

Batch job submission



- All production runs must be submitted to the batch queuing system.
- The batch scheduler on HPCx is LoadLeveler

- Submit a job with:

```
llsubmit <commandfile>
```

Command file for MPI job (head)

```
#!/bin/ksh
#
#@ job_type = parallel
#@ cpus = 256
#@ node_usage = not_shared
#@ network.MPI = csss,shared,US
#@ bulkxfer = yes
#
#@ wall_clock_limit = 00:20:00
#@ account_no = z001
#
#@ job_name = myrun
#@ output = $(job_name).$(schedd_host).$(jobid).out
#@ error = $(job_name).$(schedd_host).$(jobid).err
#@ notification = never
#@ queue
```

- See next page for following script part ...

Command file MPI (script-part)

- After the header follows a UNIX script
- Here we use **ksh**-syntax, matching the first line of our header

```
export MP_SHARED_MEMORY=yes
export MP_EAGER_LIMIT=65536
export MEMORY_AFFINITY=MCM
export MP_TASK_AFFINITY=MCM
#
poe ./my_executable
```

```
#@ shell = /bin/ksh
```

- Tell the system to use ksh for the LoadLeveler script.
- Put your choice here if you prefer different
- Recommended
- If not specified, your default shell gets used

```
#@ job_type = parallel
```

- Tell the system this is a parallel job.
Mandatory.

#@ cpus = 256

- Request the number of processors to run the job on. Mandatory.
- You can request a number which is not a multiple of 16, but you will be allocated, and charged for a whole number of nodes.
- E.g. if you request 59 CPUs you will be charged for $4 \times 16 = 64$ CPUs.

#@ node_usage = not_shared

- Says that you will not allow other jobs to share the LPARs you have been allocated.
Mandatory.

```
#@ network.MPI = csss,shared,US
```

- MPI can either use US (user space) or IP protocols. US is normally faster.
- We recommend you use US in MPI scripts
- However, US communications do not function on a single LPAR
- If you use a single LPAR (<= 16 CPUs) you may see an error similar to:

```
US is not valid for single nodes. Delete  
"<whatever #@network is given>" or use IP
```
- If this happens leave out the `#@ network` line from your LL script

#@ bulkxfer = yes

- Switches on Remote Direct Memory Access (RDMA) to boost the maximum bandwidth available via the switch network.
- This should benefit most codes
- However, a few applications might see a decrease in performance
- We recommend comparing the performance of your application with and without this line in the script.

```
#@ wall_clock_limit = 00:20:00
```

- Specifies the wall clock time limit
- Format is `mm:ss` or `hh:mm:ss`
- Mandatory.

#@ account_no = z001

- Specify the budget for the job to be charged against. Mandatory.
- You can find out which budgets you have access to and the time remaining in them with the **budgets** command.
- Parallel jobs are charged by (wall clock time) x (number of nodes)
- Serial jobs are charged by CPU time.

```
#@ job_name = fred
```

- Specify a name for the job
- Optional, but useful for identifying output files.

Select output and error file names

```
#@ output = $(job_name).$(schedd_host).$(jobid).out
```

```
#@ error = $(job_name).$(schedd_host).$(jobid).err
```

- Specify files for job output and error
- **job_name** is the name we just chose...
- **schedd_host** is the name of the host which scheduled the job.
- **jobid** is a number
- **schedd_host** and **jobid** together uniquely identify job. Don't use **\$(host)**
- Files are placed in directory where job was submitted from. Mandatory.

```
#@ notification = never
```

- Suppress notification by email.
- Optional.

To enable email notification replace this with

```
#@ notify_user = user@another.machine.somewhere.else  
#@ notification = complete
```

#@ queue

- Tell LoadLeveler to queue the job.
- Vital: don't forget this!

```
export MP_SHARED_MEMORY=yes
```

- Make sure that MPI messages between processes within a node do not use the switch.

```
export MP_EAGER_LIMIT=65536
```

- Set the maximum size for buffered messages as high as possible.
- Set this to 0 if you are developing code, to catch possible deadlocks.

```
export MEMORY_AFFINITY=MCM
```

```
export MP_TASK_AFFINITY=MCM
```

- Ask the OS to try to allocate memory on the MCM where the process is running and keep the task on this MCM

```
poe ./fred
```

- Finally (phew!) we get to run the executable.
- **poe** (Parallel Operating Environment) is the MPI job launcher.
- Note: don't need to specify number of processes here.

#@ tasks_per_node = 12

- Request the number of MPI processes per node.
- The number of CPUs requested should be a multiple of this.
- Useful for large memory jobs: each task gets 27 Gb divided by **tasks_per_node**
- Still charged for the whole node.

- A class is what would normally be called a queue in most other batch systems!
- Available classes are:

serial_*T* and **par*N*_*T***

where *N* is the number of CPUs and can have values 32,64,128,256,512,1024.

and *T* is the wall clock time in hours and can have values 1,3,6,9,12.

- LoadLeveler chooses the appropriate class for the job, based on the job type, number of nodes and wall clock time requested.

- At present the batch system is configured as follows:

Capability: 64 nodes (1024 procs) for production runs on up to 128-1024 processors

Capacity: 26 nodes (416 procs) for capacity runs on \leq 128 processors

Interactive: 2 nodes (32 processors) for interactive sessions

Development: 12 nodes (192 processors) available to specific projects

- Other nodes support login and I/O.

- Classes do not have a priority structure.
- Backfilling is in use.
 - LoadLeveler looks ahead to find a slot when large jobs can run.
 - It will not run smaller jobs now, if they will run beyond the start of the slot. Will run small jobs if run time is low enough.
 - Sometimes your job will not run, even though it appears that there are enough free processors.
 - Ensures that large jobs don't get frozen out, but has some impact on throughput.

- Job status can be monitored with **llq**

Id	Owner	Submitted	ST	PRI	Class	Running On
11f401.27287.0	fritz	4/3 20:17	R	50	par512_6	12f413
11f401.27180.0	xxc	4/3 13:28	R	50	par128_12	11f406
11f401.27377.0	sandberg	4/4 14:28	R	50	par256_6	17f411
11f402.28556.0	dlrojo	4/4 16:08	R	50	par256_6	12f407
11f401.27223.0	ttql2	4/3 15:40	R	50	par128_12	12f405
11f401.27269.0	nhine	4/3 18:09	R	50	par64_6	13f403
11f402.28436.0	ugdario	4/3 21:12	R	50	par64_6	17f403
11f402.28597.0	mjw	4/4 18:36	R	50	parn128_1	15f414
11f401.27400.0	ucsstack	4/4 15:53	R	50	parn32_12	17f414
11f402.28547.0	ucsstack	4/4 15:54	R	50	parn32_12	16f450
11f402.28551.0	ganzenmg	4/4 16:00	R	50	serial_12	11f402
11f401.27427.0	ganzenmg	4/4 17:03	R	50	serial_12	11f402
11f401.27312.0	alexr	4/4 09:04	R	50	par32_3	18f403
11f401.27448.0	pja	4/4 18:14	R	50	inter32_1	12f403
11f402.28595.0	atilocca	4/4 18:20	R	50	inter32_1	11f403
11f401.27323.0	ddundas	4/4 10:10	R	50	par32_6	12f406
11f401.27450.0	lcs	4/4 18:36	R	50	serial_12	11f402
11f401.27368.0	ibmem	4/4 13:31	I	50	par1024_1	
11f402.28488.0	mehdi	4/4 11:18	I	50	par512_12	
11f401.27387.0	ucjames	4/4 15:24	I	50	par1024_6	
11f402.28513.0	ibmem	4/4 13:28	I	50	par512_1	

- **Id** is host.jobid.stepid
- Status: R:running, I:idle, C:completed, ST: starting, H: held, NQ: not queued,....
- Priority and Running On: ignore these!
- All the info you could ever want: **llq -l**

- To cancel a waiting or running job:

```
llcancel host.jobid
```

or

```
llcancel host.jobid.stepid
```

- Check that the job has gone with `llq`

- You can see the status of all the nodes with **llstat**
- Another, probably less useful, view can be obtained using **llstatus**
- All the information you could ever want: **llstatus -l**
- If you prefer you can monitor and cancel jobs and view system status through a GUI: **xloadl**

HPC-X (phase2A) batch system LPAR allocation status at: 04/04/06 18:47:20

LPARs in use: 103 CPUs in use: 1648 Jobs running: 14

```
11f402 (serial)      13f403 [nhine    ]      14f403 [uqdarior ]      15f403 [nhine    ]
16f403 [uqdarior ]      17f403 [uqdarior ]      18f403 [alexr    ]      11f403 (inter)
12f403 (inter)      15f404 <xxc     >      16f404 <xxc     >      17f404 <ttql2   >
18f404 <ttql2   >      11f404 [nhine    ]      12f404 [uqdarior ]      13f404 [alexr    ]
14f404 [nhine    ]      11f405 <xxc     >      12f405 <ttql2   >      13f405 <xxc     >
14f405 <ttql2   >      15f405 <ttql2   >      16f405 <ttql2   >      17f405 <xxc     >
18f405 <xxc     >      11f406 <xxc     >      12f406 <ddundas >      13f406 <ttql2   >
14f406 <ttql2   >      15f406 dlrojo              16f406 sandberg      17f406 fritz
```

...

	LPARs				
	Total	Alloc	Idle		

Batch parallel (capability):	64	61	3		
Batch parallel (capacity (S)):	10	10	0	[]
Batch parallel (capacity (L)):	16	16	0	<	>
Batch parallel (development):	12	12	0	+	+
Batch parallel (test):	0	0	0	*	*
Batch parallel (course):	0	0	0	{	}
Interactive shared parallel:	2	--	--	(inter)	
Batch serial:	1	--	--	(serial)	
Unavailable:	0	--	--		

- You can see which classes are available with **llclass**

```
fiona@11f401# llclass
```

Name	MaxJobCPU d+hh:mm:ss	MaxProcCPU d+hh:mm:ss	Free Slots	Max Slots	Description
parn192_1	undefined	undefined	192	384	192 CPUs (12 nodes), 1 hour development class
parn192_20m	undefined	undefined	192	384	192 CPUs (12 nodes), 20 minute development class
parn128_1	undefined	undefined	192	384	128 CPUs (8 nodes), 1 hour development class
parn128_20m	undefined	undefined	192	384	128 CPUs (8 nodes), 20 minute development class
parn64_1	undefined	undefined	192	384	64 CPUs (4 nodes), 1 hour development class
parn64_20m	undefined	undefined	192	384	64 CPUs (4 nodes), 20 minute development class
parn32_12	undefined	undefined	192	384	32 CPUs (2 nodes), 12 hour development class
parn32_6	undefined	undefined	192	384	32 CPUs (2 nodes), 6 hour development class
parn32_1	undefined	undefined	192	384	32 CPUs (2 nodes), 1 hour development class
parn32_20m	undefined	undefined	192	384	32 CPUs (2 nodes), 20 minute development class
parn16_12	undefined	undefined	192	384	16 CPUs (1 node), 12 hour development class
parn16_6	undefined	undefined	192	384	16 CPUs (1 node), 6 hour development class
parn16_1	undefined	undefined	192	384	16 CPUs (1 node), 1 hour development class
parn16_20m	undefined	undefined	192	384	16 CPUs (1 node), 20 minute development class
par1024_12	undefined	undefined	1072	2048	1024 CPUs (64 nodes), 12 hour class
par1024_6	undefined	undefined	1072	2048	1024 CPUs (64 nodes), 6 hour class
par1024_3	undefined	undefined	1072	2048	1024 CPUs (64 nodes), 3 hour class
par1024_1	undefined	undefined	1072	2048	1024 CPUs (64 nodes), 1 hour class
par512_12	undefined	undefined	1072	2048	512 CPUs (32 nodes), 12 hour class
par256_12	undefined	undefined	1072	2048	256 CPUs (16 nodes), 12 hour class
par512_6	undefined	undefined	1072	2048	512 CPUs (32 nodes), 6 hour class
par256_6	undefined	undefined	1072	2048	256 CPUs (16 nodes), 6 hour class

- All the info you could ever want: **llclass -l**

A command file for a serial job

```
#@ job_type = serial
#@ node_usage = shared
#@ wall_clock_limit = 00:20:00
#@ account_no = z001
#@ job_name = my_serial_run
#@ output = $(job_name).$(schedd_host).$(jobid).out
#@ error = $(job_name).$(schedd_host).$(jobid).err
#@ notification = never
#
#@ queue
#
./my_serial_executable
```

Command file for interactive job

```
#@ job_type = parallel
#@ cpus = 5
#@ node_usage = shared
#@ wall_clock_limit = 00:10:00
#@ account_no = z001
#@ job_name = hello
#@ notification = never
#
#@ class = inter32_1
#
#@ queue
#
```

Note: NO poe command here!



- Interactive queue is meant for debugging and visualisation.
 - Do NOT use it for production jobs!
1. Create command file as shown.
 2. Set environment variables on the command line.
 3. Submit job from command line with

```
poe ./my_executable -llfile ./my_script.ll
```

Command file for OpenMP jobs

```
#@ job_type = parallel
#@ cpus = 1
#@ node_usage = not_shared
#@ wall_clock_limit = 2:00:00
#@ account_no = z001
#@ job_name = fred
#@ output = $(job_name).$(schedd_host).$(jobid).out
#@ error = $(job_name).$(schedd_host).$(jobid).err
#@ notification = never
#@ queue
export OMP_NUM_THREADS=4
export XLSMPOPTS=spins=0:yields=0
./fred
```

One process

Four threads

Forces threads to
adopt busy-waiting

Command file for OpenMP+MPI jobs

```
#@ job_type = parallel
#@ cpus = 4
#@ tasks_per_node = 2
#@ node_usage = not_shared
#@ network.MPI = csss,shared,US
#@ wall_clock_limit = 2:00:00
#@ account_no = z001
#@ job_name = fred
#@ output = $(job_name).$(schedd_host).$(jobid).out
#@ error = $(job_name).$(schedd_host).$(jobid).err
#@ notification = never
#@ queue

export OMP_NUM_THREADS=8
export XLSMPOPTS=spins=0:yields=0
poe ./fred
```

Four MPI tasks
Two tasks per node
8 threads per task
32 processors in all

- The HPCx website contains records of all jobs run and the resources consumed.
- Log on to the secure website, and click on “Go to report generator”
- Choose the period you are interested in (default: last month) and click on “Generate report”

- All the disk space on the system is attached to a set of I/O nodes, and is managed by GPFS (General Parallel File System).
- Local disk space on the compute nodes is *not* user accessible.
- Total storage is ~100Tb, of which ~34Tb is on disk, the rest on tape.
 - files can be archived to tape: see web site.
- Your disk space comes in two parts: home and work

- home is your login directory (e.g. `/hpcx/home/z001/z001/john`). This is backed up.
- work is additional space for working files. (e.g. `/hpcx/work/z001/z001/john`). This is not backed up.
- Quotas are per project: now 80Gb of home and 400Gb of work. PIs can set user quotas.

`/usr/lpp/mmfs/bin/mmlsquota (-g <group>)`
shows user and group quotas

- IBM RedBook: Workload management with LoadLeveler

<http://www.redbooks.ibm.com/SG246038.html>

- LoadLeveler documents

http://www-1.ibm.com/servers/eserver/pseries/library/sp_books/loadleveler.html

See "User Tasks" chapter in "Using and Administering" book.