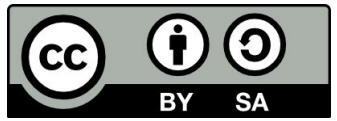
Computación de Altas Prestaciones

Gestión de recursos distribuidos

José Luis Risco Martín
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid



Índice

Need for Resource Management

- Protecting significant infrastructure investment
  - Supercomputing hardware
  - Hosting data center
  - Power delivery (cabling, UPSs, generators, flywheels)
  - Cooling infrastructure (HVAC units, heat exchangers, pipes for water cooling, etc.)
- Cost of ownership
  - Electricity (about $1M/1MW per year)
  - Support contracts
  - Maintenance crew (system administrators, technicians)
- Getting the most for your $
  - Efficient workload scheduling
  - Maximizing hardware resource utilization
  - Compute job prioritization

Types of Managed Resources

- Compute nodes
- Processing cores
- Interconnect
- Permanent storage and I/O
- Accelerators

Compute Nodes

- Each node provides a fixed compute and storage footprint
- Constitutes a fundamental resource allocation unit for a compute job
- Increasing the number of nodes enables straightforward application scaling
- Resources may vary depending on node type
  - CPU type and clock frequency
  - Main memory capacity
  - Interconnect bandwidth and latency
  - Optionally: secondary storage speed and capacity

Processing Cores

- Provide local parallelism:
  - Multiple cores per processor
  - Possibly multiple processors (sockets) per node
- Sharing of processing elements between different applications:
  - Depends on application type and usage context
  - Single compute core is a typical allocation unit
  - Shared mode enables better resource utilization
    - Parameter space sweeps
    - Large number of unrelated small jobs
  - Exclusive mode allocates full node to a single application
    - Benchmarking (minimizes intrusions from unrelated jobs)
    - Applications that demand large memory capacity
    - Applications generating large number of I/O requests to local storage
    - Bandwidth-intensive distributed jobs
    - Affinity to other hardware components of the node (e.g., PCI-Express)

Interconnect

- Typically one type per system
  - 10G Ethernet
  - InfiniBand
  - 1G Ethernet (GigE)
- Installations with multiple networks possible
  - To permit experiments with different bandwidth and latency profiles
  - To accommodate bandwidth-insensitive applications along with network “hogs”
  - To enable high-performance mechanisms available in certain implementations (such as RDMA)
  - To utilize channel bonding for increased communication bandwidth

Permanent Storage

- Typically implemented as a shared file system
  - Exported system-wide, thus accessible from every node
  - High capacity to accommodate all users
  - Optimized for performance and reliability (e.g., RAID)
- Multiple file systems may be provided for different usage scenarios
  - Users' home directories, often quota-limited
  - Scratch space for larger data volumes, but with limited retention (e.g. automatic deletion after one week of last access)
  - Archive storage with very large capacity, long term retention, but potentially limited access bandwidth
  - Local file systems
- Localized storage may provide better aggregate data bandwidth
  - Example 1: disk drives in each node
  - Example 2: burst buffers in Cray machines (SSD pools shared by groups of nodes)
  - But: limited data access options (the user must log into the same machine or group of nodes where dataset was saved)

Accelerators

- Special purpose processors
  - Heterogeneous (different hardware class to main CPU)
  - May be application specific
  - Expose large number of fine-grain processing elements
  - Consume less energy and time to accomplish certain tasks compared to the host processor
  - Use different compilers and libraries to generate executables
  - Knowledge of internal organization usually a must to develop efficient applications
  - Often require explicit code and data offload to processing unit
- Commonly used variants
  - General Purpose Graphics Processing Unit (GPGPU)
  - Intel Xeon Phi
  - Field Programmable Gate Array (FPGA)
- System hardware may not be uniformly populated with accelerators
  - Some nodes may be equipped with several accelerators while the others contain none
  - Resource managers should support allocation of accelerated nodes when required by application

Compute Job

- A self-contained work unit that may need specific input data and produces a result in the course of its execution (output)
  - Input may be typed by the user or a file
  - Output could be a file, a printout on the console or graphics displayed on screen
- Primary modes of job processing
  - Interactive: requires interaction with user
  - Batch: resource requirements, job description, and inputs are fully specified at job submission time
- Batch processing accounts for most of machine's computational throughput
- May be subdivided into smaller *steps* or *tasks*
  - Data staging
  - Data pre- and post-processing
  - Visualization
  - Processing application pipelines

Job Queue

- Defines order in which jobs are executed on the machine
  - Typically FIFO ("first-in, first-out")
  - Job schedulers may deviate from the prearranged order to improve utilization/throughput or due to operator's override
- Groups jobs targeting the same set of resources
  - Most systems have multiple job queues, for example
    - Production
      - ✓ Job size
      - ✓ Job duration
    - Debug
    - Interactive
    - Managing specific resources, such as large memory nodes or accelerators

Job Scheduling

- Critical to achieving good utilization of the machine
- Must accommodate potentially hundreds or thousands of waiting jobs and a significant number of concurrently executing jobs
- Parameters affecting job scheduling
  - Resource availability
  - Job priority
  - Resources allocated to the user
  - Maximum number of jobs
  - Requested execution time
  - Elapsed execution time
  - Job dependencies and prerequisites
  - Occurrence of specified events
  - Operator availability
  - Software license availability

Common Resource Managers

- Simple Linux Utility for Resource Management (SLURM)
- Portable Batch System (PBS)
- OpenLava based on the Load Sharing Facility (LFS)
- Moab Cluster Suite
- Tivoli Workload Scheduler LoadLeveler
- Univa Grid Engine
- HTCondor
- OAR
- Hadoop Yet Another Resource Negotiator (YARN)

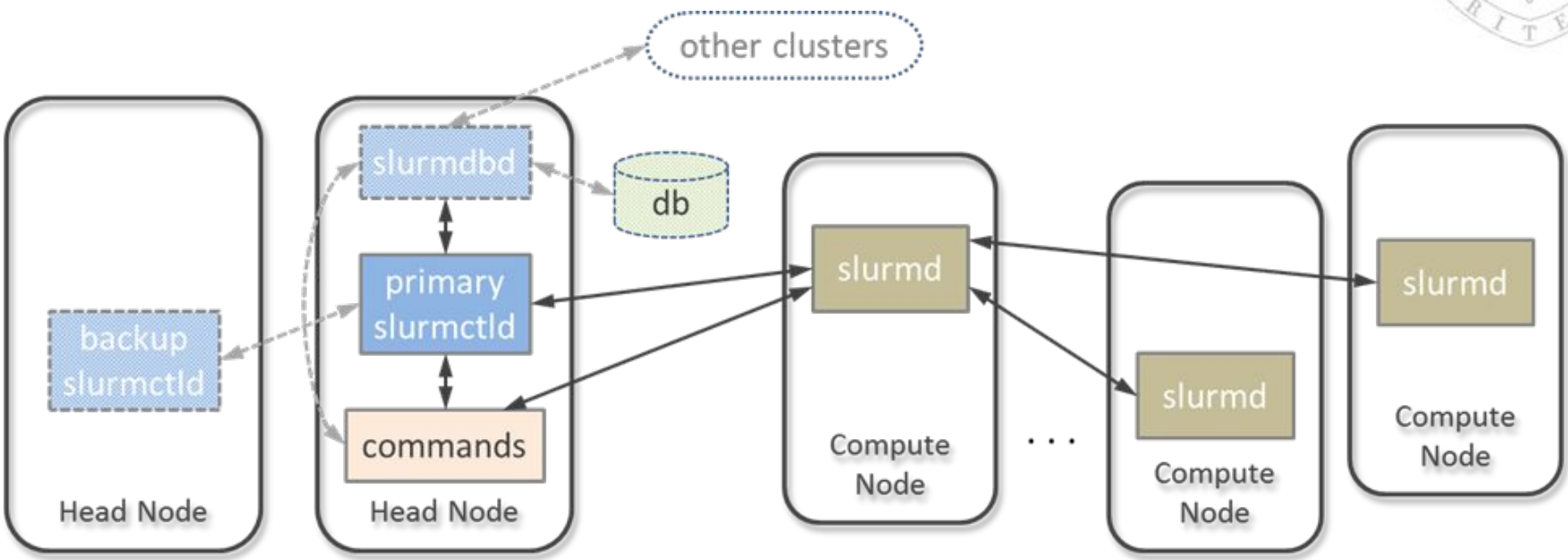
SLURM Introduction

- One of the most popular open-source managers
- Originated in early 2000s at Lawrence Livermore National Laboratory (Morris Jette)
- Currently close to 200 contributors (such as SchedMD LLC, HP, Bull, Cray, ORNL, LANL, Intel, Nvidia, and others)
- Utilized in about 60% of machines in the TOP500 list (mid-2014)

SLURM Features

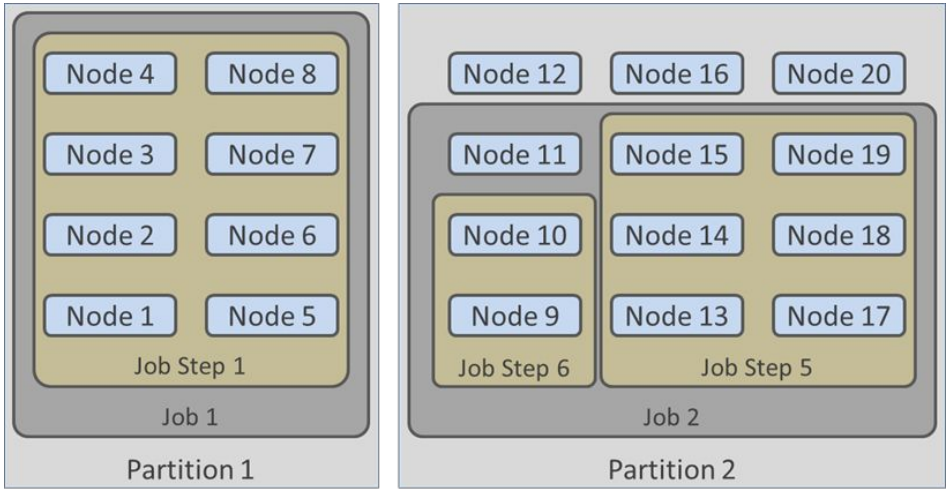
- Complex configuration options
  - Different interconnect types
  - Scheduling algorithms
  - MPI implementations
  - Job accounting
- Scalable to over 10 million cores (TaihuLight)
- Manages up to a thousand job submissions and 500 job executions per second
- Multiple strategies for power optimization
- Increased reliability through multiple daemons
- Network topology aware
- Decisions influenced by node architecture (NUMA, core distribution, thread affinities)
- Supports expanding and shrinking job's resource footprint during its execution
- A number of scheduling algorithms with a broad range of control parameters
- Accelerator support
- Extensible via plugins in C and Lua
- Optional support for job profile database

SLURM Architecture



Workload Organization

- Partitions (queues)
  - Group execution resources
  - Uniquely named
  - May overlap
- Job steps may utilize all or a fraction of nodes allocated to a parent job
- Job arrays are a means to manage a large number of similar jobs (e.g., for throughput computing)



SLURM Scheduling

- Event-triggered scheduling as default
  - Considers only a limited number of jobs at the head of the queue
  - Fast and low overhead
- Backfill scheduler
  - Starts lower priority jobs if that will not affect the expected start time of higher priority jobs
  - Fair treatment of lower priority jobs that require large amount of resources
  - Improves overall utilization
- Gang scheduling
  - For multiple similar jobs sharing the same set of resources
- Preemption
  - Stopping lower priority jobs to launch higher priority jobs
- Generic Resources (GRES) driven scheduling
  - Based on hardware device availability (e.g., accelerators)
- Elastic computing
  - Permits growing or shrinking of the overall resource footprint (both provided by the system or requested by the job)
- High throughput computing

srun

- Starts a parallel job or job steps
- Performs resource allocation for the job if such has not been performed yet
- Syntax: srun [<options>] <executable> [<arguments>]
- Frequently used options:

  -t or --time=<[hours:]minutes:seconds>
  -p or --partition=<name>
  -N or --nodes=<min_nodes>
  -n or --ntasks=<number_of_tasks>
  -c or --cpus-per_task=<number_of_cpus>
  --mem=<megabytes>
  --exclusive
  -s or --oversubscribe

salloc

- Performs resource allocation and runs the command specified by the user
- Accepts the same options as *srun*
- Syntax: salloc [<options>] [<command> [<arguments>]]

sbatch

- Submits a job script for batch processing
- Exits as soon as job parameters are accepted by the controller daemon (it does not wait for job execution)
- Job output is stored in file named “slurm-%j.out”, where %j is the job number
- Syntax: sbatch [<options>] [<script> [<arguments>]]
- Uses the same resource options as *srun*

Example MPI+OpenMP Job Script

```
#!/bin/bash
#SBATCH --nodes=16
#SBATCH --cpus-per-task=8
#SBATCH time=2:00:00

# The hello_world application will be started
# on 16 nodes with 8 threads per process;
# its execution time is limited to 2 hours

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
mpirun hello_world
```

squeue

- Displays information about queued jobs
- May be used to examine the job's status as well as resource allocations
- Syntax: squeue [<options>]
- Frequently used options:
  --all
  -l or --long
  -M or --clusters=<list_of_clusters>
  -p or --partition=<name>
  -t or --states=<list_of_states>
  -u or --user=<list_of_users>
  -i or --iterate=<seconds>
- Common job states include (abbreviated form in parentheses): PENDING (PD), RUNNING (R), SUSPENDED (S), STOPPED (ST), COMPLETING (CG), CANCELLED (CA), FAILED (F), TIMEOUT (TO), PREEMPTED (PR)

scancel

- Used to cancel submitted jobs or deliver signals to them
- Syntax: scancel [<options>] [<jobid>]…
- Frequently used options:
  -s or --signal=<signal_name>
  -n or --name=<job_name>
  -p or --partition=<partition_name>
  -t or --state=<job_state>
  -u or --user=<user_id>
  -i or --interactive
- The permitted job state include PENDING, RUNNING or SUSPENDED

sacct

- Retrieves job information from Slurm logs
- May be used to access status and exit code of no longer existing jobs
- Syntax: sacct [<options>]
- Frequently used options:
  - -a or allusers
  - -L or allclusters
  - -l or long
  - -j or job=<job_id>
  - --name=<list_of_job_names>
  - -s or state_list=<list_of_states>
- The permitted states are same as for the *squeue* command

sinfo

- Displays system's partition and node information
- Syntax: sinfo [<options>]
- Frequently used options:
  - *squeue's* options: all, long, clusters partition, iterate have the same meaning here

  -r or --responding

  -d or --dead

  -e or --exact

  -N or --Node

Important Environment Variables

- Accessible from within job scripts
- Permit runtime customization of job execution
- Often used variable list
  - SLURM_NTASKS
  - SLURM_NTASKS_PER_NODE
  - SLURM_CPUS_PER_TASK
  - SLURM_JOB_NUM_NODES (total number of nodes allocated to job)
  - SLURM_CPUS_ON_NODE (number of cores on the current node)
  - SLURM_SUBMIT_HOST
  - SLURM_CLUSTER_NAME
  - SLURM_JOB_PARTITION
  - SLURM_JOB_ID
  - SLURM_JOB_NODELIST (list of node names allocated to the job)
  - SLURM_TASKS_PER_NODE (list of tasks on each node corresponding to the host order in SLURM_JOB_NODELIST)
  - SLURM_SUBMIT_DIR

SLURM Tips and Tricks

- Launch interactive shell:

```
srun -N2 -t60 --pty /bin/bash
```

- Enable X window forwarding:

```
srun -N1 -t20 --x11 xterm
```

- Figure out the estimated start time of queued job:

```
squeue --start -j <job_id>
```

- Email notification when job terminates or fails:

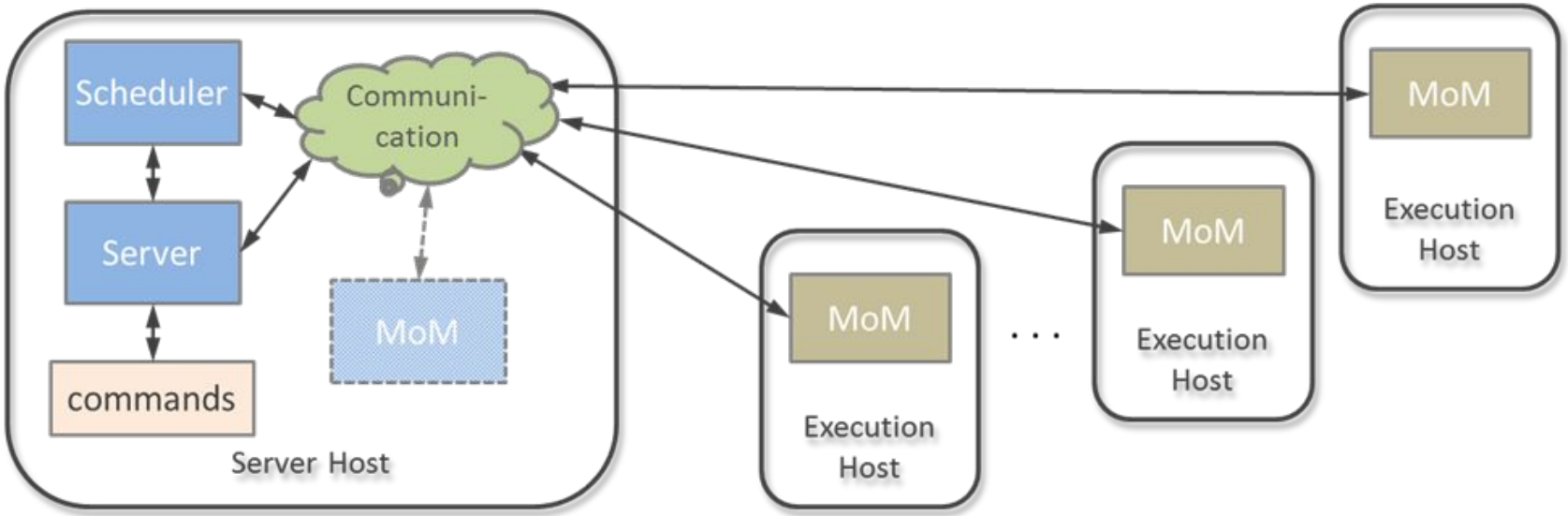
```
sbatch --mail-type=END,FAIL myjobscript
```

PBS Overview

- Started as a contract for NASA Ames developed by MRJ Technology Solutions in 1991
- Based on POSIX 1003.2d standard
- Includes contributions from NASA Ames, LLNL, NERSC
- Commercial version (PBS Pro) released by Veridian Corp. in 2000
- IP acquired by Altair Engineering (current owner)
- Open sourced in 2016
- Capable of grid workload execution using Globus toolkit
- Supports advanced reservation, ability to dynamically add and remove execution nodes, topology-aware scheduling, GPUs
- Used by many commercial and academic institutions as well as national laboratories
- Other, mostly compatible, implementations include:
  - OpenPBS (no longer maintained)
  - TORQUE (Terascale Open-source Resource and QUEue manager) developed by Adaptive Computing

PBS Architecture

- Server daemon processes user commands and creates, monitors, and dispatches jobs
- Scheduler monitors system resources (by polling MoMs) and allocates them to jobs
- Machine Oriented Mini-servers (MoMs) are a job executors
- Servers may be replicated in larger systems



qsub

- Submits a job to batch processing system
- Syntax: qsub [<options>] <script>
- Frequently used options:
  - -l <resource_list> (explained in the next slide)
  - -q <queue_name>
  - -N <job_name>
  - -V (export environment variables to job's environment)
  - -o <path> (capture standard output to a file)
  - -e <path> (capture standard error to a file)

Specifying Resources in PBS

- Syntax: <name>=<value>[,<name>=<value>…]
- Resource types:
  - Nodes

    nodes=<node_count>[:ppn=<processors_per_node>]
  - Wall time

    walltime=[[<hours:]<minutes>:]<seconds>
  - Memory

    mem=<integer>[<unit>]

    Where <unit> may be “B” (bytes) or “W” (words), optionally prefixed with “K” (kilo), “k” ($\times 2^{10}$), “M” (mega), “m” ($\times 2^{20}$), “G” (giga) or “g” ($\times 2^{30}$)
  - Accelerator specification is site-dependent, often:

    gpu=<integer>

New Resource Specification Format

- Incompatible with the old style (don't mix them!)
- Uses the concept of virtual node (vnode)
- Identified by keyword "select"
- Syntax: select=[<number>:]<chunk>[+<number:]<chunk>…]
  - <number> determines how many instances of <chunk> should be allocated
  - <chunk> is a list of <resource>=<value> assignments, separated by colons (":")
  - Common resource names:
    - ncpus (number of cores)
    - mem (physical memory per chunk)
    - mpiprocs (number of MPI processes per chunk)
    - naccelerators (number of accelerators on host)
    - accelerator_memory (amount of required accelerator memory)
    - accelerator_model (type of accelerator)
    - ompthreads (number of OpenMP threads to use)

New Placement Specification

- Syntax: place=[<arrangement>][:<sharing>][:<grouping]
- Arrangement is one of
  - free (place job on any vnode)
  - pack (puts all chunks on a single host)
  - scatter (assigns one MPI chunk to a host)
  - vscatter (one chunk per vnode)
- Sharing may be
  - excl (vnode exclusive to the current job)
  - shared (vnode shared with other jobs)
  - exclhost (entire host exclusive to the job)
- Grouping determines how chunks are grouped according to he resource
  - Syntax: <group>=<resource>
  - Some resources may be site specific

Examples

- Allocate 24 nodes for half an hour to a job:

```
qsub -l nodes=24 -l walltime=30:00 short_job.sh
```

- Submit a job to execute on two specific hosts for two hours

```
qsub -l nodes=host005+host006 -l 2:00:00 postprocess.sh
```

- Allocate 16 chunks with four processors and 1GB memory each using new format:

```
qsub -l select=16:ncpus=4:mem=1gb mpi64.sh
```

qdel

- Deletes one or more jobs from the system
- Syntax: qdel [<options>] <job_id> …
- Frequently used options:

  -W force (forcible deletion)

  -x (removes all jobs, including moved and finished jobs, and their history)

qstat (I)

- Displays status of jobs, queues or servers
- Job status syntax: qstat [<options>][<job_id>]
  - -a (reports running and queued jobs)
  - -H (shows finished and moved jobs)
  - -i (displays information about waiting, held, and queued jobs)
  - -r (shows running and suspended jobs)
  - -t (lists jobs, job arrays, and subjobs)
  - -u <user>[,…] (restricts display to jobs owned by specific user(s))
  - -f (enables long format containing job ID, attributes, submission arguments, and executable with arguments)
  - -p (replaces time use value with completion percentage)

qstat (II)

- Queue status syntax: qstat -Q [-f] [<queue>]
  - -f (shows full status, one attribute per line)
- Server status syntax: qstat -B [<options>] [<server>]
  - -f (shows full status)
  - -G (shows size in gigabytes)
  - -M (shows size in megawords)

tracejob

- Outputs logged information about a job
- Syntax: tracejob [<options>] <job_id>
- Frequently used options:
  - -v (increases verbosity)
  - -c <number> (shows <number> of most recent occurrences)
  - -n <days> (accesses history no more than <days> old)
  - -a (suppresses accounting information)
  - -l (suppresses scheduling information)
  - -m (suppresses MoM information)
  - -s (suppresses server information)

pbsnodes

- Examines status of system hosts
- Syntax: pbsnodes [<options>] [<host> …]
- Frequently used options:
  - -a (lists all hosts and their attributes)
  - -j (displays job related information, including vnodes and their state)
  - -S (shows system oriented information, including OS resources and custom hardware)
  - -H <host>[,<host>…] (reports attributes with non-default values for specified hosts)
  - -L (produces long format output)

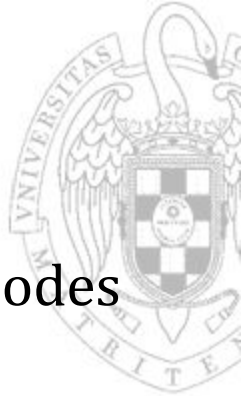
Example MPI+OpenMP PBS Job Script

```
#!/bin/bash
#PBS -l select=32:ncpus=8:ompthreads=8
#PBS -l walltime=1:30:00

# The script describes an MPI job to be run on
# 32 vnodes with 8 cores each for up to 1.5 hour.
# Since the environment is not automatically
# propagated, the script loads MPI environment
# explicitly (this part may be system-specific).

module load openmpi
mpirun mpisim simfile2.hdf5
```

Important PBS Environment Variables



- PBS_NODEFILE – path to file listing allocated execution vnodes
- NCPUS – number of available threads per vnode
- PBS_TASKNUM – number of Unix process executing on this vnode
- PBS_JOB_ID – job identifier
- PBS_JOBNAME – user-defined job name
- PBS_QUEUE – name of the queue the job was submitted to
- PBS_JOBDIR – job's execution directory
- PBS_TMPDIR – job-specific temporary directory
- PBS_O_WORKDIR – job submission directory (absolute path)
- PBS_O_HOME – value of HOME variable in submission environment
- PBS_O_HOST – name of job submission host

PBS Tips and Tricks

- Interactive execution:

```
qsub -I -l nodes=1,walltime=40:00
```

- X window forwarding:

```
qsub -X -I -l nodes=2,walltime=30:00 xterm
```

- Submitting job to the specific queue:

```
qsub -q -l nodes=4,walltime=30:00 myjob.sh
```

- Find out the estimated start of execution:

```
qstat -T <job_id>
```

- Email notification when job aborts or terminates:

```
qsub -m ae -l nodes=4,walltime=2:00:00 job.sh
```