

Uni.lu HPC School 2019

PS3: [Advanced] Job scheduling (SLURM)



Uni.lu High Performance Computing (HPC) Team

C. Parisot

University of Luxembourg (UL), Luxembourg

<http://hpc.uni.lu>



Latest versions available on **Github**:



UL HPC tutorials:

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

UL HPC School:

<http://hpc.uni.lu/hpc-school/>

PS3 tutorial sources:

ulhpc-tutorials.rtf.d.io/en/latest/scheduling/advanced/





Summary

- 1 Introduction**
- 2 SLURM workload manager
SLURM concepts and design for iris
Running jobs with SLURM
- 3 OAR and SLURM
- 4 Conclusion



Main Objectives of this Session



- **Design and usage of SLURM**
 - ↪ cluster workload manager of the UL HPC iris cluster
 - ↪ ... and future HPC systems

The tutorial will show you:

- the way SLURM was **configured, accounting** and **permissions**
- **common** and **advanced** SLURM tools and commands
 - ↪ `srun`, `sbatch`, `squeue` etc.
 - ↪ job specification
 - ↪ SLURM job types
 - ↪ comparison of SLURM (`iris`) and OAR (`gaia` & `chaos`)
- SLURM generic **launchers** you can use for your own jobs

Documentation & comparison to OAR

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



Summary

- 1 Introduction
- 2 SLURM workload manager**
SLURM concepts and design for *iris*
Running jobs with SLURM
- 3 OAR and SLURM
- 4 Conclusion

SLURM - core concepts

- SLURM manages user jobs with 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**
 - ✓ number of accelerators (**GPUs**)
 - ✓ amount of **memory**: either per node or per (logical) CPU
 - ✓ the **(wall)time** needed for the user's tasks to complete their work
 - ↪ a set of **constraints** limiting jobs to nodes with specific features
 - ↪ a requested node **partition** (job queue)
 - ↪ a requested **quality of service** (QoS) level which grants users specific accesses
 - ↪ a requested **account** for accounting purposes
- **Example**: run an interactive job Alias: `si [...]`

```
(access)$ srun -p interactive --qos qos-interactive --pty bash -i  
(node)$ echo $SLURM_JOBID  
2058
```

Simple interactive job running under SLURM



SLURM - job example (I)

```
$ scontrol show job 2058
JobId=2058 JobName=bash
  UserId=vplugaru(5143) GroupId=clusterusers(666) MCS_label=N/A
  Priority =100 Nice=0 Account=ulhpc QOS=qos-interactive
5  JobState=RUNNING Reason=None Dependency=(null)
  Requeue=1 Restarts=0 BatchFlag=0 Reboot=0 ExitCode=0:0
  RunTime=00:00:08 TimeLimit=00:05:00 TimeMin=N/A
  SubmitTime=2017-06-09T16:49:42 EligibleTime=2017-06-09T16:49:42
10 StartTime=2017-06-09T16:49:42 EndTime=2017-06-09T16:54:42 Deadline=N/A
  PreemptTime=None SuspendTime=None SecsPreSuspend=0
  Partition = interactive AllocNode:Sid=access2:163067
  ReqNodeList=(null) ExcNodeList=(null)
  NodeList=iris-081
  BatchHost=iris-081
15 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=1:0:*:* CoreSpec=*
  MinCPUsNode=1 MinMemoryCPU=4G MinTmpDiskNode=0
  Features=(null) DelayBoot=00:00:00
20 Gres=(null) Reservation=(null)
  OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
  Command=bash
  WorkDir=/mnt/irisgpf/users/vplugaru
  Power=
```

Simple interactive job running under SLURM

SLURM - job example (II)

- Many metrics available during and after job execution
 - ↪ 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)

```

$ sacct -j 2058 --format=account,user,jobid,jobname,partition,state
  Account  User      JobID  JobName  Partition  State
    ulhpc  vplugaru    2058    bash  interacti + COMPLETED

5 $ sacct -j 2058 --format=elapsed,elapsedraw,start,end
  Elapsed ElapsedRaw      Start      End
  00:02:56      176 2017-06-09T16:49:42 2017-06-09T16:52:38

10 $ sacct -j 2058 --format=maxrss,maxvmsize,consumedenergy,consumedenergyraw,nnodes,ncpus,nodelist
  MaxRSS MaxVMSize ConsumedEnergy ConsumedEnergyRaw NNodes NCPUS  NodeList
    0      299660K      17.89K      17885.000000      1      1      iris -081
  
```

Job metrics after execution ended



SLURM - design for iris (I)

Partition	# Nodes	Default time	Max time	Max nodes/user
batch*	152	0-2:0:0	5-0:0:0	unlimited
bigmem	4	0-2:0:0	5-0:0:0	unlimited
gpu	24	0-2:0:0	5-0:0:0	unlimited
interactive	8	0-1:0:0	0-4:0:0	2
long	8	0-2:0:0	30-0:0:0	2



SLURM - design for iris (I)

Partition	# Nodes	Default time	Max time	Max nodes/user
batch*	152	0-2:0:0	5-0:0:0	unlimited
bigmem	4	0-2:0:0	5-0:0:0	unlimited
gpu	24	0-2:0:0	5-0:0:0	unlimited
interactive	8	0-1:0:0	0-4:0:0	2
long	8	0-2:0:0	30-0:0:0	2

QoS	Max cores	Max jobs/user
qos-besteffort	no limit	
qos-batch	2344	100
qos-bigmem	no limit	10
qos-gpu	no limit	10
qos-interactive	168	10
qos-long	168	10



SLURM - desing for iris (II)

You have some **private** QoS not accessible to all users.

QoS	User group	Max cores	Max jobs/user
qos-besteffort	ALL	no limit	
qos-batch	ALL	2344	100
qos-batch-001	private	1400	100
qos-batch-002	private	256	100
qos-batch-003	private	256	100
qos-bigmem	ALL	no limit	10
qos-gpu	ALL	no limit	10
qos-interactive	ALL	168	10
qos-interactive-001	private	56	10
qos-long	ALL	168	10
qos-long-001	private	56	10



SLURM - design for iris (III)

- **Default partition: batch**, meant to receive most user jobs
 - ↪ we hope to see majority of user jobs being able to scale
 - ↪ shorter walltime jobs highly encouraged
- All partitions have a correspondingly named **QOS**
 - ↪ granting resource access (**long** : **qos-long**)
 - ↪ any job is tied to one **QOS** (user specified or inferred)
 - ↪ automation in place to select **QOS** based on partition
 - ↪ jobs may wait in the queue with **QOS*Limit** reason set
 - ✓ e.g. **QOSGrpCpuLimit** if group limit for CPUs was reached

SLURM - design for iris (III)

- **Default partition: batch**, meant to receive most user jobs
 - ↪ we hope to see majority of user jobs being able to scale
 - ↪ shorter walltime jobs highly encouraged
- All partitions have a correspondingly named **QOS**
 - ↪ granting resource access (**long** : **qos-long**)
 - ↪ any job is tied to one **QOS** (user specified or inferred)
 - ↪ automation in place to select **QOS** based on partition
 - ↪ jobs may wait in the queue with **QOS*Limit** reason set
 - ✓ e.g. **QOSGrpCpuLimit** if group limit for CPUs was reached
- Preemptible **besteffort** QOS available for **batch** and **interactive** partitions (but not yet for **bigmem**, **gpu** or **long**)
 - ↪ meant to ensure maximum resource utilization especially on batch
 - ↪ should be used together with restartable software
- QOSs specific to particular group accounts exist (discussed later)
 - ↪ granting additional accesses to platform contributors

SLURM - design for iris (IV)

- **Backfill** scheduling for efficiency
 - ↪ **multifactor job priority** (size, age, fair share, QOS, ...)
 - ↪ currently weights set for: job age, partition and fair share
 - ↪ other factors/decay to be tuned **as needed**
 - ✓ with more user jobs waiting in the queues
- Resource selection: **consumable resources**
 - ↪ **cores and memory** as consumable (per-core scheduling)
 - ↪ **GPUs** as consumable (4 GPUs per node in the gpu partition)
 - ↪ block distribution for cores (best-fit algorithm)
 - ↪ default memory/core: 4GB (4.1GB maximum, rest is for OS)
 - ✓ gpu and bigmem partitions: 27GB maximum

SLURM - design for iris (IV)

- **Backfill** scheduling for efficiency
 - ↳ **multifactor job priority** (size, age, fair share, QOS, ...)
 - ↳ currently weights set for: job age, partition and fair share
 - ↳ other factors/decay to be tuned **as needed**
 - ✓ with more user jobs waiting in the queues
- Resource selection: **consumable resources**
 - ↳ **cores and memory** as consumable (per-core scheduling)
 - ↳ **GPUs** as consumable (4 GPUs per node in the gpu partition)
 - ↳ block distribution for cores (best-fit algorithm)
 - ↳ default memory/core: 4GB (4.1GB maximum, rest is for OS)
 - ✓ gpu and bigmem partitions: 27GB maximum
- User process tracking with **cgroups**
 - ↳ cpusets used to constrain cores and RAM (no swap allowed)
 - ↳ task affinity used to bind tasks to cores (hwloc based)
- Hierarchical tree topology defined (for the network)
 - ↳ for optimized job resource allocation

SLURM - design for iris (IV)

- **Backfill** scheduling for efficiency
 - ↳ **multifactor job priority** (size, age, fair share)
 - ↳ currently weights set for: job age, priority, fair share
 - ↳ other factors/decay to be tuned
 - ✓ with more user jobs waiting
- Resource selection: **consumable**
 - ↳ **cores and memory** as consumable (for more scheduling)
 - ↳ **GPUs** as consumable (for more scheduling in the gpu partition)
 - ↳ block distribution (for more scheduling algorithm)
 - ↳ default memory (for more scheduling algorithm) (1GB maximum, rest is for OS)
 - ✓ gpu and memory (for more scheduling algorithm) (27GB maximum)
- User process to **groups**
 - ↳ cpuset (for more scheduling algorithm) (no swap allowed)
 - ↳ task (for more scheduling algorithm) (hwloc based)
- Hierarchical topology defined (for the network)
 - ↳ for more scheduling algorithm (job resource allocation)

Help will be needed on your part to optimize your job parameters!

A note on job priority

```
Job_priority =  
  (PriorityWeightAge) * (age_factor) +  
  (PriorityWeightFairshare) * (fair-share_factor) +  
  (PriorityWeightJobSize) * (job_size_factor) +  
  (PriorityWeightPartition) * (partition_factor) +  
  (PriorityWeightQOS) * (QOS_factor) +  
  SUM(TRES_weight_cpu * TRES_factor_cpu,  
      TRES_weight_<type> * TRES_factor_<type>,  
      ...)
```

For complete details see: slurm.schedmd.com/priority_multifactor.html

- **TRES** - Trackable RESources
 - ↪ CPU, Energy, Memory and Node tracked by default
- **GRES** - Generic RESources
 - ↪ GPU
- Corresponding weights/reset periods **tuned with your feedback**
 - ↪ we require (your & your group's) usage pattern to optimize them
 - ↪ the target is high interactivity (low time spent by the jobs waiting)



SLURM - design for iris (V)

Some details on job permissions...

- Partition limits + association-based rule enforcement
↳ association settings in SLURM's accounting database
- **QOS** limits imposed, e.g. you will see (QOSGrpCpuLimit)
- Only users with existing **associations** able to run jobs



SLURM - design for iris (V)

Some details on job permissions...

- Partition limits + association-based rule enforcement
 - ↳ association settings in SLURM's accounting database
 - **QOS** limits imposed, e.g. you will see (QOSGrpCpuLimit)
 - Only users with existing **associations** able to run jobs
-
- **Best-effort** jobs possible through preemptible QOS: **qos-besteffort**
 - ↳ of lower priority and preemptible by all other QOS
 - ↳ preemption mode is **requeue**, requeueing **disabled** by default

SLURM - design for iris (V)

Some details on job permissions...

- Partition limits + association-based rule enforcement
 - ↳ association settings in SLURM's accounting database
 - **QOS** limits imposed, e.g. you will see (QOSGrpCpuLimit)
 - Only users with existing **associations** able to run jobs
-
- **Best-effort** jobs possible through preemptible QOS: **qos-besteffort**
 - ↳ of lower priority and preemptible by all other QOS
 - ↳ preemption mode is **requeue**, requeueing **disabled** by default
 - **On metrics**: Accounting & profiling data for jobs sampled every 30s
 - ↳ tracked: cpu, mem, energy
 - ↳ energy data retrieved through the **RAPL mechanism**
 - ✓ **caveat**: for energy not all hw. that may consume power is monitored with RAPL (CPUs, GPUs and DRAM are included)



SLURM - design for iris (VI)

- **On tightly coupled parallel jobs (MPI)**
 - ↳ Process Management Interface (PMI 2) highly recommended
 - ↳ PMI2 used for better scalability and performance
 - ✓ faster application launches
 - ✓ tight integration w. SLURM's job steps mechanism (& metrics)
 - ✓ we are also testing **PMIx** (PMI Exascale) support



SLURM - design for iris (VI)

- **On tightly coupled parallel jobs (MPI)**

- ↪ Process Management Interface (PMI 2) highly recommended
- ↪ PMI2 used for better scalability and performance
 - ✓ faster application launches
 - ✓ tight integration w. SLURM's job steps mechanism (& metrics)
 - ✓ we are also testing **PMIx** (PMI Exascale) support
- ↪ PMI2 enabled in default software set for IntelMPI and OpenMPI
 - ✓ requires minimal adaptation in your workflows
 - ✓ (at minimum:) replace **mpirun** with SLURM's **srun**
 - ✓ if you compile/install your own MPI you'll need to configure it
- ↪ **Many examples at:** https://hpc.uni.lu/users/docs/slurm_launchers.html

SLURM - design for iris (VI)

- **On tightly coupled parallel jobs (MPI)**

- ↳ Process Management Interface (PMI 2) highly recommended
- ↳ PMI2 used for better scalability and performance
 - ✓ faster application launches
 - ✓ tight integration w. SLURM's job steps mechanism (& metrics)
 - ✓ we are also testing **PMIx** (PMI Exascale) support
- ↳ PMI2 enabled in default software set for IntelMPI and OpenMPI
 - ✓ requires minimal adaptation in your workflows
 - ✓ (at minimum:) replace **mpirun** with SLURM's **srun**
 - ✓ if you compile/install your own MPI you'll need to configure it
- ↳ **Many examples at:** https://hpc.uni.lu/users/docs/slurm_launchers.html

- **SSH-based connections** between computing nodes still **possible**

- ↳ other MPI implementations can still use `ssh` as launcher
 - ✓ but **really shouldn't need to**, PMI2 support is everywhere
- ↳ user jobs are tracked, no job == no access to node

SLURM - design for iris (VII)

ULHPC customization through plugins

- Job submission rule / filter
 - ↳ for now: QOS initialization (if needed)
 - ↳ more rules to come (group credits, node checks, etc.)
- Per-job temporary directories creation & cleanup
 - ↳ better security and privacy, using kernel namespaces and binding
 - ↳ `/tmp` & `/var/tmp` are `/tmp/$jobid.$rstcnt/[tmp,var_tmp]`
 - ↳ transparent for apps. ran through `srun`
 - ↳ **apps. ran with `ssh` cannot be attached, will see base `/tmp`!**
- X11 forwarding (GUI applications)
 - ↳ **Some issue prevents us to use `'-x11'` option of SLURM on iris**
 - ✓ workaround in the tutorial/FAQ
 - ✓ create job using `salloc` and then use `ssh -Y`



SLURM - design for iris (VIII)

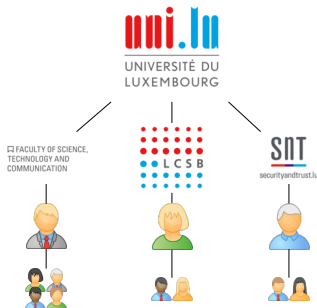
Software licenses in SLURM

- ARM (ex. Allinea) Forge and Performance Reports for now
 - ↳ static allocation in SLURM configuration
 - ↳ dynamic checks for FlexNet / RLM based apps. coming later
- Number and utilization state can be checked with:
 - ↳ `scontrol show licenses`
- Use not enforced, **honor system** applied
 - ↳ `srun [...] -L $licname:$licnumber`

```
$> srun -N 1 -n 28 -p interactive -L forge:28 --pty bash -i
```

SLURM - bank (group) accounts

- Hierarchical **bank (group) accounts**
- UL as root account, then underneath accounts for the 3 Faculties and 3 ICs
- All Prof., Group leaders and above have **bank accounts**, linked to a Faculty or IC
 - ↳ with their own name: **Name.Surname**
- All **user accounts** linked to a bank account
 - ↳ including Profs.'s own user
- Iris accounting DB contains over
 - ↳ 103 group accounts from Faculties, ICs & Externals
 - ↳ comprising 877 users



{ **Allows better usage tracking and reporting than was possible before.** }



SLURM - brief commands overview

- **squeue**: view queued jobs
- **sinfo**: view partition and node info.
- **sbatch**: submit job for batch (scripted) execution
- **srun**: submit interactive job, run (parallel) job step
- **scancel**: cancel queued jobs



SLURM - brief commands overview

- **squeue**: view queued jobs
 - **sinfo**: view partition and node info.
 - **sbatch**: submit job for batch (scripted) execution
 - **srun**: submit interactive job, run (parallel) job step
 - **scancel**: cancel queued jobs
-
- **scontrol**: detailed control and info. on jobs, queues, partitions
 - **sstat**: view system-level utilization (memory, I/O, energy)
 - ↳ for running jobs / job steps
 - **sacct**: view system-level utilization
 - ↳ for completed jobs / job steps (accounting DB)
 - **sacctmgr**: view and manage SLURM accounting data

SLURM - brief commands overview

- **squeue**: view queued jobs
 - **sinfo**: view partition and node info.
 - **sbatch**: submit job for batch (scripted) execution
 - **srun**: submit interactive job, run (parallel) job step
 - **scancel**: cancel queued jobs
-
- **scontrol**: detailed control and info. on jobs, queues, partitions
 - **sstat**: view system-level utilization (memory, I/O, energy)
 - ↳ for running jobs / job steps
 - **sacct**: view system-level utilization
 - ↳ for completed jobs / job steps (accounting DB)
 - **sacctmgr**: view and manage SLURM accounting data
-
- **sprio**: view job priority factors
 - **sshare**: view accounting share info. (usage, fair-share, etc.)



SLURM - basic commands

Action	SLURM command
Submit passive/batch job	<code>sbatch \$script</code>
Start interactive job	<code>srun --pty bash -i</code>
Queue status	<code>squeue</code>
User (own) jobs status	<code>squeue -u \$USER</code>
Specific job status (detailed)	<code>scontrol show job \$jobid</code>
Job metrics (detailed)	<code>sstat --job \$jobid -l</code>
Job accounting status (detailed)	<code>sacct --job \$jobid -l</code>
Job efficiency report	<code>seff \$jobid</code>
Delete (running/waiting) job	<code>scancel \$jobid</code>
Hold job	<code>scontrol hold \$jobid</code>
Resume held job	<code>scontrol release \$jobid</code>
Node list and their properties	<code>scontrol show nodes</code>
Partition list, status and limits	<code>sinfo</code>
Attach to running job	<code>sjoin \$jobid [\$node]</code>

QOS deduced if not specified, partition needs to be set if not "batch"



SLURM - basic options for sbatch/srun

Action	sbatch/srun option
Request n distributed nodes	-N n
Request m memory per node	--mem= m GB
Request mc memory per core (logical cpu)	--mem-per-cpu= mc GB
Request job walltime	--time=d-hh:mm:ss
Request tn tasks per node	--ntasks-per-node= tn
Request ct cores per task (multithreading)	-c ct
Request nt total # of tasks	-n nt
Request g # of GPUs per node	--gres=gpu: g
Request to start job at specific $time$	--begin $time$
Specify job name as $name$	-J $name$
Specify required node $feature$	-C $feature$
Specify job partition	-p $partition$
Specify QOS	--qos qos
Specify account	-A $account$
Specify email address	--mail-user= $email$
Request email on event	--mail-type=all[,begin,end,fail]
Use the above actions in a batch script	#SBATCH $option$

SLURM - basic options for sbatch/srun

Action	sbatch/srun option
Request \$n distributed nodes	-N \$n
Request \$m memory per node	--mem=\$mGB
Request \$mc memory per core (logical cpu)	--mem-per-cpu=\$mcGB
Request job walltime	--time=d-hh:mm:ss
Request \$tn tasks per node	--ntasks-per-node=\$tn
Request \$ct cores per task (multithreading)	-c \$ct
Request \$nt total # of tasks	-n \$nt
Request \$g # of GPUs per node	--gres=gpu:\$g
Request to start job at specific \$time	--begin \$time
Specify job name as \$name	-J \$name
Specify required node \$feature	-C \$feature
Specify job partition	-p \$partition
Specify QOS	--qos \$qos
Specify account	-A \$account
Specify email address	--mail-user=\$email
Request email on event	--mail-type=all[,begin,end,fail]
Use the above actions in a batch script	#SBATCH \$option

- Diff. between **-N**, **-c**, **-n**, **--ntasks-per-node**, **--ntasks-per-core** ?
- Normally you'd specify **-N** and **--ntasks-per-node**
 - ↳ fix the latter to 1 and add **-c** for MPI+OpenMP jobs
- If your application is scalable, just **-n** might be enough
 - ↳ **beware of running across heterogeneous nodes: use '-C'**



SLURM - more options for sbatch/srun

Start job when... (dependencies)	sbatch/srun option
these other jobs have started	-d after:\$jobid1:\$jobid2
these other jobs have ended	-d afterany:\$jobid1:\$jobid2
these other jobs have ended with no errors	-d afterok:\$jobid1:\$jobid2
these other jobs have ended with errors	-d afternok:\$jobid1:\$jobid2
all other jobs with the same name have ended	-d singleton

Job dependencies and especially "singleton" can be very useful!

SLURM - more options for sbatch/srun

Start job when... (dependencies)	sbatch/srun option
these other jobs have started	-d after:\$jobid1:\$jobid2
these other jobs have ended	-d afterany:\$jobid1:\$jobid2
these other jobs have ended with no errors	-d afterok:\$jobid1:\$jobid2
these other jobs have ended with errors	-d afternok:\$jobid1:\$jobid2
all other jobs with the same name have ended	-d singleton

Job dependencies and especially "singleton" can be very useful!

Allocate job at... (specified time)	sbatch/srun option
exact time today	--begin=16:00
tomorrow	--begin=tomorrow
specific time relative to now	--begin=now+2hours
given date and time	--begin=2017-06-23T07:30:00

Jobs run like this will wait as PD – Pending with "(BeginTime)" reason

SLURM - more options for sbatch/srun

Start job when... (dependencies)	sbatch/srun option
these other jobs have started	<code>-d after:\$jobid1:\$jobid2</code>
these other jobs have ended	<code>-d afterany:\$jobid1:\$jobid2</code>
these other jobs have ended with no errors	<code>-d afterok:\$jobid1:\$jobid2</code>
these other jobs have ended with errors	<code>-d afternok:\$jobid1:\$jobid2</code>
all other jobs with the same name have ended	<code>-d singleton</code>

Job dependencies and especially "singleton" can be very useful!

Allocate job at... (specified time)	sbatch/srun option
exact time today	<code>--begin=16:00</code>
tomorrow	<code>--begin=tomorrow</code>
specific time relative to now	<code>--begin=now+2hours</code>
given date and time	<code>--begin=2017-06-23T07:30:00</code>

Jobs run like this will wait as PD – Pending with "(BeginTime)" reason

Other scheduling requests	sbatch/srun option
Ask for minimum/maximum # of nodes	<code>-N minnodes-maxnodes</code>
Ask for minimum run time (start job faster)	<code>--time-min=d-hh:mm:ss</code>
Ask to remove job if deadline can't be met	<code>--deadline=YYYY-MM-DD[THH:MM[:SS]]</code>
Run job within pre-created (admin) reservation	<code>--reservation=\$reservationname</code>
Allocate resources as specified job	<code>--jobid=\$jobid</code>

Can use --jobid to connect to running job (different than sattach!)

SLURM - environment variables

- 53 input env. vars. can be used to define job parameters
 - ↳ almost all have a command line equivalent
- up to 59 output env. vars. available within job environment
 - ↳ some common ones:

Description	Environment variable
Job ID	\$\$SLURM_JOBID
Job name	\$\$SLURM_JOB_NAME
Name of account under which job runs	\$\$SLURM_JOB_ACCOUNT
Name of partition job is running in	\$\$SLURM_JOB_PARTITION
Name of QOS the job is running with	\$\$SLURM_JOB_QOS
Name of job's advance reservation	\$\$SLURM_JOB_RESERVATION
Job submission directory	\$\$SLURM_SUBMIT_DIR
Number of nodes assigned to the job	\$\$SLURM_NNODES
Name of nodes assigned to the job	\$\$SLURM_JOB_NODELIST
Number of tasks for the job	\$\$SLURM_NTASKS or \$\$SLURM_NPROCS
Number of cores for the job on current node	\$\$SLURM_JOB_CPUS_PER_NODE
Memory allocated to the job per node	\$\$SLURM_MEM_PER_NODE
Memory allocated per core	\$\$SLURM_MEM_PER_CPU
Task count within a job array	\$\$SLURM_ARRAY_TASK_COUNT
Task ID assigned within a job array	\$\$SLURM_ARRAY_TASK_ID

Outputting these variables to the job log is essential for bookkeeping!



Usage examples (I)

> Interactive jobs

```
srunk -p interactive --qos qos-interactive --time=0:30 -N2 --ntasks-per-node=4 --pty bash -i
salloc -p interactive --qos qos-interactive bash -c 'ssh -Y $(scontrol show hostnames | head
-n 1)'
```

```
srunk -p interactive --qos qos-besteffort --cpu-bind=none -N1 -n4 --pty bash -i
srunk -C skylake -p batch --time=0:10:0 -N1 -c28 --pty bash -i
```



Usage examples (I)

> Interactive jobs

```
srun -p interactive --qos qos-interactive --time=0:30 -N2 --ntasks-per-node=4 --pty bash -i
salloc -p interactive --qos qos-interactive bash -c 'ssh -Y $(scontrol show hostnames | head -n 1)'
```

```
srun -p interactive --qos qos-besteffort --cpu-bind=none -N1 -n4 --pty bash -i
srun -C skylake -p batch --time=0:10:0 -N1 -c28 --pty bash -i
```

> Batch jobs

```
sbatch job.sh
sbatch -N 2 job.sh
sbatch -p batch --qos qos-batch job.sh
sbatch -p long --qos qos-long job.sh
sbatch --begin=2019-11-23T07:30:00 job.sh
sbatch -p batch --qos qos-besteffort job.sh
sbatch -p gpu --qos qos-gpu --gres=gpu:4 job.sh
sbatch -p bigmem --qos qos-bigmem --mem=2T job.sh
```



Usage examples (I)

> Interactive jobs

```
srunc -p interactive --qos qos-interactive --time=0:30 -N2 --ntasks-per-node=4 --pty bash -i
salloc -p interactive --qos qos-interactive bash -c 'ssh -Y $(scontrol show hostnames | head
-n 1)'
```

```
srunc -p interactive --qos qos-besteffort --cpu-bind=none -N1 -n4 --pty bash -i
srunc -C skylake -p batch --time=0:10:0 -N1 -c28 --pty bash -i
```

> Batch jobs

```
sbatch job.sh
sbatch -N 2 job.sh
sbatch -p batch --qos qos-batch job.sh
sbatch -p long --qos qos-long job.sh
sbatch --begin=2019-11-23T07:30:00 job.sh
sbatch -p batch --qos qos-besteffort job.sh
sbatch -p gpu --qos qos-gpu --gres=gpu:4 job.sh
sbatch -p bigmem --qos qos-bigmem --mem=2T job.sh
```

Status and details for partitions, nodes, reservations

```
squeue / squeue -l / squeue -la / squeue -l -p batch / squeue -t PD
scontrol show nodes / scontrol show nodes $nodename
sinfo / sinfo -s / sinfo -N
sinfo -T
```



Usage examples (II)

Collecting job information, priority, expected start time

```
scontrol show job $jobid          # only available while job is queued + 5 minutes after completion  
sprio -l  
squeue --start -u $USER
```




Usage examples (II)

Collecting job information, priority, expected start time

```
scontrol show job $jobid          # only available while job is queued + 5 minutes after completion  
sprio -l  
squeue --start -u $USER
```

Running job metrics – sstat tool

```
sstat -j $jobid / sstat -j $jobid -l  
sstat -j $jobid1 --format=AveCPU,AveRSS,AveVMSize,MaxRSS,MaxVMSize  
sstat -p -j $jobid1,$jobid2 --format=AveCPU,AveRSS,AveVMSize,MaxRSS,MaxVMSize
```



Usage examples (II)

Collecting job information, priority, expected start time

```
scontrol show job $jobid          # only available while job is queued + 5 minutes after completion
sprio -l
squeue --start -u $USER
```

Running job metrics – sstat tool

```
sstat -j $jobid / sstat -j $jobid -l
sstat -j $jobid1 --format=AveCPU,AveRSS,AveVMSize,MaxRSS,MaxVMSize
sstat -p -j $jobid1,$jobid2 --format=AveCPU,AveRSS,AveVMSize,MaxRSS,MaxVMSize
```

Completed job metrics – sacct & seff tools

```
sacct -j $jobid / sacct -j $jobid -l
sacct -p -j $jobid --format=account,user,jobid,jobname,partition,state,elapsed,elapseddraw,
    start,end,maxrss,maxvmsize,consumedenergy,consumedenergyraw,nnodes,ncpus,nodelist
sacct --starttime 2018-11-23 -u $USER
seff $jobid          # very useful to see at a glance: CPU/memory efficiency and max. memory
```



Usage examples (III)

Controlling queued and running jobs

```
scontrol hold $jobid
scontrol release $jobid
scontrol suspend $jobid
scontrol resume $jobid
scancel $jobid
scancel -n $jobname
scancel -u $USER
scancel -u $USER -p batch
scontrol requeue $jobid
```



Usage examples (III)

Controlling queued and running jobs

```
scontrol hold $jobid
scontrol release $jobid
scontrol suspend $jobid
scontrol resume $jobid
scancel $jobid
scancel -n $jobname
scancel -u $USER
scancel -u $USER -p batch
scontrol requeue $jobid
```

Checking accounting links and QOS available for you

```
sacctmgr show user $USER format=user%20s,defaultaccount%30s
sacctmgr list association where users=$USER format=account%30s,user%20s,qos%120s
```



Usage examples (III)

Controlling queued and running jobs

```
scontrol hold $jobid
scontrol release $jobid
scontrol suspend $jobid
scontrol resume $jobid
scancel $jobid
scancel -n $jobname
scancel -u $USER
scancel -u $USER -p batch
scontrol requeue $jobid
```

Checking accounting links and QOS available for you

```
sacctmgr show user $USER format=user%20s,defaultaccount%30s
sacctmgr list association where users=$USER format=account%30s,user%20s,qos%120s
```

Checking accounting share info - usage, fair-share, etc.

```
sshare -U
sshare -A $accountname
sshare -A $(sacctmgr -n show user $USER format=defaultaccount%30s)
sshare -a
```



Job launchers - basic (I)

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --time=0-00:05:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "Hello from the batch queue on node ${SLURM_NODELIST}"
# Your more useful application can be started below!
```

{Submit it with: sbatch launcher.sh}

Job launchers - basic (II)

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH -c 28
#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}"

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
srun ./preprocess_app
srun ./main_app
```

{Submit it overriding some settings: sbatch --time=5:0:0 launcher.sh}



Job launchers - basic (III)

```
#!/bin/bash -l
#SBATCH -J MyTestJob
#SBATCH --mail-type=end,fail
#SBATCH --mail-user=Your.Email@Address.lu
#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}"
# Your more useful application can be started below!
```


Job launchers - requesting memory (I)

```
#!/bin/bash -l
#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}"
# Your more useful application can be started below!
```

Use "mem" to request (more) memory per node for low #core jobs

Job launchers - requesting memory (II)

```
#!/bin/bash -l
#SBATCH -J MyVeryLargeMemoryJob
#SBATCH --mail-type=end,fail
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 1
#SBATCH -c 64
#SBATCH --mem=2TB
#SBATCH --time=1-00:00:00
#SBATCH -p bigmem
#SBATCH --qos=qos-bigmem

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
# Your more useful application can be started below!
```

Iris compute nodes in the bigmem partition have 112C/3TB RAM.

Job launchers - node features selection

```
#!/bin/bash -l
#SBATCH -J MyJobOnSkylakeCPUs
#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
#SBATCH -C skylake
[...]
```

```
$> sinfo --format="%N %f"
NODELIST AVAIL_FEATURES
iris-[001-108] broadwell
iris-[109-168,187-190] skylake
iris-[169-186] skylake,volta
iris-[191-196] skylake,volta,volta32
```



Job launchers - accelerated nodes (I)

```
#!/bin/bash -l
#SBATCH -J MyGPUJob
#SBATCH --mail-type=all
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 1
#SBATCH -c 14
##SBATCH --gres=gpu:volta:4
#SBATCH --gres=gpu:2
#SBATCH --mem=300G
#SBATCH --time=12:00:00
#SBATCH -p gpu

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

Iris compute nodes in the gpu partition have 4xVolta V100 GPUs and 768GB RAM.





Job launchers - accelerated nodes (II)

```
#!/bin/bash -l
#SBATCH -J MyGPUJob
#SBATCH -N 1
#SBATCH -c 28
#SBATCH --gres=gpu:4
#SBATCH --time=1-0:0:0
#SBATCH -p gpu

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

# Load the Singularity HPC containers module
module load tools/Singularity
# Pull the reference TensorFlow (GPU-enabled) image from Docker hub
singularity pull docker://tensorflow/tensorflow:latest-gpu
# Run the TF container w. Singularity's nvidia support on your own model
singularity exec --nv tensorflow-latest-gpu.simg python tf-model.py
```

Job launchers - long jobs

```
#!/bin/bash -l
#SBATCH -J MyLongJob
#SBATCH --mail-type=all
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --time=3-00:00:00
#SBATCH -p long
#SBATCH --qos=qos-long

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
# Your more useful application can be started below!
```

**Long walltime possible but you should not (!) rely on it.
Always prefer parallel, short walltime, requeue-able jobs.**

Job launchers - besteffort

```
#!/bin/bash -l
#SBATCH -J MyRerunnableJob
#SBATCH --mail-type=end,fail
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 1
#SBATCH --ntasks-per-node=28
#SBATCH --time=0-12:00:00
#SBATCH -p batch
#SBATCH --qos=qos-besteffort
#SBATCH --requeue

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
# Your more useful application can be started below!
```

Many scientific applications support internal state saving and restart!
System-level checkpoint-restart possible with DMTCP.



Job launchers - threaded parallel

```
#!/bin/bash -l
#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
```

By threaded we mean pthreads/OpenMP shared-memory applications.



Job launchers - MATLAB

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH --ntasks-per-node=28
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch
#SBATCH --cpu-bind=none

module load base/MATLAB
matlab -nodisplay -nosplash < /path/to/infile > /path/to/outfile
```

**MATLAB spawns processes, limited for now to single node execution.
We are still waiting for Distributed Computing Server availability.**



Job launchers - MATLAB

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH --ntasks-per-node=28
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch
#SBATCH --cpu-bind=none

module load base/MATLAB
matlab -nodisplay -nosplash < /path/to/infile > /path/to/outfile
```

**MATLAB spawns processes, limited for now to single node execution.
We are still waiting for Distributed Computing Server availability.**

A note on parallel jobs

**As of 2019 the iris cluster is heterogeneous
(Broadwell+Skylake-gen systems)**

Its core networking is still non-blocking fat-tree.

- Simply requesting `#tasks` **may not be optimal**
 - ↪ from hardware POV - slight difference in CPU freq. for now
 - ↪ from software efficiency POV - best to have arch. opt. builds
- Many elements contribute to an optimal (fast!) execution:
 - ↪ correct division of tasks / cores-per-task and application launch
 - ↪ memory allocation
 - ↪ execution on nodes with GPU accel. and their allocation
- Different MPI implementations will **behave differently**
 - ↪ recent Intel & OpenMPI on **iris**
 - ↪ always prefer launch using **srun**



Job launchers - IntelMPI

```
#!/bin/bash -l
#SBATCH -n 128
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/intel
srun -n $SLURM_NTASKS /path/to/your/intel-toolchain-compiled-app
```

**IntelMPI is configured to use PMI2 for process management (optimal).
Bare mpirun works but not recommended.**

Job launchers - OpenMPI

```
#!/bin/bash -l
#SBATCH -n 128
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/foss
srun -n $SLURM_NTASKS /path/to/your/foss-toolchain-compiled-app
```

**OpenMPI also uses PMI2 (again, optimal).
Bare mpirun works but not recommended.**

You can easily generate a hostfile from within a SLURM job with:
`srun hostname | sort -n > hostfile`



Job launchers - MPI+OpenMP

```
#!/bin/bash -l
#SBATCH -N 10
#SBATCH --ntasks-per-node=1
#SBATCH -c 28
#SBATCH --time=0-01:00:00
#SBATCH -C skylake
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/intel
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
srun -n $SLURM_NTASKS /path/to/your/parallel-hybrid-app
```

Compile and use your applications in hybrid MPI+OpenMP mode when you can for better (best?) possible performance.

Notes on optimizing your usage

A note on CPU affinity

- Processes pinned by default to cores (**CPUs** in SLURM docs.)
- `srun` aware of requested tasks/cores configuration and pins processes/threads accordingly
- Many options to control task affinity exist, see:
 - ↳ https://slurm.schedmd.com/srun.html#OPT_cpu-bind
 - ↳ https://slurm.schedmd.com/srun.html#OPT_hint
- Can be disabled with `srun --cpu-bind=none`

If not disabled for 'interactive' jobs, all your processes will be pinned to 1st core!



Summary

- 1 Introduction
- 2 SLURM workload manager
SLURM concepts and design for iris
Running jobs with SLURM
- 3 OAR and SLURM**
- 4 Conclusion

Notes on OAR

- OAR still in use as the workload manager of Gaia and Chaos **but clusters will be decommissioned by the end of this year**
 - ↪ celebrating **4.506.192** jobs on Gaia! (2019-06-19)
 - ↪ celebrating **1.701.137** jobs on Chaos! (2018-06-19)
- Many of its features are common to other workload managers, incl. SLURM (**462.613** jobs on Iris as of 2019-06-19)
 - ↪ some things are exactly the same
 - ↪ but some things work in a different way
 - ↪ ... and some have no equivalent or are widely different
- An adjustment period for you is needed if you've only used OAR
 - ↪ next slides show a **brief transition guide**



OAR/SLURM - commands guide

Command	OAR (gaia/chaos)	SLURM (iris)
Submit passive/batch job	<code>oarsub -S \$script</code>	<code>sbatch \$script</code>
Start interactive job	<code>oarsub -I</code>	<code>srun -p interactive --pty bash -i</code>
Queue status	<code>oarstat</code>	<code>squeue</code>
User job status	<code>oarstat -u \$user</code>	<code>squeue -u \$user</code>
Specific job status (detailed)	<code>oarstat -f -j \$jobid</code>	<code>scontrol show job \$jobid</code>
Delete (running/waiting) job	<code>oardel \$jobid</code>	<code>scancel \$jobid</code>
Hold job	<code>oarhold \$jobid</code>	<code>scontrol hold \$jobid</code>
Resume held job	<code>oarresume \$jobid</code>	<code>scontrol release \$jobid</code>
Node list and properties	<code>oarnodes</code>	<code>scontrol show nodes</code>
Join a running job	<code>oarsub -C \$jobid</code>	<code>sjoin \$jobid [\$nodeid]</code>

Similar yet different?

Many specifics will actually come from the way Iris is set up.



OAR/SLURM - job specifications

Specification	OAR	SLURM
Script directive	#OAR	#SBATCH
Queue request	-q \$queue	-p \$partition
Nodes request	-l nodes=\$count	-N \$min-\$max
Cores request	-l core=\$count	-n \$count
Cores-per-node request	-l nodes=\$ncount/core=\$ccount	-N \$ncount --ntasks-per-node=\$ccount
Walltime request	-l [...],walltime=hh:mm:ss	-t \$min OR -t \$days-hh:mm:ss
Job array	--array \$count	--array \$specification
Job name	-n \$name	-J \$name
Job dependency	-a \$jobid	-d \$specification
Property request	-p "\$property=\$value"	-C \$specification
Jobs on GPU nodes	-t gpu	-p gpu
Jobs on large memory nodes	-t bigmem	-p bigmem
Besteffort jobs	-t besteffort	--qos qos-besteffort
Email on job state change	--notify mail:\$email	--mail-user=\$email

**Job specifications will need most adjustment on your side.
Iris more homogeneous than Gaia/Chaos for now.
Running things in an optimal way is easier.**

OAR/SLURM - env. vars.

Environment variable	OAR	SLURM
Job ID	<code>\$OAR_JOB_ID</code>	<code>\$SLURM_JOB_ID</code>
Resource list	<code>\$OAR_NODEFILE</code>	<code>\$SLURM_NODELIST</code> #List not file! See below.
Job name	<code>\$OAR_JOB_NAME</code>	<code>\$SLURM_JOB_NAME</code>
Submitting user name	<code>\$OAR_USER</code>	<code>\$SLURM_JOB_USER</code>
Task ID within job array	<code>\$OAR_ARRAY_INDEX</code>	<code>\$SLURM_ARRAY_TASK_ID</code>
Working directory at submission	<code>\$OAR_WORKING_DIRECTORY</code>	<code>\$SLURM_SUBMIT_DIR</code>

Check available variables: `env | egrep "OAR|SLURM"`
Generate hostfile: `srun hostname | sort -n > hostfile`



Summary

- 1 Introduction
- 2 SLURM workload manager
SLURM concepts and design for iris
Running jobs with SLURM
- 3 OAR and SLURM
- 4 **Conclusion**



Conclusion and Practical Session start

We've discussed

- ✓ The design of SLURM for the **iris** cluster
- ✓ The permissions system in use through group accounts and QOS
- ✓ Main SLURM tools and how to use them
- ✓ Job types possible with SLURM on **iris**
- ✓ SLURM job launchers for sequential and parallel applications
- ✓ Transitioning from OAR to SLURM

Conclusion and Practical Session start

We've discussed

- ✓ The design of SLURM for the **iris** cluster
- ✓ The permissions system in use through group accounts and QOS
- ✓ Main SLURM tools and how to use them
- ✓ Job types possible with SLURM on **iris**
- ✓ SLURM job launchers for sequential and parallel applications
- ✓ Transitioning from OAR to SLURM

And now...

Q&A & practical

ulhpc-tutorials.readthedocs.io/en/latest/scheduling/advanced



Thank you for your attention...

Questions?

<http://hpc.uni.lu>

High Performance Computing @ uni.lu

Prof. Pascal Bouvry
Dr. Sebastien Varrette
Valentin Plugaru
Sarah Peter
Hyacinthe Cartiaux
Clement Pariset
Dr. Frédéric Pinel
Dr. Emmanuel Kieffer

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 SLURM workload manager
SLURM concepts and design for iris

- 3 Running jobs with SLURM
- 3 OAR and SLURM
- 4 Conclusion