HPC User Environment 2

Feng Chen
HPC User Services
LSU HPC   LONI
sys-help@loni.org

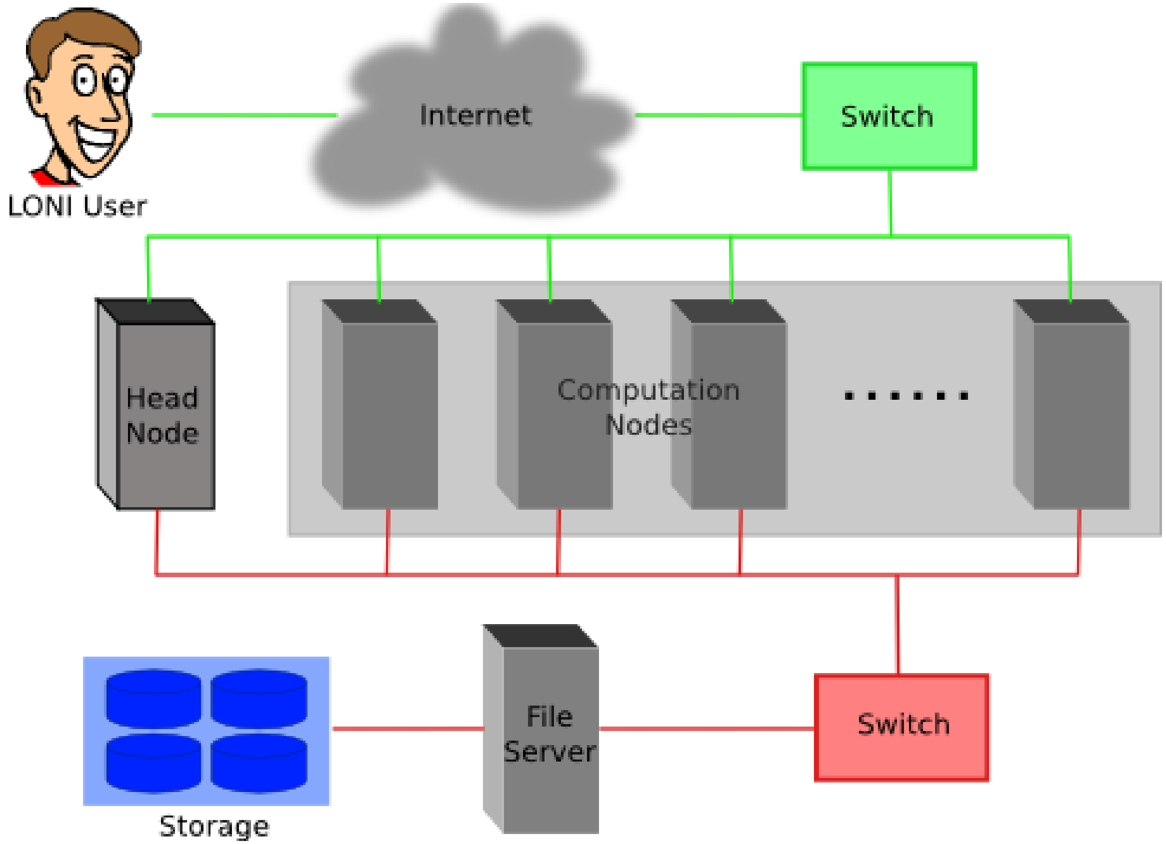
Louisiana State University
Baton Rouge
September 17, 2014

Outline

- **Last training topics**
  - Accounts and Allocations
  - Connect to clusters
  - Software management using softenv
- **Things to be covered in this training**
  - More on job management
    - Submit serial and parallel jobs
    - Job priority
    - Backfill
  - Compiling and Analyzing programs
    - Serial program
    - Parallel program
      - MPI
      - OpenMP
      - Hybrid program

Cluster Environment

- Multiple compute nodes
- Multiple users
- Each user may have multiple jobs running simultaneously



Basic HPC cluster resource terms

Term	Definition
Cluster	The top-level organizational unit of an HPC cluster, comprising a set of nodes, a queue, and jobs.
Node	A single, named host in the cluster.
Core	An individual CPU on a node. For example, a quad-core processor is considered 4 cores.
Job	A user's request to use a certain amount of resources for a certain amount of time on cluster for his work.

HPC Cluster Architectures

- **Two major architectures**
  - Intel x86_64 clusters
    - Vendor: Dell
    - Operating System: Linux (RHEL 4/5/6)
    - Processor: Intel
  - IBM PowerPC clusters
    - Vendor: IBM
    - Operating System: AIX
    - Processor: IBM power7

Job Scheduler

- **A software that manages resources (CPU time, memory etc.) and schedules job execution**
  - Linux clusters: Portable Batch System(PBS)
  - AIX clusters: Loadleveler
- **The batch queuing system determines**
  - The order jobs are executed
  - On which node(s) jobs are executed
- **Linux clusters use TORQUE, an open source version of the Portable Batch System (PBS) together with the MOAB Scheduler, to manage user jobs.**
- **Job management basics**
  - Find appropriate queue
  - Understand the queuing system and your requirements and proceed to submit jobs
  - Monitor jobs

Job Queues

- **Nodes are organized into queues. Nodes can be shared.**
- **Each job queue differs in**
  - Number of available nodes
  - Max run time
  - Max running jobs per user
  - Nodes may have special characteristics: GPU’s, Large memory, etc.
- **Jobs need to specify resource requirements**
  - Nodes, time, queue
- **Its called a queue for a reason, but jobs don’t run on a “First Come First Served” policy**

Queue Characteristics – LONI clusters

Machine	Queue	Max Runtime	# of nodes	Max running jobs per user	Max nodes per job	Use
Eric	workq	3 days	128	16	24	Unpreemptable
	checkpt		128		48	Preemptable
	single		1		1	ppn < =8
Others	workq	3 days	128	8	48	Unpreemptable
	checkpt		96		64	Preemptable
	single	14 days	16	64	1	Single processor

Queue Characteristics – LSU Linux clusters

Machine	Queue	Max Runtime	# of nodes	Max running jobs per user	Max nodes per job	Use
SuperMikeII	workq	3 days	128	34	128	Unpreemptable
	checkpt		96		128	Preemptable
	bigmem	2 days	8		2	Big memory
	gpu	1 day	50		32	Job using GPU
Philip	workq	3 days	28	12	5	Unpreemptable
	checkpt		28			Preemptable
	gpu		2			Job using GPU
	bigmem		5			Big memory
	single	14 days	24		1	Single processor

Queue Characteristics – LSU AIX Clusters

Machine	Queue	Max Runtime	# of cores	Max running jobs per user	Max cores per job	Use
Pandora	Interactive	30 minutes	8	6	8	Unpreemptable
	Workq	3 days	224		128	Preemptable
	Single	7 days	64		32	Single processor

Queue Characteristics

- "qstat -q" will give you more info on the queues

```
[fchen14@mike2 ~]$ qstat -q

server: mike3

Queue            Memory CPU Time Walltime Node  Run Que Lm  State
---------------- ------ -------- -------- ----  --- --- --  -----
workq              --      --    72:00:00   128  31   6 --   E R
mwfa               --      --    72:00:00     8   3   0 --   E R
bigmem             --      --    48:00:00     1   0   0 --   E R
lasigma            --      --    72:00:00    28  28   7 --   E R
bigmemtb           --      --    48:00:00     1   0   0 --   E R
priority           --      --    168:00:0   128   0   0 --   E R
single             --      --    72:00:00     1  62   0 --   E R
gpu                --      --    24:00:00    16   1   0 --   E R
preempt            --      --    72:00:00   --    0   0 --   E R
checkpt            --      --    72:00:00   128  31 137 --   E R
admin              --      --    24:00:00   --    0   0 --   E R
scalemp            --      --    24:00:00     1   0   0 --   E R
                                               ----- -----
                                                 156   150
```

- For a more detailed description use mdiag

Queue Querying – Linux Clusters

- **Displays information about active, eligible, blocked, and/or recently completed jobs: `showq` command**

```
$ showq
active jobs------------------------
JOBID              USERNAME      STATE PROCS   REMAINING            STARTTIME
236875              ebeigi3    Running    16     1:44:29  Mon Sep 15 20:00:22
236934                 mwu3    Running    16    00:03:27  Mon Sep 15 19:04:20
...
eligible jobs----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME
236795              dmarce1       Idle  1456    00:15:00  Mon Sep 15 16:38:45
236753               rsmith       Idle  2000     4:00:00  Mon Sep 15 14:44:52
236862              dlamas1       Idle   576     2:00:00  Mon Sep 15 17:28:57
...
121 eligible jobs
blocked jobs-----------------------
JOBID              USERNAME      STATE PROCS     WCLIMIT            QUEUETIME
232741              myagho1       Idle  2000  1:00:00:00  Mon Sep  8 07:22:12
235545              tanping       Idle     1  2:21:10:00  Fri Sep 12 16:50:49
235546              tanping       Idle     1  2:21:10:00  Fri Sep 12 16:50:50
...
```

Queue Querying - AIX clusters

- **Command: `llclass`**

```
-bash-3.2$ llclass
Name                    MaxJobCPU     MaxProcCPU  Free   Max Description
                       d+hh:mm:ss     d+hh:mm:ss Slots Slots
--------------- -------------- -------------- ----- ----- ---------------------
interactive             unlimited      unlimited     8     8 Queue for interactive jobs; maximum
runtime of 30 minutes.
workq                   unlimited      unlimited    32   224 Standard queue for job submissions;
maximum runtime of 3 days.
cheme                   unlimited      unlimited    32    96 Queue for Chemical Engineering;
maximum runtime of 3 days.
single                  unlimited      unlimited    32    64 Queue for single-node job
submissions; maximum runtime of 3 days.
--------------------------------------------------------------------------------
"Free Slots" values of the classes "workq", "cheme", "single" are constrained by the
MAX_STARTERS limit(s).
```

Two Job Types

- **Interactive job**
  - Set up an interactive environment on compute nodes for users
    - Advantage: can run programs interactively
    - Disadvantage: must be present when the job starts
  - Purpose: testing and debugging
    - Do not run on the head node!!!
    - Try not to run interactive jobs with large core count, which is a waste of resources)
- **Batch job**
  - Executed without user intervention using a job script
    - Advantage: the system takes care of everything
    - Disadvantage: can only execute one sequence of commands which cannot changed after submission
  - Purpose: production run

Submitting Jobs on Linux Clusters

- **Interactive job example:**

```
qsub -I -V \
     -l walltime=<hh:mm:ss>,nodes=<num_nodes>:ppn=<num_cores> \
     -A <Allocation> \
     -q <queue name>
```

- **Add -X to enable X11 forwarding**

- **Batch Job example:**

```
qsub job_script
```

PBS Job Script – Serial Job

```
#!/bin/bash
#PBS -l nodes=1:ppn=1         # Number of nodes and processor
#PBS -l walltime=24:00:00     # Maximum wall time
#PBS -N myjob                 # Job name
#PBS -o <file name>           # File name for standard output
#PBS -e <file name>           # File name for standard error
#PBS -q single                # The only queue that accepts serial jobs
#PBS -A <loni_allocation>     # Allocation name
#PBS -m e                     # Send mail when job ends
#PBS -M <email address>       # Send mail to this address

<shell commands>
<path_to_executable> <options>
<shell commands>
```

PBS Job Script – Parallel Job

```
#!/bin/bash
#PBS -l nodes=4:ppn=4                #Number of nodes and processors per node
#PBS -l walltime=24:00:00            #Maximum wall time
#PBS -N myjob                        #Job name
#PBS -o <file name>                  #File name for standard output
#PBS -e <file name>                  #File name for standard error
#PBS -q checkpt                      #Queue name
#PBS -A <allocation_if_needed>       #Allocation name
#PBS -m e                            #Send mail when job ends
#PBS -M <email address>              #Send mail to this address

<shell commands>
mpirun -machinefile $PBS_NODEFILE -np 16 <path_to_executable> <options>
<shell commands>
```

LoadLeveler Job Script - Parallel

```
#!/bin/sh
#@ job_type= parallel                                  Job type
#@ output = /work/default/username/$(jobid).out Standard output
#@ error = /work/default/username/$(jobid).err  Standard error
#@ notify_user= youremail@domain                       Notification
#@ notification = error                                Notify on error
#@ class = checkpt                                     Queue
#@ wall_clock_limit= 24:00:00                          Wall clock time
#@ node_usage= shared node usage
#@ node = 2                                            # of nodes
#@ total_tasks= 16                                     # of processors
#@ requirements = (Arch == “POWER7”)                   # Job requirement
#@ environment = COPY_ALL    Environment
#@ queue

<shell commands>
poe<path_to_executable> <options>
<shell commands>
```

LoadLeveler Job Script - Serial

```
#!/bin/sh
#@ job_type= serial                                    Job type
#@ output = /work/default/username/$(jobid).out Standard output
#@ error = /work/default/username/$(jobid).err  Standard error
#@ notify_user= youremail@domain                      Notification
#@ notification = error                               Notify on error
#@ class = single                                     Queue
#@ wall_clock_limit= 24:00:00                         Wall clock time
#@ requirements = (Arch == “POWER5”)                  Job requirement
#@ environment = COPY_ALL                             Environment
#@ queue

<shell commands>
poe <path_to_executable> <options>
<shell commands>
```

Submitting Jobs - AIX clusters

- **Submit jobs using llsubmit**

  `llsubmit jobscript : submit job`

  `llcancel jobid : delete job`
- **Check job status using** `llq` **and cluster status using** `llstatus`

Job Monitoring - Linux Clusters

- **Check details on your job using qstat**

```
$ qstat -f jobid      : For details on your job
$ qstat -n -u $USER   : For quick look at nodes assigned to you
$ qdel jobid          : To delete job
```

- **Check approximate start time using showstart**

```
$ showstart jobid
```

- **Check details of your job using checkjob**

```
$ checkjob jobid
```

- **Check health of your job using qshow**

```
$ qshow -j jobid
```

- **Please pay close attention to the load and the memory consumed by your job!**

Using the “top” command

- The top program provides a dynamic real-time view of a running system.

```
top - 19:39:56 up 89 days,  4:13,  1 user,  load average: 0.63, 0.18, 0.06
Tasks: 489 total,   2 running, 487 sleeping,   0 stopped,   0 zombie
Cpu(s):  6.3%us,  0.0%sy,  0.0%ni, 93.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  65909356k total,  3389616k used, 62519740k free,   151460k buffers
Swap: 207618040k total,     5608k used, 207612432k free,   947716k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 39595 fchen14   20   0  266m 257m  592 R 99.9  0.4   0:06.94 a.out
 39589 fchen14   20   0 17376 1612  980 R  0.3  0.0   0:00.05 top
 38479 fchen14   20   0  108m 2156 1348 S  0.0  0.0   0:00.03 bash
 39253 fchen14   20   0  103m 1340 1076 S  0.0  0.0   0:00.00 236297.mike3.SC
 39254 fchen14   20   0  103m 1324 1060 S  0.0  0.0   0:00.00 bm_laplace.sh
 39264 fchen14   20   0 99836 1908  992 S  0.0  0.0   0:00.00 sshd
 39265 fchen14   20   0  108m 3056 1496 S  0.0  0.0   0:00.03 bash
```

Pay attention to single queue usage

- **Single queue - Used for jobs that will only execute on a single node, i.e. nodes=1:ppn<=4.**
- **Jobs in the single queue should not use more than 2GB memory per core.**
- **If applications require more memory, scale the number of cores (ppn) to the amount of memory required i.e. max memory available for jobs in single queue is 8GB for ppn=4.**

Job Monitoring - AIX Clusters

- **Command: `showllstatus.py`**
  - Show job status and nodes running on
- **Command: `llq <options> <job_id>`**
  - All jobs are displayed if `<job_id>` is omitted
  - Display detailed information: `llq -l <job_id>`
  - Check the estimated start time: `llq -s <job_id>`
  - Show jobs from a specific user: `llq -u <username>`

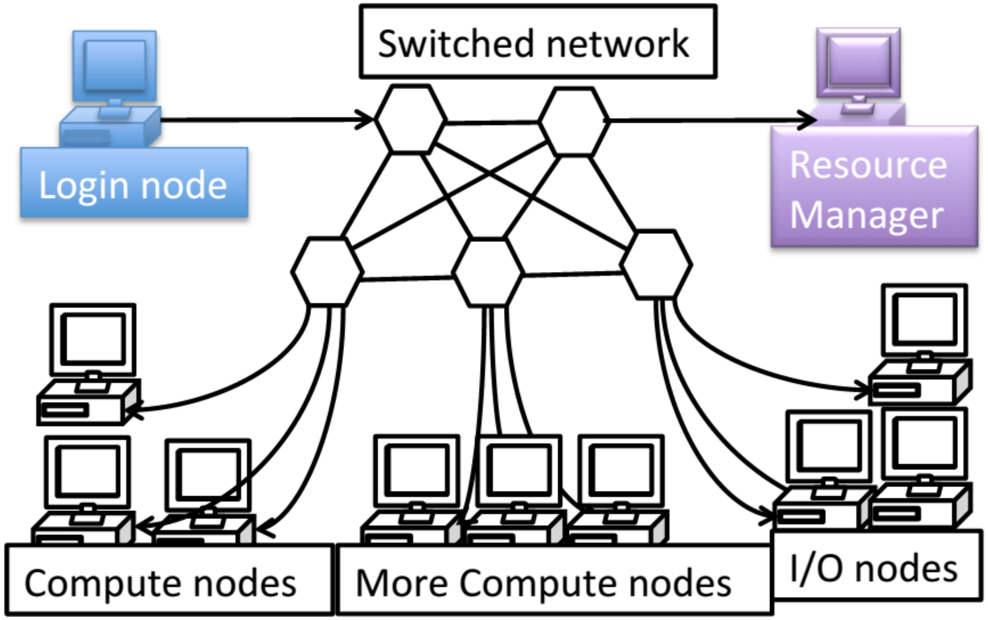
```
-bash-3.2$ llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
pandora1.19106.0         mainak      9/1  23:41 R  50  workq        pandora008
pandora1.19108.0         ghoshbd     9/2  14:58 R  50  workq        pandora005
pandora1.19109.0         ghoshbd     9/2  15:08 R  50  workq        pandora007
pandora1.19110.0         ghoshbd     9/2  15:33 R  50  workq        pandora002
pandora1.19111.0         ghoshbd     9/2  15:44 R  50  workq        pandora004
pandora1.19112.0         ghoshbd     9/2  15:58 I  50  workq
pandora1.19113.0         ghoshbd     9/2  16:10 I  50  workq
pandora1.19114.0         mainak      9/4  08:16 I  50  workq

8 job step(s) in queue, 3 waiting, 0 pending, 5 running, 0 held, 0 preempted
```

PBS Environmental Variables

```
[fchen14@mike315 ~]$ echo $PBS_
$PBS_ENVIRONMENT  $PBS_MOMPORT      $PBS_NUM_PPN      $PBS_O_MAIL
$PBS_QUEUE        $PBS_WALLTIME
$PBS_GPUFILE      $PBS_NODEFILE     $PBS_O_HOME       $PBS_O_PATH
$PBS_SERVER
$PBS_JOBCOOKIE    $PBS_NODENUM      $PBS_O_HOST       $PBS_O_QUEUE
$PBS_TASKNUM
$PBS_JOBID        $PBS_NP           $PBS_O_LANG       $PBS_O_SHELL
$PBS_VERSION
$PBS_JOBNAME      $PBS_NUM_NODES    $PBS_O_LOGNAME    $PBS_O_WORKDIR
$PBS_VNODENUM
```

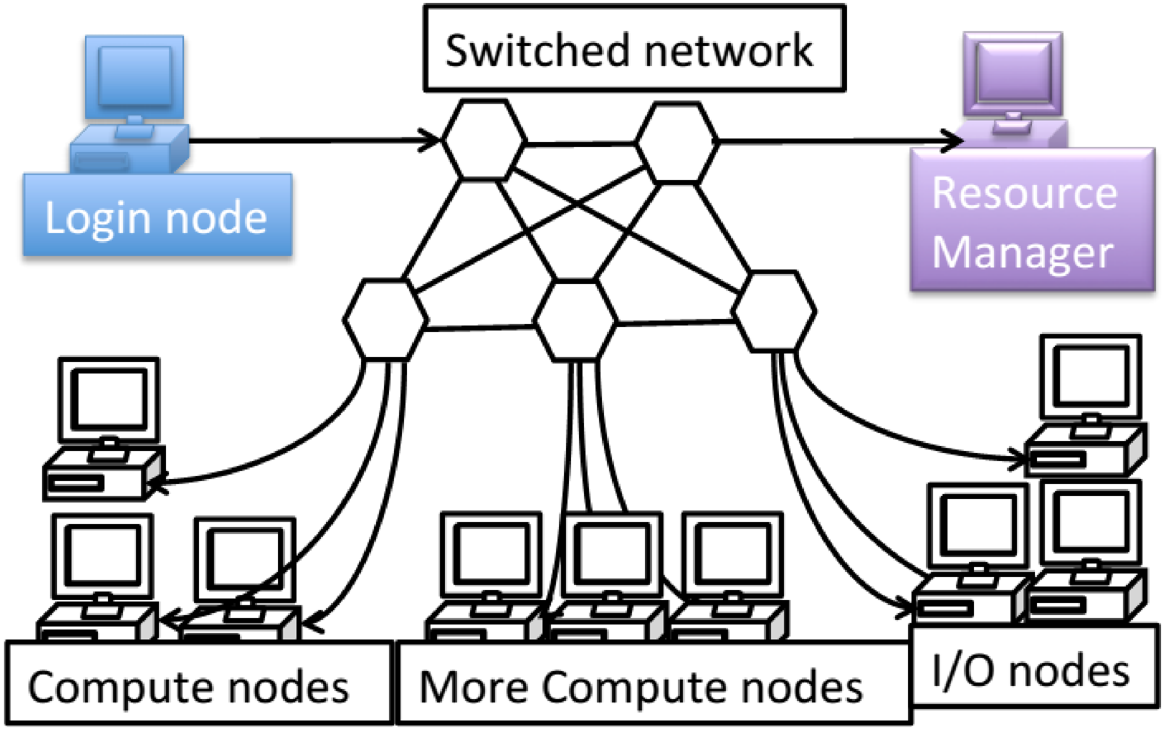
Back to Cluster Architecture

- **Resource managers give access to compute resource**
  - Takes in a resource request on login node
  - Finds appropriate resource and assigns you a priority number
  - Positions your job in a queue based on the priority assigned.
  - Starts running jobs until it cannot run more jobs with what is available.
- **Newer jobs coming in can have a higher priority as It follows a complex calculation for priority number**



Resource manager philosophy

- **Working Philosophy**
  - Prioritize workload into a queue for jobs
  - ***Backfill*** idle nodes to maximize utilization
    - Will be detailed later...



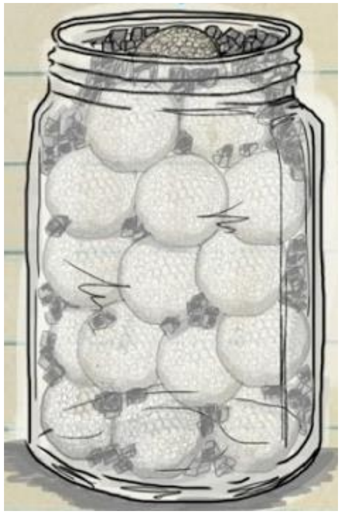
Job Priorities

- **Jobs with a higher job priority are scheduled ahead of jobs with a lower priority.**
- **Job priorities have contributions from the following:**
  - credential priority
  - fairshare priority
  - resource priority
  - service priority
- Priority determination for each queued job, use
  - `mdiag -p`:

```
$ mdiag -p
diagnosing job priority information (partition: ALL)
Job          PRIORITY*   Cred( User:Class)   FS( User:  WCA)  Serv(QTime:XFctr)   Res( Proc)
       Weights --------    100(   10:   10)  100(   10:   50)    2(    2:   20)   30(   10)

236172         246376    40.6(100.0:  0.0)   8.6( 19.6:  0.3)   4.0(1480.: 99.7)  46.8(2048.)
235440         242365    41.3(100.0:  0.0)   4.6(  8.2:  0.6)   6.6(3959.:  6.5)  47.5(512.0)
235441         242365    41.3(100.0:  0.0)   4.6(  8.2:  0.6)   6.6(3959.:  6.5)  47.5(512.0)
235442         242361    41.3(100.0:  0.0)   4.6(  8.2:  0.6)   6.6(3958.:  6.5)  47.5(512.0)
236396         241821    41.4(100.0:  0.0)   8.8( 19.6:  0.3)   2.2(664.0: 67.4)  47.6(1456.)
```

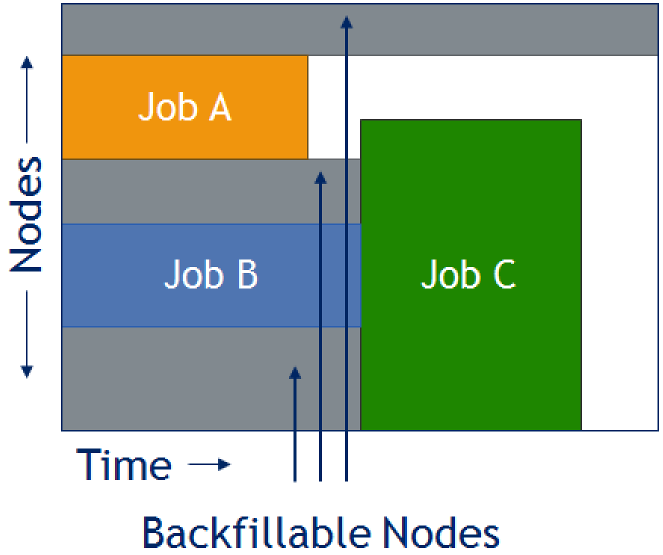
Priority components

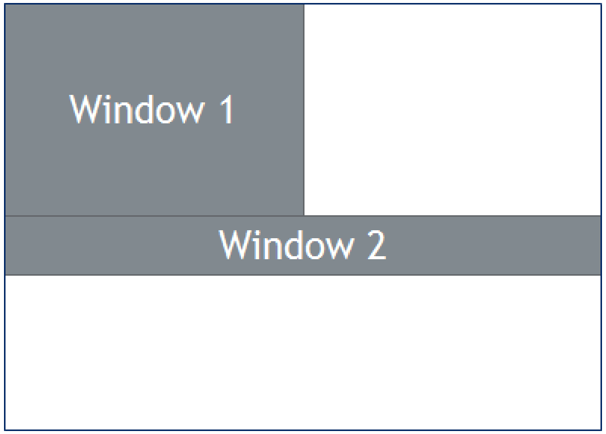
- Credential priority = credweight * (userweight * job.user.priority)
- Fairshare priority = fsweight * min (fscap, (fsuserweight * DeltaUserFSUsage))
- Resource priority = resweight * min (rescap, (procweight * TotalProcessorsRequested)
- Service priority = serviceweight * (queuetimeweight * QUEUETIME + xfactorweight * XFACTOR)
- http://www.hpc.lsu.edu/docs/pbs.php



An Overview of Backfilling (1)

- Backfill is a scheduling optimization that allows a scheduler to make better use of available resources by running jobs out of order.
- Enabling backfill allows the scheduler to start other, lower-priority jobs so long as they do not delay the highest priority job.
- If the FIRSTFIT algorithm is applied, the following steps are taken:
  - The list of feasible backfill jobs is filtered, selecting only those that will actually fit in the current backfill window.
  - The first job is started.
  - While backfill jobs and idle resources remain, repeat step 1.





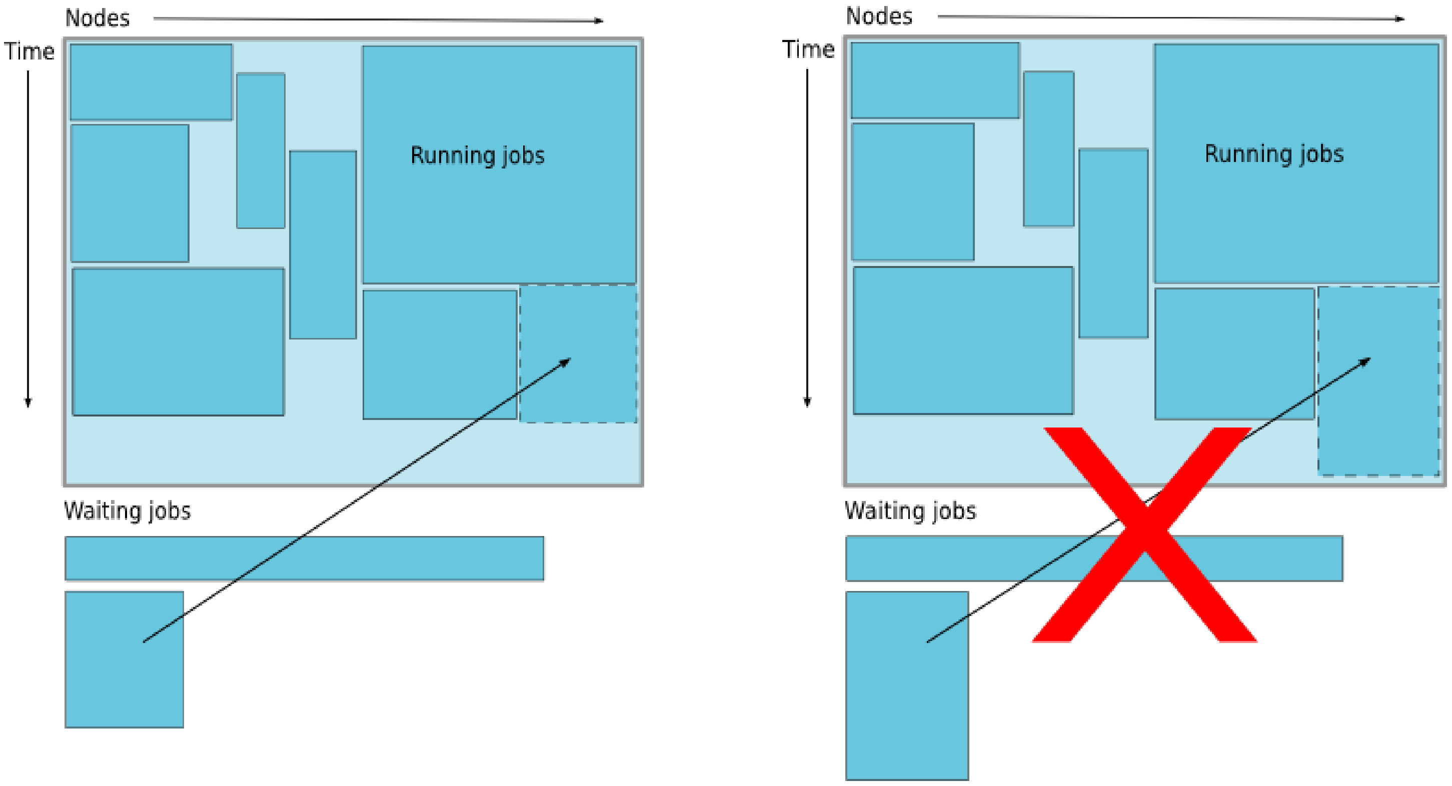
An Overview of Backfilling (2)

- **Although by default the start time of the highest priority job is protected by a reservation, there is nothing to prevent the third priority job from starting early and possibly delaying the start of the second priority job.**
- **Command to show current backfill windows:**
  - showbf
    - Shows what resources are available for immediate use. This command can be used by any user to find out how many processors are available for immediate use on the system. It is anticipated that users will use this information to submit jobs that meet these criteria and thus obtain quick job turnaround times.
  - Example:

```
[fchen14@eric2 ~]$ showbf -c workq
Partition     Tasks  Nodes      Duration   StartOffset       StartDate
---------     -----  -----  ------------  ------------  --------------
ALL              40      5      18:50:35      00:00:00  11:16:49_09/04
ALL               8      1      INFINITY      00:00:00  11:16:49_09/04
```

How Much Time Should I Ask for?

- **It should be**
  - Long enough for your job to complete
  - As short as possible to increase the chance of backfilling



Frequently Asked Questions

- **There are free nodes available, why my job is still waiting and not running?**

- **Why my job is not get accelerated when running on cluster?**
  - Is your job utilizing the parallel resource on the cluster?
  - Does you job have lots of I/O tasks?
  - See next section...

Compilers

- **Serial compilers**

Language	Linux cluster			AIX clusters
	Intel	PGI	GNU	XL
Fortran	ifort	pgf77, pgf90	gfortran	xlf, xlf90
C	icc	pgcc	gcc	xlc
C++	icpc	pgCC	g++	xlC

- **Parallel compilers**

Language	Linux clusters	AIX clusters
Fortran	mpif77, mpif90	mpxlf, mpxlf90
C	mpicc	mpcc
C++	mpiCC	mpCC

Example compiling serial code

- **icc hello_cpu_elapsed.c**
- **gfortran test_hello2.f90**

- **List symbols for executables:**

  nm - list symbols from object files
- **Example:**

```
[fchen14@mike2 hello]$ nm ./a.out | grep intel
000000000060eb60 B __intel_cpu_indicator

[fchen14@mike2 hello]$ nm ./a.out | grep gfortran
                 U _gfortran_set_args@@GFORTRAN_1.0
```

CPU time vs Elapsed time

- **CPU time (or process time):**
  - The amount of time for which a central processing unit (CPU) was used for processing instructions of a computer program or operating system, as opposed to, for example, waiting for input/output (I/O) operations or entering low-power (idle) mode.
- **Elapsed real time (or simply real time, or wall clock time)**
  - The time taken from the start of a computer program until the end as measured by an ordinary clock. Elapsed real time includes I/O time and all other types of waits incurred by the program.
- **If a program uses parallel processing, total CPU time for that program would be more than its elapsed real time.**
  - (Total CPU time)/(Number of CPUs) would be same as elapsed real time if work load is evenly distributed on each CPU and no wait is involved for I/O or other resources.

Compiling and Analyzing C serial program

```c
#include <stdio.h>
#include <time.h>
int main(char *argc, char **argv) {
    double s=0.0;
    // fundamental arithmetic type representing clock tick counts.
    clock_t start, end;
    int i;
    start = clock();
    for (i=0;i<1000000000;i++)
        s+=i*2.0; // doing some floating point operations
    end = clock();
    double time_elapsed_in_seconds = (end - start)/(double)CLOCKS_PER_SEC;
    printf("time_elapsed_in_sec: %e\n", time_elapsed_in_seconds);
    start = clock();
    system ("sleep 5"); // just sleep, does this accumulate CPU time?
    end = clock();
    time_elapsed_in_seconds = (end - start)/(double)CLOCKS_PER_SEC;
    printf("time_elapsed_in_sec: %e\n", time_elapsed_in_seconds);
    return 0;
}
```

Watch the actual cpu time

```
[fchen14@mike429 serial]$ gcc hello_cpu_elapsed.c
[fchen14@mike429 serial]$ time ./a.out
cputime_in_sec: 2.740000e+00
cputime_in_sec: 0.000000e+00

real    0m7.782s
user    0m2.750s
sys     0m0.005s
```

Example compiling threaded OpenMP code

- **Compiling OpenMP code often requires the** `openmp` **compiler flags, it varies with different compiler**
- **Environment Variable** `OMP_NUM_THREADS` **sets the number of threads**
- **Examples:**

```
[fchen14@mike2 src]$ gcc -fopenmp hello_openmp.c
[fchen14@mike2 src]$ ifort -openmp hello_openmp.f90
```

Compiler	Compiler Options	Default behavior for # of threads (OMP_NUM_THREADS not set)
GNU (gcc, g++, gfortran)	-fopenmp	as many threads as available cores
Intel (icc ifort)	-openmp	as many threads as available cores
Portland Group (pgcc,pgCC,pgf77,pgf90)	-mp	one thread

Sample OpenMP C code

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[]) {
    int nthreads, tid;
    /* Fork a team of threads with their own copies of variables */
#pragma omp parallel private(nthreads, tid)
    {
        /* Obtain thread number */
        tid = omp_get_thread_num();
        printf("Hello World from thread = %d\n", tid);
        /* Only master thread does this */
        if (tid == 0) {
            nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n", nthreads);
        }
    } /* All threads join master thread and disband */
}
```

Sample OpenMP Fortran code

```
program hello

integer nthreads,tid,omp_get_num_threads,omp_get_thread_num
! fork a team of threads giving them their own copies of variables
!$omp parallel private(nthreads, tid)
! obtain thread number
tid = omp_get_thread_num()
print *, 'hello world from thread = ', tid
! only master thread does this
if (tid .eq. 0) then
    nthreads = omp_get_num_threads()
    print *, 'number of threads = ', nthreads
end if
! all threads join master thread and disband
!$omp end parallel
end
```

Analyzing a parallel (OpenMP) program

- **What will be the CPU time and elapsed time for the following code segment:**

  see: /home/fchen14/userenv/src/openmp/hello_openmp_cpu_elapse.c

```
// fundamental arithmetic type representing clock tick counts.
clock_t start, end;
struct timeval r_start, r_end;
int i;
gettimeofday(&r_start, NULL);
start = clock();
#pragma omp parallel for // spawn the openmp threads
for (i=0;i<N;i++) a = i*2.0; // doing some floating point operations
end = clock();
gettimeofday(&r_end, NULL);
double cputime_elapsed_in_seconds = (end - start)/(double)CLOCKS_PER_SEC;
double realtime_elapsed_in_seconds = ((r_end.tv_sec * 1000000 +
r_end.tv_usec) - (r_start.tv_sec * 1000000 + r_start.tv_usec))/1000000.0;
```

Available MPI libraries on LONI & HPC

	Name	MPI Library				Default serial compiler
Cluster Resource		Mvapich	Mvapich2	OpenMPI	MPICH	
LONI	Eric	0.98, 1.1	1.4, 1.6, 1.8.1	1.3.4	X	Intel 11.1
	Other	0.98, 1.1	1.4, 1.6	1.3.4	X	Intel 11.1
LSU	SuperMikeII	X	X	1.6.2 1.6.3 1.6.5	3.0.2	Intel 13.0.0
	Philip	X	X	1.4.3, 1.6.1	1.2.7, 1.3.2, 1.4.1	Intel 11.1
	Pandora	X	X	X	X	AIX
	SuperMIC					

MPI Compilers (1)

Language	Linux clusters	AIX clusters
Fortran	mpif77, mpif90	mpxlf, mpxlf90
C	mpicc	mpcc
C++	mpiCC	mpCC

mpif90 hello.f90

mpicc hello.c

mpicxx hello.cpp

MPI Compilers (2)

- **These MPI compilers are actually wrappers**
  - They still use the compilers we've seen on the previous slide
    - Intel, PGI or GNU
  - They take care of everything we need to build MPI codes
    - Head files, libraries etc.
  - What they actually do can be reveal by the `-show` option
- **It's extremely important that you compile and run your code with the same version of MPI!**
  - Use the default version if possible

Compiling a MPI C program

- **Compiling Hello world in C version:**
  - mpicc hello_mpi.c

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv) {
    int name_len, world_size, world_rank;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    //Initialize the MPI environment
    MPI_Init(NULL, NULL);
    // Get the number and rank of processes
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    // Get the name of the processor
    MPI_Get_processor_name(processor_name, &name_len);
    // Print off a hello world message
    printf("Iam from processor %s, rank %d out of %d processors\n",
            processor_name, world_rank, world_size);
    // Finalize the MPI environment.
    MPI_Finalize();
}
```

Compiling a MPI Fortran program

- **Compiling Hello world in Fortran:**
  - mpif90 hellp_mpi.f90

```
program hello_mpi
    include 'mpif.h'
    !use mpi
    character 10 name
    ! Initialize the MPI library:
    call MPI_Init(ierr)
    ! Get size and rank
    call MPI_Comm_Size(MPI_COMM_WORLD, numtasks, ierr)
    call MPI_Comm_Rank(MPI_COMM_WORLD, rank, ierr)
    ! print date
    if (nrank == 0) then
        write( , )'System date'
        call system('date')
    endif
    call MPI_Barrier(MPI_COMM_WORLD, ierr)
    ! print rank
    call MPI_Get_Processor_Name(name, len, ierr)
    write( , )"I am ", nrank, "of", numtasks, "on ", name
    ! Tell the MPI library to release all resources it is using:
    call MPI_Finalize(ierr)
end program hello_mpi
```

Notes for compiling a MPI program (1)

- **Always verify what compiler/library is being used:**

```
$ mpicc -show
icc -I/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/include -
L/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib -lmpi -ldl -lm -
Wl,--export-dynamic -lrt -lnsl -libverbs -libumad -lpthread -lutil

$ mpif90 -show
ifort -I/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/include -
I/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib -
L/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib -lmpi_f90 -
lmpi_f77 -lmpi -ldl -lm -Wl,--export-dynamic -lrt -lnsl -libverbs -
libumad -lpthread -lutil
```

Notes for compiling a MPI program (2)

- **Always verify what library is being used: Before and after:**

```
$ ldd a.out #ldd - print shared library dependencies
        linux-vdso.so.1 =>  (0x00007fff907ff000)
        libmpi_f90.so.1 => /usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib/libmpi_f90.so.1 (0x00002b9ae577e000)
        libmpi_f77.so.1 => /usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib/libmpi_f77.so.1 (0x00002b9ae5982000)
        libmpi.so.1 => /usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib/libmpi.so.1 (0x00002b9ae5bb9000)
...
        libpthread.so.0 => /lib64/libpthread.so.0 (0x0000003b21800000)
...
        libifport.so.5 => /usr/local/compilers/Intel/composer_xe_2013.0.079/compiler/lib/intel64/libifport.so.5 (0x00002b9ae61ee000)
        libifcore.so.5 => /usr/local/compilers/Intel/composer_xe_2013.0.079/compiler/lib/intel64/libifcore.so.5 (0x00002b9ae641d000)
```

Running and Analyzing MPI program

- **Make sure you are running your jobs on the correct nodes**
- **Important if you want to run less processes than ppn**
- **Understand the usage of $PBS_NODEFILE**

```
[fchen14@mike2 ~]$ qsub -I -X -l nodes=2:ppn=16 -l walltime=01:00:00 -q gpu
...
[fchen14@mike429 ~]$ echo $PBS_NODEFILE
/var/spool/torque/aux//236660.mike3
[fchen14@mike429 ~]$ cat $PBS_NODEFILE
mike429
...          # 16 repeats of mike429
mike429
mike430
...          # 16 repeats of mike430
mike430
[fchen14@mike429 hybrid]$ cat $PBS_NODEFILE| uniq > hosts
[fchen14@mike429 hybrid]$ cat hosts
mike429
mike430
```

Running and Analyzing MPI program

```
[fchen14@mike315 mpi]$ mpicc hello_mpi.c
[fchen14@mike315 mpi]$ mpirun -np 32 -hostfile $PBSNODEFILE ./a.out
Iam from processor mike315, rank 1 out of 32 processors
Iam from processor mike315, rank 6 out of 32 processors
Iam from processor mike315, rank 9 out of 32 processors
Iam from processor mike315, rank 12 out of 32 processors
Iam from processor mike315, rank 0 out of 32 processors
Iam from processor mike315, rank 2 out of 32 processors
Iam from processor mike315, rank 3 out of 32 processors
Iam from processor mike315, rank 7 out of 32 processors
Iam from processor mike315, rank 10 out of 32 processors
Iam from processor mike315, rank 5 out of 32 processors
Iam from processor mike315, rank 13 out of 32 processors
Iam from processor mike315, rank 4 out of 32 processors
Iam from processor mike315, rank 8 out of 32 processors
Iam from processor mike334, rank 17 out of 32 processors
Iam from processor mike315, rank 11 out of 32 processors
Iam from processor mike315, rank 14 out of 32 processors
Iam from processor mike315, rank 15 out of 32 processors
Iam from processor mike334, rank 18 out of 32 processors
```

Compiling hybrid (MPI+OpenMP) program

- **See** `/home/fchen14/userenv/src/hybrid/hello_hybrid.c` **for complete source**
- **Use command:**
  - `$ mpicc -openmp hello_hybrid.c`

```
#pragma omp parallel default(shared) private(itd, np)
    {
        gtd = omp_get_num_threads(); //get total num of threads in a process
        itd = omp_get_thread_num();  // get thread id
        gid = nrank*gtd + itd;       // global id
        printf("Gid %d from thd %d out of %d from process %d out of %d on %s\n",
               gid, itd, gtd, nrank, numprocs, processor_name);
        if (nrank==0 && itd==0)
        {
            // system("pstree -ap -u $USER");
            system("for f in `cat $PBS_NODEFILE|uniq`; do ssh $f pstree -ap -u
$USER; done;");
            system("sleep 10");
        }
    }
```

Analyzing a hybrid program

```
[fchen14@mike315 hybrid]$ export OMP_NUM_THREADS=4
[fchen14@mike315 hybrid]$ mpirun -np 2 -x OMP_NUM_THREADS ./a.out
Gid 0 from thread 0 out of 4 from process 0 out of 2 on mike315
Gid 2 from thread 2 out of 4 from process 0 out of 2 on mike315
Gid 1 from thread 1 out of 4 from process 0 out of 2 on mike315
Gid 3 from thread 3 out of 4 from process 0 out of 2 on mike315
Gid 4 from thread 0 out of 4 from process 1 out of 2 on mike315
Gid 6 from thread 2 out of 4 from process 1 out of 2 on mike315
Gid 7 from thread 3 out of 4 from process 1 out of 2 on mike315
Gid 5 from thread 1 out of 4 from process 1 out of 2 on mike315
bash,108067
  |-mpirun,110651 -np 2 -x OMP_NUM_THREADS ./a.out
  |   |-a.out,110652
  |   |   |-sh,110666 -c ...
  |   |   |   `-ssh,110670 mike315 pstree -ap -u fchen14
  |   |   |-{a.out},110654
  |   |   |-{a.out},110656
  |   |   |-{a.out},110662
  |   |   |-{a.out},110663
  |   |   |-{a.out},110664
  |   |   `-{a.out},110665
  |
```

Exercise

- **Submit a small job to run "sleep 180"and "print PBS variables"**
  - Create a script to submit a 5 min job and print from within the job script PBS variables $PBS_NODEFILE, $PBS_WORKDIR. Also run "sleep 180" to give you a few minutes to verify status.
  - Once the job is running, find out the Mother Superior node and other slave nodes assigned to your job using qstat.
  - Log into MS node and verify that your job is running and find your temporary output file
  - Modify your script to print hello from each of your assigned nodes
- **Run a shell script using mpirun to print process id of shell**

Future Trainings

- **Next week training: Distributed Job Execution**
  - Wednesdays 10am, Sep 24, Frey 307 CSC
- **Programming/Parallel Programming workshops**
  - Usually in summer
- **Keep an eye on our webpage: www.hpc.lsu.edu**