SLURM Administrators Tutorial

20/01/15

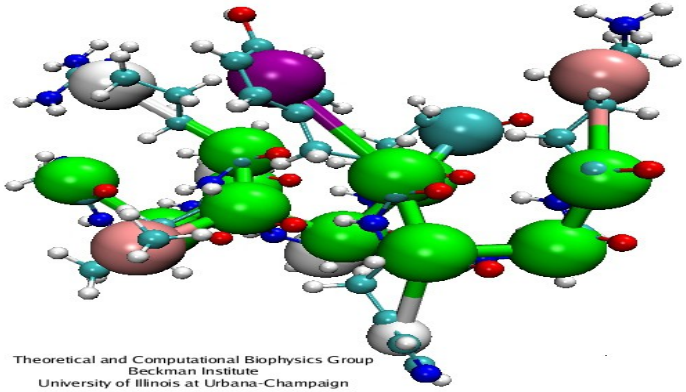
Yiannis Georgiou

Resource Management Systems Architect

- **Introduction**
- **SLURM scalable and flexible RJMS**
- **Part 1: Basics**
  - **Overview, Architecture, Configuration files, Partitions, Plugins, Reservations, Reconfiguration**
- **Part 2: Advanced Configuration**
  - **Accounting, Scheduling, Allocation, Network Topology Placement, Generic Resources, Licenses Management, Energy Reduction Techniques**
- **Part 3: Experts Configuration**
  - **CPU Management, Isolation with cgroups, Power Management, Simulation and evaluation**
- **Upcoming Features**

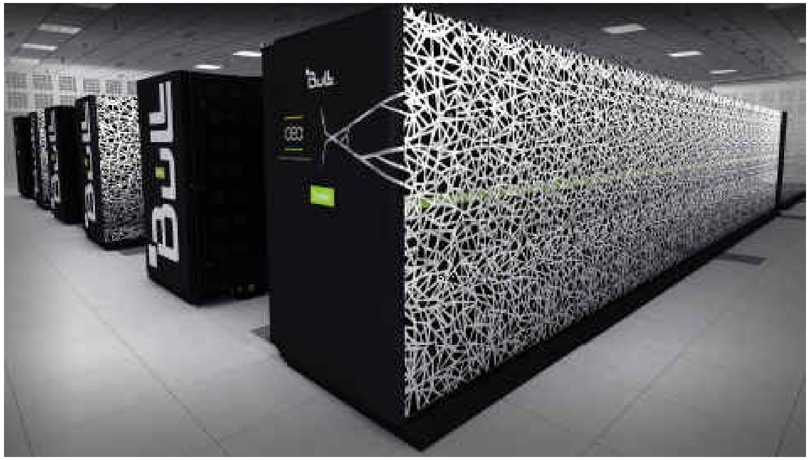
- Introduction
- SLURM scalable and flexible RJMS
- Part 1: Basics
  - Overview, Architecture, Configuration files, Partitions, Plugins, Reservations, Reconfiguration
- Part 2: Advanced Configuration
  - Accounting, Scheduling, Allocation, Network Topology Placement, Generic Resources, Licenses Management, Energy Reduction Techniques
- Part 3: Experts Configuration
  - CPU Management, Isolation with cgroups, Power Management, Simulation and evaluation
- Upcoming Features

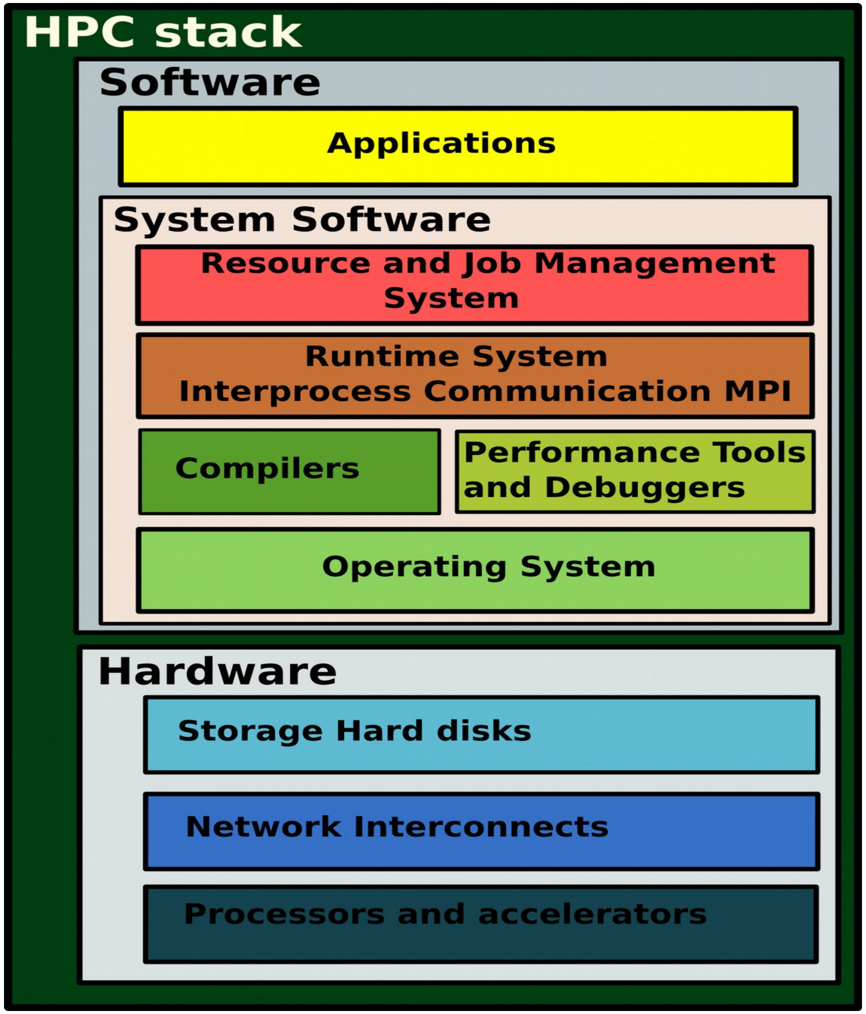
High Performance Computing



System Software:

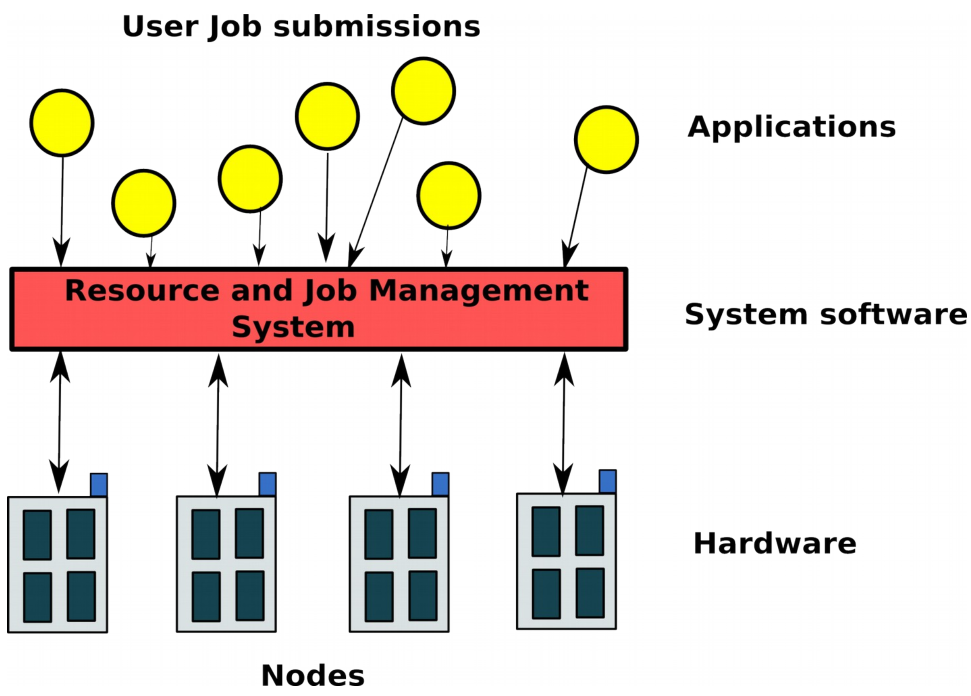
Operating System, Runtime System, Resource Management, I/O System, Interfacing to External Environments





Resource and Job Management Systems

The goal of a Resource and Job Management System (RJMS) is to satisfy users' demands for computation and assign resources to user jobs with an efficient manner.



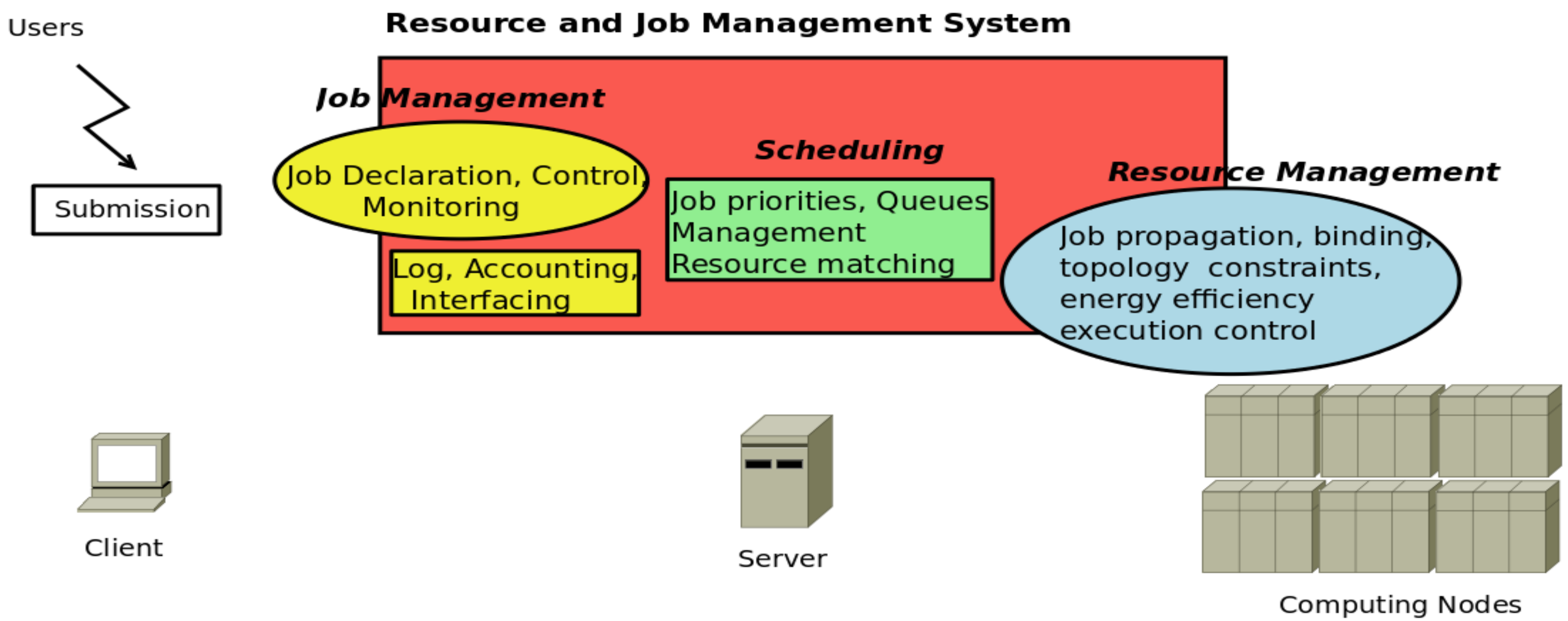
RJMS Importance
Strategic Position but complex internals:

- **Direct and constant** knowledge of resources
- **Multifacet procedures** with complex internal functions

Resource and Job Management System Layers

This assignement involves three principal abstraction layers:

- **Job Management:** declaration of a job and demand of resources and job characteristics,
- **Scheduling**: matching of the jobs upon the resources,
- **Resource Management :** launching and placement of job instances upon the computation resources along with the job's control of execution



Resource and Job Management System Concepts

RJMS subsystems		Principal Concepts	Advanced Features
Resource Management		-Resource Treatment (hierarchy, partitions,..) -Job Launcing, Propagation, Execution control -Task Placement (topology,binding,..)	- High Availability - Energy Efficiency - Topology aware placement
Job Management		-Job declaration (types, characteristics,...) -Job Control (signaling, reprioritizing,...) -Monitoring (reporting, visualization,..)	- Authentication (limitations, security,..) - QOS (checkpoint, suspend, accounting,...) - Interfacing (MPI libraries, debuggers, APIs,..)
Scheduling		-Scheduling Algorithms (builtin, external,..) -Queues Management (priorities,multiple,..)	- Advanced Reservation

- Introduction
- **SLURM scalable and flexible RJMS**
- Part 1: Basics
  - Overview, Architecture, Configuration files, Partitions, Plugins, Reservations, Reconfiguration
- Part 2: Advanced Configuration
  - Accounting, Scheduling, Allocation, Network Topology Placement, Generic Resources, Licenses Management, Energy Reduction Techniques
- Part 3: Experts Configuration
  - CPU Management, Isolation with cgroups, Power Management, Simulation and evaluation
- Upcoming Features

an atos company

SLURM scalable and flexible RJMS

- SLURM **open-source** Resource and Job Management System, sources freely available under the GNU General Public License.
- **Portable:** written in C with a GNU autoconf configuration engine.
- **Modular:** Based on a plugin mechanism used to support different kind of scheduling policies, interconnects, libraries, etc
- **Robust:** highly tolerant of system failures, including failure of the node executing its control functions.
- **Scalable:** designed to operate in a heterogeneous cluster with up to tens of millions of processors. It can accept 1,000 job submissions per second and fully execute 500 simple jobs per second (depending upon hardware and system configuration).
- **Power Management:** Job can specify their desired CPU frequency and power use by job is recorded. Idle resources can be powered down until needed.

SLURM History and Facts

•Initially developed in LLNL since 2003, passed to SchedMD in 2011

•Multiple enterprises and research centers have been contributing to the project (LANL, CEA, HP, BULL, BSC, CRAY etc)

•Large international community, active mailing lists (support by main developers)

- Contributions (various external software and standards are integrated upon SLURM)

• As of the June 2014 Top500 supercomputer list, SLURM is being used on six of the ten most powerful computers in the world including the no1 system, Tianhe-2 with 3,120,000 computing cores.

BULL and SLURM

- BULL initially started to work with SLURM in 2005
- About 6 SLURM-dedicated engineers since 2013
  - **Research** upon the field of Resource Management and Job Scheduling (National/European financed projects, PhDs) and definition of RoadMap
  - **Development** of new SLURM features: all code dropped in the open-source
  - **Support** upon clusters : Training, Confgiruation, Bugs, Feature Requests, etc
- Integrated as the default RJMS into the **BULL- HPC software stack** since 2006
- Close development **collaboration** with SchedMD and CEA
- Organaziation of Slurm User Group (SUG) Conference (User, Admin Tutorials + Technical presentation for developpers) http://www.schedmd.com/slurmdocs/publications.html

- Introduction
- SLURM scalable and flexible RJMS
- **Part 1: Basics**
  - **Overview, Architecture, Configuration files, Partitions, Plugins, Reservations, Reconfiguration**
- Part 2: Advanced Configuration
  - Accounting, Scheduling, Allocation, Network Topology Placement, Generic Resources, Licenses Management, Energy Reduction Techniques
- Part 3: Experts Configuration
  - CPU Management, Isolation with cgroups, Power Management, Simulation and evaluation
- Upcoming Features

SLURM sources and Documentation

Slurm sources :
- Download a repo (stable or development) from: http://www.schedmd.com/#repos
- Or the latest code from: *git clone git://github.com/SchedMD/slurm.git*

-For User and Admins latest **documentation**:
http://slurm.schedmd.com/documentation.html

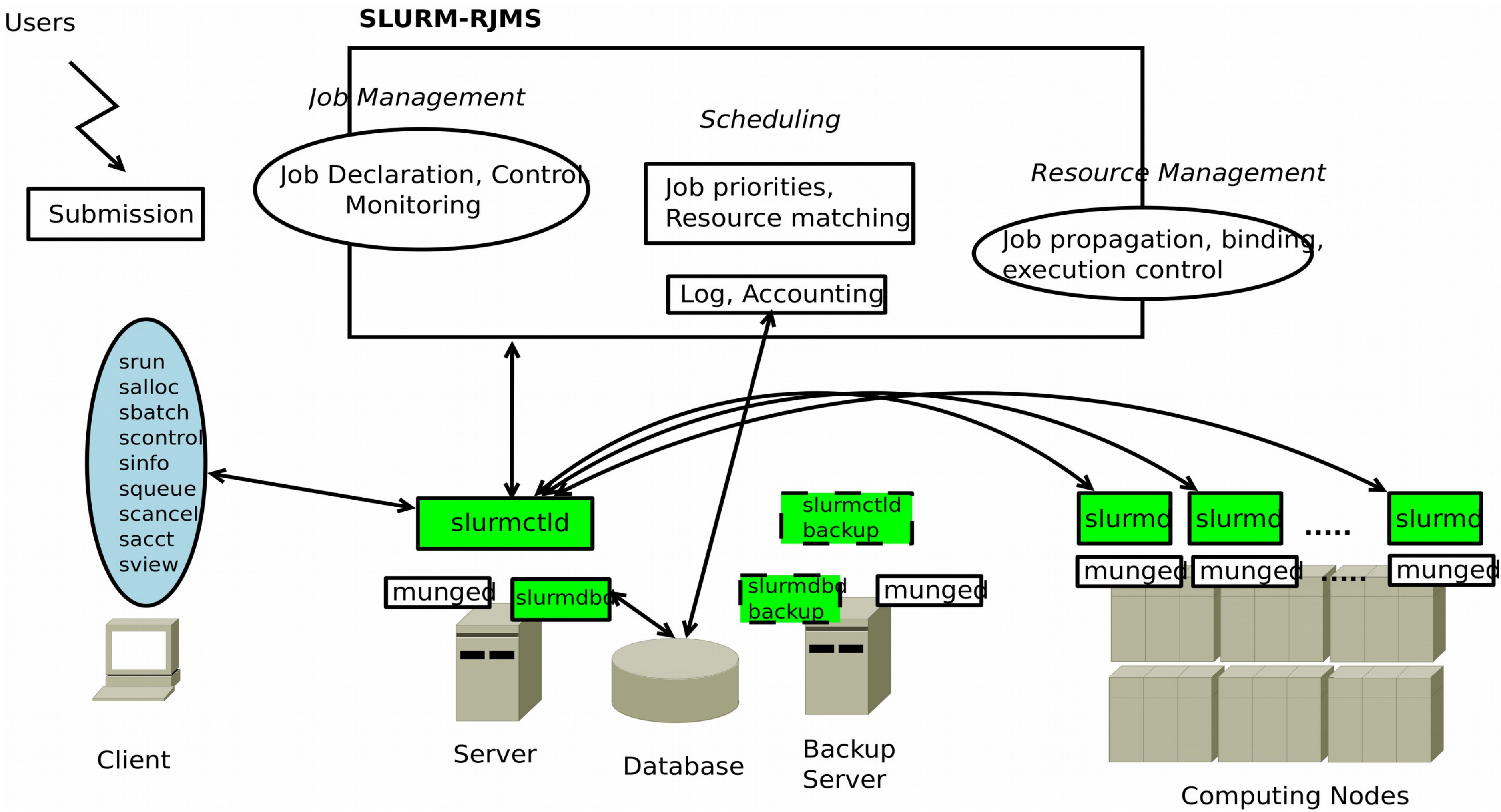
-Detailed **man pages** for commands and configuration files
http://slurm.schedmd.com/man_index.html

-All SLURM related **publications and presentations:**
http://slurm.schedmd.com/publications.html

Quick Installation Steps

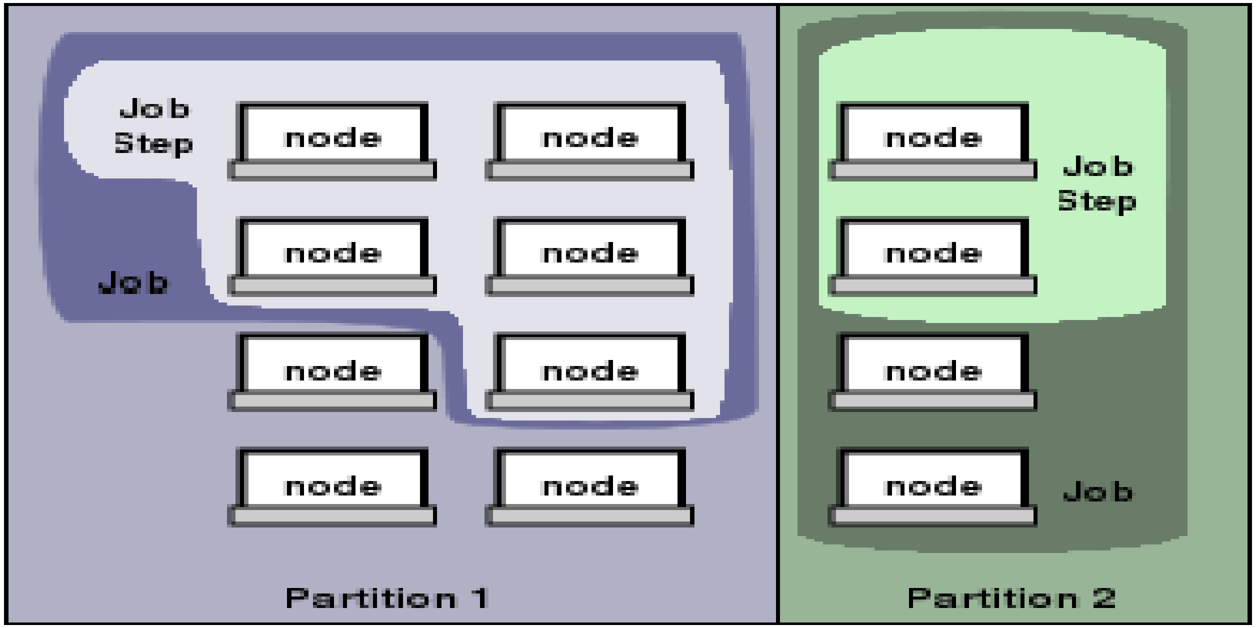
1.Install MUNGE for authentication. Make sure the MUNGE daemon, munged is started before you start the SLURM daemons.
2.Install SLURM either creating a tgz from git or downloading an existing tgz
3.cd to the directory containing the SLURM source and type ./configure with necessary options.
4.Type *make* to compile SLURM.
5.Type *make install* to install the programs, documentation, libraries, header files, etc.
6.Create the slurm User upon all nodes of the cluster.
7.Create parent directories for SLURM's log files, process ID files, state save directories, etc. are not created by SLURM. They must be created and made writable by SlurmUser as needed prior to starting SLURM daemons.
8. Create a basic slurm.conf file
9. Follow the presentation...

SLURM Architecture



SLURM Terms

•**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
•**Step**	sets of (possible parallel) tasks with a job



SLURM Principles

- Architecture Design:
  - one central controller daemon **slurmctld**
  - A daemon upon each computing node **slurmd**
  - One central daemon for the database controls slurmdbd

- Principal Concepts:
  - a general purpose **plugin mechanism** (for features such as scheduling policies, process tracking, etc)
  - the **partitions** which represent group of nodes with specific characteristics (job limits, access controls, etc)
  - one **queue** of pending work
  - The **job steps** which are sets of (possibly parallel) tasks within a job

User Commands

srun allocate resources (number of nodes, tasks, partition, constraints, etc.) launch a job that will execute on each allocated cpu.

salloc allocate resources (nodes, tasks, partition, etc.), either run a command or start a shell. Request launch srun from shell. (interactive commands within one allocation)

sbatch allocate resources (nodes, tasks, partition, etc.) Launch a script containing sruns for series of steps.

sbcast transmit file to all nodes of a running job. Used in sbatch or salloc.

sattach attach to running job for debuggers.

User & Admin Commands

sinfo display characteristics of partitions
squeue display jobs and their state
scancel cancel a job or set of jobs.
scontrol display and changes characteristics of jobs, nodes, partitions.
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

Admin Commands

sacctmgr setup accounts, specify limitations on users and groups.

sreport display information from accounting database on jobs, users, clusters.

sview graphical view of cluster. Display and change characteristics of jobs, nodes, partitions.

strigger show, set, clear event triggers. Events are usually system events such as an equipement failure.

sshare view sharing information from multifactor plugin.

Configuration

Slurm configuration: Through configuration **files** responsible, for the function of different **daemons** present on the management and the computing nodes

slurm.conf

- Indispensable on all nodes (management-compute)

slurmdbd.conf

- Used if slurmdbd accounting
- Only on management node

topology.conf

- Used if topology plugin activated
- Indispensable on all nodes (management-compute)

gres.conf

- Used if gres plugin activated
- Only on computing nodes

cgroup.conf

- Used if cgroup plugin activated
- Only on computing nodes

slurm.conf
- Low level configuration
- Management policies
- Scheduling policies
- Allocation policies
- Node definition
- Partition definition

slurmdbd.conf
- Type of persistent storage (DB)
- Location of storage

topology.conf
- Switch hierarchy

gres.conf
- Generic resources details
- Device files

cgroup.conf
- Mount point
- Release agent path
- Cgroup subsystems parameters

	Controller	Compute node
Mandatory	slurm.conf slurmdbd.conf	slurm.conf
Optional	prologs epilogs topology.conf	gres.conf cgroup.conf topology.conf

Configuration (slurm.conf) – Part 1

Node definition

- Characteristics (sockets, cores, threads, memory, features)
- Network addresses

Partition definition

- Set of nodes
- Sharing
- Priority/preemption

```
 # Compute Nodes
NodeName=cuzco[1-10] Procs=16 Sockets=2 CoresPerSocket=8 ThreadsPerCore=1 State=UNKNOWN RealMemory=38000
NodeName=cuzco[10-20] Procs=32 Sockets=2 CoresPerSocket=8 ThreadsPerCore=2 State=UNKNOWN RealMemory=46000

 # Partitioning
PartitionName=exclusive Nodes=cuzco[1-20] MaxTime=INFINITE State=UP Priority=10 Shared=Exclusive
PartitionName=shared Nodes=berlin[1-20] Default=YES MaxTime=INFINITE State=UP Priority=30
PartitionName=procs16 Nodes=berlin[1-10] MaxTime=INFINITE State=UP Priority=30
PartitionName=procs32 Nodes=berlin[10-20] MaxTime=INFINITE State=UP Priority=30
```

Partitions

- Partitions are used in SLURM to group nodes/resources characteristics

Partition 1: 32 cores and high_memory

Partition 2: 32 cores and low_memory

Partition 3: 64 cores

More on Partitions

Shared Option
Controls the ability of the partition to execute more than one job on a resource (node, socket, core)

EXCLUSIVE allocates entire node (overrides cons_res ability to allocate cores and sockets to multiple jobs)
NO sharing of any resource.
YES all resources can be shared, unless user specifies –exclusive on srun | salloc | sbatch

Important Note: To view the particular parameters of partitions users can use the “scontrol show partitions” command

Configuration (slurm.conf) – Part 2

```
#slurm.conf

# Basic parameters
ClusterName=cuzco
ControlMachine=cuzco0
#ControlAddr=127.0.0.1
SlurmUser=slurm
SlurmctldPort=6817
SlurmdPort=6818
AuthType=auth/munge

# States saving
StateSaveLocation=/var/spool/slurm
SlurmdSpoolDir=/var/spool/slurmd.%n
SlurmctldPidFile=/var/run/slurmctld.pid
SlurmdPidFile=/var/run/slurmd.%n.pid

# Logging
SlurmctldDebug=5
SlurmctldLogFile=/var/log/slurmctld.log
SlurmdDebug=5
SlurmdLogFile=/var/log/slurmd.%n.log

# Timers
SlurmctldTimeout=300
SlurmdTimeout=300
```

Management Policies

- **Location of controllers, spool, state info**
- **Authentication**
- **Logging**
- **Prolog / epilog scripts**

Configuration (slurm.conf) – Part 3

```
# Process-Task tracking
ProctrackType=proctrack/linuxproc
TaskPlugin=task/affinity
TaskPluginParam=Cpusets

# Selection of Resources
SelectType=select/cons_res
SelectTypeParameters= CR_Core_Memory

# Scheduling
SchedulerType=sched/backfill
FastSchedule=1
PreemptMode=REQUEUE
PreemptType=preempt/qos
FastSchedule=1
```

Scheduling policies

- **Priority**
- **Preemption**
- **Backfill**

Allocation policies

- **Entire nodes or 'consumable resources'**
- **Task Affinity (lock task on CPU)**
- **Topology (minimum number of switches)**

Plugins in SLURM

- **-Authentication** (i.e. munge,)
- **-Job Accounting Gather** (i.e. linux, cgroups)
- **-Accounting Storage** (i.e. mysql, postgres)
- **-Generic Resources (GRES)** (i.e. gpu, nic)
- **-Job Submission** (i.e. partitions, lua)
- **-MPI** (i.e. openmpi, pmi2)
- **-Energy Accounting** (i.e. rapl,ipmi)
- **-Preemption** (i.e. partitions,qos)
- **-Priority** (i.e. basic,multifactor)
- **-Process Tracking** (i.e. linux,cgroup)
- **-Scheduler** (i.e. builtin,backfill)
- **-Resource Selection** (i.e. linear,cons_res)
- **-Task** (i.e. affinity,cgroups)
- **-Topology** (i.e. tree,3d_torus)

Examples of info commands

```
 > sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
all*         up   infinite      4   idle trek[0-3]
P2           up   infinite      4   idle trek[0-3]
P3           up   infinite      4   idle trek[0-3]

> scontrol show node trek
NodeName=trek3 Arch=x86_64 CoresPerSocket=4
   CPUAlloc=0 CPUErr=0 CPUTot=16 Features=HyperThread
   Gres=(null)
   NodeAddr=sulu NodeHostName=sulu
   OS=Linux RealMemory=1 Sockets=2
   State=IDLE ThreadsPerCore=2 TmpDisk=0 Weight=1
   BootTime=2011-06-30T11:04:22 SlurmdStartTime=2011-07-12T06:23:43
   Reason=(null)

> scontrol show partition P2
PartitionName=P2
   AllocNodes=ALL AllowGroups=ALL Default=NO
   DefaultTime=NONE DisableRootJobs=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=UNLIMITED MinNodes=1
   Nodes=trek[0-3]
   Priority=2 RootOnly=NO Shared=NO PreemptMode=CANCEL
   State=UP TotalCPUs=40 TotalNodes=4
```

srun, salloc, sbatch

- All request an allocation of resources.
- Similar set of command line options.
- Request number of nodes, tasks, cpus, constraints, user info, dependencies, and lots more.
- **srun** launches tasks (command) on the requested nodes.
- **salloc** can launch a task such as mpirun on the client, or open a shell on the client. Srun is then used to launch tasks within the allocation.
- **sbatch** is a shell script that contains multiple sruns within the allocation. (multi step job).

Example srun

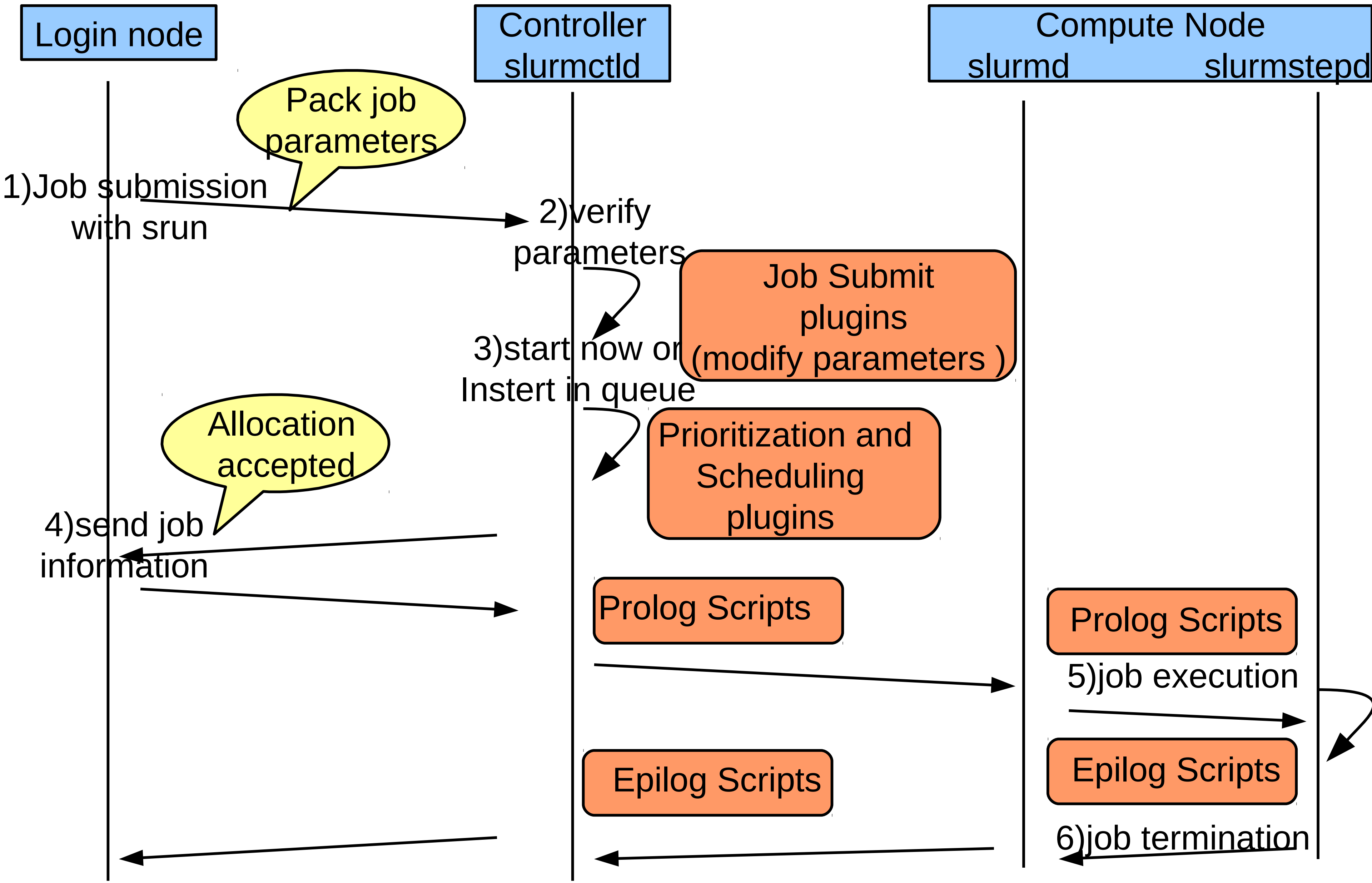
srun -l –p P2 –N2 –tasks-per-node=2 –exclusive hostname

-l	prepend task number to output (debug)
-p P2	use Partition P2
-N2	use 2 nodes
--tasks-per-node	launch 2 tasks on each node
--exclusive	do not share the nodes
hostname	command to run.

Results

```
> srun -l -p P2 -N2 --tasks-per-node=2
--exclusive hostname
0: scotty
1: scotty
2: bones
3: bones
```

Job Journey


Login node
Controller
slurmctld
Compute Node
slurmd
slurmstepd
Pack job parameters
1)Job submission with srun
2)verify parameters
Job Submit plugins (modify parameters)
3)start now or Instert in queue
Prioritization and Scheduling plugins
Allocation accepted
4)send job information
Prolog Scripts
Prolog Scripts
5)job execution
Epilog Scripts
Epilog Scripts
6)job termination

- login node :

  Using srun, all information about the job (parameters) are packed and send via RPC to the controller

- Slurm controller :

  _ call job_submit plugin to modify parameters (if needed)

  _ validate all the parameters (partition, limits, etc...)

  _ evaluate the priority of the job : it's the highest > start now,

  it's not > enqueue

  Direct path and « queued » path :

  _ find nodes matching job's recquirement

  _ give credential to srun → job can start

  _ update nodes state

Job journey - we have compute nodes, what's next ?

- srun sends all job's information (executable name, files name, etc...) to the first node of the allocation.
- Then the node forwards the request to other nodes using a spanning tree algorithm
- Each slurmd send a request to the controller to verify the job was not canceled during the process
- Each slurmd spawn a slurmstepd program to manage the job step
- Slurmstepd can now initiate the job :

  _ do the mpi setup

  _ create a container using the proctrack plugin

  _ change the uid to the user one

  _ set up environnement variables

  _ fork the task

Job journey – This is the end…

- job termination can be initiated by :
  - the task itself : end of the program

    → Slurmd will warn slurmctld which will log the termination

  - the user : job cancelation,
  - Slurm : the job has exceed a limit (timelimit, memory usage, etc…)

    → in both case slurmd will :

    _ send a SIGTERM to any task

    _ wait kilwait seconds

    _ send a SIGKILL

    _ execute any Epilog

    _ send a final message to slurmctld

JobSubmit Plugin in SLURM

JobSubmitPlugins Parameter in **slurm.conf**

A comma delimited list of job submission plugins to be used. These are intended to be site-specific plugins which can be used *to set default job parameters* and/or logging events. Plugins that can be modified and used: "defaults", "partition", "lua",etc.

For examples of use, see the SLURM code in "src/plugins/job_submit" and "contribs/lua/job_submit*.lua" then modify the code to satisfy your needs.

Example:
Associate user jobs to projects: A user may only charge jobs to one of his projects

Parameter	Location	Invoked by	User	When executed
Prolog (from slurm.conf)	Compute or front end node	slurmd daemon	SlurmdUser (normally user root)	First job or job step initiation on that node (by default);
PrologSlurmctld (from slurm.conf)	Head node (where slurmctld daemon runs)	slurmctld daemon	SlurmctldUser	At job allocation
Epilog (from slurm.conf)	Compute or front end node	slurmd daemon	SlurmdUser (normally user root)	At job termination
EpilogSlurmctld (from slurm.conf)	Head node (where slurmctld daemon runs)	slurmctld daemon	SlurmctldUser	At job termination

Prolog / Epilog for jobstep allocation

Parameter	Location	Invoked by	User	When executed
SrunProlog (from slurm.conf) or srun --prolog	srun invocation node	srun command	User invoking srun command	Prior to launching job step
TaskProlog (from slurm.conf) or srun –task-prolog	Compute node	slurmstepd daemon	User invoking srun command	Prior to launching job step
TaskEpilog (from slurm.conf) or srun –task-epilog	Compute node	slurmstepd daemon	User invoking srun command	Completion job step
SrunEpilog (from slurm.conf) or srun --epilog	srun invocation node	srun command	User invoking srun command	Completion job step

- Once the principal configuration parameters are correctly set the services can be started on management and computing nodes by **launching the particular scripts** on all nodes:
/etc/init.d/slurm {start, stop, restart, …}

- Alternatively the services can be started by **executing the commands** slurmctld on the controller and slurmd on the computing nodes

- The services are normally launched in the background with logging in the particular files set in the slurm.conf. However it is possible to **start the deamons in the foreground** with -D followed by v for different verbosity levels. This is useful for testing.
slurmctld -Dvvvvvv
slurmd -Dvvvvvv

- Every **change of** configuration parameters should be followed by a **copy of slurm.conf** on all computing nodes and restart of SLURM deamons.

- In case other configuration files are changed they should be equally copied on all computing nodes (except of slurmdbd.conf which is used only on the management nodes)

- **Reconfiguration** can also be enabled dynamically (running jobs continue execution) without copying the slurm.conf file with scontrol reconfigure (doesn't update slurm.conf files)

- **ALL** SLURM daemons should be shutdown and restarted if any of these parameters are to be changed: AuthType, BackupAddr, BackupController, ControlAddr, ControlMach, PluginDir,StateSaveLocation, SlurmctldPort or SlurmdPort

scontrol command can be also used for reservations
It provides the ability to create, update and delete advanced reservations for resources allocations

Basic parameters that need to be used:
Starttime,Duration, User, NodeCnt or NodeList
Once the reservation is made the user can submit a job upon the reserved Resources and this job will start on the starttime.

Examples:

```
>scontrol: create res StartTime=2009-04-01T08:00:00 Duration=5:00:00 Users=toto NodeCnt=10
    Reservation created: toto_1
>scontrol: update Reservation=toto_1 Flags=Overlap NodeCnt=20
```

An alternative way to start a job in a particular moment in the future is the –begin-time option of the submission commands

- Introduction
- SLURM scalable and flexible RJMS
- Part 1: Basics
  - Overview, Architecture, Configuration files, Partitions, Plugins, Reservations, Reconfiguration
- **Part 2: Advanced Configuration**
  - **Accounting, Scheduling, Allocation, Network Topology Placement, Generic Resources, Licenses Management, Energy Reduction Techniques**
- Part 3: Experts Configuration
  - CPU Management, Isolation with cgroups, Power Management, Simulation and evaluation
- Upcoming Features

Bull
an atos company

SLURM Accounting

- Accounting based upon Mysql database
- Robust and scalable (confirmed upon Tera100 cluster)
- Command for database and accounting configuration: *sacctmgr*
- **Fairsharing and Preemption** scheduling techniques based upon the accounting infrastructure

```
>sacctmgr add cluster snowflake
>sacctmgr add account users Cluster=snowflake Description="none" Organization="none"
>sacctmgr list users
      User   Def Acct     Admin
---------- ---------- ---------
      gohn   students      None
      root       root Administ+
     slurm professors      None
```

Accounting

Commands

Sacct reports resource usage for running or terminated jobs.
Sstat reports on running jobs, including imbalance between tasks.
Sreport generates reports based on jobs executed in a time interval.
Sacctmgr is used to create account and modify account settings.

Plugins associated with resource accounting

AccountingStorageType controls how information is recorded (MySQLl with SlurmDBD is best)
JobAccntGatherType controls the mechanism used to gather data. (OS Dependent)
JobCompType controls how job completion information is recorded.

Accounting (associations)

An Association is a combination of a Cluster, a User, and an Account.

- An accounting database may be used by multiple **Clusters**.
- **Account** is a slurm entity.
- **User** is a Linux user.

Use –account srun option.

With associations, a user may have different privileges on different clusters.

A user may also be able to use different accounts, with different privileges.

Multiple users may launch jobs on a linux account.

Sacctmgr

Account Options
Clusters to which the Account has access
Name, **Description** and **Organization**.
Parent is the name of an account for which this account is a child.

User Options
Account(s) to which the user belongs.
AdminLevel is accounting privileges (for sacctmgr). None, Operator, Admin
Cluster limits clusters on which accounts user can be added to.
DefaultAccount is the account for the user if an account is not specified on srun
Partition is the a partition an association applies to.

Accounting Limits Enforcement

If a user has a limit set SLURM will read in those, if not we will refer to the account associated with the job. If the account doesn't have the limit set we will refer to the cluster's limits. If the cluster doesn't have the limit set no limit will be enforced.

Some (but not all limits are)

Fairshare= Integer value used for determining priority. Essentially this is the amount of claim this association and it's children have to the above system. Can also be the string "parent", this means that the parent association is used for fairshare.

GrpCPUMins= A hard limit of cpu minutes to be used by jobs running from this association and its children. If this limit is reached all jobs running in this group will be killed, and no new jobs will be allowed to run. (GrpCPUs, GrpJobs, GrpNodes, GrpSubmitJobs, GrpWall)

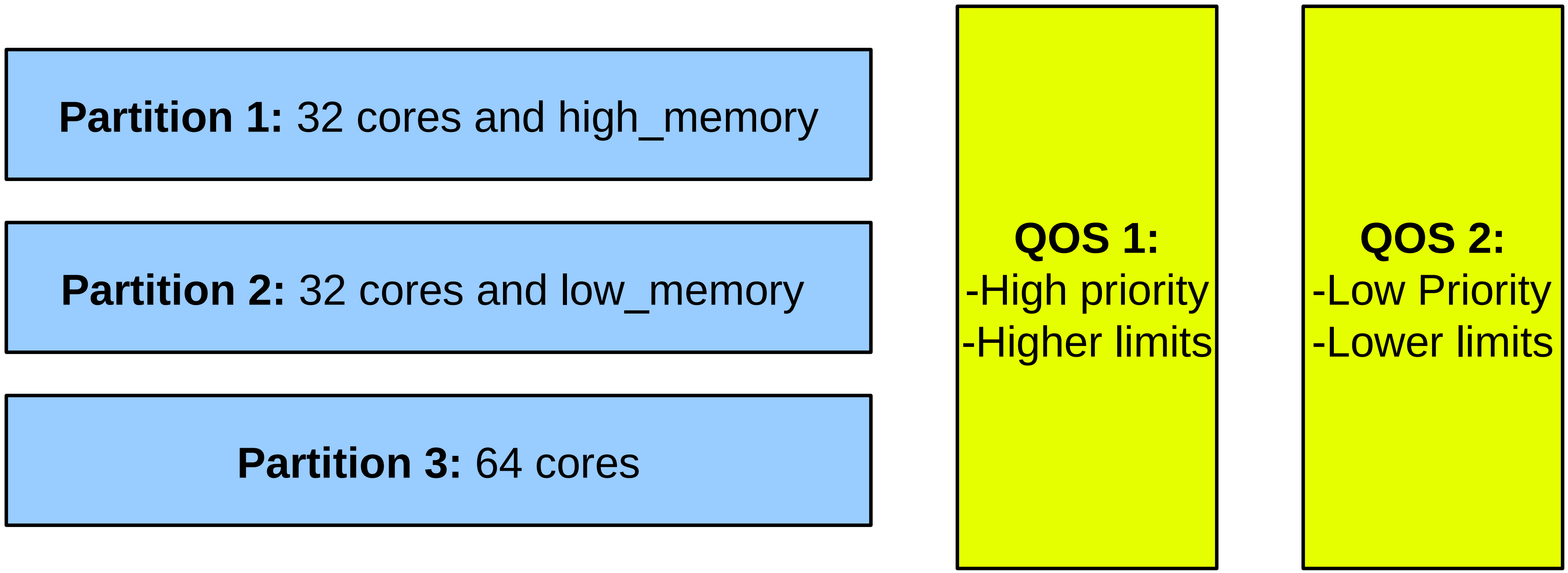
MaxCPUMinsPerJob= A limit of cpu minutes to be used by jobs running from this association. If this limit is reached the job will be killed will be allowed to run. (MaxCPUsPerJob, MaxJobs, MaxNodesPerJob, MaxSubmitJobs, MaxWallDurationPerJob)

QOS (quality of service) comma separated list of QOS's this association is able to run.

Important Note: To activate the accounting limitations and QOSyou need to add the following parameter in slurm.conf, distribute the slurm.conf on all nodes and restart the deamons: AccountingStorageEnforce=limits, qos

Partitions and QOS

- Partitions and QOS are used in SLURM to group nodes and jobs characteristics
- The use of **Partitions** and **QOS** (Quality of Services) entities in SLURM is orthogonal:
  – Partitions for grouping resources characteristics
  – QOS for grouping limitations and priorities



Partitions and QOS Configuration

Partitions Configuration:
In slurm.conf file

```
# Partition Definitions
PartitionName=all Nodes=trek[0-3] Shared=NO Default=YES
PartitionName=P2 Nodes=trek[0-3] Shared=NO Priority=2 PreemptMode=CANCEL
PartitionName=P3 Nodes=trek[0-3] Shared=Exclusive Priority=3 PreemptMode=REQUEUE
```

QOS Configuration:
In Database

```
>sacctmgr add qos name=lowprio priority=10 PreemptMode=Cancel GrpCPUs=10 MaxWall=60 MaxJobs=20
>sacctmgr add qos name=hiprio priority=100 Preempt=lowprio GrpCPUs=40 MaxWall=120 MaxJobs=50
>sacctmgr list qos
     Name   Priority Preempt PreemptMode     GrpCPUs  MaxJobs  MaxWall
---------- ---------- ---------- ---------- ----------- ------------------------------------- ---------- -------- ----------- -------
   lowprio     10                   cancel          10          20         60
    hiprio     100       lowprio                    40          50         120
```

More on QOS

Used to provide detailed limitations and prioritiees on jobs

Every user/account will have multiple allowed QOS upon which he may send jobs with the –qos parameter but only one default QOS in case he doesn't precise a –qos parameter in his submission

Important Note: To view the particular parameters of QOS provided by the admins users can use the “sacctmgr show associations” command

Fairsharing in SLURM

- User and Group accounts created in the **database**
- **Inheritance** between Groups and Users for all the different characteristics (Fairshare factors, Max number of Jobs, Max number of CPUs, etc)
- Job Priorities based on the **CPU*Time utilization** of each user
- Various **factors** can take part in the formula through the MultiFactor plugin:

  Job_priority =

  (PriorityWeightAge) * (age_factor) +

  (PriorityWeightFairshare) * (fair-share_factor) +

  (PriorityWeightJobSize) * (job_size_factor) +

  (PriorityWeightPartition) * (partition_factor)

Important Note: To activate fairsharing in SLURM you need to add the Priority/multifactor parameter in slurm.conf along with the different parameters for the particular factors that are needed for the site

Usage Guide – Accounting

sacct displays accounting information for jobs and steps

Some basic parameters for **sacct** command:

-b Displays a brief listing (jobid,status,exitcode)

-l a long listing of jobs characteristics

--format <param1,param2,> to select the actual fields to be shown

```
Example:
>sacct –format=jobid,elapsed,ncpus,ntasks,state
# sacct --format=jobid,elapsed,ncpus,ntasks,state
Jobid     Elapsed       Ncpus    Ntasks     State
---------- ---------- ---------- -------- ----------
3             00:01:30         2        1  COMPLETED
3.0           00:01:30         2        1  COMPLETED
4             00:00:00         2        2  COMPLETED
4.0           00:00:01         2        2  COMPLETED
5             00:01:23         2        1  COMPLETED
5.0           00:01:31         2        1  COMPLETED
```

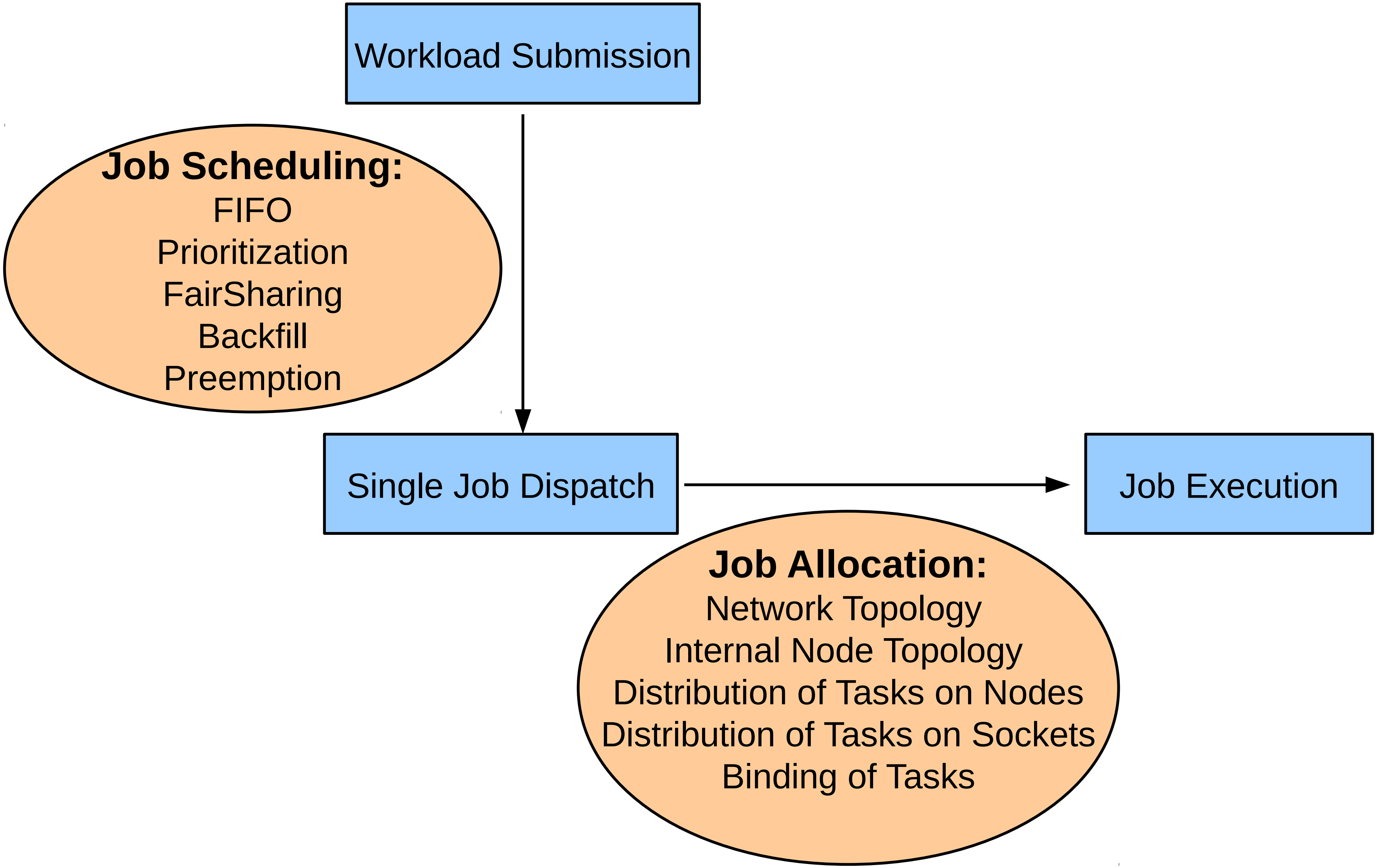
Usage Guide – Reporting

sreport generates reports of job usage and cluster utilization

The syntax of this command is like:
<type><REPORT><OPTIONS> where <type> can be cluster, job or user and each type has various reports and options
Example1: sreport job sizesbyaccount
Example2: sreport cluster AccountUtilizationByUser
Example3: sreport user topusage account=gohn

```
Example:
>sreport cluster utilization
```

SLURM scheduling / allocation procedures



SLURM Scheduling

- SLURM supports various **scheduling policies and optimization techniques** (non-exhaustive list) :
  - Backfill
  - Preemption
  - Fairsharing
  - Topology aware placement
- Advantage: Techniques can be **supported simultaneously**
- Scheduler scalable upto 130.000 cores and 90.000 jobs per hour (confirmed upon Tera 100)
- Enhancements for High Throughput Computing
- Study experimenting with scalability and network topology-aware placement has been published in 2012:

  http://link.springer.com/chapter/10.1007/978-3-642-35867-8_8#

Scheduling Policies

Scheduler Plugin Type
Sched/builtin Default FIFO
Sched/hold variation on builtin; new jobs are held if /etc/slurm.hold file exists.

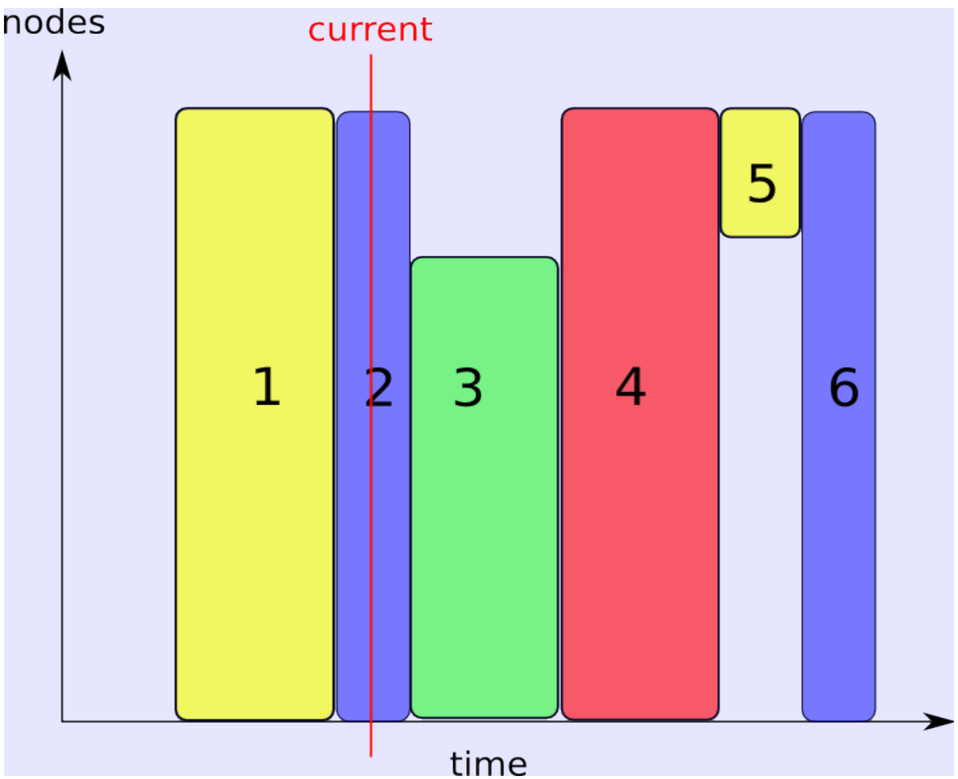
Sched/backfill schedule lower priority jobs as long as they don't delay a waiting higher priority job.

- Increases utilization of the cluster.
- Requires declaration of max execution time of lower priority jobs.
  - --time on 'srun',
  - DefaultTime or MaxTime on Partition
  - MaxWall from accounting association

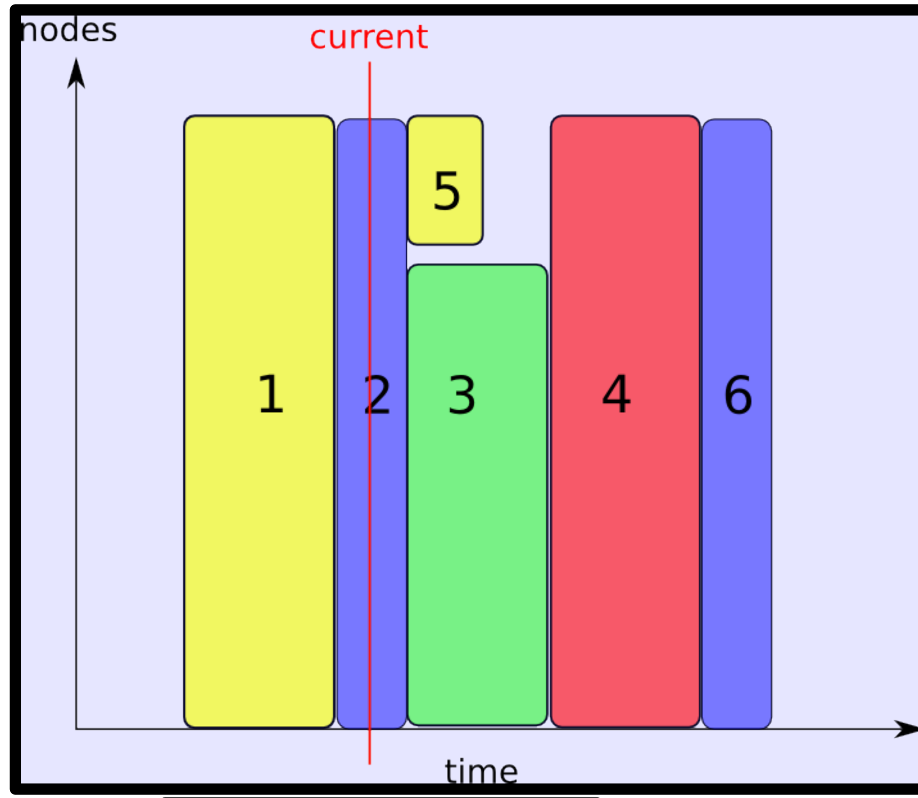
```
#slurm.conf file
SchedulerType=sched/backfill
SchedulerParameters=defer,bf_interval=60
FastSchedule=1
```

Scheduling – Backfill

Holes can be filled if previous jobs order is not changed



FIFO Scheduler



Backfill Scheduler

Scheduling Configuration Tips – Backfill

Important parameter for **backfill** to take effect is the **Walltime** of the job (Max time allowed for the job to be completed).

- Through command line option (--time=<Minutes>)
- Partitions or QOS can be declared with Walltime parameter and jobs submitted to these partitions inherit automatically those parameters.

Configuration of scheduler backfill in slurm.conf

Scheduler Parameters= bf_interval=#, bf_max_job_user=#,
bf_resolution=#,bf_window=#,max_job_bf=#

Preemption Policies

Preempt Types

None
Partition_prio priority on partition definition.
Qos quality of service defined in accounting database.

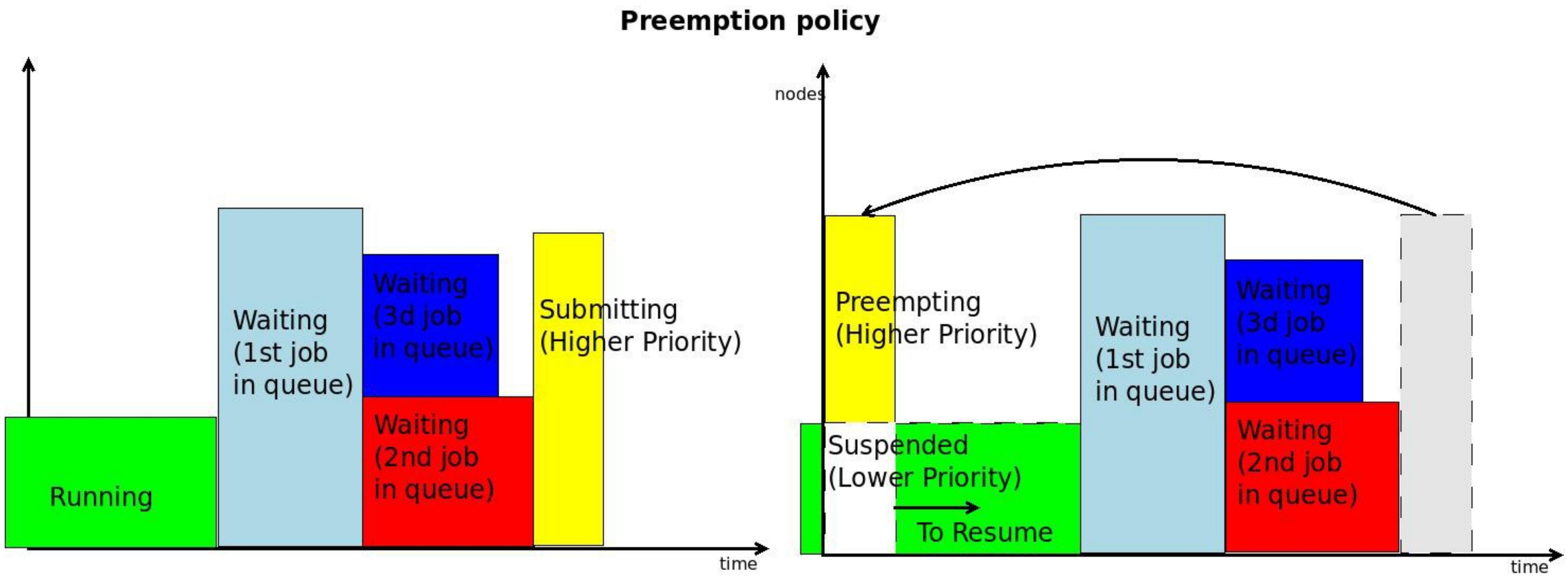
Preempt Modes

Off
Cancel preempted job is cancelled.
Checkpoint preempted job is checkpointed if possible, or cancelled.
Gang enables time slicing of jobs on the same resource.
Requeue job is requeued as restarted at the beginning (only for sbatch).
Suspend job is suspended until the higher priority job ends (requires Gang).

```
#slurm.conf file
PreemptMode=REQUEUE
PreemptType=preempt/qos
```

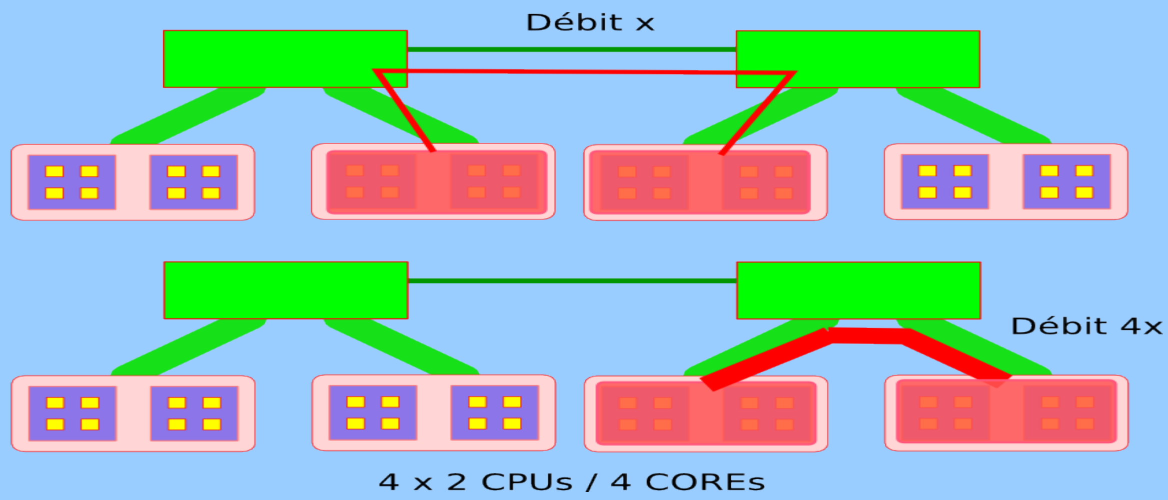
Scheduling - Preemption

Preemption policy allows higher priority jobs to execute without waiting upon the cluster resources by taking the place of the lower priority jobs



Network Topology Aware Placement

- topology/tree SLURM Topology aware plugin. **Best-Fit** selection of resources
- In fat-tree hierarchical topology: Bisection Bandwidth Constraints need to be taken into account




```
#slurm.conf file
TopologyPlugin=topology/tree
```

Configuration (topology.conf)

topology.conf file needs to exist on all computing nodes for network topology architecture description

```
# topology.conf file
SwitchName=Top
Switches=TS1,TS2,TS3,TS4,TS5,TS6,...

SwitchName=TS1 nodes=curie[1-18]
SwitchName=TS2 nodes=curie[19-37]
SwitchName=TS3 nodes=curie[38-56]
SwitchName=TS4 nodes=curie[57-75]
....
```

Network Topology Aware Placement

In the slurm.conf the **topology/tree** plugin may be activated by the admins to allow job placement according to network topology constraints

In the submission commands the users may use the **--switches=<count>[@<max-time>]** parameter to indicate how many switches their job would be ideal to execute upon: When a tree topology is used, this defines the maximum count of switches desired for the job allocation and optionally the maximum time to wait for that number of switches.

Generic Resources (Allocation of GPUs, MIC, etc)

Generic Resources (GRES) are resources associated with a specific node that can be allocated to jobs and steps. The most obvious example of GRES use would be GPUs. GRES are identified by a specific name and use an optional plugin to provide device-specific support.

SLURM supports no generic resourses in the default configuration. One must explicitly specify which resources are to be managed in the **slurm.conf** configuration file. The configuration parameters of interest are:

- **GresTypes** a comma delimited list of generic resources to be managed (e.g. GresTypes=gpu,nic). This name may be that of an optional plugin providing additional control over the resources.
- **Gres** the specific generic resource and their count associated with each node (e.g. NodeName=linux[0-999] Gres=gpu:8,nic:2) specified on all nodes and SLURM will track the assignment of each specific resource on each node. Otherwise SLURM will only track a count of allocated resources rather than the state of each individual device file.

Generic Resources (Allocation of GPUs, MIC, etc)

For configuration the new file **gres.conf** needs to exist on each compute node with gres resources

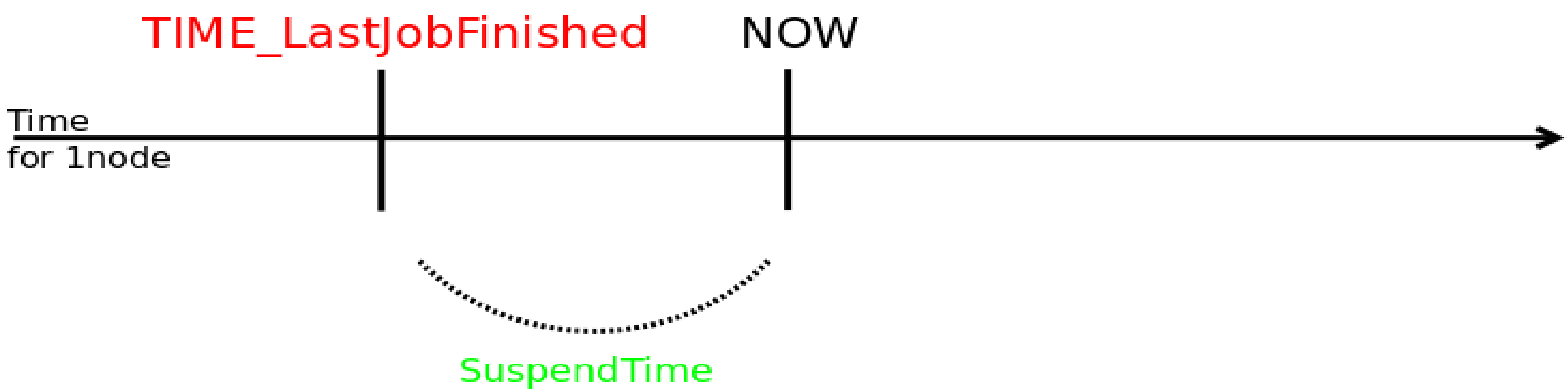
```
# Configure support for our four GPUs
Name=gpu File=/dev/nvidia0 CPUs=0,1
Name=gpu File=/dev/nvidia1 CPUs=0,1
Name=gpu File=/dev/nvidia2 CPUs=2,3
Name=gpu File=/dev/nvidia3 CPUs=2,3
```

For job execution the –gres option has to be used for to salloc, sbatch, and srun.

--gres=<list>Specifies a comma delimited list of generic consumableresources. The format of each entry on the list is**"name[:count]".**

Energy reduction techniques

- Parameters for energy reduction techniques
- Automatic node shut-down or other actions in case of resources **unutilization** during particular time.



Algorithm for SLURM Energy Reduction Techniques

```
Nodes Sleep Actions
        if    SuspendTime > A_PreDefined_Idle_TIME
                  exec SuspendProgram upon SuspendRate nodes per minute


Nodes WakeUp Actions
        if    SleepingNode_isNeeded  then
                  exec ResumeProgram upon ResumeRate nodes per minute
```

Energy reduction techniques Configuration

SuspendTime: Idle time to activate energy reduction techniques. A negative number disables power saving mode. The default value is -1 (disabled).

SuspendRate: # nodes added per minute. A value of zero results in no limits being imposed. The default value is 60. Use this to prevent rapid drops in power consumption.

ResumeRate: # nodes removed per minute. A value of zero results in no limits being imposed. The default value is 300. Use this to prevent rapid increases in power consumption.

SuspendProgram: Program to be executed to place nodes into power saving mode. The program executes as SlurmUser (as configured in slurm.conf). The argument to the program will be the names of nodes to be placed into power savings mode (using Slurm's hostlist expression format).

ResumeProgram: This program may use the scontrol show node command to insure that a node has booted and the slurmd daemon started.

SuspendTimeout, ResumeTimeout, SuspendExcNodes,SuspendExcParts, BatchStartTimeout

Controlling jobs' time limits

Srun parameter -t, --time=<time>
Slurm.conf parameter killwait

-t --time=<time>

Set a limit on the total run time of the job or job step. If the requested time limit for a job exceeds the partition's time limit, the job will be left in a PENDING state (possibly indefinitely). When the time limit is reached, each task in each job step is sent SIGTERM followed by SIGKILL.

killwait

The interval, in seconds, given to a job's processes between the SIGTERM and SIGKILL signals upon reaching its time limit. If the job fails to terminate gracefully in the interval specified, it will be forcibly terminated. default= 30sec

Controlling jobs' memory limits

Srun parameters –mem=<MB>,--mem-per-cpu=<MB>
Slurm.conf paramaters DefMemPerCPU, DefMemPerNode, MaxMemPerNode, MaxMemPerCPU

–mem=<MB>

Specify the real memory required per node in MB.Enforcement of memory limits currently relies upon enabling of accounting, which samples memory use on a periodic basis (data need not be stored, just collected). In both cases memory use is based upon the job's Resident Set Size (RSS). A task may exceed the memory limit until the next periodic accounting sample.

MaxMemPerNode

Maximum real memory size available per allocated node in MegaBytes. Used to avoid over-subscribing memory and causing paging.

Using different MPI libraries

OpenMPI

The system administrator must specify the range of ports to be reserved in the slurm.conf file using the MpiParams parameter. For example:

MpiParams=ports=12000-12999

Launch tasks using the srun command plus the option --resv-ports.

Alternately define the environment variable SLURM_RESV_PORT

srun –resv-ports -n <num_procs> a.out

If OpenMPI is configured with --with-pmi either pmi or pmi2 the OMPI jobs can be launched directly using the srun command. This is the preferred way. If the pmi2 support is enabled then the command line options '--mpi=pmi2' has to be specified on the srun command line.

srun --mpi=pmi2 -n <num_procs> a.out

Intel-MPI

Set the *I_MPI_PMI_LIBRARY* environment variable to point to the SLURM Process Management Interface (PMI) library:

export I_MPI_PMI_LIBRARY=/path/to/slurm/pmi/library/libpmi.so

Use the *srun* command to launch the MPI job:

srun -n <num_procs> a.out

Licenses Management

For local cluster configuration use parameter for slurm.conf Licenses
Specification of licenses (or other resources available on all nodes of the cluster) which can be allocated to jobs. License names can optionally be followed by a colon and count with a default count of one. Multiple license names should be comma separated (e.g. "Licenses=foo:4,bar").

For global cluster configuration use sacctmgr add/modify/delete resource
Example: sacctmgr add resource name=flexlis cluster=nazgul count=100 percentallowed=50 server=flex_host servertype=flexlm type=license
Check with scontrol show license

For execution use srun parameter -L, --licenses=<license>
Specification of licenses (or other resources available on all nodes of the cluster) which must be allocated to this job. License names can be followed by a colon and count (the default count is one). (e.g. "--licenses=foo:4,bar").

- Introduction
- SLURM scalable and flexible RJMS
- Part 1: Basics
  - Overview, Architecture, Configuration files, Partitions, Plugins, Reservations, Reconfiguration
- Part 2: Advanced Configuration
  - Accounting, Scheduling, Allocation, Network Topology Placement, Generic Resources, Licenses Management, Energy Reduction Techniques
- **Part 3: Experts Configuration**
  - **CPU Management, Isolation with cgroups, Power Management, Simulation and evaluation**
- Upcoming Features

SLURM Resource Management

- Fine Resource Management through submission command parameters
  - Adapted for **MultiCore/MultiThread** Architectures
  - Able to treat all kind of consumable resources (nodes, CPU hierarchies, memory, GPUs, etc)
- Based upon plpa/hwloc or Cpusets for **CPUs confinement** and optional **tasks binding** upon particular CPUs

CPU Management Documentation

HTML documents

- CPU Management User and Administrator Guide (http://www.schedmd.com/slurmdocs/cpu_management.html)

man pages

- slurm.conf
- srun
- salloc
- sbatch
- sacctmgr (for accounting limits)

Allocation Policies

Select Plugin Types

Linear entire nodes are allocated, regardless of the number of tasks (cpus) required.

Cons_res cpus and memory as a consumable resource. Individual resources on a node may be allocated (not shared) to different jobs. Options to treat CPUs, Cores, Sockets, and memory as individual resources that can be independently allocated. Useful for nodes with several sockets and several cores per socket.

```
#slurm.conf file
SelectType=select/cons_res
SelectTypeParameters=CR_Core_Memory,CR_CORE_DEFAULT_DIST_BLOCK
```

Task Assignment Policies

Task Plugin controls assignment (binding) of tasks to CPUs

None All tasks on a node can use all cpus on the node.
Cgroup cgroup subsystem is used to contain CPUs allocated to a job. Portable Hardware Locality (hwloc) library used to bind tasks to CPUs.
Affinity Bind tasks with one of the following *methods*
- **Cpusets** use cpuset subsystem to contain cpus assigned to tasks.
- **Sched** use sched_setaffinity to bind tasks to cpus.

In addition to the method, a *binding unit* may also be specified. It can be one of **Sockets, Cores, Threads, None**
Both the method and unit are specified on the TaskPluginParam statement.

```
#slurm.conf file
TaskPlugin=task/affinity
TaskPluginParam=Cpusets,Cores
```

CPU Management Terminology

Allocation

Assignment of a specific set of CPU resources (nodes, sockets, cores and/or threads) to a specific job or step

Distribution

1. Assignment of a specific task to a specific node, or
2. Assignment of a specific task to a specific set of CPUs within a node (used for optional Task-to-CPU binding)

Binding

Confinement/locking of a specific set of tasks to a specific set of CPUs within a node

SLURM Nodes/CPUs allocation procedure

SLURM uses four basic steps to manage CPU resources for a job/step:

Step 1: Selection of Nodes
Step 2: Allocation of CPUs from the selected Nodes
Step 3: Distribution of Tasks to the selected Nodes
Step 4: Optional Distribution and Binding of Tasks to CPUs within a Node

- SLURM provides a rich set of configuration and command line options to control each step
- Many options influence more than one step
- Interactions between options can be complex and difficult to predict
- Users may be constrained by Administrator's configuration choices

Notable Options for Step 1: Selection of Nodes

Configuration options in **slurm.conf**

Nodename: Defines a node and its characteristics. This includes the layout of sockets, cores, threads and the number of logical CPUs on the node.

FastSchedule: Allows administrators to define "virtual" nodes with different layout of sockets, cores and threads and logical CPUs than the physical nodes in the cluster.

PartitionName: Defines a partition and its characteristics. This includes the set of nodes in the partition.

Command line options on **srun/salloc/sbatch** commands

--partition, --nodelist: Specifies the set of nodes from which the selection is made

-N, --nodes: Specifies the minimum/maximum number of nodes to be selected

-B, --sockets-per-node, --cores-per-socket, --threads-per-core: Limits node selection to nodes with the specified characteristics

Notable Options for Step 2: Allocation of CPUs from Selected Nodes

Configuration options in **slurm.conf:**

SelectType:

SelectType=select/linear: Restricts allocation to whole nodes

SelectType=select/cons_res: Allows allocation of individual sockets, cores or threads as consumable resources

SelectTypeParameters: For select/cons_res, specifies the consumable resource type and default allocation method within nodes

Command line options on **srun/salloc/sbatch**:

-n, --ntasks: Specifies the number of tasks. This may affect the number of CPUs allocated to the job/step

-c, --cpus-per-task: Specifies the number of CPUs per task. This may affect the number of CPUs allocated to the job/step

Notable Options for Step 3: Distribution of Tasks to Nodes

Configuration options in **slurm.conf**:

MaxTasksPerNode: Specifies maximum number of tasks per node

Command Line options on **srun/salloc/sbatch**:

-m, --distribution: Controls the order in which tasks are distributed to nodes.

Configuration options in **slurm.conf:**

TaskPlugin:

TaskPlugin=task/none: Disables this step.

TaskPlugin=task/affinity: Enables task binding using the task affinity plugin.

TaskPlugin=task/cgroup: Enables task binding using the new task cgroup plugin.

TaskPluginParam: For task/affinity, specifies the binding unit (sockets, cores or threads) and binding method (sched_setaffinity or cpusets)

Command Line options on **srun/salloc/sbatch**:

--cpu_bind: Controls many aspects of task affinity

-m, --distribution: Controls the order in which tasks are distributed to allocated CPUs on a node for binding

Allocation & Distribution Methods

SLURM uses two default methods for allocating and distributing individual CPUs from a set of resources

- **block** method: Consume all eligible CPUs consecutively from a single resource before using the next resource in the set
- **cyclic** method: Consume eligible CPUs from each resource in the set consecutively in a round-robin fashion

The following slides illustrate the default method used by SLURM for each step.

Different ways of selecting resources in SLURM:

- Cyclic method (Balance between nodes / Round Robin)
- Block method (Minimization of fragmentation)

- **Cyclic**

```
[bench@wardlaw0 ~]$ srun -n10 -N2 –exclusive /bin/hostname
wardlaw67
wardlaw67
wardlaw67
wardlaw67
wardlaw67
wardlaw66
wardlaw66
wardlaw66
wardlaw66
wardlaw66
```

- **Block**

```
[bench@wardlaw0 ~]$ srun -n10 -N2 /bin/hostname
wardlaw67
wardlaw67
wardlaw67
wardlaw67
wardlaw67
wardlaw67
wardlaw67
wardlaw67
wardlaw67
wardlaw66
```

Allocation & Distribution Methods

Step 2a: Allocation of CPUs from a set of Nodes

The default allocation method for this case is **block**
Example: A partition contains 3 nodes. Each node has 8 available CPUs.

```
srun –-nodes=3 –-ntasks=15 ...
```

Node	n0	n1	n2
Number of allocated CPUs	8	6	1

Users can override this default using the appropriate command line options, e.g.
`--nodes, --ntasks-per-node`

Allocation & Distribution Methods

Step 2b: Allocation of CPUs from a set of Sockets within a Node

The default allocation method for this case is **cyclic**
Example: A node contains 2 sockets. Each socket has 4 available CPUs.

```
srun –-ntasks=6 ...
```

Socket#	0	1
Number of allocated CPUs	3	3

Administrators can change the default allocation method for this case to **block** with `SelectTypeParameters=CR_CORE_DEFAULT_DIST_BLOCK`

Users can override the default using the command line option `--distribution`

Allocation & Distribution Methods

Step 3: Distribution of Tasks to Nodes

The default distribution method for this case is **block**
Example: A partition has 3 nodes. Each node has 8 available cpus.

```
srun –-nodes=3 –-ntasks=8 –-cpus-per-task=2 ...
```

Node	n0	n1	n2
Number of allocated CPUs	8	6	2
Distribution of Tasks, by Task#	0 - 3	4 - 6	7

Users can override this default using the command line option `--distribution`. The option supports three alternate methods for distributing tasks to nodes: **cyclic**, **plane** and **arbitrary**.

Allocation & Distribution Methods

Step 4: Distribution of Tasks to Allocated CPUs within a Node for Task-to-CPU binding

The default distribution method for this case is **cyclic**
Example: A node has 2 sockets. Each socket has 4 available CPUs (cores).

```
srun –-ntasks=8 --cpu_bind=cores
```

Node	n0							
Socket#	**0**				**1**			
CPU#	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Bound Task#	0	2	4	6	1	3	5	7

Users can override this default and specify **block** distribution of tasks to CPUs using the command line option `–-distribution`

CPU Resource Sharing

- **<u>CPU Resource Sharing between Jobs</u>**

- **Shared** keyword on partition definition in slurm.conf
- **--share**, **--exclusive** options on srun/salloc/sbatch command line

Example:

```
Nodename=n0 CPUs=8
PartitionName=p1 Nodes=n0 Shared=YES

    srun –-partition=p1 –-ntasks=8 –-share …
    srun –-partition=p1 –-ntasks=8 –-share …
```

CPU Resource Sharing

- **CPU Resource Sharing between Tasks of the same Job**

- **--overcommit** option on srun/salloc/sbatch command line

  Example:

```
NodeName=n0 CPUs=8
PartitionName=p1 Nodes=n0

srun ––partition=p1 ––ntasks=12 ––overcommit …
```

Note:

It is important to understand that the Linux scheduler is not aware of CPU allocations by SLURM. Unless Task-to-CPU binding is used, Linux may run any task on any CPU on the node to which the task was distributed. In this way, CPUs may be shared between tasks and jobs even in the absence of shared CPU allocation by SLURM.

CPU management by SLURM users is subject to limits imposed by SLURM Accounting. Accounting limits may be applied on CPU usage at the level of users, associations and clusters. CPU-related accounting limits include the following:

MaxNodes: Maximum number of nodes that can be selected for each job.

MaxCPUs: Maximum number of CPUs that can be allocated to each job.

GrpNodes: Maximum number of nodes that can be selected for all running jobs combined in this association

GrpCPUs: Maximum number of CPUs that can be allocated to all running jobs combined in this association

For more details, see the sacctmgr man page.

Information about Node Selection and CPU allocation (Steps 1 & 2)

```
[sulu] (slurm) mnp> srun --nodes=2 --ntasks=3 --cpus-per-task=2 sleep 60
[2] 22841

[sulu] (slurm) mnp> squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
    309  allnodes     sleep    slurm  R       0:02      2  n[6-7]

[sulu] (slurm) mnp> scontrol --details show job 309
   . . .
   NumNodes=2 NumCPUs=6 CPUs/Task=2 ReqS:C:T=*:*:*
     Nodes=n6 CPU_IDs=4-7 Mem=0
     Nodes=n7 CPU_IDs=6-7 Mem=0
   . . .
```

Note:
The CPU_IDs reported by scontrol are SLURM abstract CPU numbers, not physical CPU numbers known to Linux.

Getting Information About CPU Usage (cont'd)

Information about Distribution of Tasks to Nodes (Step 3)

```
[sulu] (slurm) mnp> srun --nodes=3 --ntasks=5 sleep 60 &


[sulu] (slurm) mnp> squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
    311  allnodes    sleep    slurm   R       0:03      3 n[6-7,13]


[sulu] (slurm) mnp> sstat --allsteps --jobs 311 --pidformat
       JobID                 Nodelist                 Pids
------------ -------------------- --------------------
311.0                             n13            7994,7995
311.0                              n6            3838,3839
311.0                              n7                 7199
```

Information about Task-to-CPU binding (Step 4)

```
[sulu] (slurm) mnp> srun --partition=bones-scotty --nodes=2 --ntasks=4
--cpu_bind=cores,verbose --label cat /proc/self/status | grep
Cpus_allowed_list

0: cpu_bind=MASK - scotty, task  0  0 [4070]: mask 0x8 set
3: cpu_bind=MASK - bones, task  3  0 [23808]: mask 0x80 set
1: cpu_bind=MASK - scotty, task  1  1 [4071]: mask 0x20 set
2: cpu_bind=MASK - scotty, task  2  2 [4072]: mask 0x80 set

3: Cpus_allowed_list:   7
0: Cpus_allowed_list:   3
1: Cpus_allowed_list:   5
2: Cpus_allowed_list:   7
```

<u>Example 2: Consumable Resources with balanced allocation across Nodes</u>

A job requires 9 CPUs (3 tasks and 3 CPUs per task). Allocate 3 CPUs from each of the 3 nodes in the default partition.

<u>In slurm.conf</u>:
```
SelectType=select/cons_res
SelectTypeParameters=CR_Core
```

<u>srun command line</u>:
```
srun --ntasks=3 –-cpus-per-task=3 –-ntasks-per-node=1 ...
```

To satisfy `--ntasks-per-node=1`, SLURM must allocate 3 CPUs on each of the 3 nodes in the default partition. This overrides the default method for allocating CPUs from a set of nodes (block allocation).

Example 3: Consumable Resources with minimization of resource fragmentation

A job requires 12 CPUs. Allocate CPUs using the minimum number of nodes (2 nodes) and the minimum number of sockets (3 sockets) required for the job in order to minimize fragmentation of allocated/unallocated CPUs in the cluster.

In slurm.conf:

```
SelectType=select/cons_res
SelectTypeParameters=CR_Core,CR_CORE_DEFAULT_DIST_BLOCK
```

srun command line:

```
srun --ntasks=12
```

The default allocation method across a selection of nodes is block. This minimizes the number of nodes used for the job. The configuration option `CR_CORE_DEFAULT_DIST_BLOCK` sets the default allocation method across sockets within a node to block. This minimizes the number of sockets used for the job within a node.

Example 4: Consumable resources with Task-to-CPU binding and non-default allocation and distribution methods

A job requires 18 CPUs. Allocate 6 CPUs on each node using block allocation within nodes. Use cyclic distribution of tasks to nodes and block distribution of tasks to CPUs for binding.

In slurm.conf:

```
SelectType=select/cons_res
SelectTypeParameters=CR_Core
TaskPlugin=task/affinity
TaskPluginParam=sched
```

srun command line:

```
srun --ntasks=18 –-ntasks-per-node=6 --distribution=cyclic:block
--cpu_bind=cores ...
```

(continued)

- `--ntasks-per-node=6` overrides the default allocation method across nodes (block) and causes SLURM to allocate 6 CPUs on each of the 3 nodes in the default partition.

- `--distribution cyclic:xxxxx` overrides the default method for distributing tasks to nodes (block).

- `task/affinity` plus `––cpu_bind=cores` causes SLURM to bind each task to a single allocated core.

- `--distribution xxxxx:block` overrides the default allocation method within nodes (cyclic) and the default method for distributing tasks to CPUs for binding (cyclic).

The following table depicts a possible pattern of allocation, distribution and binding for this job.

(continued)

```
srun --ntasks=18 --ntasks-per-node=6 --distribution=cyclic:block
 --cpu_bind=cores ...
```

Node		n0								n1								n2							
Socket#		0				1				0				1				0				1			
Number of Allocated CPUs		4				2				4				2				4				2			
Allocated CPU#'s		0 - 5								0 - 5								0 - 5							
Number of Tasks		6								6								6							
Distribution of Tasks to Nodes, by Task#		0, 3, 6, 9, 12, 15								1, 4, 7, 10, 13, 16								2, 5, 8, 11, 14, 17							
Task-to-CPU Binding	CPU#	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
	Task#	0	3	6	9	12	15	-	-	1	4	7	10	13	16	-	-	2	5	8	11	14	17	-	-

Issues with no Task Confinement upon the allocated resources

- SMP system without some means of CPU placement, any task can run on any CPU.

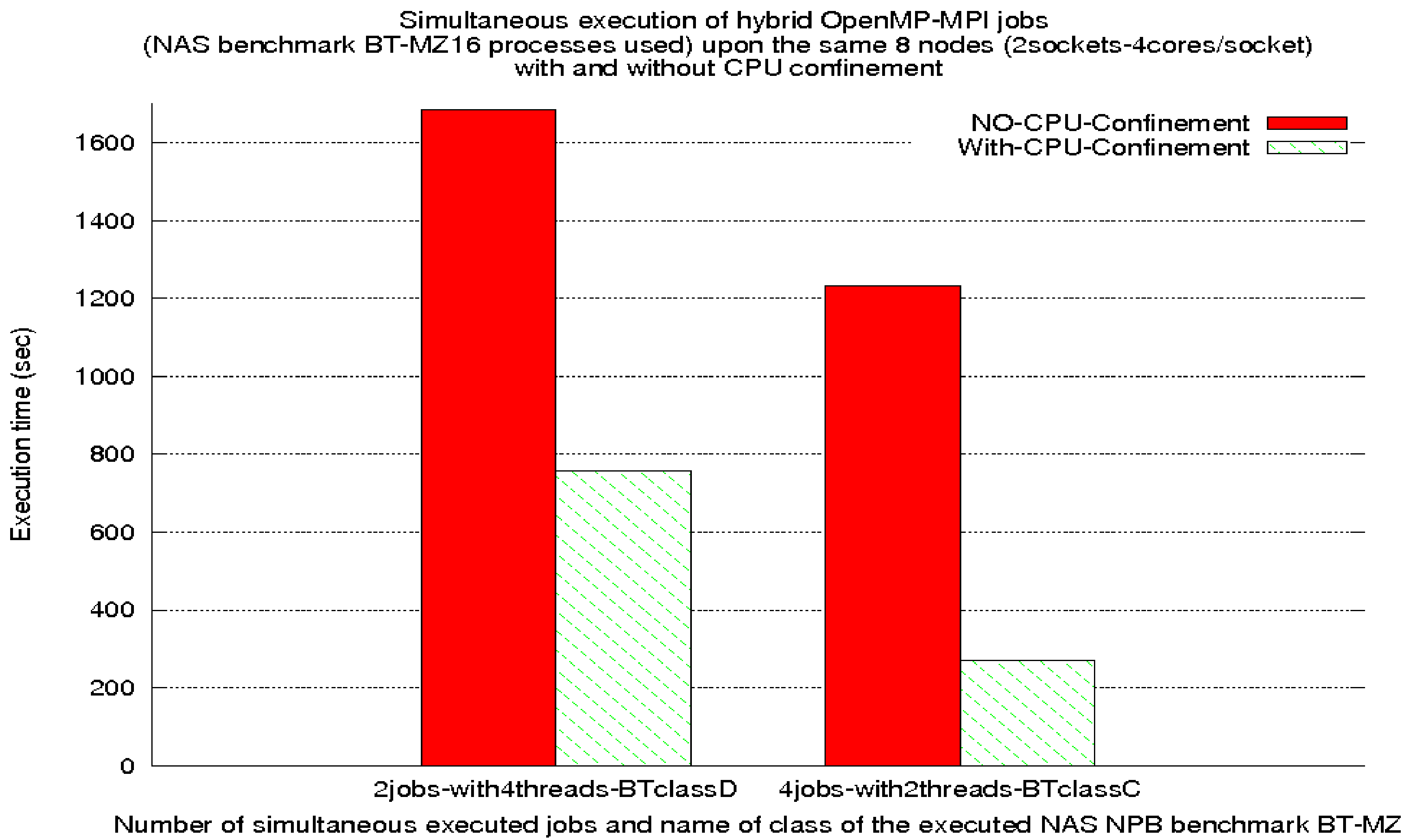
  This may cause CPU idleness while other CPUs are shared and system time spent on migrating tasks between processors

- NUMA system, any memory page can be allocated on any node.

  This can cause both poor cache locality and poor memory access times.

Issues with no Task Confinement upon the allocated resources

Execution of NAS benchmarks (MPI-OpenMP) upon 8 nodes SMP cluster (2sockets-4cores/socket) with and without CPU confinement



Issues with no Task Confinement upon the allocated resources

CPU instant utilization during execution of 2 BT-MZ jobs with 16 tasks and 4 threads per task without CPU confinement

```
top - 13:32:36 up  1:21,  1 user,  load average: 7.05, 5.18, 6.79
Tasks: 256 total,   9 running, 247 sleeping,   0 stopped,   0 zombie
Cpu0  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu4  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu5  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu6  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu7  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  18395656k total,  3346396k used, 15049260k free,    5764k buffers
Swap:  1022752k total,        0k used,  1022752k free,   114104k cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  P COMMAND
 3582 georgioy  20   0  920m 713m 3956 R 54.8  4.0   0:45.34 3 bt-mz.D.16
 3581 georgioy  20   0  921m 717m 4192 R 52.5  4.0   0:45.47 1 bt-mz.D.16
 3592 georgioy  20   0  921m 713m 3924 R 51.2  4.0   0:43.02 2 bt-mz.D.16
 3577 georgioy  20   0  920m 717m 3940 R 50.8  4.0   0:44.81 0 bt-mz.D.16
 3578 georgioy  20   0  921m 713m 3924 R 50.2  4.0   0:45.37 3 bt-mz.D.16
 3594 georgioy  20   0  920m 717m 3940 R 48.5  4.0   0:43.48 0 bt-mz.D.16
 3598 georgioy  20   0  921m 717m 4192 R 48.2  4.0   0:43.14 1 bt-mz.D.16
 3597 georgioy  20   0  920m 713m 3956 R 43.9  4.0   0:43.18 2 bt-mz.D.16
    1 root      20   0 21336 1548 1280 S  0.0  0.0   0:03.60 5 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 5 kthreadd
```

CPUs are shared between jobs

While there are idle CPUs

Advantages: cgroups support for HPC

- To guarantee that every consumed resources is consumed the way it's planned to be
  - leveraging Linux latest features in terms of process control and resource management
  - Enabling node sharing
- While enhancing the connection with Linux systems
  - Improve **tasks isolation** upon resources
  - Improve **efficiency** of resource management activities (e.g., process tracking, collection of accounting statistics)
  - Improve **robustness** (e.g. more reliable cleanup of jobs)
- And simplifying the addition of **new controlled resources and features**
  - prospective management of network and I/O as individual resources

Introduction to cgroups

Control Groups (cgroups) is a **Linux kernel mechanism** (appeared in 2.6.24) to limit, isolate and monitor resource usage (CPU, memory, disk I/O, etc.) of groups of processes.

<u>Features</u>

- ***Resource Limiting*** (i.e. not to exceed a memory limit)
- ***Prioritization*** (i.e. groups may have larger share of CPU)
- ***Isolation*** (i.e. isolate GPUs for particular processes)
- ***Accounting*** (i.e. montior resource usage for processes)
- ***Control*** (i.e. suspending and resuming processes)

Cgroups Model and Concepts

Model

Cgroups **similar** to Linux processes:

- Hierarchical
- Inheritance of attributes from parent to child

but **different** because:

- **multiple hierarchies** of cgroups may exist that are attached to one or more subsystems

Concepts

- **Cgroup** – a group of processes with the same characteristics
- **Subsystem** – a module that applies parameters to a group of processes (cgroup)
- **Hierarchy** – a set of cgroups organized in a tree, plus one or more subsystems associated with that tree

Cgroups subsystems

- **cpuset** – assigns tasks to individual CPUs and memory nodes in a cgroup
- **cpu** – schedules CPU access to cgroups
- **cpuacct** – reports CPU resource usage of tasks of a cgroup
- **memory** – set limits on memory use and reports memory usage for a cgroup
- **devices** – allows or denies access to devices (i.e. gpus) for tasks of a cgroup
- **freezer** – suspends and resumes tasks in a cgroup
- **net_cls** – tags network packets in a cgroup to allow network traffic priorities
- **ns** – namespace subsystem
- **blkio** – tracks I/O ownership, allowing control of access to block I/O resources

Cgroups functionality rules

- Cgroups are represented as **virtual file systems**
  - Hierarchies are directories, created by mounting subsystems, using the mount command; subsystem names specified as mount options
  - Subsystem parameters are represented as files in each hierarchy with values that apply only to that cgroup
- **Interaction with cgroups** take place by manipulating directories and files in the cgroup virtual file system using standard shell commands and system calls (mkdir, mount, echo, etc)
  - *tasks* file in each cgroup directory lists the tasks (pids) in that cgroup
  - Tasks are automatically removed from a cgroup when they terminate or are added to a different cgroup in the same hierarchy
  - Each task is present in only one cgroup in each hierarchy
- Cgroups have a mechanism for **automatic removal** of abandoned cgroups (release_agent)

Cgroups subsystems parameters

cpuset subsystem

cpuset.cpus: defines the set of cpus that the tasks in the cgroup are allowed to execute on

cpuset.mems: defines the set of memory zones that the tasks in the cgroup are allowed to use

memory subsystem

memory.limit_in_bytes: defines the memory limit for the tasks in the cgroup

memory.swappiness: controls kernel reclamation of memory from the tasks in the cgroup (swap priority)

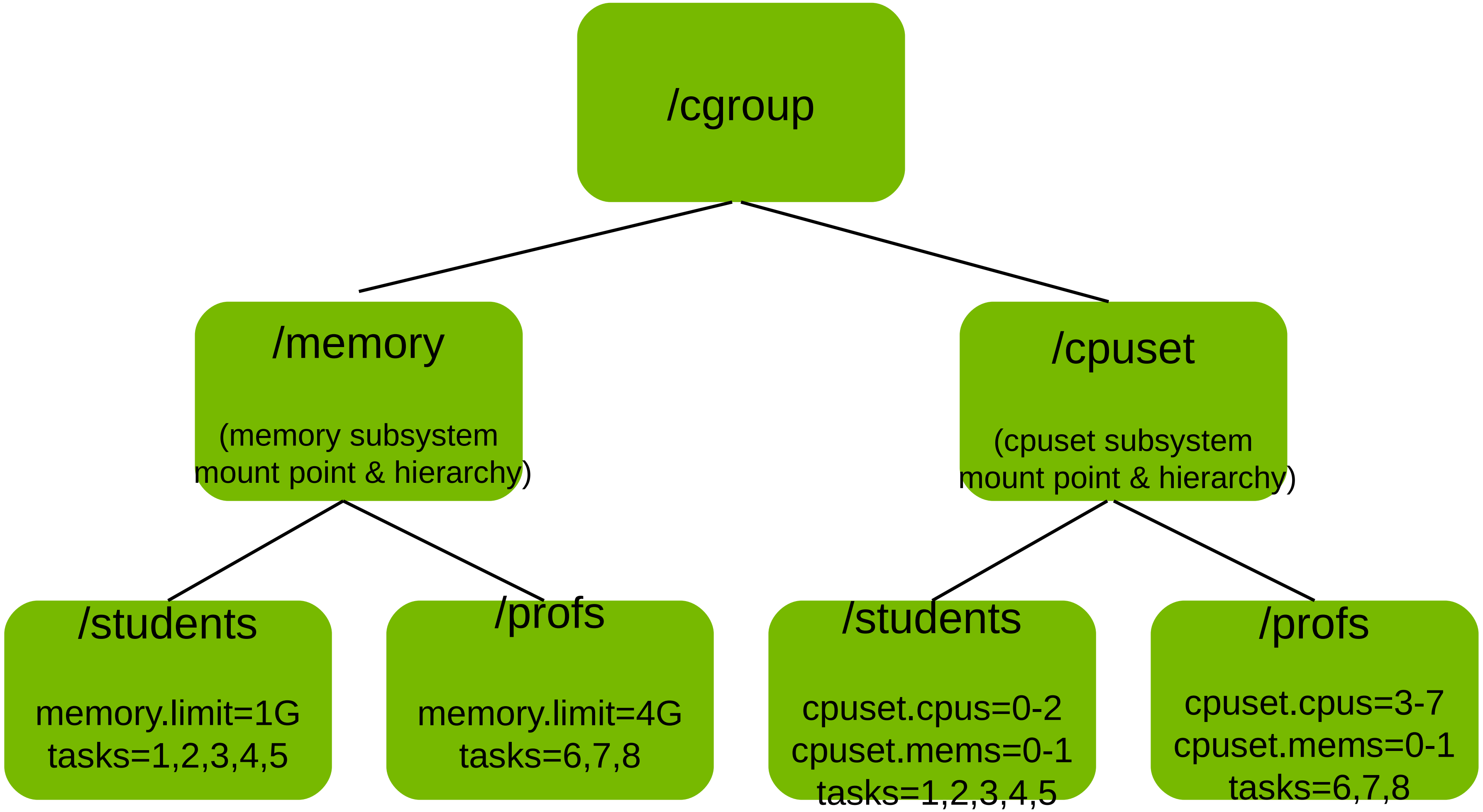
freezer subsystem

freezer.state: controls whether tasks in the cgroup are active (runnable) or suspended

devices subsystem

devices_allow: specifies devices to which tasks in a cgroup have acces

Cgroups functionality example



Cgroups functionality example

```
[root@mordor:~]# mkdir /cgroup
[root@mordor:~]# mkdir /cgroup/cpuset
[root@mordor:~]# mount -t cgroup -o cpuset none /cgroup/cpuset
[root@mordor:~]# ls /cgroup/cpuset/
cpuset.cpus  cpuset.mems tasks notify_on_release release_agent
[root@mordor:~]# mkdir /cgroup/cpuset/students
[root@mordor:~]# mkdir /cgroup/cpuset/profs
[root@mordor:~]# echo 0-2 > /cgroup/cpuset/students/cpuset.cpus
[root@mordor:~]# echo 0 > /cgroup/cpuset/students/cpuset.mems
[root@mordor:~]# echo $PIDS_st > /cgroup/cpuset/students/tasks
[root@mordor:~]# echo 3-7 > /cgroup/cpuset/profs/cpuset.cpus
[root@mordor:~]# echo 1 > /cgroup/cpuset/profs/cpuset.mems
[root@mordor:~]# echo $PIDS_pr > /cgroup/cpuset/profs/tasks
```

Process Tracking with Cgroups

Track job processes using the freezer subsystem

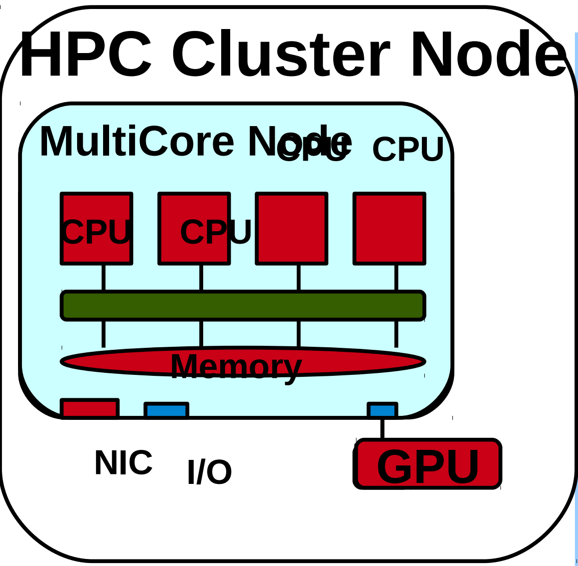
- Every spawned process is tracked
  - Automatic inheritance of parent's cgroup
  - No way to escape the container
- Every processes can be frozen
  - Using the Thawed|Frozen state of the subsystem
  - No way to avoid the freeze action

Cgroup **Proctrack** plugin: **freezer** subsystem

[mat@leaf slurm]$ srun sleep 300

```
[root@leaf ~]# cat /cgroup/freezer/uid_500/job_53/step_0/freezer.state
THAWED
[root@leaf ~]# scontrol suspend 53
[root@leaf ~]# ps -ef f | tail -n 2
root     15144     1  0 17:10 ?        Sl     0:00 slurmstepd: [53.0]
mat      15147 15144  0 17:10 ?        T      0:00  \_ /bin/sleep 300
[root@leaf ~]# cat /cgroup/freezer/uid_500/job_53/step_0/freezer.state
FREEZING
[root@leaf ~]# scontrol resume 53
[root@leaf ~]# ps -ef f | tail -n 2
root     15144     1  0 17:10 ?        Sl     0:00 slurmstepd: [53.0]
mat      15147 15144  0 17:10 ?        S      0:00  \_ /bin/sleep 300
[root@leaf ~]# cat /cgroup/freezer/uid_500/job_53/step_0/freezer.state
THAWED
[root@leaf ~]#
```

Task confinement for allocated resources

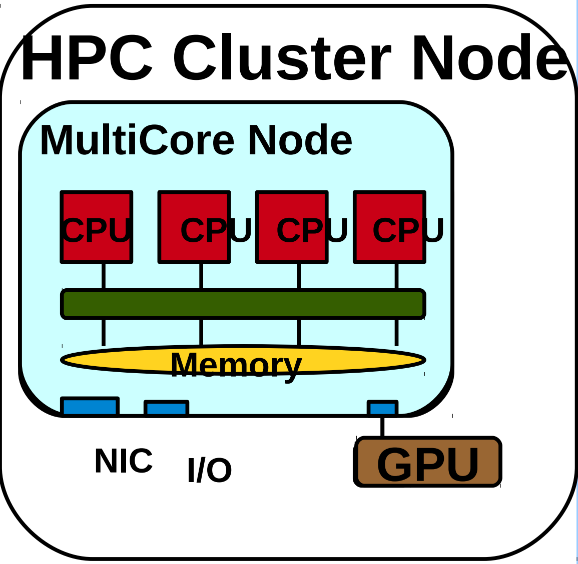


Constrain jobs tasks to the allocated resources

- 3 independant layers of managed resources using 3 subsystems
  - Cores (**cpuset**), Memory (**memory**),

    GRES (**devices**)
- Every spawned process is tracked
  - Automatic inheritance of parent's cgroup
  - No escape, no way to use additional resources,
- Each layer has its own additional parameters
- More resources could be added in the future

Task confinement for cpus



Constrain jobs tasks to the allocated cores

- Configurable feature
  - ConstrainCores=yes|no
- Use step's allocated cores with “exclusive steps”
  - Otherwise, let steps use job's allocated cores
- Basic affinity management as a configurable sub-feature
  - TaskAffinity=yes|no in cgroup.conf (rely on HWLOC)
  - Automatic block and cyclic distribution of tasks

Cgroup **Task** plugin : **cpuset** subsystem

```
[mat@leaf slurm]$ salloc --exclusive srun -n1 --cpu_bind=none sleep
 3000
salloc: Granted job allocation 55
```

```
[root@leaf ~]# egrep "Cores|Affinity" /etc/slurm/cgroup.conf
ConstrainCores=yes
TaskAffinity=yes
[root@leaf ~]# tail -f /var/log/slurmd.leaf10.log |grep task/cgroup
[2011-09-16T17:24:59] [55.0] task/cgroup: now constraining jobs allocated
 cores
[2011-09-16T17:24:59] [55.0] task/cgroup: loaded
[2011-09-16T17:24:59] [55.0] task/cgroup: job abstract cores are '0-31'
[2011-09-16T17:24:59] [55.0] task/cgroup: step abstract cores are '0-31'
[2011-09-16T17:24:59] [55.0] task/cgroup: job physical cores are '0-31'
[2011-09-16T17:24:59] [55.0] task/cgroup: step physical cores are '0-31'
[2011-09-16T17:24:59] [55.0] task/cgroup: task[0] is requesting no affinity
```

Cgroup **Task** plugin : **cpuset** subsystem

```
[mat@leaf slurm]$ salloc --exclusive srun -n1 --cpu_bind=cores sleep 3000
salloc: Granted job allocation 57
```

```
[root@leaf ~]# egrep "Cores|Affinity" /etc/slurm/cgroup.conf
ConstrainCores=yes
TaskAffinity=yes
[root@leaf ~]# tail -f /var/log/slurmd.leaf10.log |grep task/cgroup
[2011-09-16T17:31:17] [57.0] task/cgroup: now constraining jobs allocated cores
[2011-09-16T17:31:17] [57.0] task/cgroup: loaded
[2011-09-16T17:31:17] [57.0] task/cgroup: job abstract cores are '0-31'
[2011-09-16T17:31:17] [57.0] task/cgroup: step abstract cores are '0-31'
[2011-09-16T17:31:17] [57.0] task/cgroup: job physical cores are '0-31'
[2011-09-16T17:31:17] [57.0] task/cgroup: step physical cores are '0-31'
[2011-09-16T17:31:17] [57.0] task/cgroup: task[0] is requesting core level binding
[2011-09-16T17:31:17] [57.0] task/cgroup: task[0] using Core granularity
[2011-09-16T17:31:17] [57.0] task/cgroup: task[0] taskset '0x00000001' is set
```

Cgroup **Task** plugin : **cpuset** subsystem

```
[mat@leaf slurm]$ salloc --exclusive srun -n1 --cpu_bind=socket sleep 3000
salloc: Granted job allocation 58
```

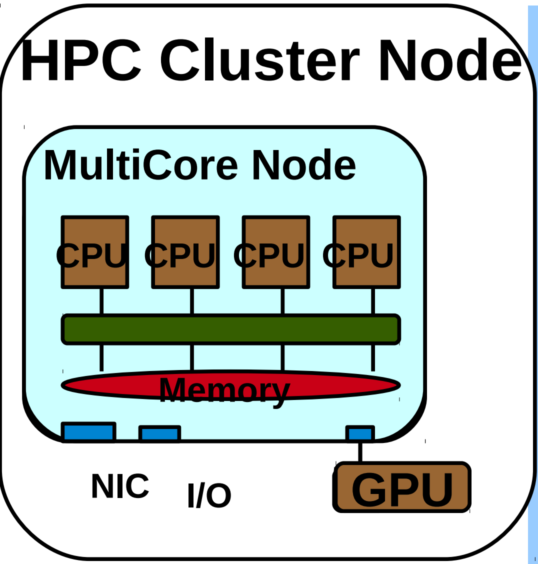
```
 [root@leaf ~]# egrep "Cores|Affinity" /etc/slurm/cgroup.conf
ConstrainCores=yes
TaskAffinity=yes
[root@leaf ~]# tail -f /var/log/slurmd.leaf10.log |grep task/cgroup
[2011-09-16T17:33:31] [58.0] task/cgroup: now constraining jobs allocated cores
[2011-09-16T17:33:31] [58.0] task/cgroup: loaded
[2011-09-16T17:33:31] [58.0] task/cgroup: job abstract cores are '0-31'
[2011-09-16T17:33:31] [58.0] task/cgroup: step abstract cores are '0-31'
[2011-09-16T17:33:31] [58.0] task/cgroup: job physical cores are '0-31'
[2011-09-16T17:33:31] [58.0] task/cgroup: step physical cores are '0-31'
[2011-09-16T17:33:31] [58.0] task/cgroup: task[0] is requesting socket level binding
[2011-09-16T17:33:31] [58.0] task/cgroup: task[0] using Socket granularity
[2011-09-16T17:33:31] [58.0] task/cgroup: task[0] taskset '0x00000003' is set
```

Cgroup **Task** plugin : **cpuset** subsystem

```
[mat@leaf slurm]$ salloc --exclusive srun -n2 --cpu_bind=socket sleep 3000
salloc: Granted job allocation 60
```

```
[root@leaf ~]# egrep "Cores|Affinity" /etc/slurm/cgroup.conf
ConstrainCores=yes
TaskAffinity=yes
[root@leaf ~]# tail -f /var/log/slurmd.leaf10.log |grep task/cgroup[2011-09-16T17:36:18] [60.0]
 task/cgroup: now constraining jobs allocated cores
[2011-09-16T17:36:18] [60.0] task/cgroup: loaded
[2011-09-16T17:36:18] [60.0] task/cgroup: job abstract cores are '0-31'
[2011-09-16T17:36:18] [60.0] task/cgroup: step abstract cores are '0-31'
[2011-09-16T17:36:18] [60.0] task/cgroup: job physical cores are '0-31'
[2011-09-16T17:36:18] [60.0] task/cgroup: step physical cores are '0-31'
[2011-09-16T17:36:18] [60.0] task/cgroup: task[0] is requesting socket level binding
[2011-09-16T17:36:18] [60.0] task/cgroup: task[1] is requesting socket level binding
[2011-09-16T17:36:18] [60.0] task/cgroup: task[1] using Core granularity
[2011-09-16T17:36:18] [60.0] task/cgroup: task[1] higher level Socket found
[2011-09-16T17:36:18] [60.0] task/cgroup: task[1] taskset '0x00000003' is set
[2011-09-16T17:36:18] [60.0] task/cgroup: task[0] using Core granularity
[2011-09-16T17:36:18] [60.0] task/cgroup: task[0] higher level Socket found
[2011-09-16T17:36:18] [60.0] task/cgroup: task[0] taskset '0x00000003' is set
```

Task confinement for memory : **memory** subsystem



Constrain jobs tasks to the allocated amount of memory

- Configurable feature
  - ConstrainRAMSpace=yes|no
  - ConstrainSwapSpace=yes|no
- Use step's allocated amount of memory with “exclusive steps”
  - Else, let steps use job's allocated amount
- Both RSS and swap are monitored
- Trigger OOM killer on the cgroup's tasks when reaching limits
- Tolerant mechanism
  - AllowedRAMSpace , AllowedSwapSpace percents

Cgroup **Task** plugin : **memory** subsystem

```
[mat@leaf slurm]$ salloc --exclusive --mem-per-cpu 100 srun -n1 sleep 3000
salloc: Granted job allocation 67
```

```
[root@leaf ~]# tail -f /var/log/slurmd.leaf10.log |grep task/cgroup
[2011-09-16T17:55:20] [67.0] task/cgroup: now constraining jobs allocated memory
[2011-09-16T17:55:20] [67.0] task/cgroup: loaded
[2011-09-16T17:55:20] [67.0] task/cgroup: job mem.limit=3520MB memsw.limit=3840MB
[2011-09-16T17:55:20] [67.0] task/cgroup: step mem.limit=3520MB memsw.limit=3840MB
```

```
[mat@leaf slurm]$ salloc --exclusive --mem-per-cpu 100 srun –
 exclusive -n1 sleep 3000
salloc: Granted job allocation 68
```

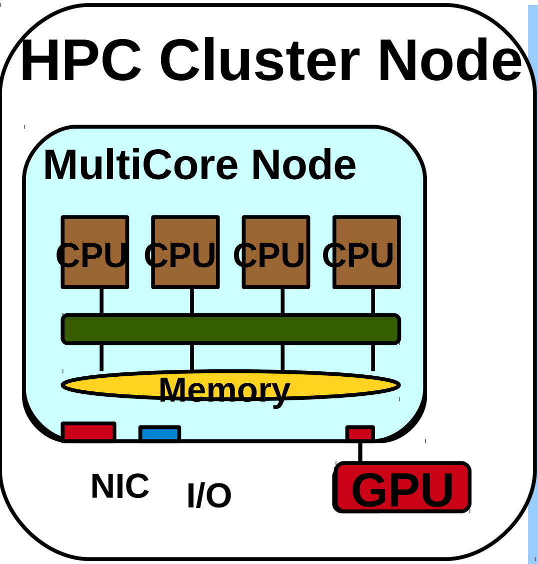
```
[root@leaf ~]# tail -f /var/log/slurmd.leaf10.log |grep task/cgroup
[2011-09-16T17:57:31] [68.0] task/cgroup: now constraining jobs allocated memory
[2011-09-16T17:57:31] [68.0] task/cgroup: loaded
[2011-09-16T17:57:31] [68.0] task/cgroup: job mem.limit=3520MB memsw.limit=3840MB
[2011-09-16T17:57:31] [68.0] task/cgroup: step mem.limit=110MB memsw.limit=120MB
```

Cgroup **Task** plugin : **memory** subsystem
OOM killer usage

```
[mat@leaf slurm]$ salloc --exclusive --mem-per-cpu 100 srun -n1 sleep
 3000
salloc: Granted job allocation 67
```

```
slurmd[berlin27]: Step 268.0 exceeded 1310720 KB
memory limit, being killed
srun: Exceeded job memory limit
srun: Job step aborted: Waiting up to 2 seconds for job step
to finish.
slurmd[berlin27]: *** STEP 268.0 KILLED AT 2012-03-
31T15:50:36 WITH SIGNAL 9 ***
srun: error: berlin27: tasks 0,1: Killed
```

Tasks confinement for devices: **devices** subsystem



Constrain jobs tasks to the allocated system devices

- Based on the **GRES** plugin for generic resources allocation (NIC, GPUs, etc) and built upon the cgroup task plugin
  - Each task is allowed to access to a number of devices by default
  - Only the tasks that have granted allocation on the **GRES** devices will be allowed to have access on them.
  - Tasks with no granted allocation upon **GRES** devices will not be able to use them.

Cgroup Devices Configuration Example

```
[root@mordor cgroup]# egrep "Devices" /etc/slurm/cgroup.conf
ConstrainDevices=yes
AllowedDevicesFile="/etc/slurm/allowed_devices.conf"
```

```
[root@mordor cgroup]# cat /etc/slurm/allowed_devices.conf
/dev/sda*
/dev/null
/dev/zero
/dev/urandom
/dev/cpu/*/*
```

Cgroup **Task** plugin : **devices** subsystem

Cgroup Devices Logic as implemented in task plugin

1) Initialization phase (information collection gres.conf file, major, minor, etc)

2) Allow all devices that should be allowed by default (allowed_devices.conf)

3) Lookup which gres devices are allocated for the job

- Write allowed gres devices to devices.allow file
- Write denied gres devices to devices.deny file

4) Execute **2** and **3** for job and steps tasks (different hierarchy level in cgroups)

Cgroups **devices** subsystem : Usage Example

```
[root@mordor cgroup]# egrep "Gres" /etc/slurm/slurm.conf
GresTypes=gpu
NodeName=cuzco[57,61] Gres=gpu:2 Procs=8 Sockets=2 CoresPerSocket=4
```

```
[root@cuzco51]# cat /etc/slurm/allowed_devices.conf
/dev/sda*
/dev/null
```

```
[gohn@cuzco0]$ cat gpu_test.sh
#!/bin/sh
sleep 10
echo 0 > /dev/nvidia0
echo 0 > /dev/nvidia1
```

Cgroups **devices** subsystem : Usage Example

```
[gohn@cuzco0]$ srun -n1 –gres=gpu:1 -o output ./gpu_test.sh
```

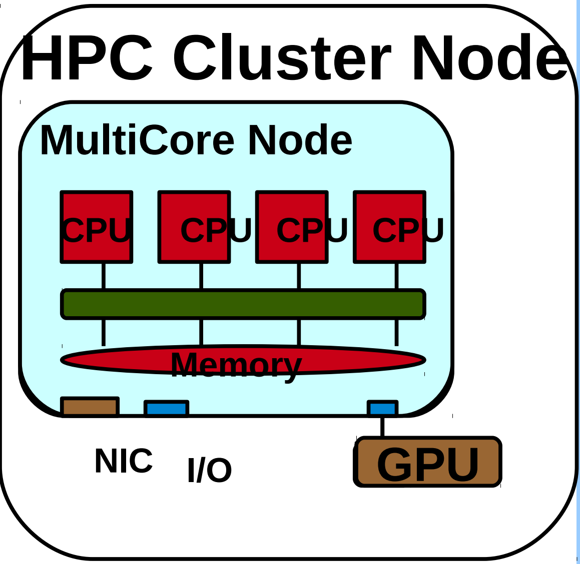
```
[root@cuzco51 ~]# tail -f /var/log/slurmd.cuzco51.log
[2011-09-20T03:10:02] [22.0] task/cgroup: manage devices jor job '22'
[2011-09-20T03:10:02] [22.0] device : /dev/nvidia0 major 195, minor 0
[2011-09-20T03:10:02] [22.0] device : /dev/nvidia1 major 195, minor 1
[2011-09-20T03:10:02] [22.0] device : /dev/sda2 major 8, minor 2
[2011-09-20T03:10:02] [22.0] device : /dev/sda1 major 8, minor 1
[2011-09-20T03:10:02] [22.0] device : /dev/sda major 8, minor 0
[2011-09-20T03:10:02] [22.0] device : /dev/null major 1, minor 3
[2011-09-20T03:10:02] [22.0] Default access allowed to device b 8:2 rwm
[2011-09-20T03:10:02] [22.0] parameter 'devices.allow' set to 'b 8:2 rwm' for '/cgroup/devices/uid_50071/job_22/step_0'
[2011-09-20T03:10:02] [22.0] Default access allowed to device b 8:1 rwm
[2011-09-20T03:10:02] [22.0] parameter 'devices.allow' set to 'b 8:1 rwm' for '/cgroup/devices/uid_50071/job_22/step_0'
[2011-09-20T03:10:02] [22.0] Default access allowed to device b 8:0 rwm
[2011-09-20T03:10:02] [22.0] parameter 'devices.allow' set to 'b 8:0 rwm' for '/cgroup/devices/uid_50071/job_22/step_0'
[2011-09-20T03:10:02] [22.0] Default access allowed to device c 1:3 rwm
[2011-09-20T03:10:02] [22.0] parameter 'devices.allow' set to 'c 1:3 rwm' for '/cgroup/devices/uid_50071/job_22/step_0'
[2011-09-20T03:10:02] [22.0] Allowing access to device c 195:0 rwm
[2011-09-20T03:10:02] [22.0] parameter 'devices.allow' set to 'c 195:0 rwm' for '/cgroup/devices/uid_50071/job_22/step_0'
[2011-09-20T03:10:02] [22.0] Not allowing access to device c 195:1 rwm
[2011-09-20T03:10:02] [22.0] parameter 'devices.deny' set to 'c 195:1 rwm' for '/cgroup/devices/uid_50071/job_22/step_0'
```

Cgroups **devices** subsystem : Usage Example

```
[root@cuzco51 ~]# cat /cgroup/devices/uid_50071/job_22/step_0/tasks
4875
4879
4882
[root@cuzco51 ~]# cat /cgroup/devices/uid_50071/job_22/step_0/devices.list
b 8:2 rwm
b 8:1 rwm
b 8:0 rwm
c 1:3 rwm
c 195:0 rwm
```

```
[gohn@cuzco0]$ cat output
/home/GPU/./gputest.sh: line 4: echo: write error: Invalid argument
/home/GPU/./gputest.sh: line 5: /dev/nvidia1: Operation not permitted
```

Monitoring Resource Usage: **cpuacct** and **memory subsystems**



Monitoring cpu usage with cpuacct subsystem and memory usage with memory subsystem

- Implemented as a jobacct_gather plugin for SLURM
- Collects information concerning CPU time and Memory RSS consumed for each task of the cgroup
- Values reported as a new job characteristics in the accounting database of SLURM
- Values can be used for billing purposes
- Monitor per job energy consumption (not through cgroups)

Monitoring Resources: **cpuacct -memory** subsystems

```
[gohn@cuzco0]$ srun -n32  ./malloc
[gohn@cuzco0]$ sacct -j 167
```

```
    JobID    JobName Partition  MaxRSS   AveRSS   MaxPages  AvePages
  MinCPU       AveCPU  Elapsed  State  Ntasks  AllocCPUs  ExitCode

------------ ---------- -------------- -------------- ---------- ---------- ---------- ----------

    167.0    malloc  shared  61311K   57221K   239.24G  99893120K
00:03.000    00:03.000  00:01:10  COMPLETED  32  32  0.0
```

Cgroup **Task** plugin : **devices** subsystem

Cgroup Devices Logic as implemented in task plugin

1) Initialization phase (information collection gres.conf file, major, minor, etc)

2) Allow all devices that should be allowed by default (allowed_devices.conf)

3) Lookup which gres devices are allocated for the job

- Write allowed gres devices to devices.allow file
- Write denied gres devices to devices.deny file

4) Execute **2** and **3** for job and steps tasks (different hierarchy level in cgroups)

Energy accounting and control

Summary of the energy accounting and control features

- Power and Energy consumption monitoring per node level.
- Energy consumption accounting per step/job on SLURM DataBase
- Power profiling per step/job on the end of job
- Frequency Selection Mechanisms for user control of job energy consumption

Summary of the energy accounting and control features

- Power and Energy consumption monitoring per node level.
- Energy consumption accounting per step/job on SLURM DataBase
- Power profiling per step/job on the end of job
- Frequency Selection Mechanisms for user control of job energy consumption

How this takes place:

- Dedicated Plugins for Support of in-band collection of energy/power data (IPMI / RAPL)
- Dedicated Plugins for Support of out-of-band collection of energy/power data (RRD databases)
- Power data job profiling with HDF5 file format
- SLURM Internal power-to-energy and energy-to-power calculations

Summary of the energy accounting and control features

- Power and Energy consumption monitoring per node level.
- Energy consumption accounting per step/job on SLURM DataBase
- Power profiling per step/job on the end of job
- Frequency Selection Mechanisms for user control of job energ

•**Overhead:** In-band Collection
•**Precision:** of the measurements and internal calculations
•**Scalability:** Out-of band Collection

How this

- Dedicated y/power data (IPM
- Dedicated Plugins for Support of out-of-band collection of energy/power data (RRD databases)
- Power data job profiling with HDF5 file format
- SLURM Internal power-to-energy and energy-to-power calculations

In-band collection of power/energy data with IPMI

- **IPMI** is a message-based, hardware-level interface specification (may operate in-band or out-of-band)
- Communication with the Baseboard Management Controller BMC which is a specialized microcontroller embedded on the motherboard of a computer
- SLURM support is based on the FreeIPMI API:

  http://www.gnu.org/software/freeipmi/
  - FreeIPMI includes a userspace driver that works on most motherboards without any required driver.
  - No thread interferes with application execution
- The data collected from IPMI are currently instantaneous measures in Watts
- SLURM individual polling frequency (>=1sec)
  - direct usage for power profiling
  - but internal SLURM calculations for energy reporting per job

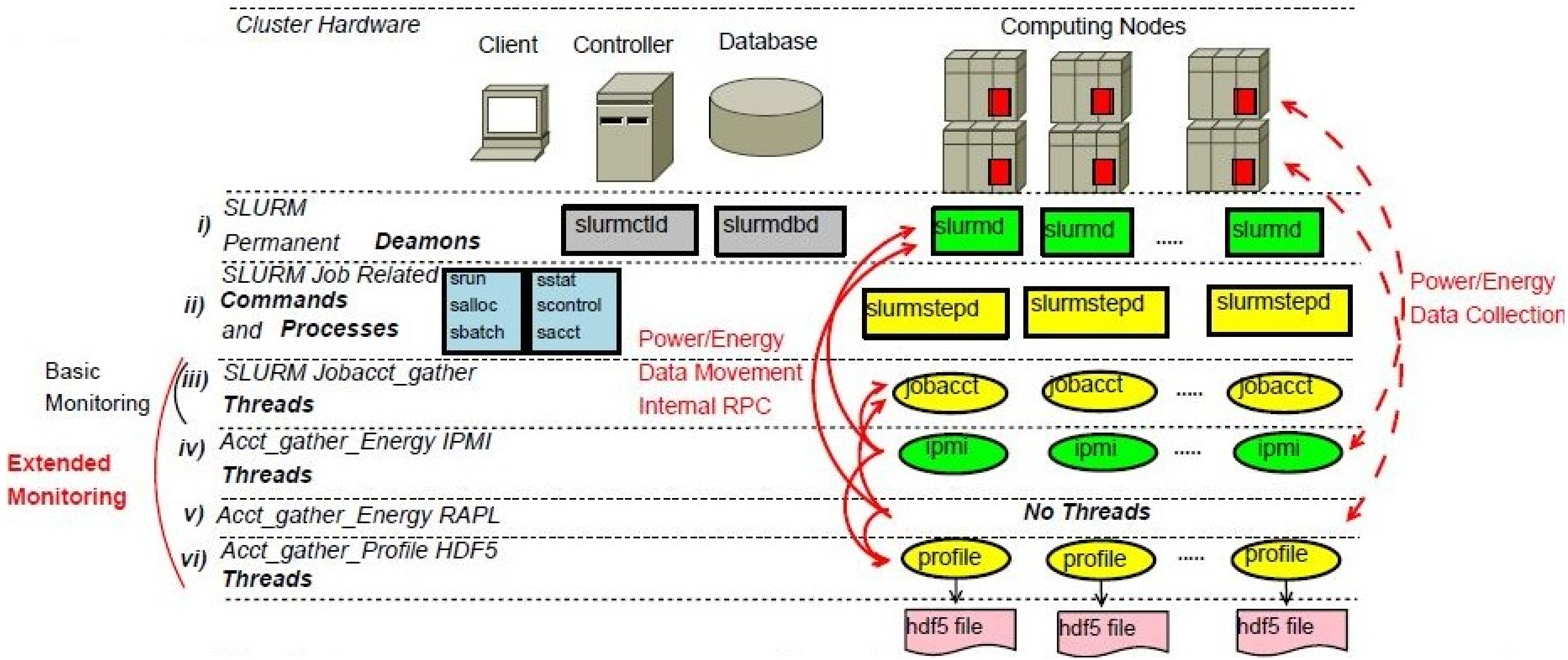
In-band collection of power/energy data with RAPL

- **RAPL** (Running Average Power Limit) are particular interfaces on Intel Sandy Bridge processors (and later models) implemented to provide a mechanism for keeping the processors in a particular user-specified power envelope.
- Interfaces can estimate current energy usage based on a software model driven by hardware performance counters, temperature and leakage models
  - Linux supports an ’MSR’ driver and access to the register can be made through /dev/cpu/*/msr with priviledged read permissions
- The data collected from RAPL is energy consumption in Joules (since the last boot of the machine)
- SLURM individual polling frequency (>=1sec)
  - direct usage for energy reporting per job
  - but internal SLURM calculations for power reporting

Power Profiling

- Job profiling to periodically capture the task's usage of various resources like CPU, Memory, Lustre, Infiniband and Power per node
- Resource Independent polling frequency configuration
- Based on **hdf5** file format http://www.hdfgroup.org opensource software library
  - versatile data model that can represent very complex data objects and a wide variety of metadata
  - portable file format with no limit on the number or size of data objects stored
- Profiling per node (one hdf5 file per job on each node)
- Aggregation on one hdf5 file per job (after job termination)
- Slurm built-in tools for extraction of hdf5 profiling data

Energy Accounting and Power Profiling Architecture



acct_gather_energy Plugin - Overview

- One of a new family of **acct_gather** plugins that collect resource usage data for accounting, profiling and monitoring.
- Loaded by **slurmd** on each compute node.
- Called by **jobacct_gather** plugin to collect energy consumption accounting data for jobs and steps.
- Called separately via RPC from the **slurmctld background** thread to collect energy consumption data for nodes.
- Calls **acct_gather_profile** plugin to provide energy data samples for profiling.

acct_gather_energy Plugin - Configuration

In **slurm.conf**

To configure plugin:

AcctGatherEnergyType=acct_gather_energy/rapl *or*
AcctGatherEnergyType=acct_gather_energy/ipmi

Frequency of node energy sampling controlled by:
AcctGatherNodeFreq=<seconds>
Default value is 0, which disables node energy sampling

Collection of energy accounting data for jobs/steps requires:
JobAcctGatherType=jobacct_gather/linux *or*
JobAcctGatherType=jobacct_gather/cgroup
Frequency of job accounting sampling controlled by:
JobAcctGatherFrequency=task=<seconds>
Default value is 30 seconds

In **acct_gather.conf** (new config file), for **acct_gather_energy/ipmi** only:

```
EnergyIPMIFrequency
EnergyIPMICalcAdjustment
EnergyIPMIPowerSensor
EnergyIPMIUsername
EnergyIPMIPassword
```

acct_gather_energy Plugin – Data Reporting

- For running jobs, energy accounting data is reported by **sstat**.
- If accounting database is configured, energy accounting data is included in accounting records and reported by **sacct** and **sreport**.
- If **acct_gather_profile** plugin is configured, energy profiling data is reported by the method specified by the profile plugin type.
- Energy consumption data for nodes is reported by **scontrol show node**.
- Cumulative/total energy consumption is reported in units of **joules**.
- Instantaneous rate of energy consumption (power) is reported in units of **watts**.

Out-of-band collection of power/energy data

- **External Sensors** Plugins to allow out-of-band monitoring of cluster sensors
- Possibility to Capture energy usage and temperature of various components (nodes, switches, rack-doors, etc)
- Framework generic but initial Support for RRD databases through rrdtool API (for the collection of energy/temperature data)
  - Plugin to be used with real wattmeters or out-of-band IPMI capturing
- Power data captured used for per node power monitoring (scontrol show node) and per job energy accounting (Slurm DB)
  - direct usage for energy reporting per job
  - but internal SLURM calculations for power reporting

External Sensors Plugin - Purpose

Plugin Name: ext_sensors

Purpose: To collect environmental-type data from external sensors or sources for the following uses:

- Job/step accounting – Total energy consumption by a completed job or step (no energy data while job/step is running).
- Hardware monitoring – Instantaneous and cumulative energy consumption for nodes; instantaneous temperature of nodes.
- Future work will add additional types of environmental data, such as energy and temperature data for network switches, cooling system, etc. Environmental data may be used for resource management.

ext_sensors Plugin - Overview

- Loaded by **slurmctld** on management node.
- Collects energy accounting data for jobs and steps independently of the **acct_gather** plugins.
  - Called by slurmctld request handler when step starts.
  - Called by slurmctld step manager when step completes.
- Since energy use by jobs/steps is measured only at completion (i.e., no sampling), does not support energy profiling or energy reporting for running jobs/steps (sstat).
- Called separately from the **slurmctld background** thread to sample energy consumption and temperature data for nodes.

ext_sensors Plugin – Data Reporting

- If accounting database is configured, energy data is included in accounting records and reported by **sacct** and **sreport**.

- Energy consumption data for nodes is reported by **scontrol show node**.

- Cumulative/total energy consumption reported in **joules**.
- Instantaneous energy consumption rate (power) for nodes reported in **watts**.
- Node temperature reported in **celsius**.

ext_sensors Plugin - Versions

- One version of **ExtSensorsType** plugin currently supported:
  - **ext_sensors/rrd**

    External sensors data is collected using RRD. RRDtool is GNU-licensed software that creates and manages a linear database used for sampling or logging. The database is populated with energy data using out-of-band IPMI collection.

- Plugin API is described in Slurm developer documentation:
  - http://slurm.schedmd.com/ext_sensorsplugins.html

ext_sensors Plugin - Configuration

- In **slurm.conf**

  To configure plugin:
  `ExtSensorsType=ext_sensors/rrd`

  Frequency of node energy sampling controlled by:
  `ExtSensorsFreq=<seconds>`
  Default value is 0, which disables node energy sampling

  Collection of energy accounting data for jobs/steps requires:
  `JobAcctGatherType=jobacct_gather/linux` *or* **`cgroup`**

- In **ext_sensors.conf** (new configuration file)

  `JobData`=**`energy`** Specify the data types to be collected by the plugin for jobs/steps.
  `NodeData`=**`[energy|temp]`**Specify the data types to be collected by the plugin for nodes.
  `SwitchData`=**`energy`** Specify the data types to be collected by the plugin for switches.
  `ColdDoorData`=**`temp`** Specify the data types to be collected by the plugin for cold doors.
  `MinWatt`=**`<number>`** Minimum recorded power consumption, in watts.
  `MaxWatt`=**`<number>`** Maximum recorded power consumption, in watts.
  `MinTemp`=**`<number>`** Minimum recorded temperature, in celsius.
  `MaxTemp`=**`<number>`** Maximum recorded temperature, in celsius.
  `EnergyRRA`=**`<name>`** Energy RRA name.
  `TempRRA`=**`<name>`** Temperature RRA name.
  `EnergyPathRRD`=**`<path>`** Pathname of energy RRD file.
  `TempPathRRD`=**`<path>`** Pathname of temperature RRD file.

Example 1 – Node energy monitoring using acct_gather_energy/rapl

```
[sulu] (slurm) mnp> scontrol show config
...
AcctGatherEnergyType    = acct_gather_energy/rapl
AcctGatherNodeFreq      = 30 sec
...

[sulu] (slurm) mnp> scontrol show node n15
NodeName=n15 Arch=x86_64 CoresPerSocket=8
   CPUAlloc=0 CPUErr=0 CPUTot=32 CPULoad=0.00 Features=(null)
   Gres=(null)
   NodeAddr=drak.usrnd.lan NodeHostName=drak.usrnd.lan
   OS=Linux RealMemory=1 AllocMem=0 Sockets=4 Boards=1
   State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1
   BootTime=2013-08-28T09:35:47 SlurmdStartTime=2013-09-05T14:31:21
   CurrentWatts=121 LowestJoules=69447 ConsumedJoules=8726863
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

Example 2 – Energy accounting using acct_gather_energy/rapl

```
[sulu] (slurm) mnp> scontrol show config
...
JobAcctGatherType       = jobacct_gather/linux
JobAcctGatherFrequency  = task=10
AcctGatherEnergyType    = acct_gather_energy/rapl
AccountingStorageType   = accounting_storage/slurmdb
...

[sulu] (slurm) mnp> srun test/memcputest 100 10000 &
[1] 20712
[sulu] (slurm) mnp> 100 Mb buffer allocated

[sulu] (slurm) mnp> squeue
             JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
               120 drak-only memcpute    slurm   R       0:03      1 n15

[sulu] (slurm) mnp> sstat -j 120 -o ConsumedEnergy
ConsumedEnergy
--------------
          2149

[sulu] (slurm) mnp> sstat -j 120 -o ConsumedEnergy
ConsumedEnergy
--------------
          2452

[sulu] (slurm) mnp> sstat -j 120 -o ConsumedEnergy
ConsumedEnergy
--------------
          2720
[sulu] (slurm) mnp> Finished: j = 10001, c = 2990739969

[1]+  Done                    srun test/memcputest 100 10000

[sulu] (slurm) mnp> sacct -j 120 -o ConsumedEnergy
ConsumedEnergy
--------------
          3422
```

Example 3 – Energy accounting using acct_gather_energy/ipmi

```
[root@cuzco108 bin]# scontrol show config
...
JobAcctGatherType       = jobacct_gather/linux
JobAcctGatherFrequency  = task=10
AcctGatherEnergyType    = acct_gather_energy/ipmi
AccountingStorageType   = accounting_storage/slurmdb
...

[root@cuzco108 bin]# cat /usr/local/slurm2.6/etc/acct_gather.conf

EnergyIPMIFrequency=10
#EnergyIPMICalcAdjustment=yes
EnergyIPMIPowerSensor=1280


[root@cuzco108 bin]# srun -w cuzco113 memcputest 100 10000 &
[1] 26138
[root@cuzco108 bin]# 100 Mb buffer allocated

[root@cuzco108 bin]# squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               101 exclusive memcpute     root  R       0:04      1 cuzco113
[root@cuzco108 bin]# sstat -j 101 -o ConsumedEnergy
ConsumedEnergy
--------------
           570

[root@cuzco108 bin]# sstat -j 101 -o ConsumedEnergy
ConsumedEnergy
--------------
         1.74K
```

Example 3 – continued

```
[root@cuzco108 bin]# Finished: j = 10001, c = 2990739969

[1]+  Done                    srun -w cuzco113 memcputest 100 10000
[root@cuzco108 bin]# sacct -j 101 -o ConsumedEnergy
ConsumedEnergy
--------------
         1.74K
```

Example 4 – Node energy and temperature monitoring using ext_sensors/rrd

```
[root@cuzco0 ~]# scontrol show config
...
ExtSensorsType          = ext_sensors/rrd
ExtSensorsFreq          = 10 sec
...

[root@cuzco108 slurm]# cat /usr/local/slurm2.6/etc/ext_sensors.conf
#
# External Sensors plugin configuration file
#

JobData=energy
NodeData=energy,temp

EnergyRRA=1
EnergyPathRRD=/BCM/data/metric/%n/Power_Consumption.rrd

TempRRA=1
TempPathRRD=/BCM/data/metric/%n/Temperature.rrd

MinWatt=4
MaxWatt=200


[root@cuzco0 ~]# scontrol show node cuzco109

NodeName=cuzco109 Arch=x86_64 CoresPerSocket=4
   CPUAlloc=0 CPUErr=0 CPUTot=8 CPULoad=0.00 Features=(null)
   Gres=(null)
   NodeAddr=cuzco109 NodeHostName=cuzco109
   OS=Linux RealMemory=24023 AllocMem=0 Sockets=2 Boards=1
   State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1
   BootTime=2013-09-03T17:39:00 SlurmdStartTime=2013-09-10T22:58:10
   CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
   ExtSensorsJoules=4200 ExtSensorsWatts=105 ExtSensorsTemp=66
```

Example 5 – Energy accounting comparison using ext_sensors/rrd and acct_gather_energy/ipmi

The accuracy/consistency of energy measurements may be inaccurate if the run time of the job is short and allows for only a few samples. This effect should be reduced for longer jobs.

The following example shows that the **ext_sensors/rrd** and **acct_gather_energy/ipmi** plugins produce very similar energy consumption results for a MPI benchmark job using 4 nodes and 32 CPUs, with a run time of ~9 minutes.

Example 5 – continued

acct_gather_energy/ipmi

```
[root@cuzco108 bin]# scontrol show config | grep acct_gather_energy
AcctGatherEnergyType    = acct_gather_energy/ipmi

[root@cuzco108 bin]# srun -n32 --resv-ports ./cg.D.32 &

[root@cuzco108 bin]# squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               122 exclusive  cg.D.32     root  R       0:02      4 cuzco[109,111-113]

[root@cuzco108 bin]# sacct -o "JobID%5,JobName,AllocCPUS,NNodes%3,NodeList%22,State,Start,End,Elapsed,ConsumedEnergy%9"
JobID    JobName  AllocCPUS NNo               NodeList      State               Start                 End    Elapsed ConsumedE
----- ---------- ---------- --- ---------------------- ---------- ------------------- ------------------- ---------- ---------
  127    cg.D.32         32   4     cuzco[109,111-113]  COMPLETED 2013-09-12T23:12:51 2013-09-12T23:22:03   00:09:12   490.60K
```

ext_sensors/rrd

```
[root@cuzco108 bin]# scontrol show config | grep ext_sensors
ExtSensorsType          = ext_sensors/rrd

[root@cuzco108 bin]# srun -n32 --resv-ports ./cg.D.32 &

[root@cuzco108 bin]# squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               128 exclusive  cg.D.32     root  R       0:02      4 cuzco[109,111-113]

[root@cuzco108 bin]# sacct -o "JobID%5,JobName,AllocCPUS,NNodes%3,NodeList%22,State,Start,End,Elapsed,ConsumedEnergy%9"
JobID    JobName  AllocCPUS NNo               NodeList      State               Start                 End    Elapsed ConsumedE
----- ---------- ---------- --- ---------------------- ---------- ------------------- ------------------- ---------- ---------
  128    cg.D.32         32   4     cuzco[109,111-113]  COMPLETED 2013-09-12T23:27:17 2013-09-12T23:36:33   00:09:16   498.67K
```

Profiling Configuration

- Configuration parameters

The profile plugin is enabled in the **slurm.conf** file, but is internally configured in the **acct_gather.conf** file.

slurm.conf parameters

- **AcctGatherProfileType**=acct_gather_profile/hdf5 enables the HDF5 Profile Plugin
- **JobAcctGatherFrequency**={energy=freq {,lustre=freq {,network=freq , {task=freq}}}} sets default sample frequencies for data types.
- One or more of the following plugins must also be configured.
  - AcctGatherEnergyType=acct_gather_energy/ipmi
  - AcctGatherEnergyType=acct_gather_energy/rapl
  - AcctGatherFilesystemType=acct_gather_filesystem/lustre
  - AcctGatherInfinibandType=acct_gather_infiniband/ofed
  - JobAcctGatherType=job_acct_gather/linux

Sample conf files

slurm.conf

```
DebugFlags=Profile
AcctGatherProfileType=acct_gather_profile/hdf5

JobAcctGatherType=jobacct_gather/linux
JobAcctGatherFrequency=energy=5,lustre=60,network=60,task=60
AcctGatherEnergyType=acct_gather_energy/ipmi
AcctGatherFilesystemType=acct_gather_filesystem/lustre
AcctGatherInfinibandType=acct_gather_infiniband/ofed
```

acct_gather.conf

```
# Parameters for AcctGatherEnergy/ipmi plugin
EnergyIPMIFrequency=10
EnergyIPMICalcAdjustment=yes
#
# Parameters for AcctGatherProfileType/hdf5 plugin
ProfileHDF5Dir=/app/Slurm/profile_data
# Parameters for AcctGatherInfiniband/ofed plugin
InfinibandOFEDFrequency=4
InfinibandOFEDPort=1
```

Energy Data

- AcctGatherEnergyType=acct_gather_energy/ipmi is required in slurm.conf to collect energy data.
- JobAcctGatherFrequeny=Energy=<freq> should be set in either slurm.conf or via –acctg-freq command line option.

The IPMI energy plugin also needs the EnergyIPMIFrequency value set in the acct_gather.conf file. This sets the rate at which the plugin samples the external sensors. This value should be the same as the energy=sec in either JobAcctGatherFrequency or --acctg-freq.

Note that the IPMI and profile sampling is not synchronous. The profile sample simply takes the last available IPMI sample value. If the profile energy sample is more frequent than the IPMI sample rate, the IPMI value will be repeated. If the profile energy sample is greater than the IPMI rate, IPMI values will be lost.

Also note that smallest effective IPMI (EnergyIPMIFrequency) sample rate for 2013 era Intel processors is 3 seconds.

Note that Energy data is collected for the entire node so it is only meaningful for exclusive allocations.

- Each data sample in the Energy Time Series contains the following data items.

Date Time Time of day at which the data sample was taken.
This can be used to correlate activity with other sources such as logs.
Time Elapsed time since the beginning of the step.
Power Power consumption during the interval.
CPU Frequency CPU Frequency at time of sample in kilohertz.

Lustre Data

- AcctGatherFilesystemType=acct_gather_filesystem/lustre is required in Slurm.conf to collect lustre data.
- JobAcctGatherFrequeny=Lustre=<freq> should be set in either Slurm.conf or via –acctg-freq command line option.

- Each data sample in the Lustre Time Series contains the following data items.

Date Time	Time of day at which the data sample was taken. This can be used to correlate activity with other sources such as logs.
Time	Elapsed time since the beginning of the step.
Reads	Number of read operations.
MegabytesRead	Number of megabytes read.
Writes	Number of write operations.
MegabytesWrite	Number of megabytes written.

Network (Infiniband) Data

- AcctGathertInfinibandType=acct_gather_infiniband/ofed is required in Slurm.conf to collect Network data.
- JobAcctGatherFrequeny=Network=<freq> should be set in either Slurm.conf or via –acctg-freq command line option.

- Each data sample in the Network Time Series contains the following data items.

Date Time	Time of day at which the data sample was taken. This can be used to correlate activity with other sources such as logs.
Time	Elapsed time since the beginning of the step.
PacketsIn	Number of packets coming in.
MegabytesIn	Number of megabytes coming in through the interface.
PacketsOut	Number of packets going out.
MegabytesOut	Number of megabytes going out through the interface.

Task Data

- JobAcctGatherType=jobacct_gather/linux is required in Slurm.conf to collect task data
- JobAcctGatherFrequeny=Task=<freq> should be set in either Slurm.conf or via –acctg-freq command line option.

  The frequency should be set to at least 30 seconds for CPU utilization to be meaningful (since the resolution of cpu time in linux is 1 second)

- Each data sample in the Task Time Series contains the following data items.

Date Time Time of day at which the data sample was taken.
This can be used to correlate activity with other sources such as logs.
Time Elapsed time since the beginning of the step.
CPUFrequency CPU Frequency at time of sample.
CPUTime Seconds of CPU time used during the sample.
CPUUtilization CPU Utilization during the interval.
RSS Value of RSS at time of sample.
VMSize Value of VM Size at time of sample.
Pages Pages used in sample.
ReadMegabytes Number of megabytes read from local disk.
WriteMegabytes Number of megabytes written to local disk.

Emulation and Performance Evaluation

Activating emulation technique within SLURM

- Multiple slurmd technique can be used to experiment with larger scales:

  - the idea is that multiple slurmd deamons use the same IP address but different ports

  - all controller side plugins and mechanisms will function

  - ideal for scheduling, internal communications and scalability experiments

1. You need to run ./configure with –enable-multiple-slurmd parameter (make, make install, etc)
2. Perform the necessary changes in the slurm.conf file similarly the following example:

Activating emulation technique within SLURM

```
SlurmdPidFile=/usr/local/slurm-test/var/run/slurmd-%n.pid
SlurmdSpoolDir=/tmp/slurm-%n
SlurmdLogFile=/tmp/slurmd-%n.log
FastSchedule=2
PartitionName=exclusive Nodes=virtual[0-40] Default=YES MaxTime=INFINITE State=UP Priority=10
Shared=EXCLUSIVE
NodeName=DEFAULT Sockets=2 CoresPerSocket=8 ThreadsPerCore=1 RealMemory=21384 State=IDLE
NodeName=virtual0 NodeHostName=nazgul NodeAddr=127.0.0.1 Port=17000.
NodeName=virtual1 NodeHostName=nazgul NodeAddr=127.0.0.1 Port=17001
NodeName=virtual2 NodeHostName=nazgul NodeAddr=127.0.0.1 Port=17002
......
```

3. You can start the slurmd deamons with:
   - Either through a script such as:

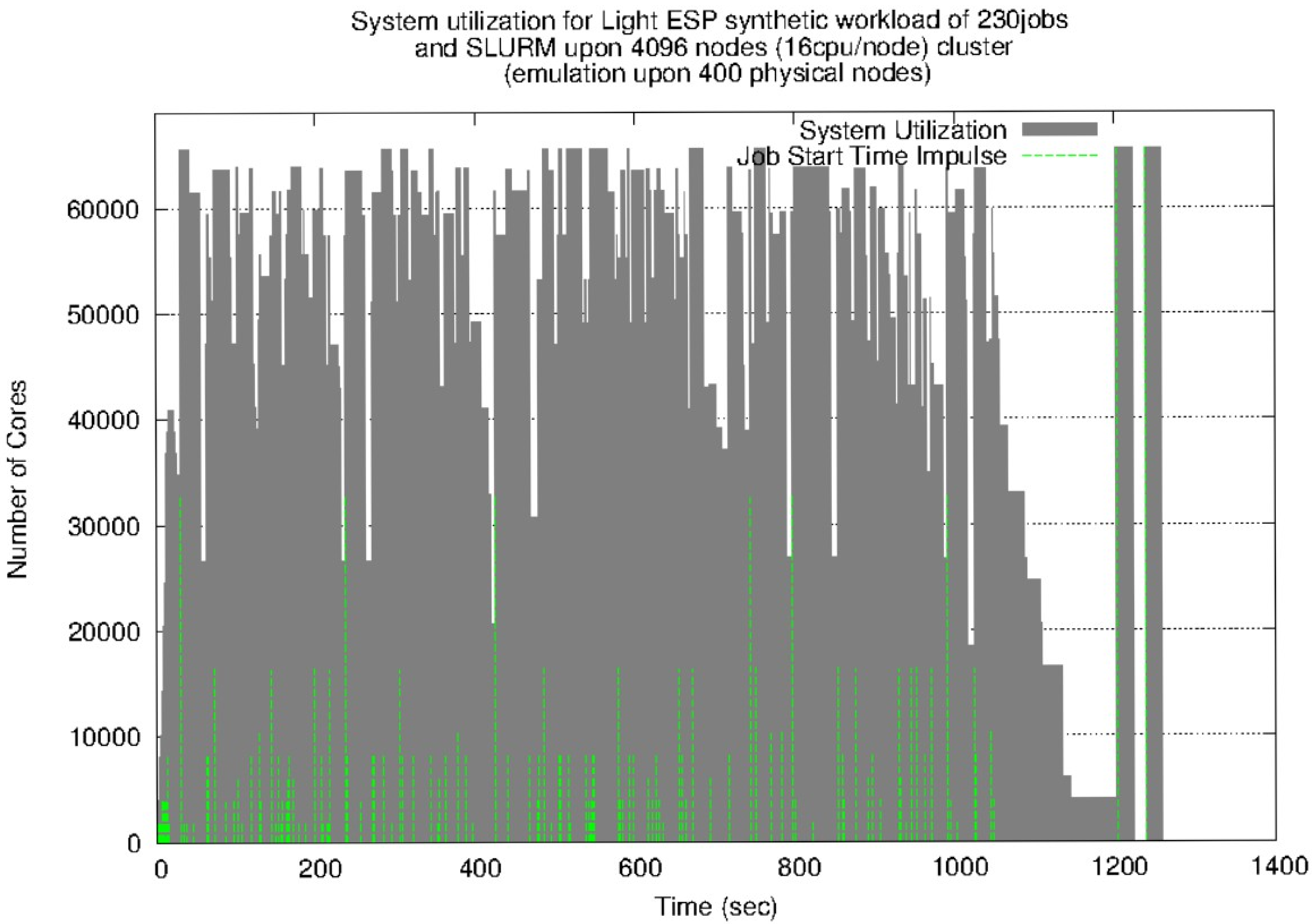
     for i in {0..40}; do slurmd -N virtual$i; done

   - Or by exporting: MULTIPLE_SLURMD="$(grep NodeHostName=$(hostname) /etc/slurm.conf | cut -d ' ' -f 1 | cut -d'=' -f 2)"

     on /etc/sysconfig/slurm and starting with /etc/init.d/slurm

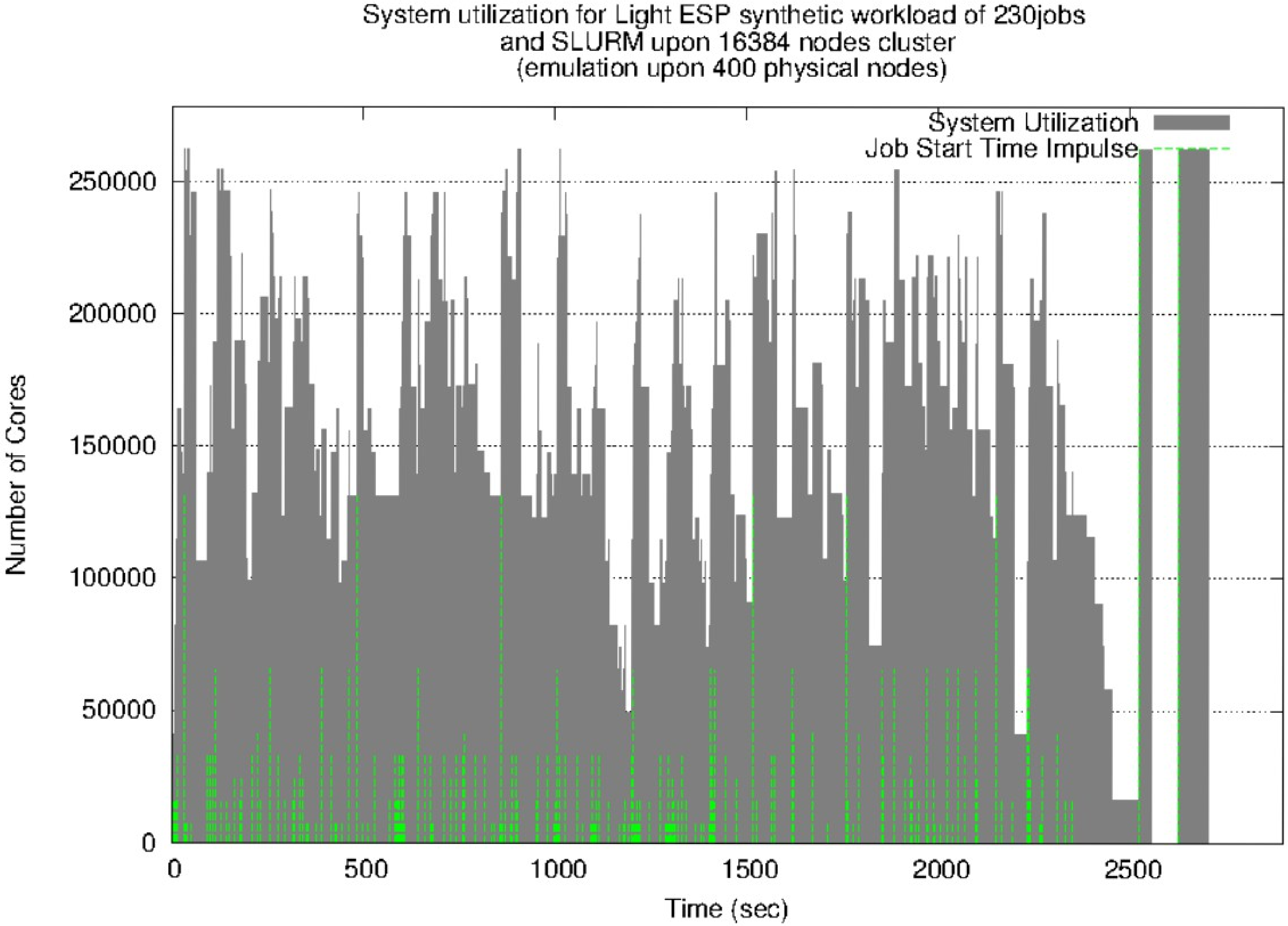
Examples of performance evaluation with emulation

4096 emulated nodes upon 400 physical nodes



Examples of performance evaluation with emulation

16384 emulated nodes upon 400 physical nodes



- Introduction
- SLURM scalable and flexible RJMS
- Part 1: Basics
  - Overview, Architecture, Configuration files, Partitions, Plugins, Reservations, Reconfiguration
- Part 2: Advanced Configuration
  - Accounting, Scheduling, Allocation, Network Topology Placement, Generic Resources, Licenses Management, Energy Reduction Techniques
- Part 3: Experts Configuration
  - CPU Management, Isolation with cgroups, Power Management, Simulation and evaluation
- **Upcoming Features**

Bull
an atos company

New Slurm features under development

- Heterogeneous Environment
  - Asymmetric Resources and MPMD model
  - GPU Affinity
- Scalability
  - Support of PMI-x project
  - Messages Aggregation
  - HDF5 Profiling Framework
- Power Management and Energy Efficiency
  - Extension of Energy Accounting and Power Profiling Framework
  - Power-Capping logic in Job Scheduling
  - Energetic Fairsharing

References

- Y. Georgiou: Contributions for Resource and Job Management in High Performance Computing. Ph.D. thesis, 2010
- Y. Georgiou, M. Hautreux: Evaluating scalability and efficiency of the resource and job management system on large hpc clusters. In: JSSPP 2012
- Y. Georgiou: Resource Management with Linux Control Groups in HPC Clusters Yiannis Georgiou, 6th Linux Collaboration Summit, 2012
- Y. Georgiou, T. Cadeau, D. Glesser, D. Auble, M. Jette, and M. Hautreux, "Energy accounting and control with slurm resource and job management system," in ICDCN, 2014.
- Y. Georgiou, D. Glesser, K. Rzadca, D. Trystram : A Scheduler-Level Incentive Mechanism for Energy Efficiency in HPC, to appear in CCGrid 2015
- Y. Georgiou, D. Glesser, D. Trystram: Adaptive Resource and Job Management for limited power consumption, pending for HPPAC-IPDPS 2015

Bull

an atos company