's highly recommended the number of threads is set to the number of compute cores
- Your program needs to be written to use multi-threading

```
#!/bin/bash
#BSUB -J myjob
#BSUB -P YourAllocationAccount
#BSUB -q express
#BSUB -n 4
#BSUB -R "span[hosts=1]"
#BSUB -R rusage[mem=12000]
#BSUB -W 01:00
#BSUB -o %J.stdout
#BSUB -eo %J.stderr
#BSUB -L /bin/bash

cd $LS_SUBCWD
export OMP_NUM_THREADS=4          #sets the number of threads
/my/bin/executable < my_data.in
```

Job submission script example: star.lsf

```
#!/bin/bash
#BSUB -J mySTARjob                  # Job name
#BSUB -P acc_PLK2                   # allocation account
#BSUB -q premium                    # queue
#BSUB -n 8                          # number of compute cores
#BSUB -W 12:00                      # walltime in HH:MM
#BSUB -R rusage[mem=4000]           # 32 GB of memory (4 GB per core)
#BSUB -R span[hosts=1]              # all cores from one node
#BSUB -o %J.stdout                  # output log (%J : JobID)
#BSUB -eo %J.stderr                 # error log
#BSUB -L /bin/bash                  # Initialize the execution environment

module load star
WRKDIR=/sc/orga/projects/hpcstaff/benchmark_star
STAR --genomeDir $WRKDIR/star-genome --readFilesIn Experiment1.fastq --runThreadN 8
--outFileNamePrefix Experiment1Star
```

Submit the script with the **bsub** command:

```
bsub < star.lsf
```

Specifying a resource - OpenMP job

Span: define the shape of the slots you ask for:

-n 12 -R span[hosts=1] - allocate all 12 cores to one host

-n 12 -R span[ptile=12] - all 12 slots/cores must be on 1 node

-n 24 -R span[ptile=12] - allocate 12 cores per node = 2 nodes

OMP_NUM_THREADS must be set in script:

- **bsub -n 12 -R span[hosts=1] < my_parallel_job**

  export OMP_NUM_THREADS=12
- **bsub -n 12 -R span[ptile=12] -a openmp < my_parallel_job**

  LSF sets it for you as number of procs per node
- **bsub -n 1 -R "affinity[core(12)]" -R "rusage[mem=12000]" -a openmp < my_parallel_job**
  - 1 job slot with 12 cores, 12000MB merory to that job slot...not per core
  - Advantage: Can vary number of cores and/or memory without making any other changes or calculations

A Bravura Submission - Mixing it all together

Suppose you want to run a combined MPI-openMP job. One mpi process per node, openMP in each MPI Rank:

bsub -n 20 -R span[ptile=1] -R affinity[core(8)] -a openmp < my_awsome_job

ptile=1 - one slot on each node

core(8) - 8 cores per job slot

openmp - will set OMP_NUM_THREADS on each node to 8

GPGPU (General Purpose Graphics Processor Unit)

- GPGPU resources on Minerva
  - Interactive queue (1 GPU node)
  - gpu queue for batch (11 GPU nodes)
    - Can be quite busy sometimes

number of nodes	12
GPU card	4 v100
CPU cores	32
host memory	384GB
GPU memory	16 GB

```
#BSUB -q gpu                                  # submit to gpu queue
#BSUB -n Ncpu                                 # Ncpu is 1~32 on v100

#BSUB -R v100                                 # request specified gpu node v100
#BSUB -R "gpu rusage[ngpus_excl_p=2]"         # The number of GPUs requested per node ( 1 by
                                              default)
module purge
module load anaconda3 ( or 2)                 # to access tensorflow
module load cuda                              # to access the drivers and supporting
source activate tfGPU                         subroutines

python -c "import tensorflow as tf"
```

GPGPU (continue)

- Request multiple GPU cards across different GPU nodes

```
#BSUB -q gpu                                          # submit to gpu queue
#BSUB -n 8                                            # 8 compute cores  requested
#BSUB -R span[ptile=2]                                # 2 cores per node, so 4 nodes in total requested
#BSUB -R v100                                         # request specified gpu node v100
#BSUB -R "gpu rusage[ngpus_excl_p=2]"                 # 2 GPUs requested per node
```

Checkpoint/Restart

https://hpc.mssm.com->Minerva Documentation->Checkpoint-Restart

- Checkpoint: Save the state of a process at a particular point in the computation
- Restart: Restore the state of a process and continue the computation from the saved state.

Checkpoint/Restart

https://hpc.mssm.com->Minerva Documentation->Checkpoint-Restart

- The long-time standard BLCR method is no longer supported
- It has been replaced by the more modern method: Checkpoint/Restart In User space (CRIU)

bsub -k "checkpoint_dir [init=initial_checkpoint_period] [check-point_period] [method=method_name]"

E.g.,

bsub -k "chkpntDir init=10 90 method=criu"

Checkpoint/Restart

https://hpc.mssm.com->Minerva Documentation->Checkpoint-Restart

- To restart, use brestart command
- Must restart on same type of machine.
- Can increase memory, change queue, add dependency, etc (see man page)

brestart [options] checkpointFolder jobid

brestart -W 4:00 -R rusage[mem=26000] chkpnt 193876

BONUS: You may be able to checkpoint a process even if you didn't set it up via LSF.

See HPC web site for details.

More bsub [options]

https://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_command_ref/bsub.heading_options.1.html

Tips for efficient usage of the queuing system

- User limitation
  - Max pending job per user: 20,000
  - Heavy users: depending on the resource requested
- Find appropriate queue and nodes
  - use -q interactive: for debug (both CPU and GPU with internet access)
  - use -q express if walltime < 12h
  - use himem node for memory intensive job
- Request reasonable resource
  - Prior knowledge needed (run test program and use top or others to monitor)
  - Keep it simple
- Job not start after a long pending time
  - Whether the resource requested is non-exist: -R rusage[mem = 10000] -n 20
  - Run into PM:

```
NOTE: Because of PM reservations, job may not run
      until after Sat 21 Mar at  8:00PM
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Job <6628109> is submitted to queue <premium>.
```

- If you see memory not enough
  - Think about shared memory vs distributed memory job……
  - Use -R span[hosts=1] where needed

Final Friendly Reminder

- Never run jobs on login nodes
  - For file management, coding, compilation, etc., purposes only
- Never run jobs outside LSF
  - Fair sharing
  - Scratch disk not backed up, efficient use of limited resources
  - Job temporary dir configured to /local/JOBS instead of /tmp.
- Logging onto compute nodes is no longer allowed

- Follow us by visiting https://labs.icahn.mssm.edu/minervalab , weekly update and twitter
- Acknowledge Scientific Computing at Mount Sinai should appear in your publications
  - This work was supported in part through the computational resources and staff expertise provided by Scientific Computing at the Icahn School of Medicine at Mount Sinai.
  - If you are using BODE: “Research reported in this paper was supported by the Office of Research Infrastructure of the National Institutes of Health under award numbers S10OD026880. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

Last but not Least

- Got a problem? Need a program installed? Send an email to:

hpchelp@hpc.mssm.edu