

SC-Camp 2017: Resource Manager & Job Scheduler

On the efficient use of HPC facility



UL High Performance Computing (HPC) Team

Dr. S. Varrette

Oct. 23th, 2017

University of Luxembourg (UL), Luxembourg

8th Intl. SuperComputing Camp (SC-Camp 2017), Cadiz, Spain

Shared Etherpad Notes:

<https://im.csc.uni.lu/p/SC-Camp2017>

Latest versions available on **GitLab**:



UL HPC tutorials:

<https://github.com/ULHPC/tutorials>

SC-Camp:

<http://www.sc-camp.org/>

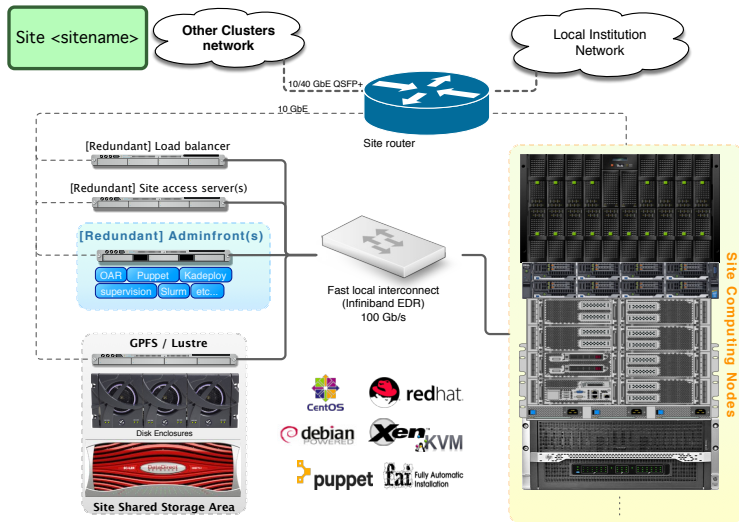




Summary

- 1 **Introduction**
- 2 The OAR Batch Scheduler
- 3 The SLURM Batch Scheduler
 - Overview
 - Slurm commands
 - Slurm vs. OAR commands
 - ULHPC Slurm Configuration
 - Usage Example
 - Slurm Launcher
- 4 Conclusion

UL HPC: General cluster organization



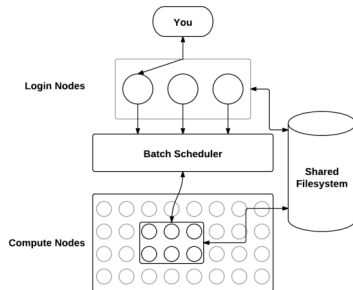
HPC Components: Software Stack

- **Remote connection to the platform** SSH
- **Identity Management / SSO:** LDAP, Kerberos, IPA...
- **Resource management:** job/batch scheduler
 - ↪ SLURM, OAR, PBS, MOAB/Torque...
- **(Automatic) Node Deployment:**
 - ↪ FAI, Kickstart, Puppet, Chef, Ansible, Kadeploy...
- **(Automatic) User Software Management:**
 - ↪ Easybuild, Environment Modules, LMod
- **Platform Monitoring:**
 - ↪ Nagios, Icinga, Ganglia, Foreman, Cacti, Alerta...

Resource and Job Management Systems

• Resource and Job Management System (RJMS)

- “Glue” for a parallel computer to execute parallel jobs
- **Goal:** satisfy users' demands for computation
 - ✓ assign resources to user jobs with an efficient manner



Resource and Job Management Systems

• Resource and Job Management System (RJMS)

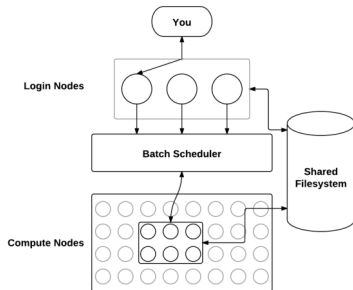
- “Glue” for a parallel computer to execute parallel jobs
- **Goal:** satisfy users’ demands for computation
 - ✓ assign resources to user jobs with an efficient manner

• HPC Resources:

- Nodes (typically a unique IP address)
 - ✓ Sockets / Cores / Hyperthreads
 - ✓ Memory
 - ✓ Interconnect/switch resources
- Generic resources (e.g. GPUs)
- Licenses

• Strategic Position

- Direct/constant knowledge of resources
- Launch and otherwise manage jobs



RJMS Layers

- Resource Allocation involves three principal abstraction layers:

- ↳ **Job Management:**

- ✓ declaration of a job & demand of resources and job characteristics,

- ↳ **Scheduling:** matching of the jobs upon the resources,

- ↳ **Resource Management:**

- ✓ launching and placement of job instances. . .
- ✓ . . . along with the job's control of execution

- **When there is more work than resources**

- ↳ the job scheduler manages queue(s) of work

- ✓ supports complex scheduling algorithms

- ↳ Supports resource limits (by queue, user, group, etc.)

RJMS Detailed Components

- **Resource Management**

- ↪ Resource Treatment (hierarchy, partitions,..)
- ↪ Job Launching, Propagation, Execution control
- ↪ Task Placement (topology, binding,..)
- ↪ **Advanced Features:**
 - ✓ High Availability, Energy Efficiency, Topology aware placement

RJMS Detailed Components

● Resource Management

- ↪ Resource Treatment (hierarchy, partitions,..)
- ↪ Job Launching, Propagation, Execution control
- ↪ Task Placement (topology, binding,..)
- ↪ **Advanced Features:**
 - ✓ High Availability, Energy Efficiency, Topology aware placement

● Job Management

- ↪ Job declaration and control (signaling, reprioritizing,...)
- ↪ Monitoring (reporting, visualization,..)
- ↪ **Advanced Features:**
 - ✓ Authentication (limitations, security,..)
 - ✓ QOS (checkpoint, suspend, accounting,...)
 - ✓ Interfacing (MPI libraries, debuggers, APIs,..)

RJMS Detailed Components

• Resource Management

- ↪ Resource Treatment (hierarchy, partitions,..)
- ↪ Job Launching, Propagation, Execution control
- ↪ Task Placement (topology, binding,..)
- ↪ **Advanced Features:**
 - ✓ High Availability, Energy Efficiency, Topology aware placement

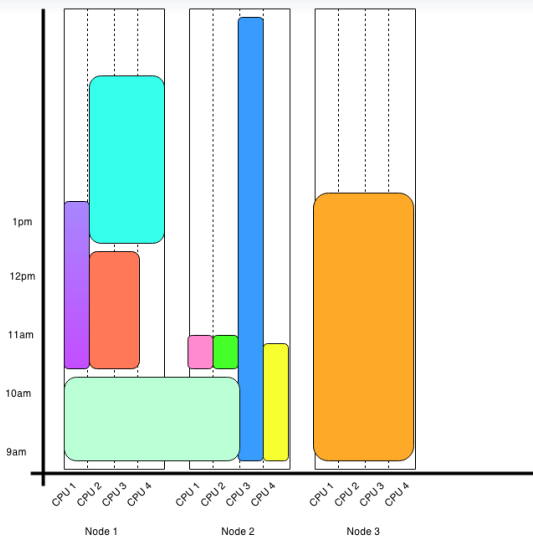
• Job Management

- ↪ Job declaration and control (signaling, reprioritizing,...)
- ↪ Monitoring (reporting, visualization,..)
- ↪ **Advanced Features:**
 - ✓ Authentication (limitations, security,..)
 - ✓ QOS (checkpoint, suspend, accounting,...)
 - ✓ Interfacing (MPI libraries, debuggers, APIs,..)

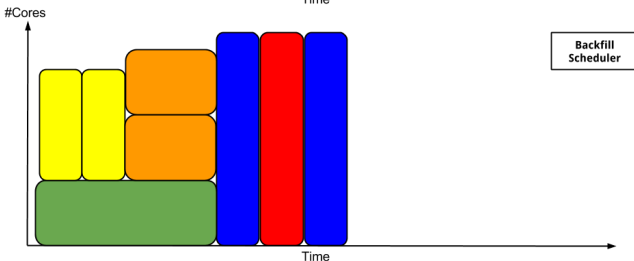
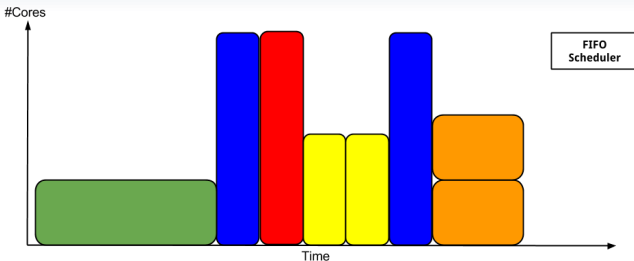
• Scheduling

- ↪ Queues Management (priorities, multiple,..)
- ↪ Advanced Reservation

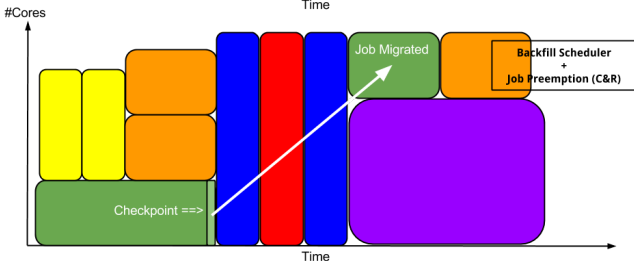
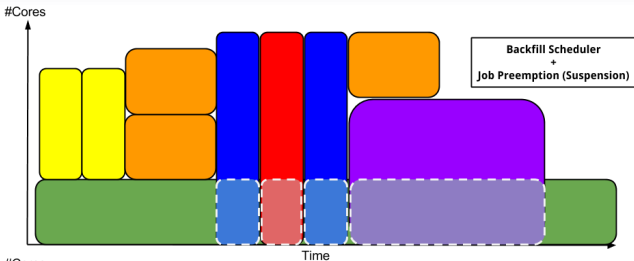
Job Scheduling



Job Scheduling (backfilling)



Job Scheduling (suspension & requeue)



Main Job Schedulers

Name	Company	Version*
SLURM	SchedMD	17.02.8
LSF	IBM	10.1
OpenLava	LSF Fork	2.2
MOAB/Torque	Adaptative Computing	6.1
PBS	Altair	13.0
OAR (PBS Fork)	LIG	2.5.7
Oracle Grid Engine (formely SGE)	Oracle	

*: As of Oct. 2017

Main Job Schedulers

Name	Company	Version*
SLURM	SchedMD	17.02.8
LSF	IBM	10.1
OpenLava	LSF Fork	2.2
MOAB/Torque	Adaptative Computing	6.1
PBS	Altair	13.0
OAR (PBS Fork)	LIG	2.5.7
Oracle Grid Engine (formely SGE)	Oracle	

*: As of Oct. 2017



Summary

- 1 Introduction
- 2 The OAR Batch Scheduler
- 3 The SLURM Batch Scheduler
 - Overview
 - Slurm commands
 - Slurm vs. OAR commands
 - ULHPC Slurm Configuration
 - Usage Example
 - Slurm Launcher
- 4 Conclusion

UL HPC resource manager: OAR

The OAR Batch Scheduler

<http://oar.imag.fr>

- Versatile resource and task manager

- ↳ schedule **jobs** for users on the cluster **resource**
- ↳ OAR resource = a node or part of it (CPU/core)
- ↳ OAR job = execution time (**walltime**) on a set of resources



UL HPC resource manager: OAR

The OAR Batch Scheduler

<http://oar.imag.fr>

- Versatile resource and task manager
 - ↪ schedule **jobs** for users on the cluster **resource**
 - ↪ OAR resource = a node or part of it (CPU/core)
 - ↪ OAR job = execution time (**walltime**) on a set of resources



OAR main features includes:

- **interactive vs. passive (aka. batch) jobs**
- **best effort jobs**: use more resource, accept their release any time
- **deploy jobs (Grid5000 only)**: deploy a customized OS environment
 - ↪ ... and have full (root) access to the resources
- **powerful resource filtering/matching**

Main OAR commands

- oarsub** submit/reserve a job (by default: **1 core for 2 hours**)
- oardel** delete a submitted job
- oarnodes** shows the resources states
- oarstat** shows information about running or planned jobs

	Submission
interactive	<code>oarsub [options] -I</code>
passive	<code>oarsub [options] scriptName</code>

- Each created job receive an identifier JobID
 - ↪ Default passive job log files: `OAR.JobID.std{out,err}`
- You can make a reservation with `-r "YYYY-MM-DD HH:MM:SS"`

Main OAR commands

- oarsub** submit/reserve a job (by default: **1 core for 2 hours**)
- oardel** delete a submitted job
- oarnodes** shows the resources states
- oarstat** shows information about running or planned jobs

	Submission
interactive	<code>oarsub [options] -I</code>
passive	<code>oarsub [options] scriptName</code>

- Each created job receive an identifier JobID
 - ↪ Default passive job log files: `OAR.JobID.std{out,err}`
- You can make a reservation with `-r "YYYY-MM-DD HH:MM:SS"`

Direct access to nodes by `ssh` is forbidden: use `oarsh` instead

OAR job environment variables

Once a job is created, some environments variables are defined:

Variable	Description
\$OAR_NODEFILE	Filename which lists all reserved nodes for this job
\$OAR_JOB_ID	OAR job identifier
\$OAR_RESOURCE_PROPERTIES_FILE	Filename which lists all resources and their properties
\$OAR_JOB_NAME	Name of the job given by the "-n" option of oarsub
\$OAR_PROJECT_NAME	Job project name

Useful for MPI jobs for instance:

```
$> mpirun -machinefile $OAR_NODEFILE /path/to/myprog
```

... Or to collect how many cores are reserved per node:

```
$> cat $OAR_NODEFILE | uniq -c
```

OAR job types

Job Type	Max Walltime (hour)	Max #active_jobs	Max #active_jobs_per_user
interactive	12:00:00	10000	5
default	120:00:00	30000	10
besteffort	9000:00:00	10000	1000

cf /etc/oar/admission_rules/*.conf

- **interactive**: useful to test / prepare an experiment
 - ↪ you get a shell on the first reserved resource
- **best-effort vs. default**: nearly unlimited constraints **YET**
 - ↪ a besteffort job can be killed as soon as a default job as no other place to go
 - ↪ enforce checkpointing (and/or idempotent) strategy

Characterizing OAR resources

Specifying wanted resources in a hierarchical manner

- Use the `-l` option of `oarsub`. Main constraints:

<code>enclosure=N</code>	number of enclosure
<code>nodes=N</code>	number of nodes
<code>core=N</code>	number of cores
<code>walltime=hh:mm:ss</code>	job's max duration

Specifying OAR resource properties

- Use the `-p` option of `oarsub`:

Syntax: `-p "property='value'"`

<code>gpu='{YES,NO}'</code>	has (or not) a GPU card
<code>host='fqdn'</code>	full hostname of the resource
<code>network_address='hostname'</code>	Short hostname of the resource
(Chaos only) <code>nodeclass='{k,b,h,d,r}'</code>	Class of node



OAR (interactive) job examples

- 2 cores on 3 nodes (same enclosure) for 3h15:

Total: 6 cores

```
(frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15
```

OAR (interactive) job examples

- 2 cores on 3 nodes (same enclosure) for 3h15:

Total: 6 cores

```
(frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15
```

- 4 cores on a GPU node for 8 hours

Total: 4 cores

```
(frontend)$> oarsub -I -l /core=4,walltime=8 -p "gpu='YES'"
```

OAR (interactive) job examples

- 2 cores on 3 nodes (same enclosure) for 3h15: Total: 6 cores

```
(frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15
```

- 4 cores on a GPU node for 8 hours Total: 4 cores

```
(frontend)$> oarsub -I -l /core=4,walltime=8 -p "gpu='YES'"
```

- 2 nodes among the h-cluster1-* nodes (Chaos only) Total: 24 cores

```
(frontend)$> oarsub -I -l nodes=2 -p "nodeclass='h'"
```

OAR (interactive) job examples

- 2 cores on 3 nodes (same enclosure) for 3h15: Total: 6 cores

```
(frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15
```

- 4 cores on a GPU node for 8 hours Total: 4 cores

```
(frontend)$> oarsub -I -l /core=4,walltime=8 -p "gpu='YES'"
```

- 2 nodes among the h-cluster1-* nodes (Chaos only) Total: 24 cores

```
(frontend)$> oarsub -I -l nodes=2 -p "nodeclass='h'"
```

- 4 cores on 2 GPU nodes + 20 cores on other nodes Total: 28 cores

```
$> oarsub -I -l "{gpu='YES'}/nodes=2/core=4+{gpu='NO'}/core=20"
```

OAR (interactive) job examples

- 2 cores on 3 nodes (same enclosure) for 3h15: Total: 6 cores

```
(frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15
```

- 4 cores on a GPU node for 8 hours Total: 4 cores

```
(frontend)$> oarsub -I -l /core=4,walltime=8 -p "gpu='YES'"
```

- 2 nodes among the h-cluster1-* nodes (Chaos only) Total: 24 cores

```
(frontend)$> oarsub -I -l nodes=2 -p "nodeclass='h'"
```

- 4 cores on 2 GPU nodes + 20 cores on other nodes Total: 28 cores

```
$> oarsub -I -l "{gpu='YES'}/nodes=2/core=4+{gpu='NO'}/core=20"
```

- A full big SMP node Total: 160 cores on gaia-74

```
$> oarsub -t bigsmp -I 1 node=1
```

Some other useful features of OAR

Connect to a running job

```
(frontend)$> oarsub -C JobID
```

Cancel a job

```
(frontend)$> oardel JobID
```

Status of a jobs

```
(frontend)$> oarstat -state -j JobID
```

View the job

```
(frontend)$> oarstat
```

```
(frontend)$> oarstat -f -j JobID
```

Get info on the nodes

```
(frontend)$> oarnodes
```

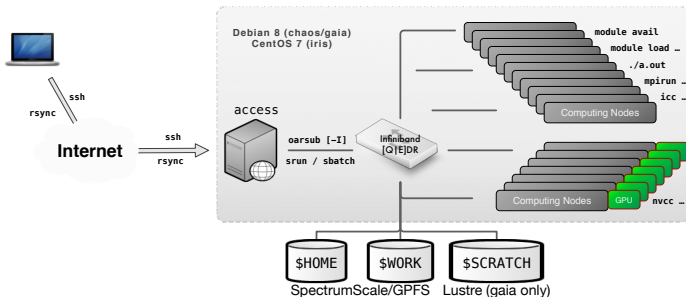
```
(frontend)$> oarnodes -l
```

```
(frontend)$> oarnodes -s
```

Run a best-effort job

```
(frontend)$> oarsub -t besteffort ...
```

OAR Practical session



Demo Time

- gaia or chaos UL cluster access
- Interactive / Passive job submission

Designing efficient OAR job launchers

Resources/Example

<https://github.com/ULHPC/launcher-scripts>



- UL HPC grant access to **parallel computing** resources
 - ideally: OpenMP/MPI/CUDA/OpenCL jobs
 - if serial jobs/tasks: run them efficiently
- Avoid to submit purely serial jobs to the OAR queue a
 - waste the computational power (11 out of 12 cores on gaia).
 - use whole nodes by running at least 12 serial runs at once
- **Key:** understand difference between **Task** and **OAR job**

Designing efficient OAR job launchers

Resources/Example

<https://github.com/ULHPC/launcher-scripts>



- UL HPC grant access to **parallel computing** resources
 - ↪ ideally: OpenMP/MPI/CUDA/OpenCL jobs
 - ↪ if serial jobs/tasks: run them efficiently
- Avoid to submit purely serial jobs to the OAR queue a
 - ↪ waste the computational power (11 out of 12 cores on gaia).
 - ↪ use whole nodes by running at least 12 serial runs at once
- **Key:** understand difference between **Task** and **OAR job**

For more information...

- Incoming Practical Session
 - ↪ HPC workflow with sequential jobs (C,python,java etc.)

OAR Simple Example of usage

```
# Simple interactive job
(access)$> oarsub -I
(node)$> echo $OAR_JOBID
4239985
(node)$> echo $OAR_NODEFILE
/var/lib/oar//4239985
(node)$> cat $OAR_NODEFILE | wc -l
8
(node)$> cat $OAR_NODEFILE
moonshot1-39
moonshot1-39
moonshot1-39
moonshot1-39
moonshot1-40
moonshot1-40
moonshot1-40
moonshot1-40
```

View existing job

```
# View YOUR jobs (remove -u to view all)
```

```
(access)$> oarstat -u
```

Job id	Name	User	Submission Date	S Queue
4239985		svarrette	2017-10-23 12:33:41	R default

View Detailed info on jobs

```
(access)$> oarstat -f -j 4239985
```

```
Job_Id: 4239985
```

```
[...]
```

```
state = Running
```

```
wanted_resources = -l "{type = 'default'}/ibpool=1/host=2,walltime=2:0:0
```

```
types = interactive, inner=4236343, moonshot
```

```
assigned_resources = 3309+3310+3311+3312+3313+3314+3315+3316
```

```
assigned_hostnames = moonshot1-39+moonshot1-40
```

```
queue = default
```

```
launchingDirectory = /home/users/svarrette
```

```
stdout_file = OAR.4239985.stdout
```

```
stderr_file = OAR.4239985.stderr
```

```
jobType = INTERACTIVE
```

```
properties = (((bigmem='NO' AND bigsmp='NO') AND dedicated='NO') AND os=
```

```
walltime = 2:0:0
```

```
initial_request = oarsub -I -l nodes=2 -t moonshot -t inner=4236343
```

```
message = R=8,W=2:0:0,J=I,T=inner|interactive|moonshot (Karma=1.341)
```

Access to an existing job: Attempt 1

```
# Get your job ID...  
(access)$> oarstat -u
```

- **Attempt 1:** Get assigned resources and ...

Access to an existing job: Attempt 1

```
# Get your job ID...
(access)$> oarstat -u
```

- **Attempt 1:** Get assigned resources and ... ssh to it!

```
# Collect the assigned resources
(access)$> oarstat -f -j 4239985 | grep hostname
    assigned_hostnames = moonshot1-39+moonshot1-40
```

```
(access)$> ssh moonshot1-39
[...]
```

```
=====
/!\ WARNING:   Direct login by ssh is forbidden.
```

Use `oarsub(1)` to reserve nodes, and `oarsh(1)` to connect to your reserved nodes, typically by:

```
OAR_JOB_ID=<jobid> oarsh <nodename>
```

Access to an existing job

- Using oarsh:

```
# Get your job ID...  
(access)$> oarstat -u  
# ... get the hostname of the nodes allocated ...  
(access)$> oarstat -f -j 4239985 | grep hostname  
# ... and connect to it with oarsh  
(access)$> OAR_JOB_ID=4239985 oarsh moonshot1-39
```

Access to an existing job

- Using oarsh:

```
# Get your job ID...  
(access)$> oarstat -u  
# ... get the hostname of the nodes allocated ...  
(access)$> oarstat -f -j 4239985 | grep hostname  
# ... and connect to it with oarsh  
(access)$> OAR_JOB_ID=4239985 oarsh moonshot1-39
```

- (better) Using oarsub -C:

```
# Get your job ID...  
(access)$> oarstat -u  
# ... and connect to the FIRST node of the reservation  
(access)$> oarsub -C 4239985
```


MPI jobs

- Intel MPI

```
(node)$> module load toolchain/intel  
# ONLY on moonshot node have no IB card: export I_MPI_FABRICS=tcp  
(node)$> mpirun -hostfile $OAR_NODEFILE /path/to/mpiprogram
```

- OpenMPI:

```
(node)$> module load mpi/OpenMPI  
(node)$> mpirun -hostfile $OAR_NODEFILE -x PATH -x LD_LIBRARY_PATH \  
/path/to/mpiprogram
```

- For more details: See MPI sessions

https://github.com/ULHPC/launcher-scripts/blob/devel/bash/MPI/mpi_launcher.sh

OAR Launcher Scripts

```
$> oarsub -S <scriptname>           # -S: interpret #OAR comments
```

- Our launcher scripts on Github: <https://github.com/ULHPC/launcher-scripts>
↪ see in particular [our generic launcher](#) compliant w. OAR & SLURM
- **Example:**

```
#!/bin/bash
#OAR -l nodes=2/core=1,walltime=1
#OAR -n MyNamedJob
# Prepare UL HPC modules
if [ -f /etc/profile ]; then
. /etc/profile
fi

module load toolchain/intel
/path/to/prog <ARGS>
```



Summary

- 1 Introduction
- 2 The OAR Batch Scheduler
- 3 The SLURM Batch Scheduler**
 - Overview
 - Slurm commands
 - Slurm vs. OAR commands
 - ULHPC Slurm Configuration
 - Usage Example
 - Slurm Launcher
- 4 Conclusion

Slurm Workload Manager

- **Simple Linux Utility for Resource Management**
 - ↪ Development started in 2002
 - ✓ initially as a simple resource manager for Linux clusters
 - ↪ Has evolved into a capable job scheduler through use of opt. plugins
 - ↪ About 500,000 lines of C code today.
 - ↪ Supports AIX, Linux, Solaris, other Unix variants
- Used on many of the world's largest computers

Slurm Workload Manager

- **Simple Linux Utility for Resource Management**
 - ↪ Development started in 2002
 - ✓ initially as a simple resource manager for Linux clusters
 - ↪ Has evolved into a capable job scheduler through use of opt. plugins
 - ↪ About 500,000 lines of C code today.
 - ↪ Supports AIX, Linux, Solaris, other Unix variants
 - Used on many of the world's largest computers
-
- **Now deployed on new UL HPC clusters**
 - ↪ starting iris cluster (2017)

Slurm Design Goals

- **Small and simple**
 - **Highly scalable and Fast**
 - ↪ managing 1.6 million core IBM BlueGene/Q,
 - ↪ tested to 33 million cores using emulation
 - ↪ throughput: up to 600 jobs p.s. & 1000 job submissions p.s.
 - **Modular:**
 - ↪ plugins to support \neq scheduling policies, MPI librairies. . .
 - **Secure and Fault-tolerant**
 - ↪ highly tolerant of system failures
 - **Power Management** and detailed monitoring
-
- **Open source:** GPL v2, active world-wide development
 - **Portable:** written in C with a GNU autoconf configuration engine

Slurm Docs and Resources

- User and Admins latest documentation:
↳ <http://slurm.schedmd.com/documentation.html>
- Detailed man pages for commands and configuration files
↳ http://slurm.schedmd.com/man_index.html
- All SLURM related publications and presentations:
↳ <http://slurm.schedmd.com/publications.html>

ULHPC Documentation & comparison to OAR

<https://hpc.uni.lu/users/docs/scheduler.html>

Other Resources

• Puppet module ULHPC/slurm

- Developed by the UL HPC Team – see <https://forge.puppet.com/ULHPC/>
- Used in production on iris cluster
- see also Slurm Control Repo Example

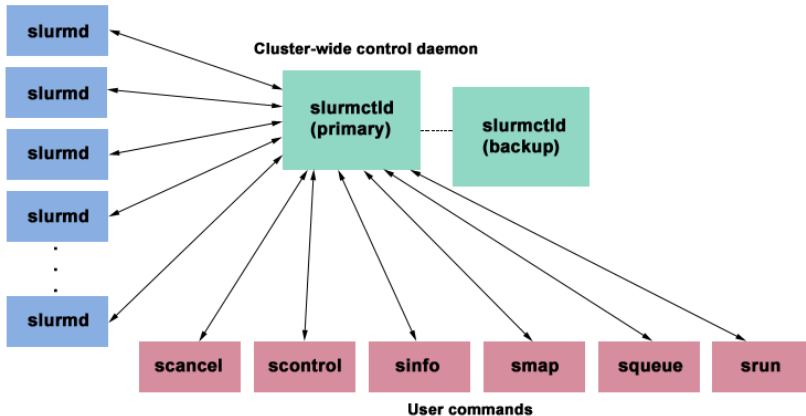


Puppet Class	Description
<code>slurm</code>	The main slurm class, piloting all aspects of the configuration
<code>slurm::slurmdbd</code>	Specialized class for <code>Slurmdbd</code> , the Slurm Database Daemon.
<code>slurm::slurmctld</code>	Specialized class for <code>Slurmctld</code> , the central management daemon of Slurm.
<code>slurm::slurmd</code>	Specialized class for <code>Slurmd</code> , the compute node daemon for Slurm.
<code>slurm::munge</code>	Manages <code>MUNGE</code> , an authentication service for creating and validating credentials.
<code>slurm::pam</code>	Handle PAM aspects for SLURM (Memlock for MPI etc.)

Puppet Defines	Description
<code>slurm::download</code>	takes care of downloading the SLURM sources for a given version passed as resource name
<code>slurm::build</code>	building Slurm sources into packages (<i>_i.e.</i> RPMs for the moment)
<code>slurm::install::packages</code>	installs the Slurm packages, typically built from <code>slurm::build</code>
<code>slurm::acct::*</code>	adding (or removing) accounting resources to the slurm database

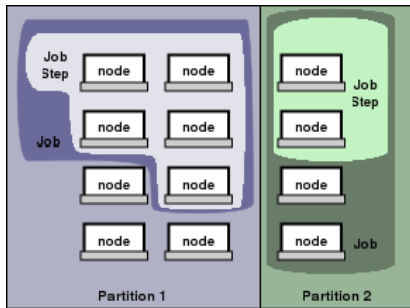
SLURM Architecture

One daemon per node

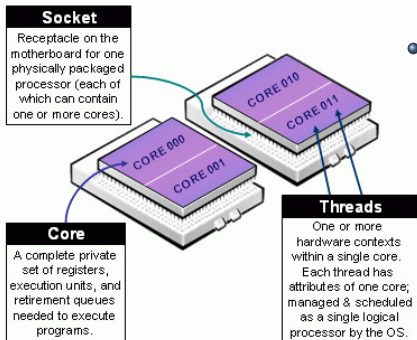


SLURM Entities

Entity	Description
Computing node	Computer used for the execution of programs
Partition	Group of nodes into logical sets
Job	Allocation of resources assigned to a user for some time
Job Step	Sets of (possible parallel) tasks with a job



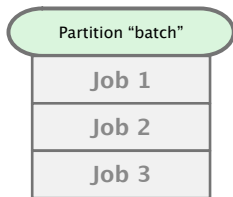
SLURM Multi-Core/Thread Support



- Nodes hierarchy

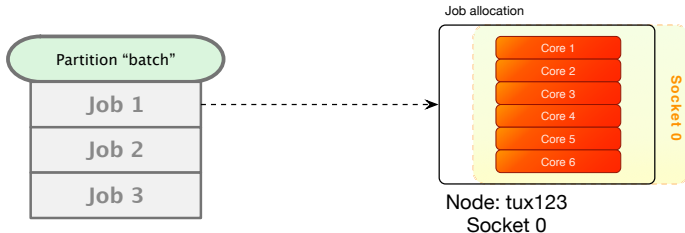
- ↪ NUMA [base]board
 - ✓ Socket/Core/Thread
- ↪ Memory
- ↪ Generic Resources GRES (e.g. GPUs)

SLURM Entities example



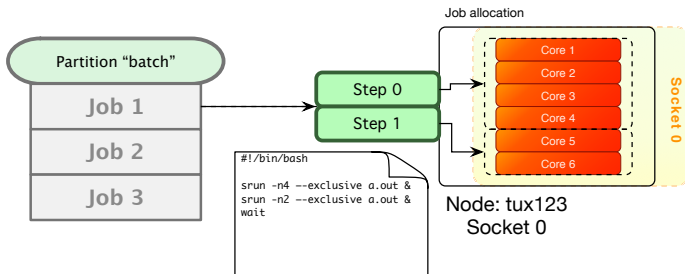
- Users submit jobs to a partition (queue)

SLURM Entities example



- Jobs are allocated resources

SLURM Entities example



- Jobs spawn steps, which are allocated resources from within the job's allocation

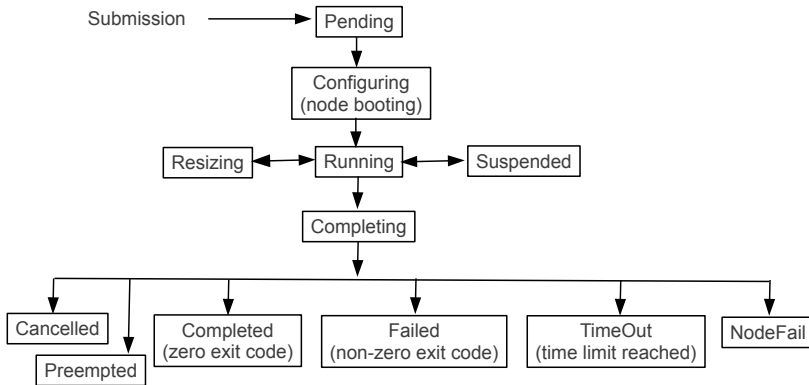
Node State Information

- NUMA boards, Sockets, Cores, Threads
- CPUs
 - ↪ can treat each core or each thread as a CPU for scheduling purposes
- Memory size
- Temporary disk space
- Features (arbitrary string, e.g. OS version)
- Weight (scheduling priority, . . .)
 - ↪ can favor least capable node that satisfies job requirement
- Boot time
- CPU Load
- State (e.g. drain, down, etc.)
- Reason, time and user ID
 - ↪ e.g. "Bad PDU [operator@12:40:10T12/20/2011]"

Queues/Partitions State Information

- Associated with specific set of nodes
 - ↳ Nodes can be in more than one partition (not the case in `iris`)
- Job size and time limits
- Access control list (by Linux group) / QoS
- Preemption rules
- State information (e.g. drain)
- Over-subscription and gang scheduling rules

Job State



Slurm Daemons

- **slurmctld**: Central controller (typically one per cluster)
 - Optional backup with automatic fail over
 - Monitors state of resources
 - Manages job queues and Allocates resources
- **slurmd**: Compute node daemon
 - typically one per compute node, one or more on front-end nodes
 - Launches and manages tasks
 - Small and very light-weight (low memory and CPU use)

- **Common configuration file**: `/etc/slurm/slurm.conf`
 - Other interesting files: `/etc/slurm/{topology,gres}.conf`

- **slurmdbd**: database daemon (typically one per site)
 - Collects accounting information
 - Uploads configuration information (limits, fair-share, etc.)

Slurm in iris cluster

- Predefined **Queues/Partitions**:

- ↪ batch (Default)

Max: 30 nodes, 5 days walltime

- ↪ interactive

Max: 2 nodes, 4h walltime, 10 jobs

- ↪ long

Max: 2 nodes, 30 days walltime, 10 jobs

- Corresponding Quality of Service (QOS)
- Possibility to run besteffort jobs via the qos-besteffort QOS
- Accounts associated to supervisor (multiple associations possible)
- Proper group/user accounting

Slurm Job Management

- User jobs have the following key characteristics:
 - ↪ set of requested resources:
 - ✓ number of computing resources: **nodes** (including all their CPUs and cores) or **CPUs** (including all their cores) or **cores**
 - ✓ amount of **memory**: either per node or per CPU
 - ✓ **(wall)time** needed for the user's tasks to complete their work
 - ↪ a requested node **partition** (job queue)
 - ↪ a requested **quality of service** (QoS) level which grants users specific accesses
 - ↪ a requested **account** for accounting purposes

By default...

- users submit jobs to a particular partition, **and** under a particular account (pre-set per user).

Slurm Commands: General Info

- Man pages available for all commands, daemons and config. files
 - ↪ `--help` option prints brief description of all options
 - ↪ `--usage` option prints a list of the options
 - ↪ `-v` | `-vv` | `-vvv`: verbose output
- Commands can be run on any node in the cluster
- Any failure results in a non-zero exit code
- APIs make new tool development easy
 - ↪ Man pages available for all APIs
- Almost all options have two formats
 - ↪ A single letter option (e.g. `-p batch` for partition 'batch')
 - ↪ A verbose option (e.g. `--partition=batch`)
- Time formats: `DD-HH:MM::SS`

User Commands: Job/step Allocation

- **sbatch**: Submit script for later execution (batch mode)
 - ↪ allocate resources (nodes, tasks, partition, etc.)
 - ↪ Launch a script containing `sruns` for series of steps on them.
- **salloc**: Create job allocation & start a shell to use it
 - ↪ allocate resources (nodes, tasks, partition, etc.),
 - ↪ either run a command or start a shell.
 - ↪ Request launch `srun` from shell. (interactive commands within one allocation)
- **srun**: Create a job allocation (if needed) and launch a job step (typically an MPI job)
 - ↪ allocate resources (number of nodes, tasks, partition, constraints, etc.)
 - ↪ launch a job that will execute on them.
- **sattach**: attach to running job for debuggers



User & Admin Commands: System Information

- **sinfo**: Report system status (nodes, partitions etc.)
- **squeue**: display jobs[steps] and their state
- **scancel**: cancel a job or set of jobs.
- **scontrol**: view and/or update system, nodes, job, step, partition or reservation status
- **sstat**: show status of running jobs.
- **sacct**: display accounting information on jobs.
- **sprio**: show factors that comprise a jobs scheduling priority
- **smap**: graphically show information on jobs, nodes, partitions
 - ↪ not available on iris

Slurm Admin Commands

- **sacctmgr**: setup accounts, specify limitations on users and groups.
- **sshare**: view sharing information from multifactor plugin.
- **sreport**: display information from accounting database on jobs, users, clusters.
- **sview**: graphical view of cluster. Display and change characteristics of jobs, nodes, partitions.
 - ↪ not yet available on iris cluster
- **strigger**: show, set, clear event triggers. Events are usually system events such as an equipment failure.

Slurm vs. OAR Main Commands

Action	SLURM command	OAR Command
Submit passive/batch job	<code>sbatch [...] \$script</code>	<code>oarsub [...] \$script</code>
Start interactive job	<code>srun [...] --pty bash</code>	<code>oarsub -I [...]</code>
Queue status	<code>squeue</code>	<code>oarstat</code>
User job status	<code>squeue -u \$user</code>	<code>oarstat -u \$user</code>
Specific job status (detailed)	<code>scontrol show job \$jobid</code>	<code>oarstat -f -j \$jobid</code>
Job accounting status (detailed)	<code>sacct --job \$jobid -l</code>	
Delete (running/waiting) job	<code>scancel \$jobid</code>	<code>oardel \$jobid</code>
Hold job	<code>scontrol hold \$jobid</code>	<code>oarhold \$jobid</code>
Resume held job	<code>scontrol release \$jobid</code>	<code>oarresume \$jobid</code>
Node list and their properties	<code>scontrol show nodes</code>	<code>oarnodes</code>

Job Specifications

Specification	SLURM	OAR
Script directive	#SBATCH	#OAR
<n> Nodes request	-N <n>	-l nodes=<n>
<n> Cores/Tasks request	-n <n>	-l core=<n>
<c> Cores-per-node request	--ntasks-per-node=<c>	-l nodes=<n>/core=<c>
<c> Cores-per-task request (multithreading)	-c<c>	
<m> GB memory per node request	--mem=<m>GB	
Walltime request	-t <mm>/<days>-hh[:mm:ss]	-l walltime=hh[:mm:ss]
Job array	--array <specification>	--array <count>
Job name	-J <name>	-n <name>
Job dependency	-d <specification>	-a <jobid>
Property request	-C <specification>	-p "<property>=<value>"
Specify job partition/queue	-p <partition>	-t <queue>
Specify job qos	--qos <qos>	
Specify account	-A <account>	
Specify email address	--mail-user=<email>	--notify "mail:<email>"

Typical Workflow

```
# Run an interactive job -- make an alias 'si [...]'
$> srun -p interactive --qos qos-interactive --pty bash
# Ex: interactive job for 30 minutes, with 2 nodes/4 tasks per node
$> si --time=0:30:0 -N 2 --ntasks-per-node=4
# Run a [passive] batch job -- make an alias 'sb [...]'
$> sbatch -p batch --qos qos-batch /path/to/launcher.sh
# Will create (by default) slurm-<jobid>.out file
```

Environment variable	SLURM	OAR
Job ID	<code>\$SLURM_JOB_ID</code>	<code>\$OAR_JOB_ID</code>
Resource list	<code>\$SLURM_NODENAME</code> #List not file!	<code>\$OAR_NODEFILE</code>
Job name	<code>\$SLURM_JOB_NAME</code>	<code>\$OAR_JOB_NAME</code>
Submitting user name	<code>\$SLURM_JOB_USER</code>	<code>\$OAR_USER</code>
Task ID within job array	<code>\$SLURM_ARRAY_TASK_ID</code>	<code>\$OAR_ARRAY_INDEX</code>
Working directory at submission	<code>\$SLURM_SUBMIT_DIR</code>	<code>\$OAR_WORKING_DIRECTORY</code>
Number of nodes assigned to the job	<code>\$SLURM_NNODES</code>	
Number of tasks of the job	<code>\$SLURM_NTASKS</code>	<code>\$(wc -l \${OAR_NODEFILE})</code>

- **Note:** create the equivalent of `$OAR_NODEFILE` in Slurm:

```
↪ srun hostname | sort -n > hostfile
```

Available Node partitions

- Slurm Command Option `-p, --partition=<partition>`
 ↪ **Ex:** `{srun,sbatch} -p batch [...]`
- Date format: `-t <minutes>` or `-t <D>-<H>:<M>:<S>`

Partition	#Nodes	Default time	Max time	Max nodes/user
batch	80%	0-2:0:0 [2h]	5-0:0:0 [5d]	unlimited
interactive	10%	0-1:0:0 [1h]	0-4:0:0 [4h]	2
long	10%	0-2:0:0 [2h]	30-0:0:0 [30d]	2

Quality of Service (QoS)

- Slurm Command Option `--qos=<qos>`
- **There is no default QOS** (due to the selected scheduling model)
 - you **MUST** provide upon **any job submission**
 - a default qos is guessed from the partition **i.e.** `qos-<partition>`

QoS	User group	Max nodes	Max jobs/user	Description
qos-besteffort	ALL	no limit		Preemptible jobs, requested on preemption
qos-batch	ALL	30	100	Normal usage of the batch partition
qos-interactive	ALL	8	10	Normal usage of the interactive partition
qos-long	ALL	8	10	Normal usage of the long partition
qos-batch-###	TBD	TBD	100	Special usage of the batch partition
qos-interactive-###	TBD	TBD	10	Special usage of the interactive partition
qos-long-###	TBD	TBD	10	Special usage of the long partition

Accounts

- Every user job runs under a group account
 ↳ granting access to specific QOS levels.

Account	Parent Account
UL	
FSTC	UL
FDEF	UL
FLSHASE	UL
LCSB	UL
SNT	UL
Professor \$X	<i>FACULTY/IC</i>
Group head \$G	<i>FACULTY/IC</i>
Researcher \$R	Professor \$X
Researcher \$R	Group head \$G
Student \$S	Professor \$X
Student \$S	Group head \$G
External collaborator \$E	Professor \$X
External collaborator \$E	Group head \$G

```
$> sacctmgr list associations where users=$USER \
      format=Account%30s,User,Partition,QOS
```

Other Features

- **Checkpoint / Restart**

- ↪ Based on DMTCP: Distributed MultiThreaded CheckPointing
- ↪ see the official DMTCP launchers
- ↪ ULHPC example

- Many metrics can be extracted from user jobs

- ↪ with SLURM's own tools (sacct/sstat)
- ↪ within the jobs with e.g. **PAPI**
- ↪ easy to bind executions with Allinea Performance Report

- Advanced admission rules

- ↪ to simplify CLI

- Container **Shifter** / **Singularity**

- ↪ Work in progress, not yet available on **iris**



Simple Example of usage

```
# Simple interactive job
(access)$> srun -p interactive [--qos qos-interactive] --pty bash
(node)$> echo $SLURM_NTASKS
1
(node)$> echo $SLURM_JOBID
59900
```


Simple Example of usage

```
# Simple interactive job  
(access)$> srun -p interactive [--qos qos-interactive] --pty bash  
(node)$> echo $SLURM_NTASKS  
1  
(node)$> echo $SLURM_JOBID  
59900
```

```
$> squeue -u $USER -l    # OR 'sq'
```

Simple Example of usage

```
# Simple interactive job
(access)$> srun -p interactive [--qos qos-interactive] --pty bash
(node)$> echo $SLURM_NTASKS
1
(node)$> echo $SLURM_JOBID
59900
```

```
$> squeue -u $USER -l # OR 'sq'
```

- Many metrics during (scontrol)/after job execution (sacct)
 - ↳ including energy (J) – but with caveats
 - ↳ job **steps** counted individually
 - ↳ enabling advanced application debugging and optimization
- Job information available in easily parseable format (add -p/-P)

Live Job Statistics

```
$> scontrol show job 59900
JobId=59900 JobName=bash
UserId=<login>(<uid>) GroupId=clusterusers(666) MCS_label=N/A
Priority=6627 Nice=0 Account=ulhpc QOS=qos-interactive
JobState=RUNNING Reason=None Dependency=(null)
RunTime=00:04:19 TimeLimit=01:00:00 TimeMin=N/A
SubmitTime=2017-10-22T23:07:02 EligibleTime=2017-10-22T23:07:02
StartTime=2017-10-22T23:07:02 EndTime=2017-10-23T00:07:02 Deadline=N/A
PreemptTime=None SuspendTime=None SecsPreSuspend=0
Partition=interactive AllocNode:Sid=access1:72734
[...]
NodeList=iris-002
NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
TRES=cpu=1,mem=4G,node=1
Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
MinCPUsNode=1 MinMemoryCPU=4G MinTmpDiskNode=0
[...]
Command=bash
WorkDir=/mnt/irisgpfs/users/<login>
```

Node/Job Statistics

```
$> sinfo
PARTITION  AVAIL  TIMELIMIT  NODES  STATE NODELIST
interactive up    4:00:00    10   idle iris-[001-010]
long       up   30-00:00:0    2   resv iris-[019-020]
long       up   30-00:00:0    8   idle iris-[011-018]
batch*     up   5-00:00:00    5   mix  iris-[055,060-062,101]
batch*     up   5-00:00:00   13  alloc iris-[053-054,056-059,102-108]
batch*     up   5-00:00:00   70   idle iris-[021-052,063-100]
```

Node/Job Statistics

```
$> sinfo
```

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
interactive	up	4:00:00	10	idle	iris-[001-010]
long	up	30-00:00:0	2	resv	iris-[019-020]
long	up	30-00:00:0	8	idle	iris-[011-018]
batch*	up	5-00:00:00	5	mix	iris-[055,060-062,101]
batch*	up	5-00:00:00	13	alloc	iris-[053-054,056-059,102-108]
batch*	up	5-00:00:00	70	idle	iris-[021-052,063-100]

```
$> sacct --format=account,user,jobid,jobname,partition,state -j <JOBID>
```

```
$> sacct --format=elapsed,elapseddraw,start,end -j <JOBID>
```

```
$> sacct -j <JOBID> --format=maxrss,maxvmsize,consumedenergy,\
      consumedenergyraw,nodelist
```



Playing with hostname and task ID label

```
$> srun [-N #node] [-n #task] [--ntasks-per-node #n] [] CMD
```

```
# -n: #tasks
```

```
$> srun -n 4 -l hostname
```

```
1: iris-055
```

```
2: iris-055
```

```
3: iris-055
```

```
0: iris-055
```

Playing with hostname and task ID label

```
$> srun [-N #node] [-n #task] [--ntasks-per-node #n] [] CMD
```

-n: #tasks

```
$> srun -n 4 -l hostname
```

```
1: iris-055
```

```
2: iris-055
```

```
3: iris-055
```

```
0: iris-055
```

-N: #nodes

```
$> srun -N 4 -l hostname
```

```
3: iris-058
```

```
2: iris-057
```

```
1: iris-056
```

```
0: iris-055
```

Playing with hostname and task ID label

```
$> srun [-N #node] [-n #task] [--ntasks-per-node #n] [] CMD
```

-n: #tasks

```
$> srun -n 4 -l hostname
```

```
1: iris-055
```

```
2: iris-055
```

```
3: iris-055
```

```
0: iris-055
```

-c: #cpus/task ~#thread/task

```
$> srun -c 4 -l hostname
```

```
0: iris-055
```

-N: #nodes

```
$> srun -N 4 -l hostname
```

```
3: iris-058
```

```
2: iris-057
```

```
1: iris-056
```

```
0: iris-055
```


Playing with hostname and task ID label

```
$> srun [-N #node] [-n #task] [--ntasks-per-node #n] [] CMD
```

-n: #tasks

```
$> srun -n 4 -l hostname  
1: iris-055  
2: iris-055  
3: iris-055  
0: iris-055
```

-N: #nodes

```
$> srun -N 4 -l hostname  
3: iris-058  
2: iris-057  
1: iris-056  
0: iris-055
```

-c: #cpus/task ~#thread/task

```
$> srun -c 4 -l hostname  
0: iris-055
```

```
$> srun -N 2 -n 4 -l hostname
```

```
3: iris-056
```

```
0: iris-055
```

```
1: iris-055
```

```
2: iris-055
```

```
$> srun -N 2 --ntasks-per-node 2 -l hostname
```

```
3: iris-056
```

```
2: iris-056
```

```
1: iris-055
```

```
0: iris-055
```

Job submission with salloc

```
$> salloc [-N #node] [-n #task] [--ntasks-per-node #n]
```

```
$> salloc -N 4
salloc: Granted job allocation 59955
salloc: Waiting for resource configuration
salloc: Nodes iris-[055,060-062] are ready for job
$> env | grep SLURM
$> hostname
access1.iris-cluster.uni.lux
$> srun -l hostname
0: iris-055
2: iris-061
1: iris-060
3: iris-062
```



Reservations and scontrol features

```
$> scontrol show job <JOBID>
```

Job info

Reservations and scontrol features

```
$> scontrol show job <JOBID>
```

Job info

```
$> scontrol show {partition,topology}
```

Reservations and scontrol features

```
$> scontrol show job <JOBID> # Job info
```

```
$> scontrol show {partition, topology}
```

```
$> scontrol show reservations # Show existing reservations
```

```
$> scontrol create reservation
ReservationName=<name>
accounts=<account_list>                               licenses=<license>
corecnt=<num>                                           nodecnt=<count>
duration=[days-]hours:minutes:seconds                nodes=<node_list>
endtime=yyyy-mm-dd[thh:mm[:ss]]                       partitionname=<partition(s)>
features=<feature_list>                                starttime=yyyy-mm-dd[thh:mm[:ss]]
flags=maint,overlap,ignore_jobs,daily,weekly          users=<user_list>
```

Basic Slurm Launcher Examples

Documentation

https://hpc.uni.lu/users/docs/slurm_launchers.html

See also **PS1**, **PS2** and **PS3**

```
#!/bin/bash -l
# Request one core for 5 minutes in the batch queue

#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --time=0-00:05:00
#SBATCH -p batch
#SBATCH --qos=qos-batch
```

[...]

Basic Slurm Launcher Examples (cont.)

```
#!/bin/bash -l
# Request two cores on each of two nodes for 3 hours

#SBATCH -N 2
#SBATCH --ntasks-per-node=2
#SBATCH --time=0-03:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"

[...]
```

Basic Slurm Launcher Examples (cont.)

```
#!/bin/bash -l
# Request one core and half the memory available on an iris cluster
# node for one day
#
#SBATCH -J MyLargeMemorySequentialJob
#SBATCH --mail-type=end,fail
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --mem=64GB
#SBATCH --time=1-00:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
```


pthread/OpenMP Slurm Launcher

```
#!/bin/bash -l
# Single node, threaded (pthread/OpenMP) application launcher,
# using all 28 cores of an iris cluster node:

#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -c 28
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
/path/to/your/threaded.app
```

MATLAB Slurm Launcher

```
#!/bin/bash -l
# Single node, multi-core parallel application (MATLAB, Python, R...)
# launcher, using all 28 cores of an iris cluster node:

#SBATCH -N 1
#SBATCH --ntasks-per-node=28
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load base/MATLAB
matlab -nodisplay -nosplash < /path/to/inputfile > /path/to/outputfile
```

Intel MPI Slurm Launchers

- Official SLURM guide for Intel MPI

```
#!/bin/bash -l
# Multi-node parallel application IntelMPI launcher,
# using 128 distributed cores:

#SBATCH -n 128
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/intel
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
srun -n $SLURM_NTASKS /path/to/your/intel-toolchain-compiled-application
```

OpenMPI Slurm Launchers

- Official SLURM guide for Open MPI

```
#!/bin/bash -l
# Multi-node parallel application openMPI launcher,
# using 128 distributed cores:

#SBATCH -n 128
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/foss
mpirun -n $SLURM_NTASKS /path/to/your/foss-toolchain-compiled-application
```

Hybrid IntelMPI+OpenMP Launcher

```
#!/bin/bash -l
# Multi-node hybrid application IntelMPI+OpenMP launcher,
# using 28 threads per node on 10 nodes (280 cores):

#SBATCH -N 10
#SBATCH --ntasks-per-node=1
#SBATCH -c 28
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

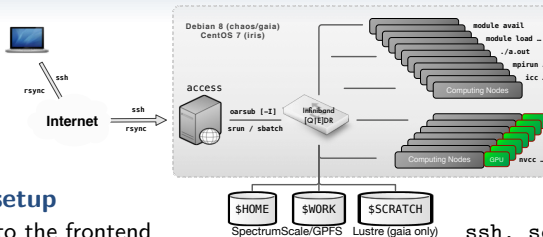
module load toolchain/intel
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
srun -n $SLURM_NTASKS /path/to/your/parallel-hybrid-app
```



Summary

- 1 Introduction
- 2 The OAR Batch Scheduler
- 3 The SLURM Batch Scheduler
 - Overview
 - Slurm commands
 - Slurm vs. OAR commands
 - ULHPC Slurm Configuration
 - Usage Example
 - Slurm Launcher
- 4 Conclusion

Typical Workflow on UL HPC resources



• Preliminary setup

① Connect to the frontend

② Synchronize you code

③ Reserve a few interactive resources

or,

✓ (eventually) build your program

✓ Test on small size problem

✓ Prepare a launcher script

ssh, screen

scp/rsync/svn/git

oarsub -I [...]

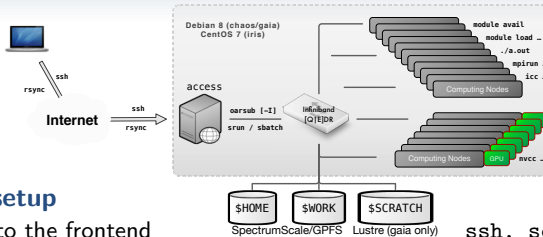
on iris: srun -p interactive [...]

gcc/icc/mpicc/nvcc..

mpirun/srun/python/sh...

<launcher>.{sh|py}

Typical Workflow on UL HPC resources



• Preliminary setup

① Connect to the frontend

② Synchronize you code

③ Reserve a few interactive resources

or,

✓ (eventually) build your program

✓ Test on small size problem

✓ Prepare a launcher script

ssh, screen

scp/rsync/svn/git

oarsub -I [...]

on iris: srun -p interactive [...]

gcc/icc/mpicc/nvcc..

mpirun/srun/python/sh...

<launcher>.{sh|py}

• Real Experiment

① Reserve passive resources

oarsub [...] <launcher>

or,

on iris: sbatch -p {batch|long} [...] <launcher>

② Grab the results

scp/rsync/svn/git

Questions?

<http://hpc.uni.lu>

High Performance Computing @ UL

Prof. Pascal Bouvry

Dr. Sebastien Varrette & the UL HPC Team

(V. Plugaru, S. Peter, H. Cartiaux & C. Parisot)

University of Luxembourg, Belval Campus

Maison du Nombre, 4th floor

2, avenue de l'Université

L-4365 Esch-sur-Alzette

mail: hpc@uni.lu



- 1 Introduction
- 2 The OAR Batch Scheduler
- 3 The SLURM Batch Scheduler
 - Overview
 - Slurm commands
 - Slurm vs. OAR commands
 - ULHPC Slurm Configuration
 - Usage Example
 - Slurm Launcher
- 4 Conclusion