Minerva Scientific Computing Environment

Load Sharing Facility (LSF)

https://hpc.mssm.edu

Patricia Kovatch
Eugene Fluder, PhD
Hyung Min Cho, PhD
Lili Gai, PhD*
Bhupender Thakur, PhD
Francesca Tartaglione, MS
Dansha Jiang, PhD

Sep 26, 2018

Mount Sinai

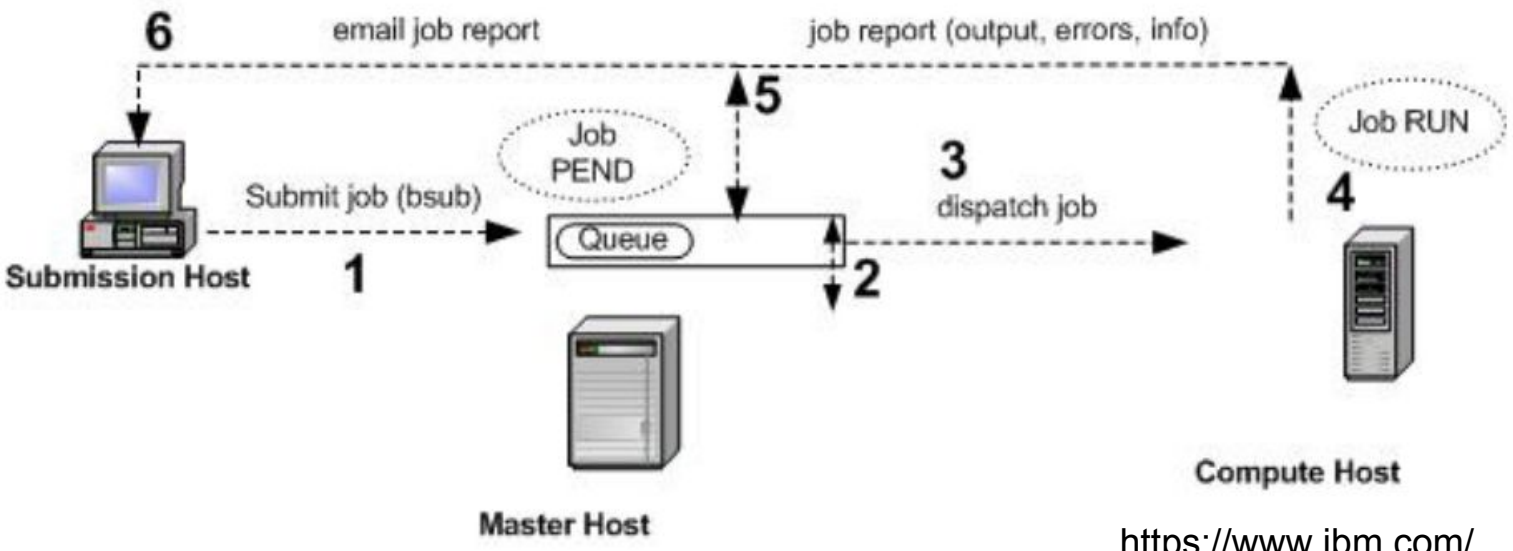
Outline

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

Distributed Resource Management System (DRMS)

- Goal is to achieve best utilization of resources and maximize throughput for high-performance computing systems
- Control usage of hard resources
  - CPU cycles; Memory; Disk Space
- Can be decomposed into subsystems:
  - Job management; Physical resource management; Scheduling and queuing
- 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
6. send Email to client (disabled on Minerva)

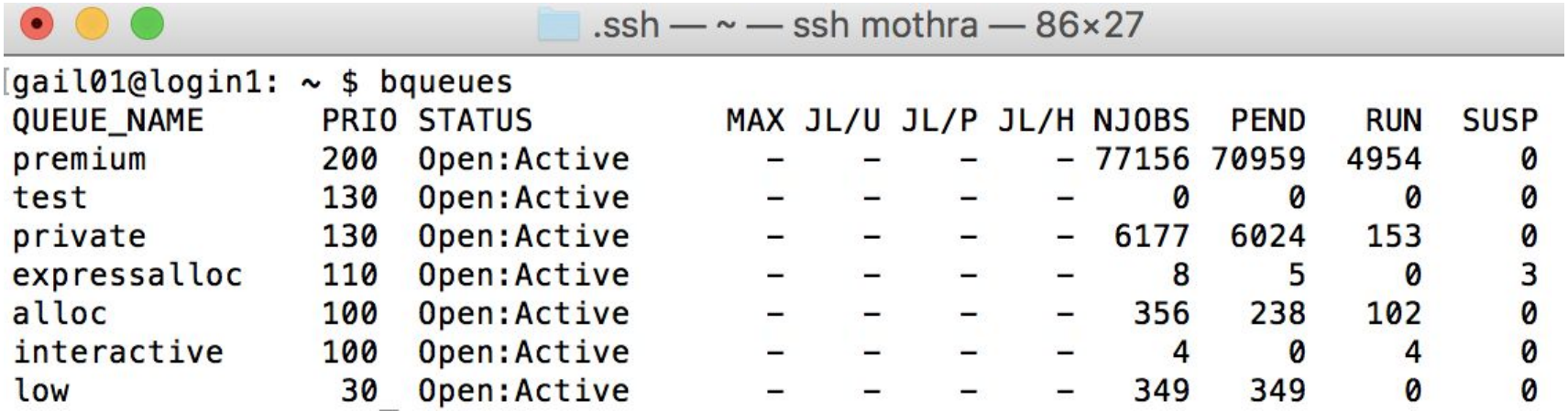
LSF Useful Commands

bhosts: Displays hosts and their static and dynamic resources

.ssh — ~ — ssh mothra — 86×27

```
gail01@login1: ~ $ bhosts
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
bashia02-01        ok              -     36     25     25      0      0      0
bashia02-02        closed          -     24     24     24      0      0      0
bashia02-03        ok              -     24      0      0      0      0      0
bashia02-04        ok              -     24      0      0      0      0      0
bashia02-05        closed          -     24     24     24      0      0      0
db2                ok              -     24      0      0      0      0      0
dor01-1            closed          -     48     48     48      0      0      0
filizm02-1         ok              -     24      0      0      0      0      0
filizm02-2         ok              -     24      0      0      0      0      0
filizm02-3         closed          -     24     24     24      0      0      0
loosr01-1          ok              -     24      0      0      0      0      0
master_hosts       closed          -      0      0      0      0      0      0
node19-1           ok              -     16      0      0      0      0      0
node19-2           ok              -     20      5      3      0      0      2
node21-1           ok              -     12      7      7      0      0      0
node21-10          closed          -     12     12     12      0      0      0
node21-11          closed          -     12     12     12      0      0      0
node21-12          closed          -     12     12     11      0      0      1
node21-13          closed          -     12     12     12      0      0      0
node21-14          closed          -     12     12     12      0      0      0
node21-15          closed          -     12     12     12      0      0      0
node21-16          closed          -     12     12     12      0      0      0
node21-17          closed          -     12     12     12      0      0      0
node21-18          closed          -     12     12     12      0      0      0
node21-19          closed          -     12     12     12      0      0      0
```

bqueues: displays information about 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:Inact_Adm - - - - 0 0 0 0 0 0
Interval for a host to accept two jobs is 0 seconds

DEFAULT LIMITS:
….
MAXIMUM LIMITS:
RUNLIMIT
720.0 min of mgmt2
……

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

Interactive jobs:

- This will set up an interactive environment on compute nodes for users
- Useful for testing and debugging jobs

bsub -XF -P acc_hpcstaff -q interactive -n 5 -W 1:10 -R rusage[mem=3000] -Ip /bin/bash

- -q : to specify the queue-name from where to get the nodes
- -Ip: 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@login1: ~ $ bsub -XF -P acc_hpcstaff -q interactive -n 5 -W 1:10 -R rusage[mem=
3000] -Ip /bin/bash
Job <109679591> is submitted to queue <interactive>.
<<ssh X11 forwarding job>>
<<Waiting for dispatch ...>>
<<Starting on node7-13>>
gail01@node7-13: ~ $
```

bsub - batch job

Create job scripts containing job info and commands to the LSF

bsub [options] < my_batch_job or **bsub [options] my_batch_job**

- With "<" will interpret the #BSUB cookies in the script.
- Options on the command line override what is in the script

```
gail01@login1: ~ $ cat helloworld.lsf
#!/bin/bash
#BSUB -J myjob                      # name of the job
#BSUB -P acc_hpcstaff               # the CPU allocation account used
#BSUB -q alloc                      # Job queue
#BSUB -n 1                          # number of cores ( job slot). Try to fill a node first
#BSUB -R rusage[mem=2000]           # memory requested each core
#BSUB -W  02:00                     # time limit in HH:MM, no second!
#BSUB -m manda                      # specific host
#BSUB -o t.out                      # output is sent to file t.out
#BSUB -e t.err                      # error  is sent to file t.out

#<shell command>                    # shell command
echo "salve munde!"                 # execution
#<shell command>

gail01@login1: ~ $ bsub < helloworld.lsf
Job <109680072> is submitted to queue <alloc>.
```

bjobs - status of jobs

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

```
gail01@login1: ~ $ bjobs
     JOBID     USER    JOB_NAME   STAT     QUEUE  FROM_HOST    EXEC_HOST   SUBMIT_TIME    START_TIME
TIME_LEFT
  109680072   gail01       myjob   PEND     alloc     login1      -       Sep 20 10:31      -         -
```

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

```
     JOBID     USER    JOB_NAME   STAT     QUEUE  FROM_HOST    EXEC_HOST   SUBMIT_TIME    START_TIME
TIME_LEFT
  111628054     lij43      mark35    RUN    premium   minerva4    node22-12  Sep 20 10:22  Sep 20 10:29    11:53 L
  104423808   xdayam01  simulate_h0  USUSP expressall   minerva2    node22-12  Jun  1 18:14  Jun  1 18:16     1:8 L
  111628104    leee19    Act-colon    RUN    premium   minerva4    node22-11  Sep 20 10:26  Sep 20 10:31    79:54 L
  104423811   xdayam01  simulate_h0  USUSP expressall   minerva2    node22-11  Jun  1 18:14  Jun  1 18:16     1:8 L
  104423819   xdayam01  simulate_h0  USUSP expressall   minerva2    node22-13  Jun  1 18:14  Jun  1 18:16     1:8 L
  ……...
```

- Long format with option -l

bmod - modify submission options of pending jobs

bmod takes similar options to bsub

- bmod -R rusage[mem=20000] <jobID>
- bmod -q expressalloc <jobID>

```
gail01@login1: ~ $ bmod -q expressalloc 109680072
Parameters of job <109680072> are being changed
```

bpeek - display output of the job produced so far

bpeek <jobID>

```
gail01@login1: ~ $ bpeek 109680072
<< output from stdout >>
“salve munde!”

<< 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@login1: ~ $ bhist -n 3 -l 109680070

Job <109680070>, Job Name <myjob>, User <gail01>, Project <acc_hpcstaff>, Appli
                     cation <default>, Command <#!/bin/bash     ;#BSUB -J myjob
                      ;#BSUB -P acc_hpcstaff;#BSUB -q alloc;#BSUB -n 1;#BSUB -R
                      rusage[mem=2000];#BSUB -W  02:00;#BSUB -m manda;#BSUB -o
                     t.out;#BSUB -e t.err; #<shell command>;echo “salve munde
                     !”;#<shell command>>
Thu Aug 23 14:17:22: Submitted from host <login1>, to Queue <alloc>, CWD <$HOME
                     >, Output File <t.out>, Error File <t.err>, Requested Reso
                     urces <rusage[mem=2000]>, Specified Hosts <manda>;

 RUNLIMIT
 120.0 min of login1

 MEMLIMIT
    1.9 G
Thu Aug 23 14:19:19: Dispatched to <node5-12>, Effective RES_REQ <select[((heal
                     thy=1)) && (type == any)] order[!-slots:-maxslots] rusage[
                     mem=2000.00] same[model] affinity[core(1)*1] >;
Thu Aug 23 14:19:20: Starting (Pid 56658);
Thu Aug 23 14:19:21: Running with execution home </hpc/users/gail01>, Execution
                      CWD </hpc/users/gail01>, Execution Pid <56658>;
Thu Aug 23 14:19:22: Done successfully. The CPU time used is 0.4 seconds;
Thu Aug 23 14:19:23: Post job process done successfully;

MEMORY USAGE:
MAX MEM: 23 Mbytes;  AVG MEM: 10 Mbytes

Summary of time in seconds spent in various states by  Thu Aug 23 14:19:23
  PEND     PSUSP    RUN      USUSP    SSUSP    UNKWN    TOTAL
  117      0        3        0        0        0        120
```

bhist - common errors of exited jobs

1. Valid allocation account needed in the submission script

Cannot open your job file: /tmp/1533305843.108403997
TERM_ADMIN: job killed by root or an administrator.
Exited with signal termination: Interrupt.

Thu Aug 23 23:09:58: External Message "Job was terminated due to reservation failure in GOLD

- $mybalance

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 2000MB as default
- multithreaded applications need to be on the same node, such as STAR, bwa

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 premium
#BSUB -W 1:00
#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/Exec_1 < my_input_parameters_1 > output_1.log

/my/bin/path/Exec_3 < my_input_parameters_3 > output_3.log

.

/my/bin/path/Exec_9999 < my_input_parameters_9999 > output_9999.log

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

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 or OpenCL

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

```
#!/bin/bash

#BSUB -W 1:0
#BSUB -q expressalloc
#BSUB -J "jobarraytest[1-10]"
#BSUB -o logs/out.%J.%I
#BSUB -e logs/err.%J.%I
Working on file.$LSB_JOBINDEX
```

```
gail01@minerva4: $ bsub < myjob.sh
Job <108519892> is submitted to queue <expressalloc>.
gail01@minerva4  $ bjobs
     JOBID      USER    JOB_NAME   STAT     QUEUE  FROM_HOST    EXEC_HOST
SUBMIT_TIME    START_TIME  TIME_LEFT
  108519892    gail01 *rraytest[1]   PEND expressall   minerva4     -       Aug  6 09:58      -         -
  108519892    gail01 *rraytest[2]   PEND expressall   minerva4     -       Aug  6 09:58      -         -
  108519892    gail01 *rraytest[3]   PEND expressall   minerva4     -       Aug  6 09:58      -         -
  108519892    gail01 *rraytest[4]   PEND expressall   minerva4     -       Aug  6 09:58      -         -
  108519892    gail01 *rraytest[5]   PEND expressall   minerva4     -       Aug  6 09:58      -         -
  108519892    gail01 *rraytest[6]   PEND expressall   minerva4     -       Aug  6 09:58      -         -
  108519892    gail01 *rraytest[7]   PEND expressall   minerva4     -       Aug  6 09:58      -         -
  108519892    gail01 *rraytest[8]   PEND expressall   minerva4     -       Aug  6 09:58      -         -
  108519892    gail01 *rraytest[9]   PEND expressall   minerva4     -       Aug  6 09:58      -         -
  108519892    gail01 *raytest[10]   PEND expressall   minerva4     -       Aug  6 09:58      -
-
```

MPI Jobs

- This example requests 48 cores on the Mothra side (intel cores) for 2 hours in the "alloc" queue.

```
#!/bin/bash
#BSUB -J myjobMPI
#BSUB -P YourAllocationAccount
#BSUB -q alloc
#BSUB -n 48

#BSUB -W 02:00
#BSUB -m mothra
#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 Job - 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 alloc
#BSUB -n 4
#BSUB -R "span[hosts=1]"
#BSUB -R rusage[mem=12000]
#BSUB -W 01:00
#BSUB -m manda
#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
```

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)] -app 1switch -a openmp < my_awsome_job

1switch - All nodes on same network switch, to minimize communication latency (24 mothra nodes =24*12 cores)

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)

- Minerva has two types of GPGPU resources

Partition	P100	K20Xm
number of nodes	1	8
GPU card	4	2
CPU cores	20 (Intel Broadwell)	24(Intel Ivy Bridge)
host memory	128 GB	258GB
GPU memory	12 GB	6 GB

```
#BSUB -n Ncpu                                  # Ncpu is 1~24 on k20xm, 1~20 on P100
#BSUB -R “gpu”                                 # request gpu resource

#BSUB -m gpu-k20xm ( or gpu-p100)              # request specified gpu node
#BSUB -R "gpu rusage[ngpus_excl_p=2]"          # The number of GPUs requested per node ( 1 by default)

module purge
module load anaconda2 ( or 3)                  # to access tensorflow
module load cuda/9.1.85                        # to access the drivers and supporting
module load cudnn/7.1.1 gcc/5.3.0              subroutines
source activate tfGPU

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

Restarting job with checkpoint - BLCR

- For jobs beyond 144 hours of walltime

#BSUB -k "checkpoint_dir [init=initial_checkpoint_period] [checkpoint_period] [method=method_name]"

- checkpoint_dir - folder where checkpoints are to be stored. Will be created if needed.
- initial_checkpoint_period - wait this many minutes before taking the first checkpoint
- checkpoint_period - checkpoint every this many minutes after the initial checkpoint is taken.
- method_name - **blcr, and blcr_R for R users**

```
#!/bin/bash
#            myJob.lsf
#BSUB -P acc_hpcstaff
#BSUB -n 1
#BSUB -R affinity[core(4)]
#BSUB -q premium
#BSUB -W 144:00
#BSUB -k "chkpt_dir init=1 1200 method=blcr"
#BSUB -app chkpnt
#BSUB -oo %J.out
#BSUB -eo %J.err

module load gcc
cr_run ./myLooooooongProgram> test.out 2>test.err
```

Checkpoint - BLCR

- [user01@login1 lsf]$ bsub < myjob.lsf

Job <96610603> is submitted to queue <premium>

```
$ ls chkpt_dir
96610603


$ ls chkpt_dir/96610603/
1516635926.96610603 1516635926.96610603.out chklog
1516635926.96610603.err 1516635926.96610603.shell context.31117
```

- To restart the job,

brestart [bsub_options] checkpoint_dir [job_ID]

brestart -w 144:0 chkpt_dir 96610603

Checkpoint - BLCR

- Checkpoint programs only
  - Enable checkpoint for the program at startup. **Programs must be dynamically linked** (use *file* command to check).

```
gail01@login1: ~ $ file hello
hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked
(uses shared libs), for GNU/Linux 2.6.18, not stripped
```

  - All files open at the time of a checkpoint must exist at their original locations for the restart.
  - Be sure you restart a program on the same type of node as it was checkpointed on. E.g. checkpointed on Manda node, restart on Manda node.
  - Checkpointing of MPI programs: No

- More at https://hpc.mssm.edu/about/checkpoint

Tips for efficient usage of the queuing system

- Jobs with a higher job priority
  - Job priorities factors: fairshare, service and resource priority
  - Backfill idle nodes to maximize utilization
- Find appropriate queue
  - -q expressalloc: walltime < 6h
  - -q interactive: for debug
- 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, e.g., -m mothra -R rusage[mem = 10000] -n 10 -span[ptile=1]
  - Too much memory: -R rusage[mem = 10000] -n 10
  - Run into PM: -W 144h
- If you see memory not enough
  - Think about shared memory vs distributed memory job……..

Last but not Least

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

hpchelp@hpc.mssm.edu