RUNNING PARALLEL APPLICATIONS ON A CLUSTER

Module 3.8

LEARNING OBJECTIVES

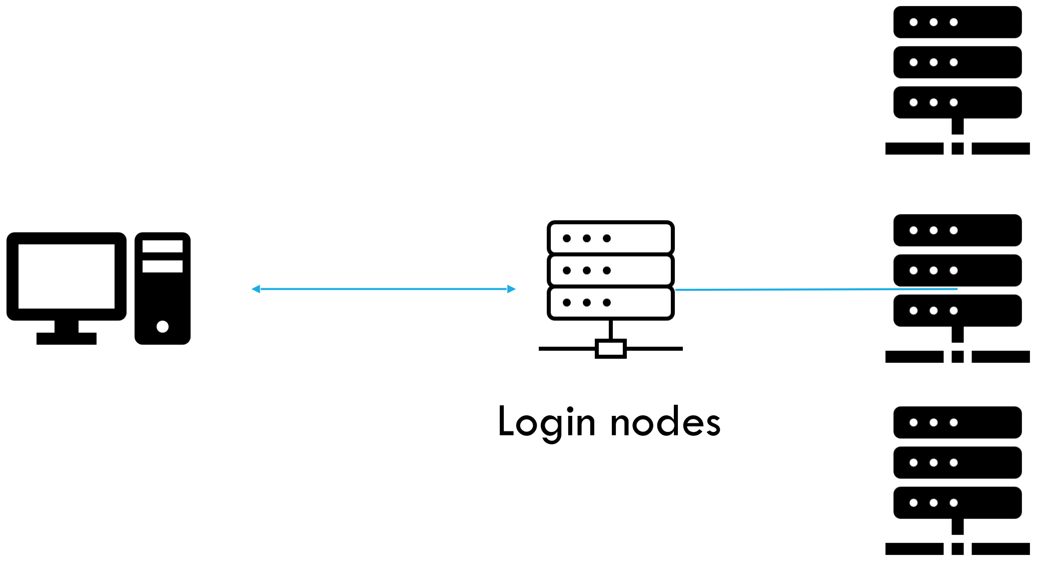
Running jobs on the compute nodes of a cluster:

- Running multicore jobs in a cluster
- Running multimode jobs in a cluster
- Running GPU accelerated jobs in a cluster

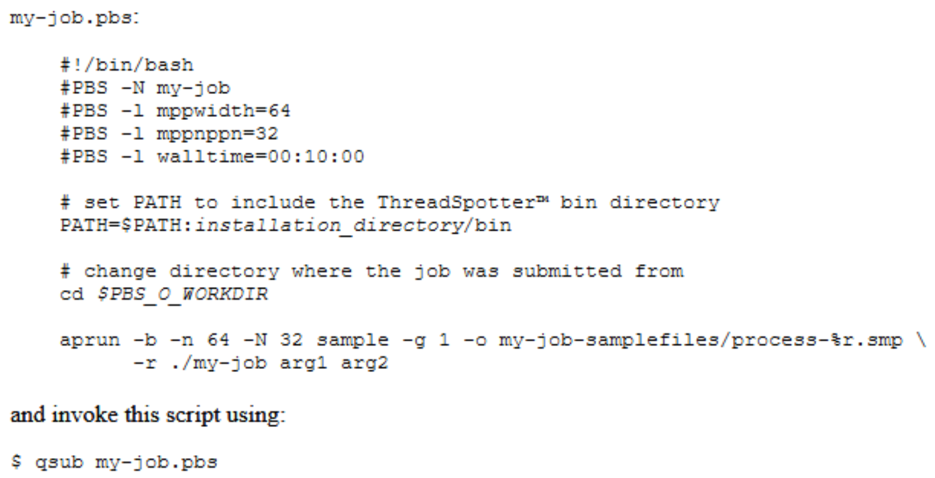
CLUSTER ARCHITECTURE

A supercomputer is a shared resource that is used by several users concurrently. To optimize access to the resource the minimal unit of execution of a program is defined as a job. Jobs are scheduled to run in a group of nodes for a given amount of time as requested by the user. The queue manager, so-called scheduler, ensures that jobs are queued using some fair use policy (e.g., TORQUE).

Your jobs should run on the **compute nodes** of the cluster.



The action of sending a job to the queueing systems is called job submission. Therefore, in contrast to running programs on a local computer, programs in a supercomputer are first queued and subsequentely managed by a workload manager (e.g., Moab, Slurm).



my-job.pbs:

```
#!/bin/bash
#PBS -N my-job
#PBS -l mppwidth=64
#PBS -l mppnppn=32
#PBS -l walltime=00:10:00

# set PATH to include the ThreadSpotter™ bin directory
PATH=$PATH:installation_directory/bin

# change directory where the job was submitted from
cd $PBS_O_WORKDIR

aprun -b -n 64 -N 32 sample -g 1 -o my-job-samplefiles/process-%r.smp \
      -r ./my-job arg1 arg2
```

and invoke this script using:

```
$ qsub my-job.pbs
```

To submit the job you must write a submission script.

FILE SYSTEMS

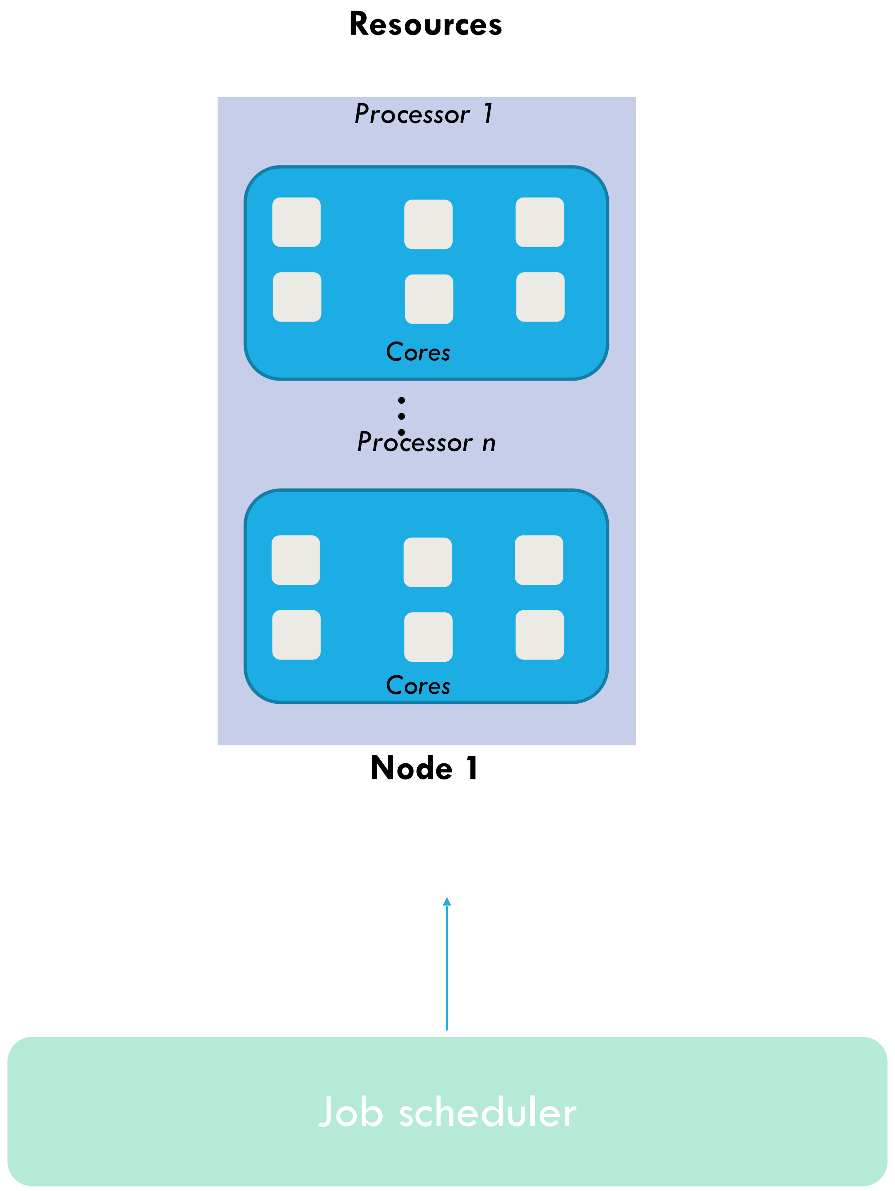
Typically in a cluster there are several filesystems with different purposes.

File system	Best Storage Practices	Best Activities
$HOME	cron jobs small scripts environment settings	compiling, editing
$WORK	software installations original datasets that can't be reproduced job scripts and templates	staging datasets
$SCRATCH	temporary datasets I/O files job files	all job I/O activity

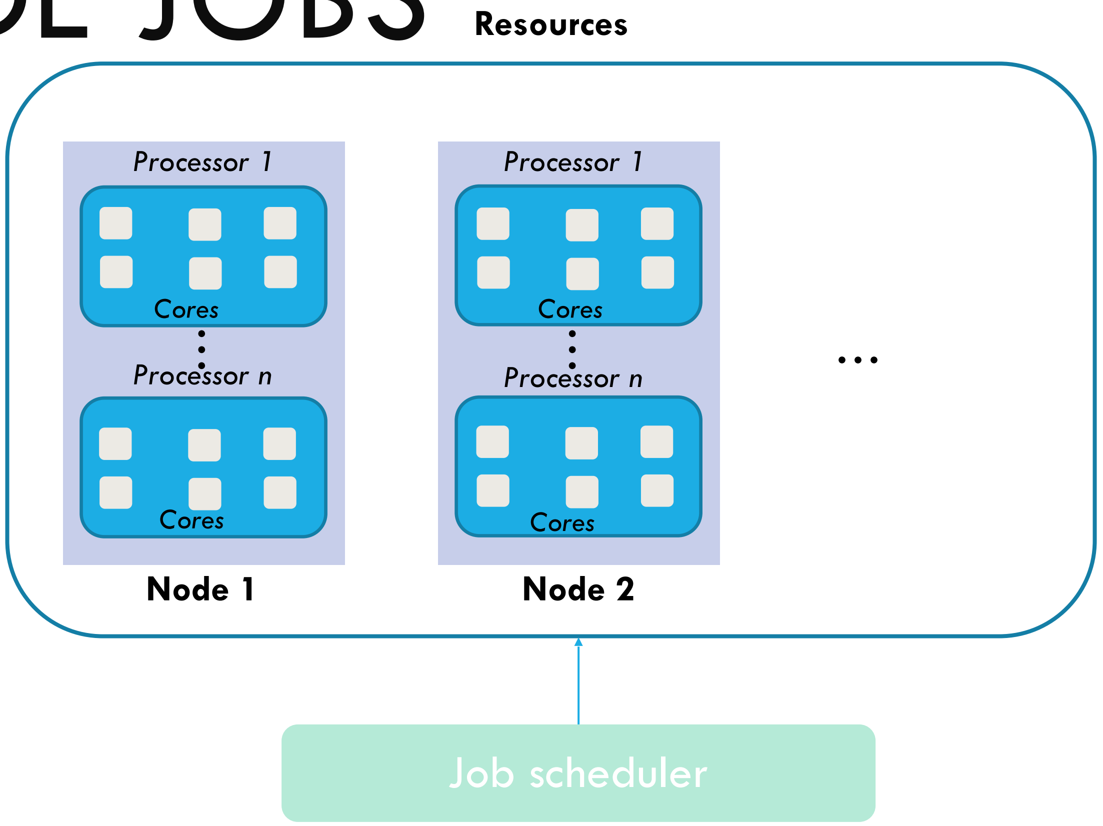
CASE STUDY ON FRONTERA: RUNNING JOBS

Frontera uses the Slurm Workload Manager as its job scheduler.

- Resources are requested through the job scheduler:
  - Single node:
    - Serial
    - Threaded jobs (OpenMP)
  - Multi node (MPI):
    - Multicore (OpenMP or SMP)



RUNNING MULTI NODE JOBS



Allocation of resources consist in requesting a given **number of nodes** and the **number of processes per node.**

According to the request of resources, the total **number of MPI ranks** are distributed along the nodes and cores.

Commands to control cpu-affinity include:
--cpus-per-task
--cpu-bind

$\longrightarrow$

Assign the number of CPUs per process and arrange CPU-binding according to Non-Uniform Memory Access **(NUMA)** design

USING CLUSTER-SPECIFIC LAUNCHERS

Launcher managers

Once resources are allocated, the jobs are launched by the generic job manager supported on the processors.

For parallel jobs (using MPI):

Launch a single job parallelizing the steps involved in the workflow

Launch a job multiple times over the resources allocated.

srun

srun enables setting interactive sessions for resource allocation and job submission

MPI jobs

--*distribution* ; Controls the placements of ranks across sockets within the nodes requested

--*cpu_bind* ; Distributes ranks across physical CPUs or logical cores (hyperthreading).

--*map_cpu* ; List of CPUIDs mapped to the total number of ranks on every node.

ibrun

TACC-specific MPI launcher.

ibrun allocates the total number of ranks according to the resource allocation set with *Slurm*.

Additional flags include:

-*n* ; Total number of ranks

-*N* ; Total number of nodes

export IBRUN_TASKS_PER_NODE= *number of ranks per node*

In the absence of flags, the *numactl* command enables the control of process and memory affinity

-*N* ; socket affinity

-C ; core affinity

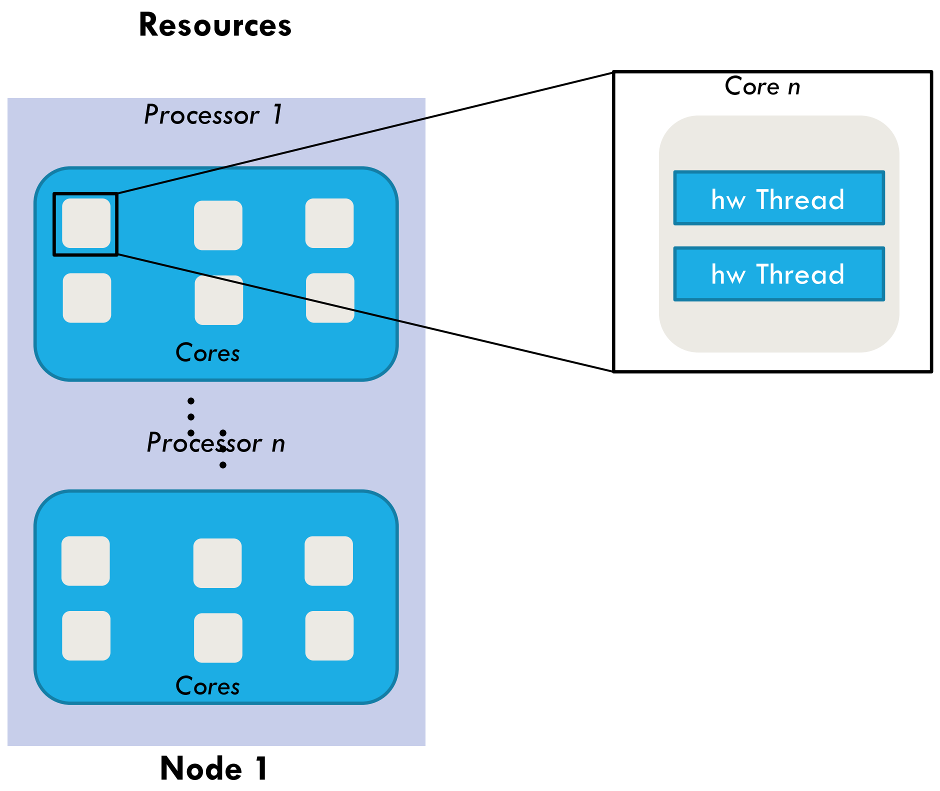
USING CLUSTER-SPECIFIC LAUNCHERS

Running one job per hw thread

For a single node, processes are bound to each of the cores allocated by the resource manager. Hence, for each processor, one job is allocated per physical core.

Running one job per logical thread

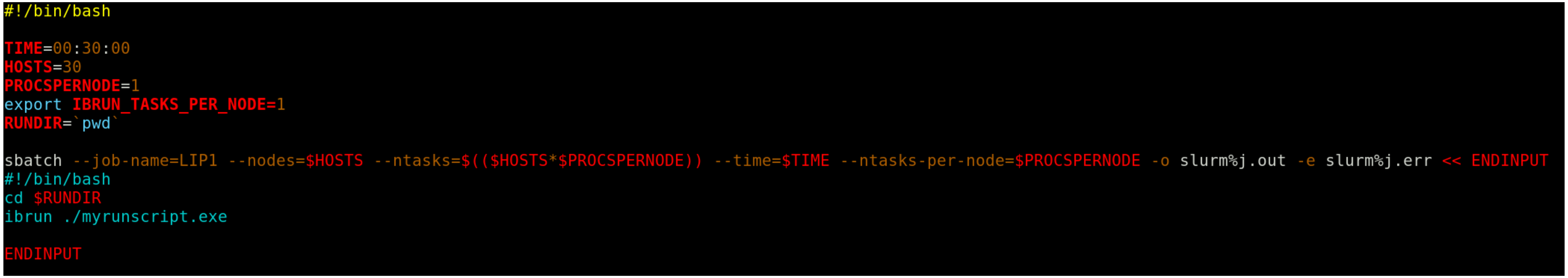
Hyperthreading enables the distribution of multiple processes across the logical cores on a processor. For instance, one process can be bound to each of the threads on each core. The result is the distribution of more than one job per physical core.



EXAMPLE

The following script corresponds to a submission example to allocate 30 MPI-ranks on 30 nodes, placing 1 node per rank. Upon resource allocation, the execution script is run with the MPI-launcher *ibrun*.

Allocation of more than one rank per node is enabled by increasing the number of processes/tasks per node.



```
#!/bin/bash

TIME=00:30:00
HOSTS=30
PROCSPERNODE=1
export IBRUN_TASKS_PER_NODE=1
RUNDIR=`pwd`

sbatch --job-name=LIP1 --nodes=$HOSTS --ntasks=$(($HOSTS*$PROCSPERNODE)) --time=$TIME --ntasks-per-node=$PROCSPERNODE -o slurm%j.out -e slurm%j.err << ENDINPUT
#!/bin/bash
cd $RUNDIR
ibrun ./myrunscript.exe

ENDINPUT
```

- Using aprun: https://bluewaters.ncsa.illinois.edu/using-aprun
- Queueing policies: https://bluewaters.ncsa.illinois.edu/queues-and-scheduling-policies
- Accessing compute nodes: https://bluewaters.ncsa.illinois.edu/accessing

REFERENCES