Slurm Advanced Scheduling

(and how to improve various metrics)

Yiannis Georgiou - CTO Ryax Technologies

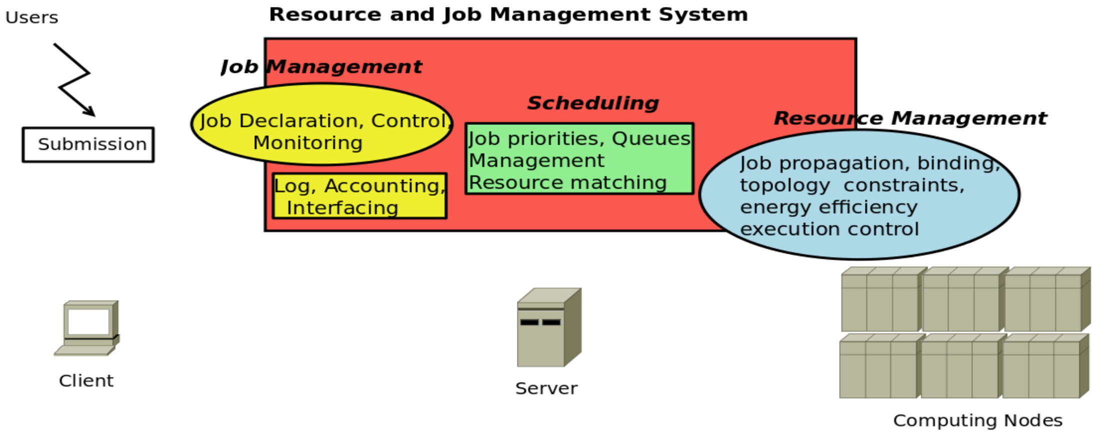
ryax technologies

Resource and Job Management System Layers

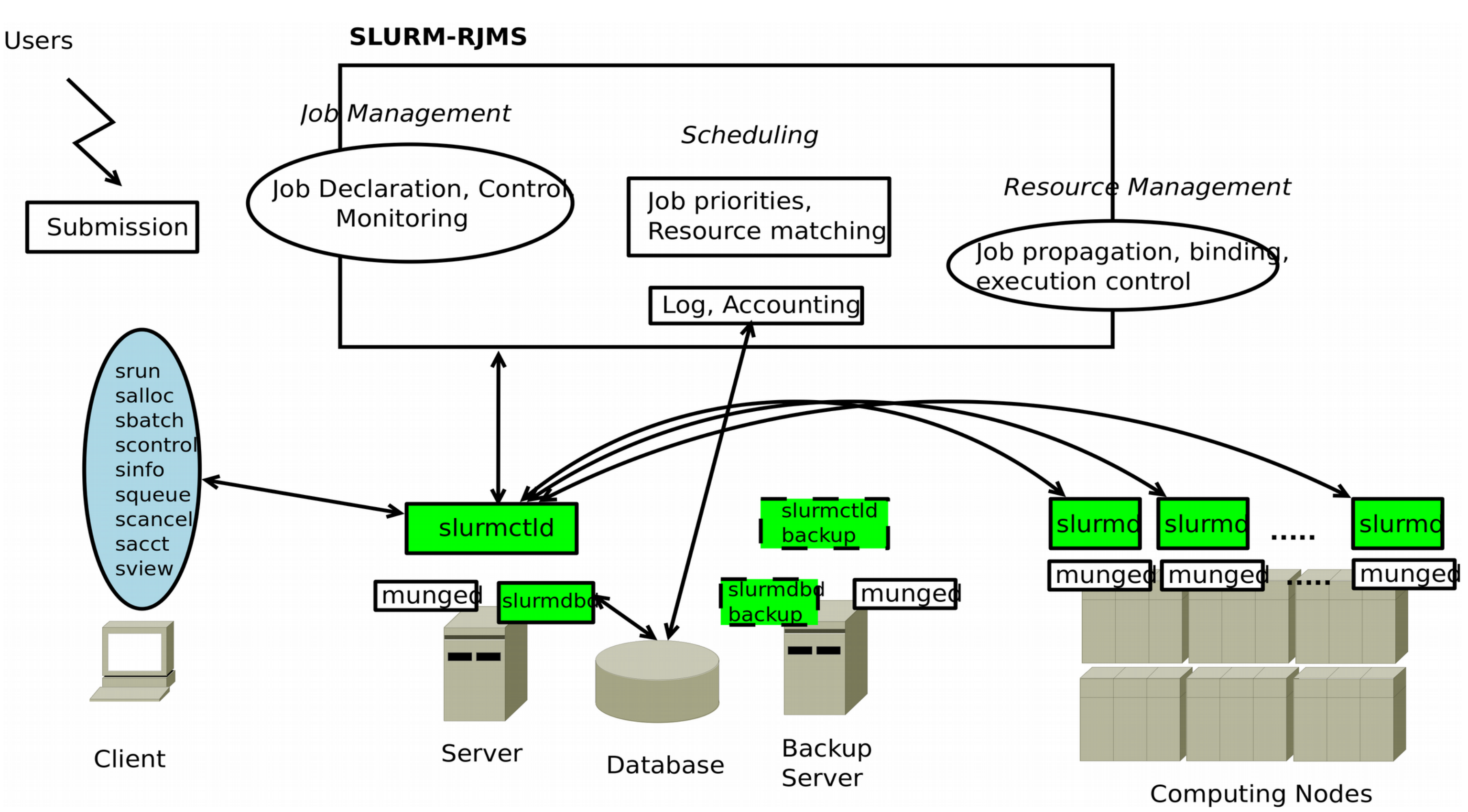
The goal of a Resource and Job Management System (RJMS) is to satisfy users' demands for computation and assign resources to user jobs with an efficient manner.

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

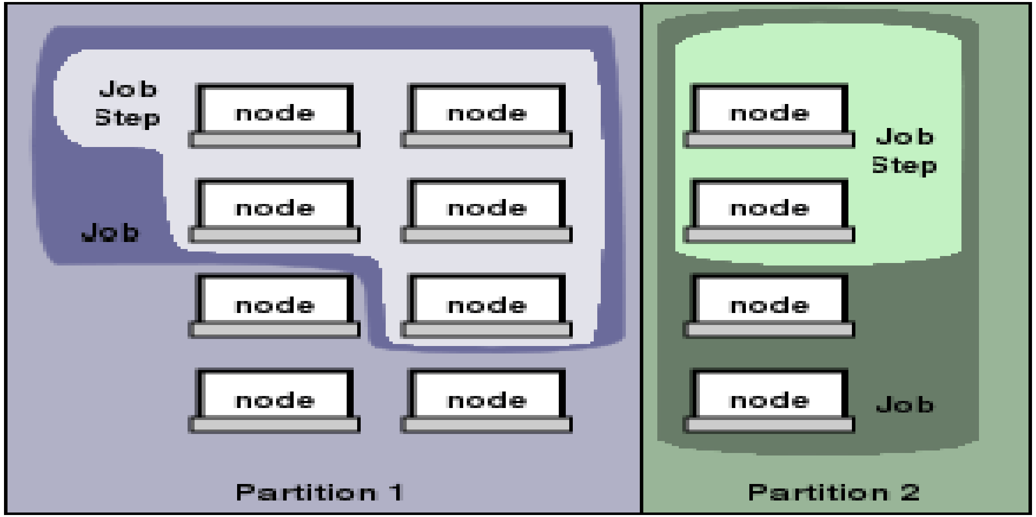


SLURM Architecture



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

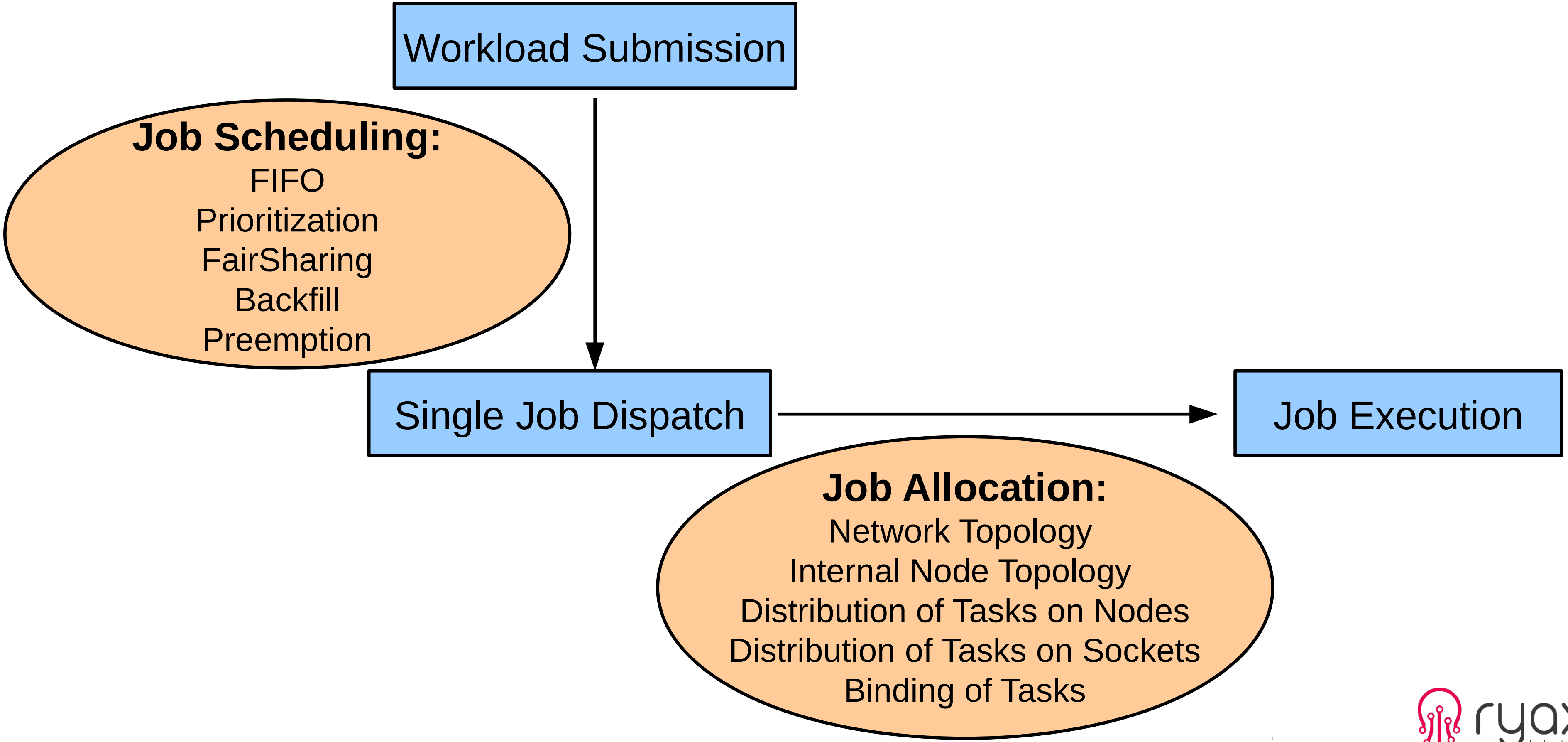
Outline

1) Typical Job Scheduling and Allocation Mechanisms
2) Energy Reductions and Powercapping Techniques
3) Topology Aware Job Mapping
4) Techniques for scheduling experimentation

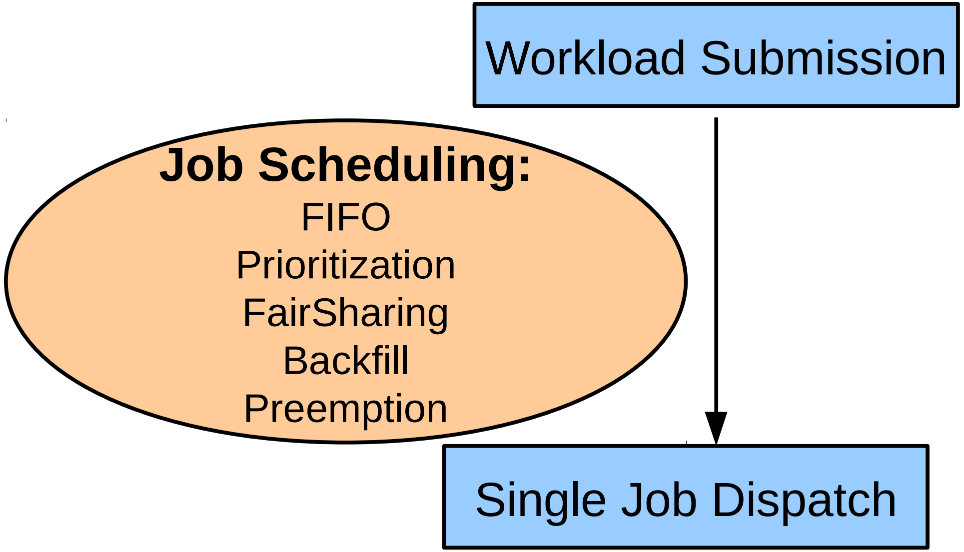
Outline

1) **Typical Job Scheduling and Allocation Mechanisms**
2) Energy Reductions and Powercapping Techniques
3) Topology Aware Job Mapping
4) Techniques for scheduling experimentation

SLURM scheduling / allocation procedures



SLURM scheduling



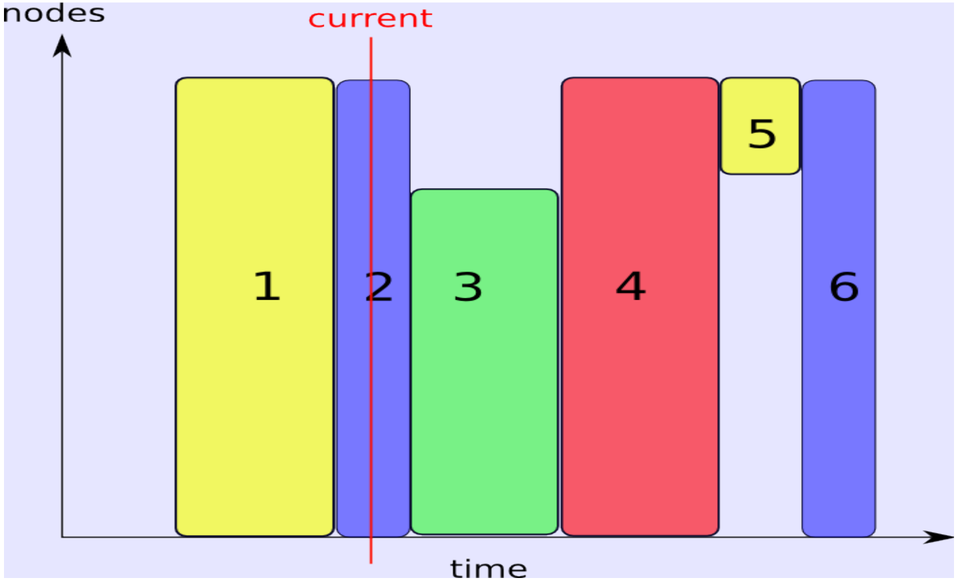
SLURM Scheduling

- SLURM supports various **scheduling policies and optimization techniques** such as :
  - Backfill
  - Preemption
  - Multi-factor priority
  - Fairsharing

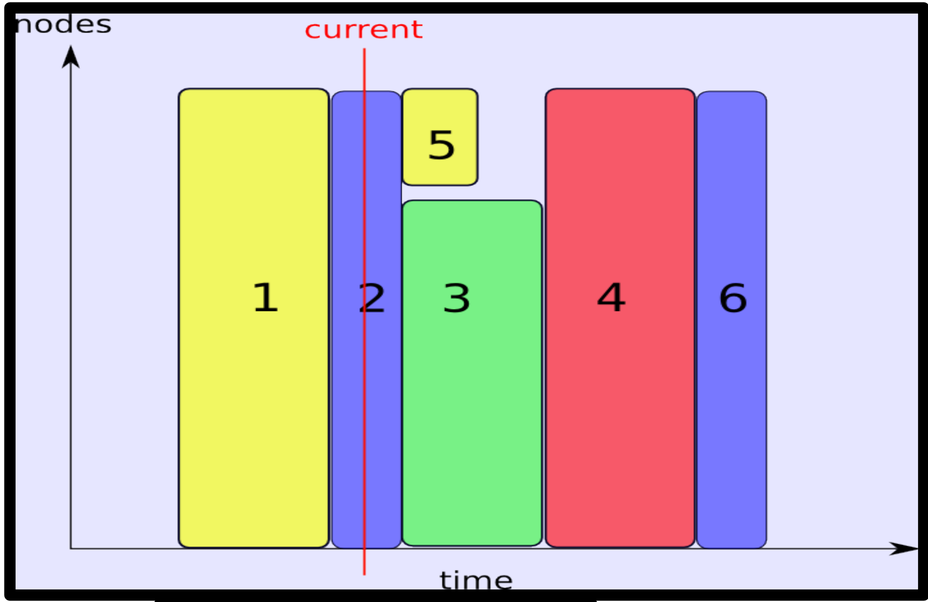
- Note: Techniques can be **supported simultaneously**

Scheduling – Backfill

Holes can be filled if previous jobs order is not changed



FIFO Scheduler



Backfill Scheduler

Scheduling Policies

Sched/backfill

schedules 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.

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

Backfill Configuration

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.

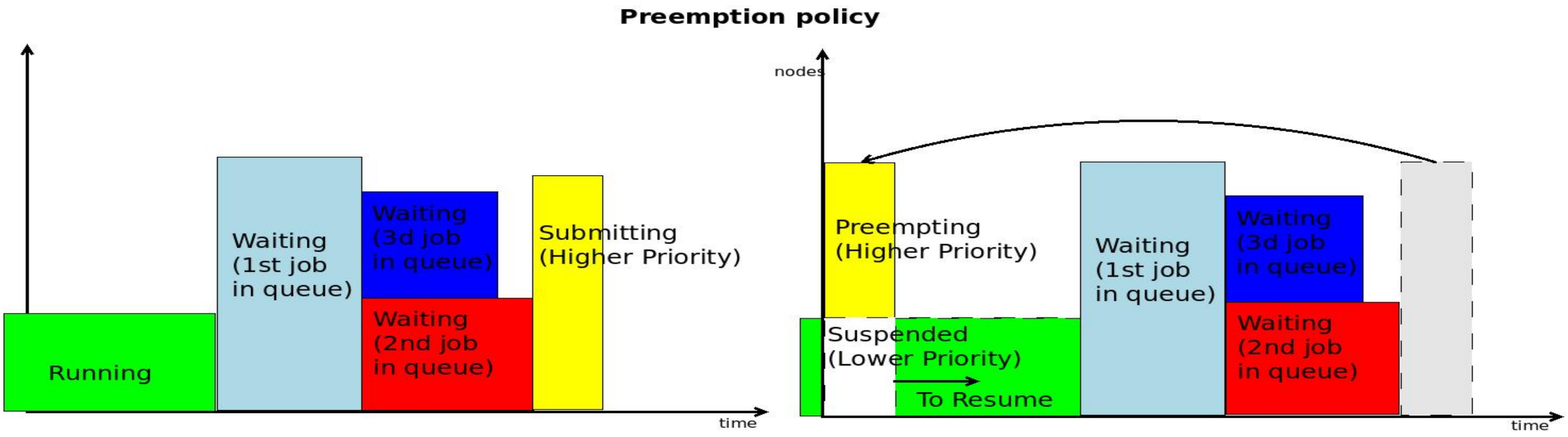
•Configuration of scheduler backfill in slurm.conf

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

Metrics Improvements: System Utilization and Jobs Waiting times

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



Preemption Policies

Preempt Modes

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=SUSPEND
PreemptType=preempt/qos
```

Metrics Improvements:
Respect SLAs

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

Partition 1: 32 cores and high_memory

Partition 2: 32 cores and low_memory

Partition 3: 64 cores

QOS 1:
-High priority
-Higher limits

QOS 2:
-Low Priority
-Lower limits

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
```

Multifactor Priority in SLURM

- 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)+
  SUM(TRES_weight_cpu * TRES_factor_cpu,

  TRES_weight_<type> * TRES_factor_<type>,...)

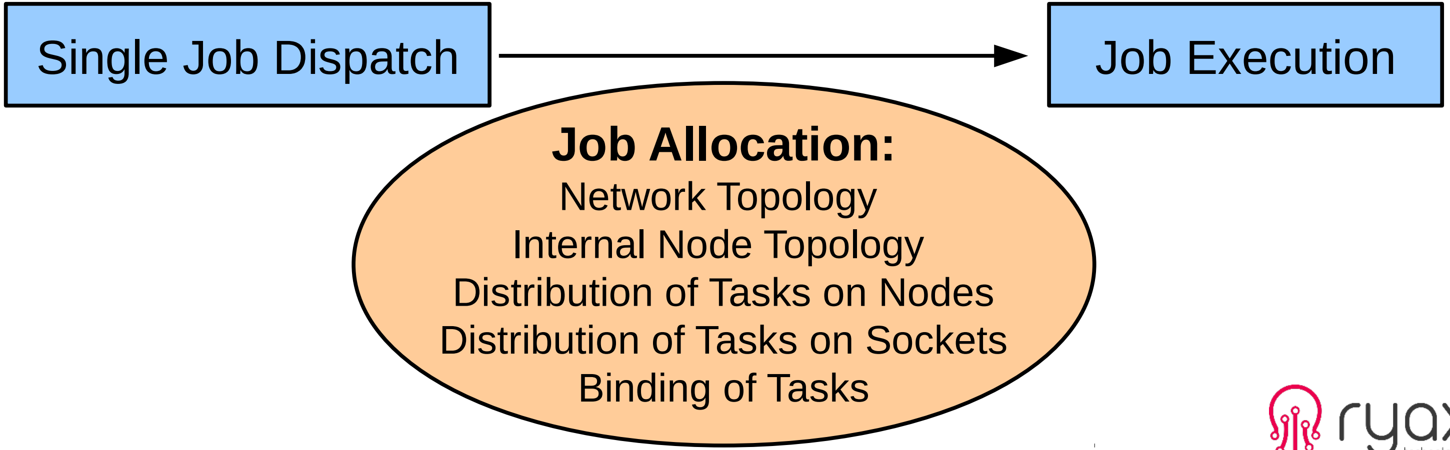
Fairsharing in SLURM

- 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** by default or usage of TRESBillingWeights which tracks utilization of selected Trackable Resources (CPU, Memory, GPUs, etc) of each user

Metrics Improvements: Minimize Job Starvation, Balanced usage

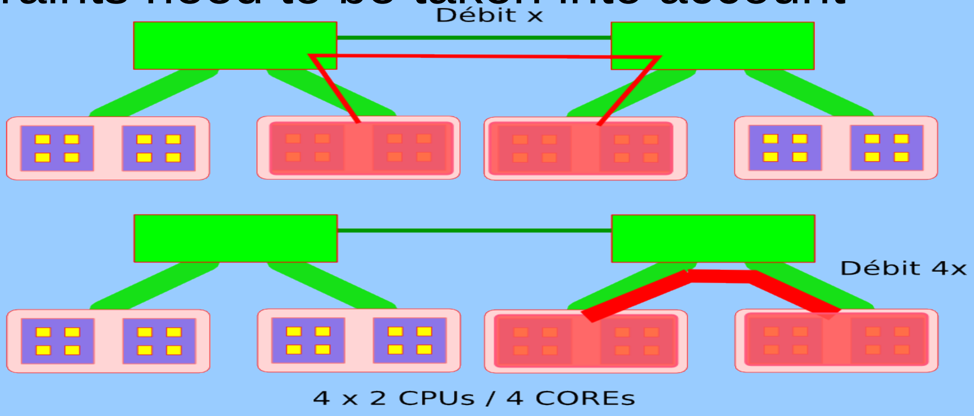
ryax technologies

SLURM allocation



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
```

ryax technologies

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

Metrics Improvements: Application Performance, Job Execution Time

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.

Internal node topology/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

Metrics Improvements: Application Performance, Job Execution Time

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

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

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.

Options for Step 4: Optional Distribution & Binding

<u>Configuration options in **slurm.conf:**</u>

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)

<u>Command Line options on **srun/salloc/sbatch**:</u>

--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.

Distribution of Resources

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

ryax technologies

Generic Resources (Allocation of GPUs, ...)

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, ...)

For configuration the 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.

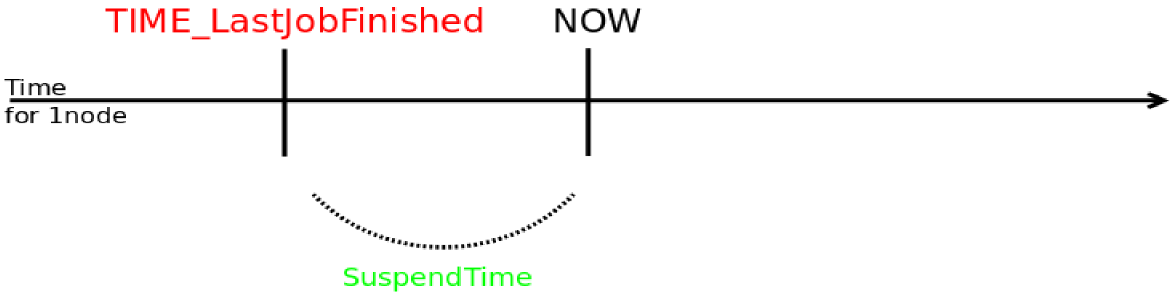
--gres=<list>Specifies a comma delimited list of generic consumableresources. The format of each entry on the list is**"name[:count]".**

Outline

1) Typical Job Scheduling and Allocation Mechanisms
2) **Energy Reductions and Powercapping Techniques**
3) Topology Aware Job Mapping
4) Techniques for scheduling experimentation

Energy Reduction Techniques

- Framework for energy reductions through unutilized nodes
  - Administrator configurable actions (hibernate, DVFS, power off, etc)
  - Automatic 'wake up' when jobs arrive



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.

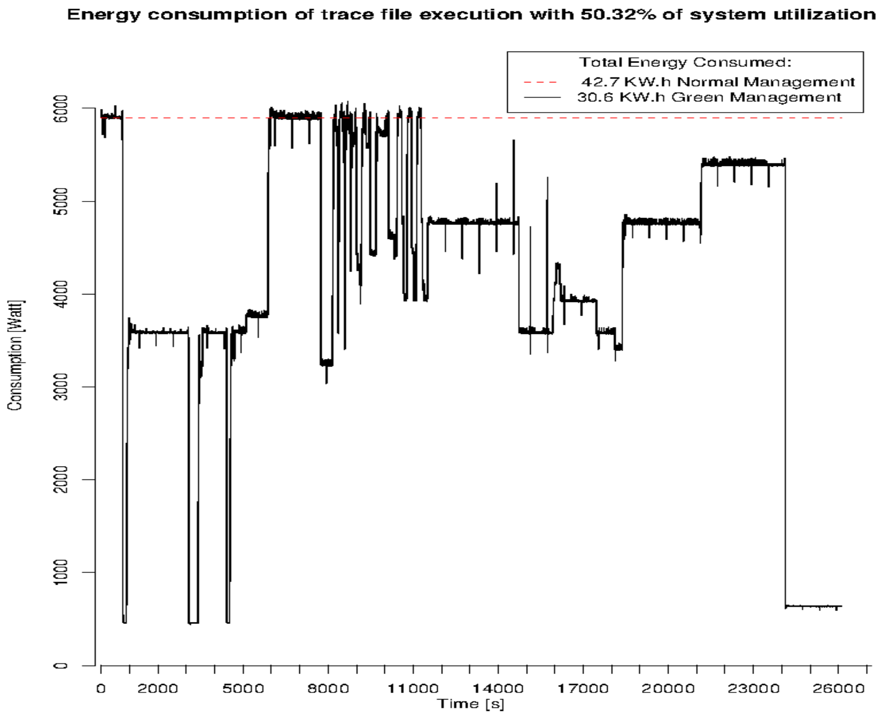
ResumeRate: # nodes removed per minute. A value of zero results in no limits being imposed. The default value is 300. Use this to prevent rapid increases in power consumption.

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.

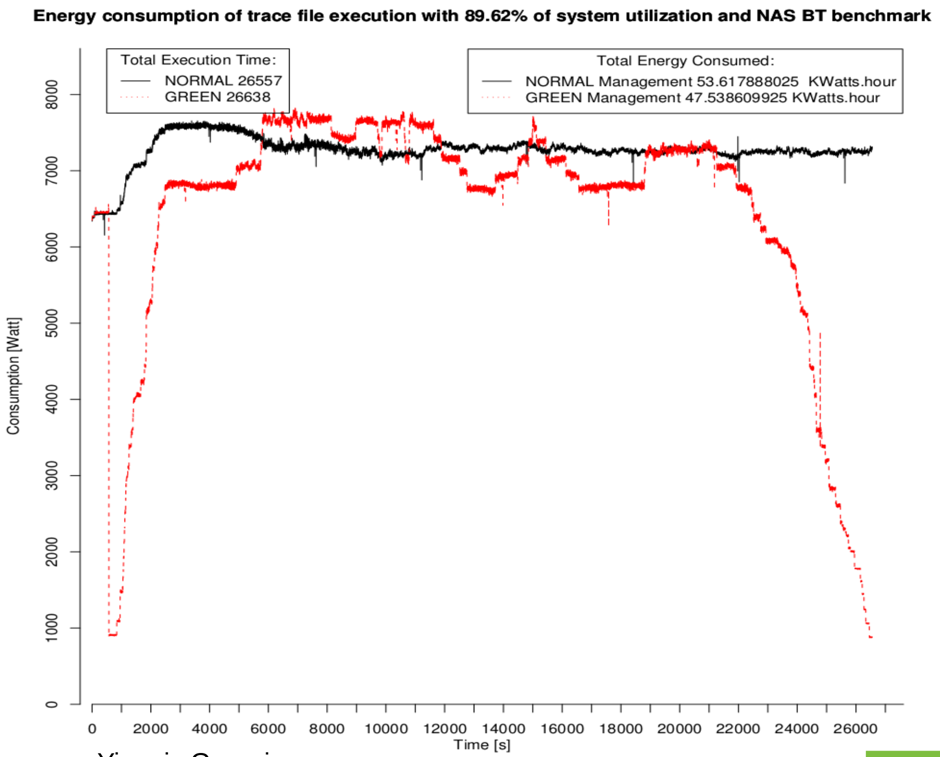
SuspendTimeout, ResumeTimeout, SuspendExcNodes,SuspendExcParts, BatchStartTimeout

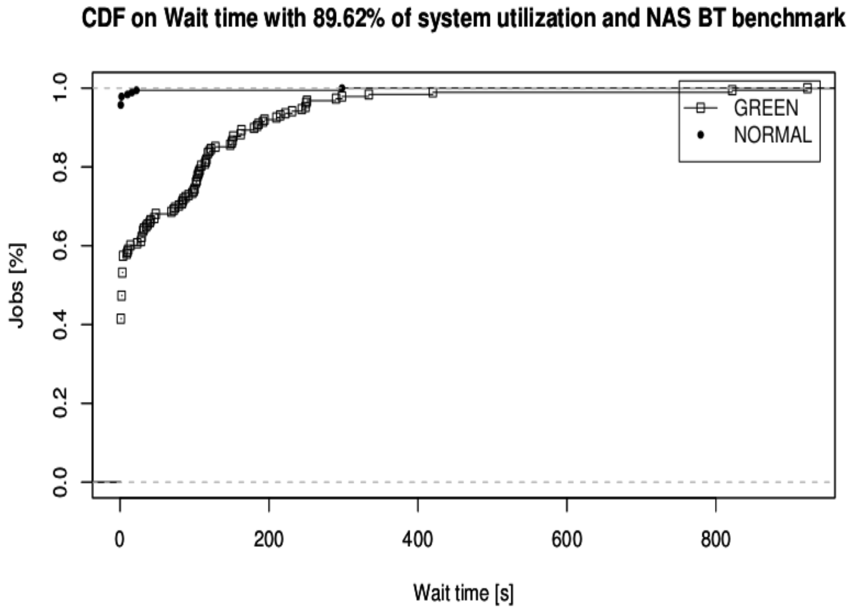
Energy Reduction Techniques



Georges Da Costa, Marcos Dias de Assuncao, Jean-Patrick Gelas, Yiannis Georgiou, Laurent Lefevre, Anne-Cecile Orgerie, Jean-Marc Pierson, Olivier Richard and Amal Sayah
Multi-facet approach to reduce energy consumption in clouds and grids: The green-net framework.
(In proceedings of e-Energy 2010)

Energy Reduction Techniques

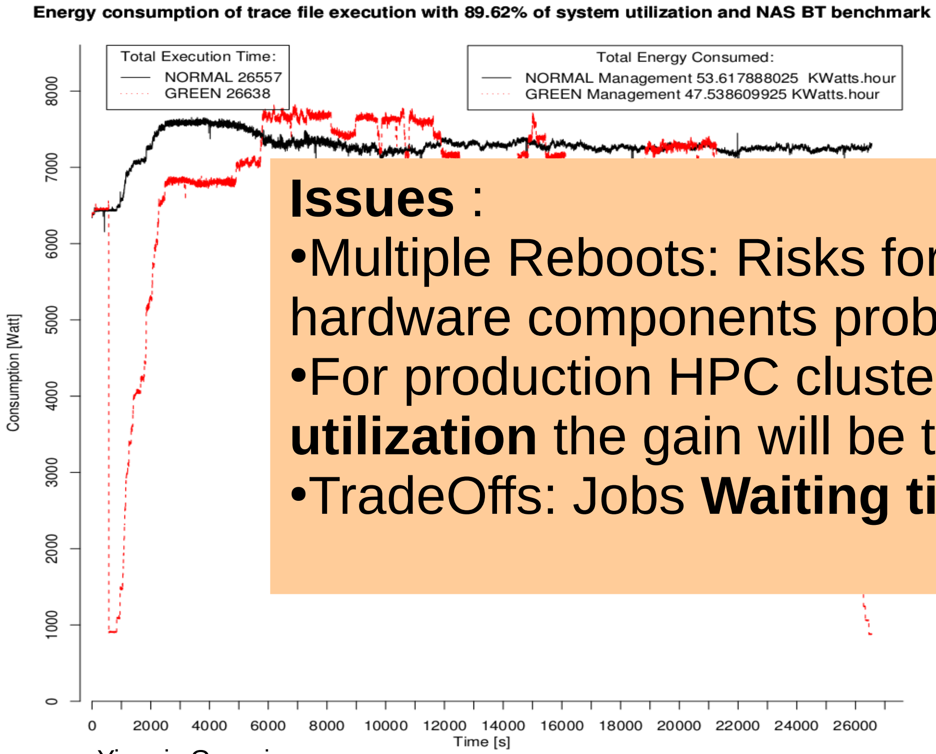


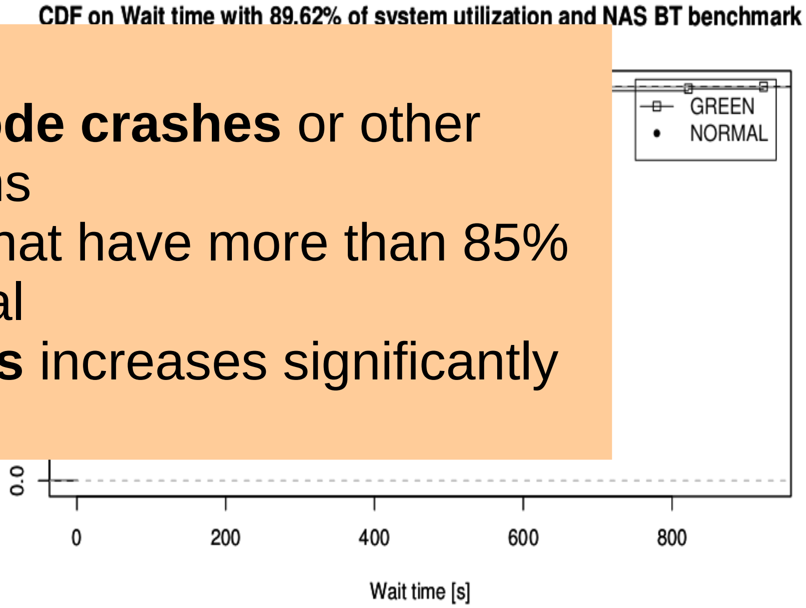


Yiannis Georgiou
Contributions for Resource and Job Management
in High Performance Computing
(PhD Thesis 2010)

Metrics Improvements: Cluster total energy consumption and TCO

Energy Reduction Techniques





Issues :

•Multiple Reboots: Risks for **node crashes** or other hardware components problems

•For production HPC clusters that have more than 85% **utilization** the gain will be trivial

•TradeOffs: Jobs **Waiting times** increases significantly

Yiannis Georgiou
Contributions for Resource and Job Management
in High Performance Computing
(PhD Thesis 2010)

ryax technologies

Power adaptive scheduling

- Provide **centralized mechanism** to dynamically **adapt the instantaneous power** consumption of the whole platform
  - Reducing the number of usable resources or running them with lower power
- Provide technique to plan in advance for **future power adaptations**
  - In order to align upon dynamic energy provisioning and **electricity prices**

Power adaptive scheduling in Slurm v15.08 and later

The implementation appeared in 15.08 has the following characteristics:

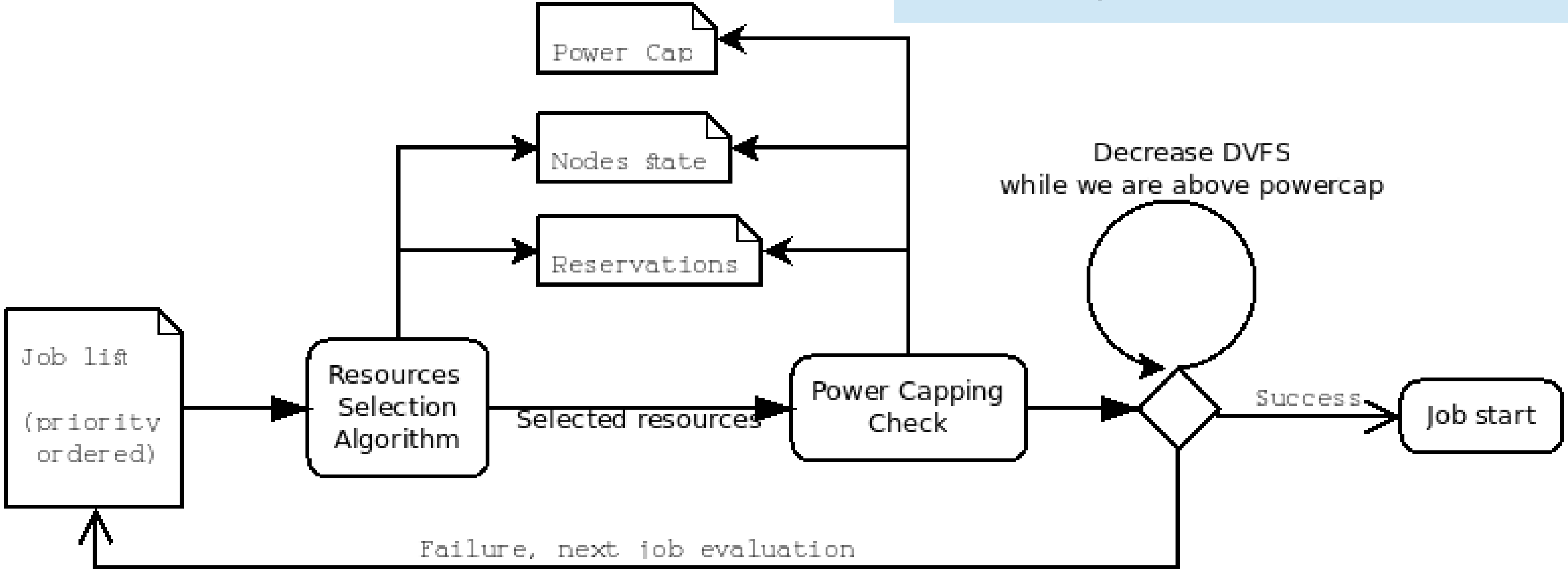
- Based upon layouts framework
  - for internal representation of resources power consumption
  - Only logical/static representation of power
  - Fine granularity down to cores

- Reductions take place through following techniques coordinated by the scheduler:
  - Letting Idle nodes
  - Powering-off unused nodes (using default SLURM mechanism)
  - Running nodes in lower CPU Frequencies (respecting –-cpu-freq allowed frequencies)

Power adaptive scheduling – algorithm

- Implementation based upon new **layouts framework** within SLURM
  - Key/value store
  - Map power consumption upon components

- Power reductions take place through following **coordinated mechanisms:**
  - Letting Idle nodes
  - Powering-off unused nodes
  - Running nodes in lower CPU Frequencies



Power adaptive scheduling - Algorithm

Logic within the Powercapping Check

- Calculate what power consumption the cluster would have if the job was executed
- If higher than the allowed power budget, check if DVFS is allowed for the job (usage of ––cpu-freq parameter with MIN and MAX)
  - If yes then calculate what power consumption the cluster would have if the job was executed with its different allowed CPU-Frequencies
  - Try with the optimal CPU-Frequency which is the one that would allow all the idle resources to become allocated
- If neither the optimal nor the MIN allowed CPU-Frequency for the job results in lower power consumption than the powercap then job pending else running

Power adaptive scheduling – Architecture

Architecture of the Powercapping Check

- Based upon the different nodes bitmaps states
- Using Layouts for collecting and setting nodes and cores power consumption (both get and set functions)
- Each CPU Frequency is represented/considered to have its own power consumption (based on measures or hardware provider specifications)

Power adaptive scheduling – Configuration

- Set parameter within slurm.conf

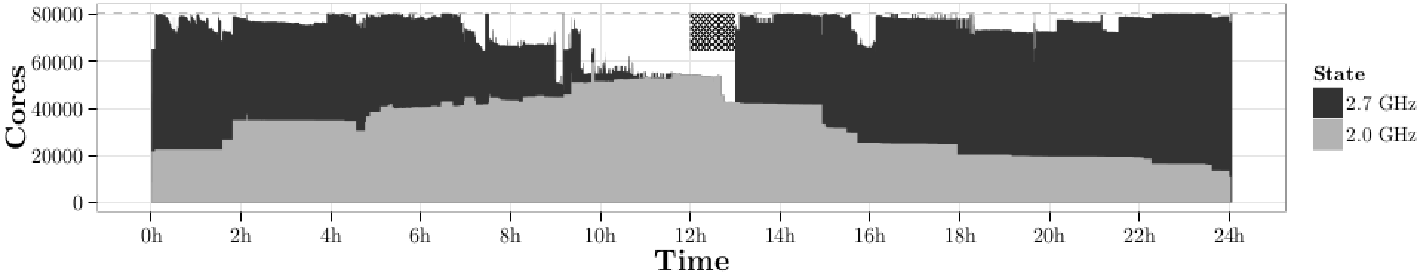
```
[root@nd25 slurm]#cat /etc/slurm.conf |grep power
Layouts=power/cpufreq
```

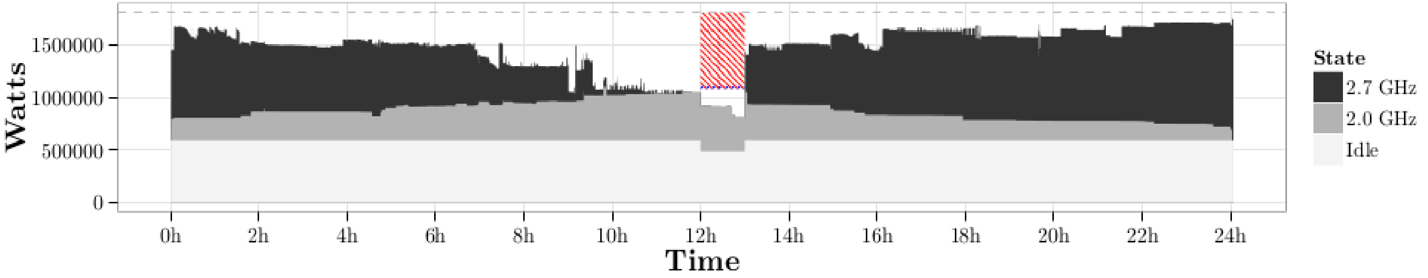
- Set new /etc/layouts.d/power.conf file

```
[root@nd25 slurm]#cat /etc/layouts.d/power.conf

Entity=Cluster Type=Center CurrentSumPower=0 IdleSumWatts=0 MaxSumWatts=0 Enclosed=virtual[0-5039]

Entity=virtualcore[0-80639] Type=Core CurrentCorePower=0 IdleCoreWatts=7 MaxCoreWatts=22 CurrentCoreFreq=0
Cpufreq1Watts=12 Cpufreq2Watts=13 Cpufreq3Watts=15 Cpufreq4Watts=16 Cpufreq5Watts=17 Cpufreq6Watts=18
Cpufreq7Watts=20

Entity=virtual0 Type=Node CurrentPower=0 IdleWatts=0 MaxWatts=0 DownWatts=14 PowerSaveWatts=14 CoresCount=0
LastCore=15 Enclosed=virtualcore[0-15] Cpufreq1=1200000 Cpufreq2=1400000 Cpufreq3=1600000 Cpufreq4=1800000
Cpufreq5=2000000 Cpufreq6=2200000 Cpufreq7=2400000 NumFreqChoices=7

Entity=virtual1 Type=...
```

Power adaptive scheduling

System utilization in terms of cores (top) and power (bottom) for MIX policy during a 24 hours workload of Curie system with a powercap reservation (hatched area) of 1 hour of 40% of total power. Cores switched-off represented by a dark-grey hatched area.





Yiannis Georgiou, David Glesser, Denis Trystram
Adaptive Resource and Job Management for limited power consumption
In proceedings of IPDPS-HPPAC 2015

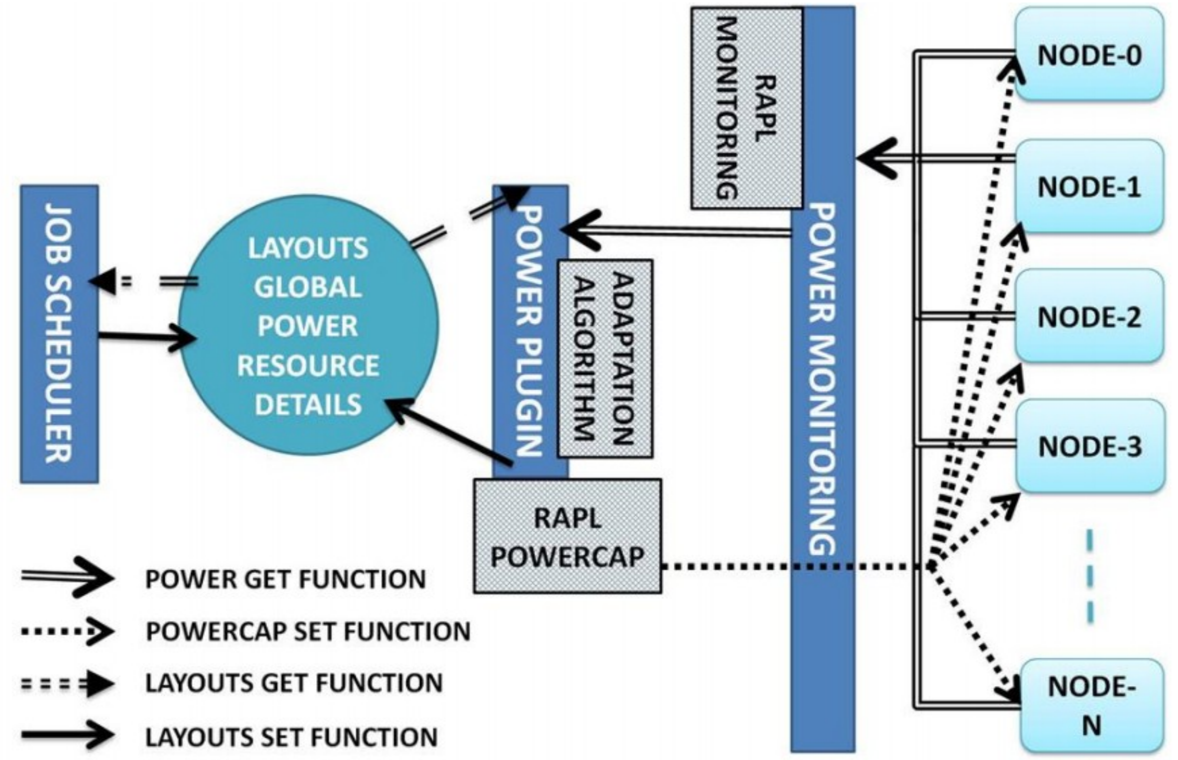
Metrics Improvements: TCO

Enhanced Power adaptive scheduling

- The E-PAS algorithm is capable of efficiently improving the system resource utilization and significantly reducing job waiting times by redistributing the system power under strict powercap regime.
- E-PAS extends the previously conceived PAS algorithm by performing power aware optimizations on the basis of real power monitoring data and has been evaluated on both Intel and ARM architectures.

Dineshkumar Rajagopal, Daniele Tafani, Yiannis Georgiou, David Glesser, Michael Ott:
A Novel Approach for Job Scheduling Optimizations Under Power Cap for ARM and Intel HPC Systems.
In proceedings of HiPC 2017: 142-151

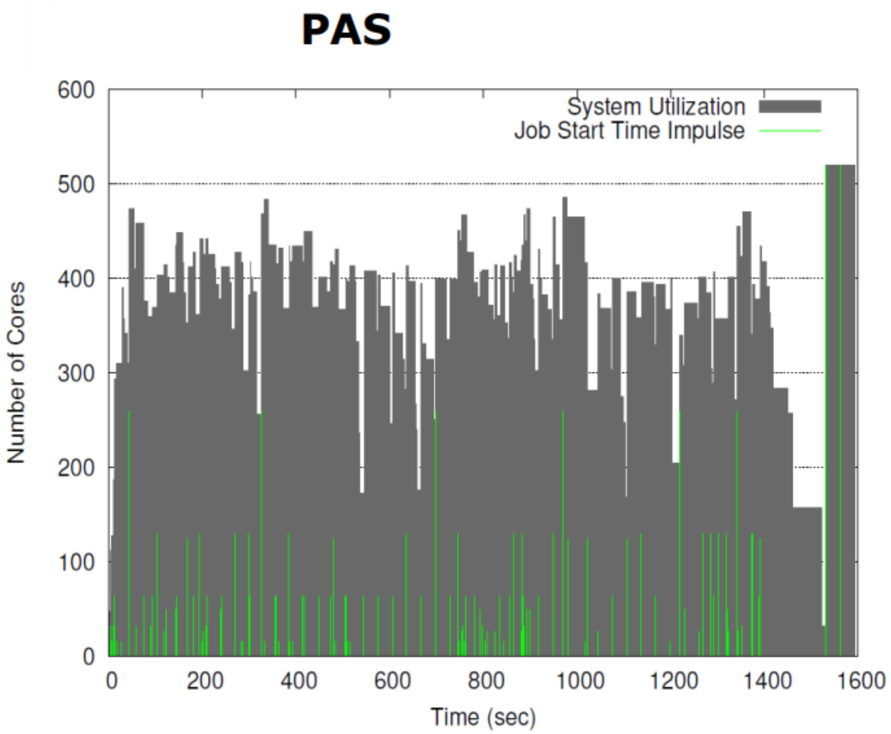
Enhanced Power adaptive scheduling

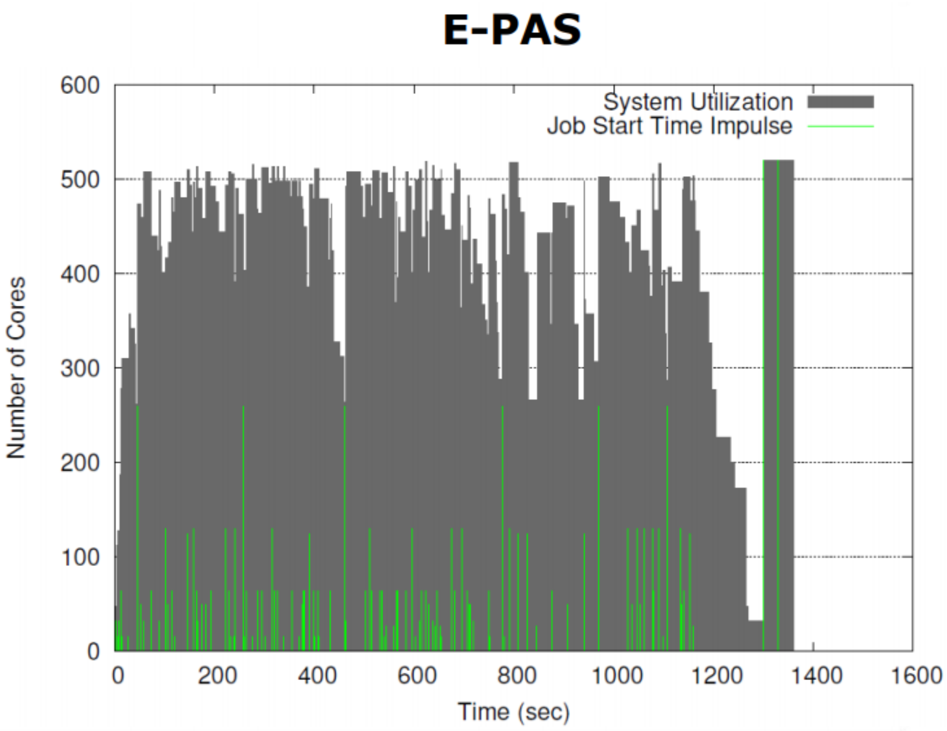


Dineshkumar Rajagopal, Daniele Tafani, Yiannis Georgiou, David Glesser, Michael Ott:
A Novel Approach for Job Scheduling Optimizations Under Power Cap for ARM and Intel HPC Systems.
In proceedings of HiPC 2017: 142-151

ryax technologies

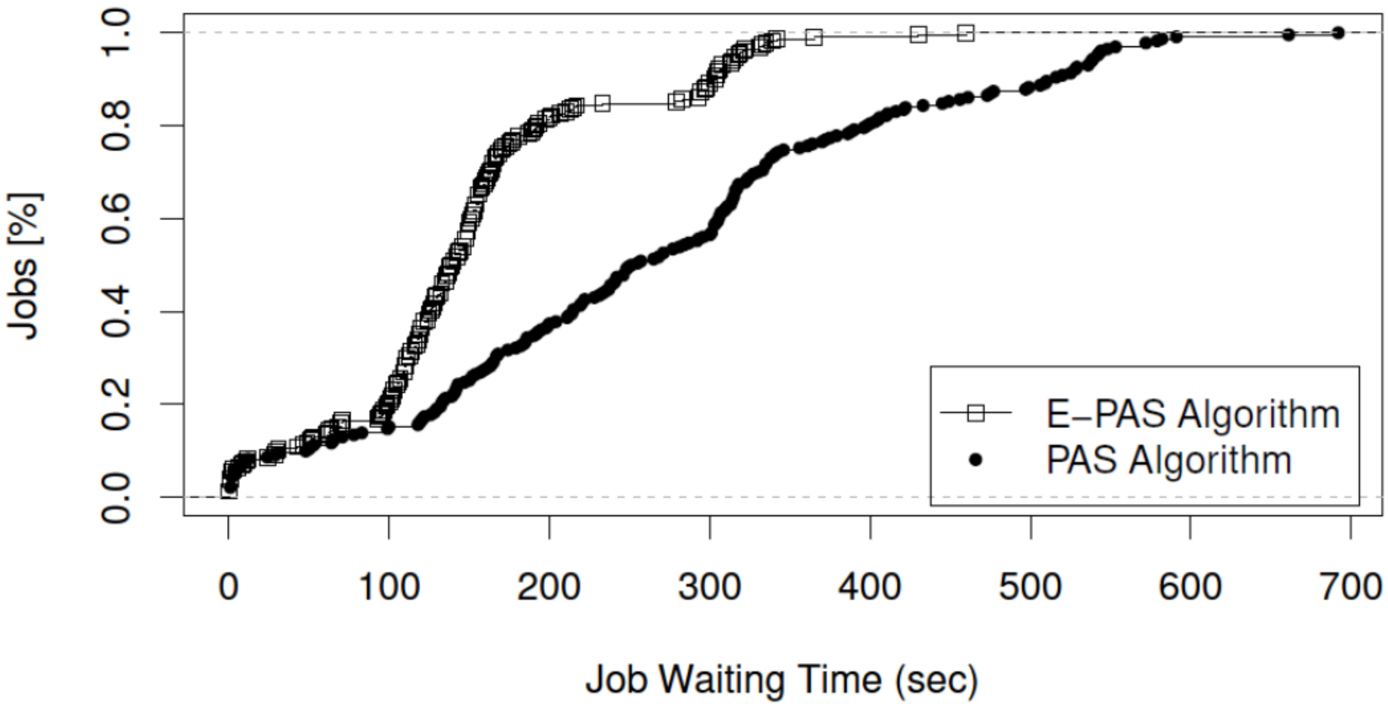
Enhanced Power adaptive scheduling





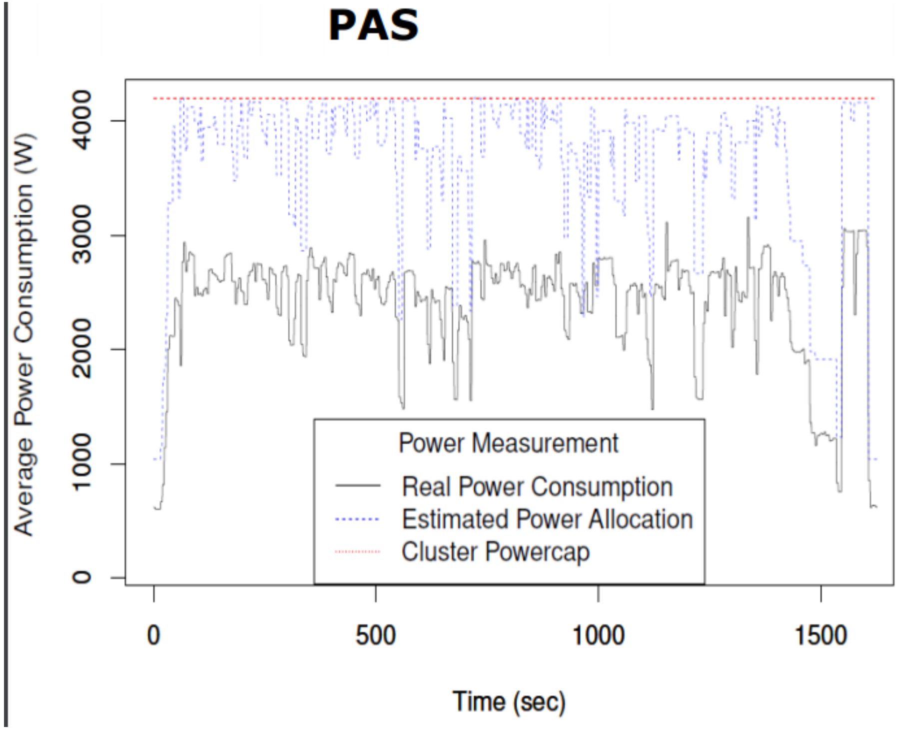
Dineshkumar Rajagopal, Daniele Tafani, Yiannis Georgiou, David Glesser, Michael Ott:
A Novel Approach for Job Scheduling Optimizations Under Power Cap for ARM and Intel HPC Systems.
In proceedings of HiPC 2017: 142-151

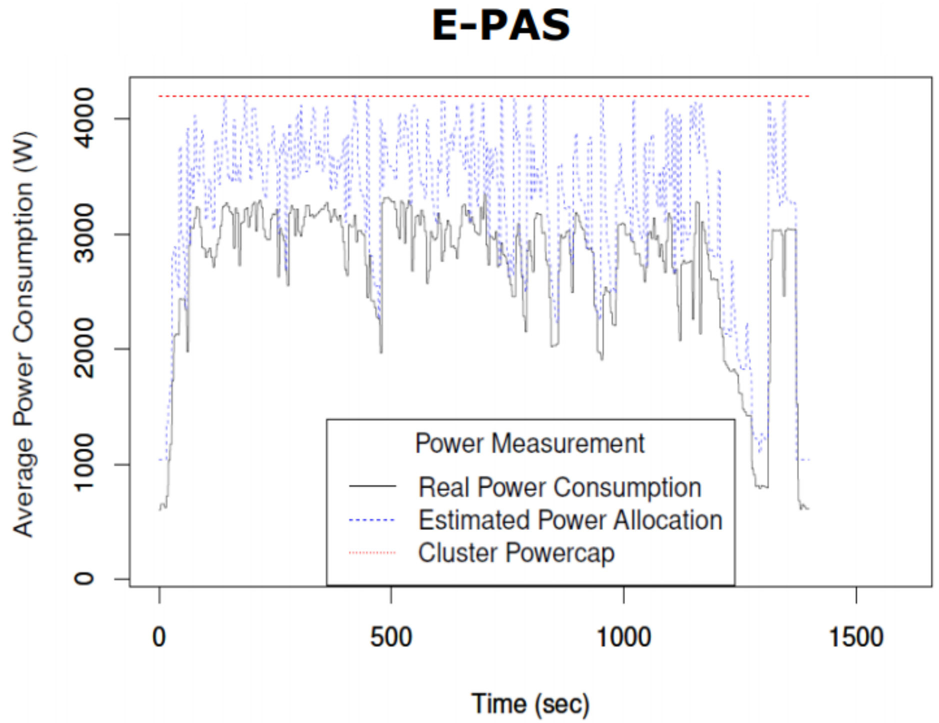
Enhanced Power adaptive scheduling



Dineshkumar Rajagopal, Daniele Tafani, Yiannis Georgiou, David Glesser, Michael Ott:
A Novel Approach for Job Scheduling Optimizations Under Power Cap for ARM and Intel HPC Systems.
In proceedings of HiPC 2017: 142-151

Enhanced Power adaptive scheduling





Dineshkumar Rajagopal, Daniele Tafani, Yiannis Georgiou, David Glesser, Michael Ott:
A Novel Approach for Job Scheduling Optimizations Under Power Cap for ARM and Intel HPC Systems.
In proceedings of HiPC 2017: 142-151

Metrics Improvements: TCO with acceptable system utilization

Outline

1) Typical Job Scheduling and Allocation Mechanisms
2) Energy Reductions and Powercapping Techniques
3) **Topology Aware Job Mapping**
4) Techniques for scheduling experimentation

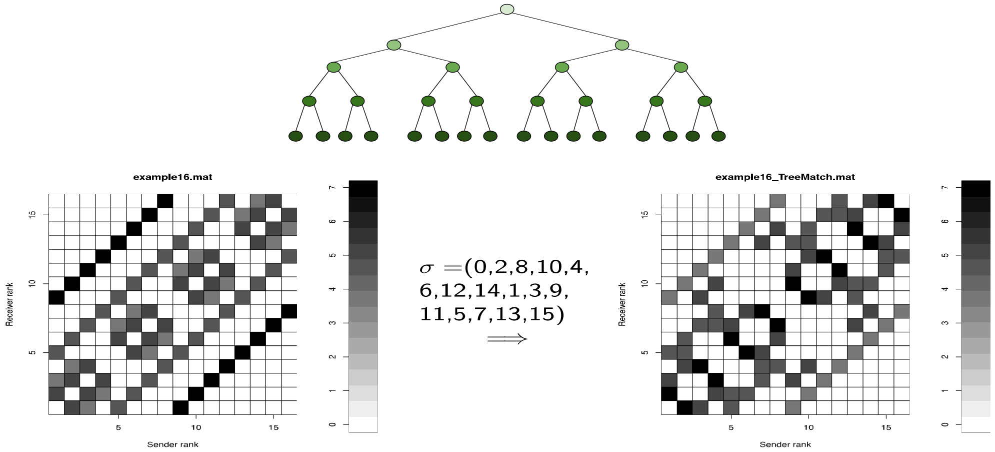
Topology-aware task placement

Improve the way the application exchanges its data:

- based on the expression of bytes/messages exchanged by the application processes (communication pattern)
- match this pattern to the available resources of the underlying architecture by placing processes that communicate more to cores that are closer to each other

Process Placement with Treematch

- Not all the processes exchange the same amount of data
- The speed of the communications, and hence the performance of the application depends on the way processes are mapped to resources.

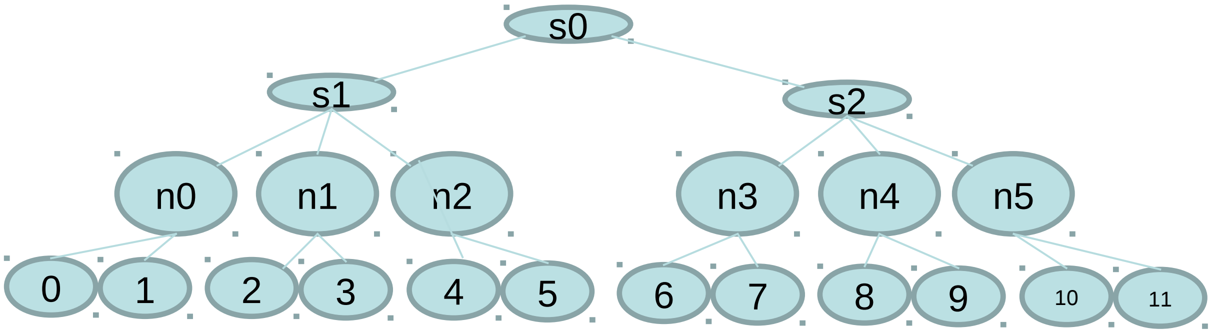


- Communication matrix + Tree Topology = Process permutation

http://treematch.gforge.inria.fr/

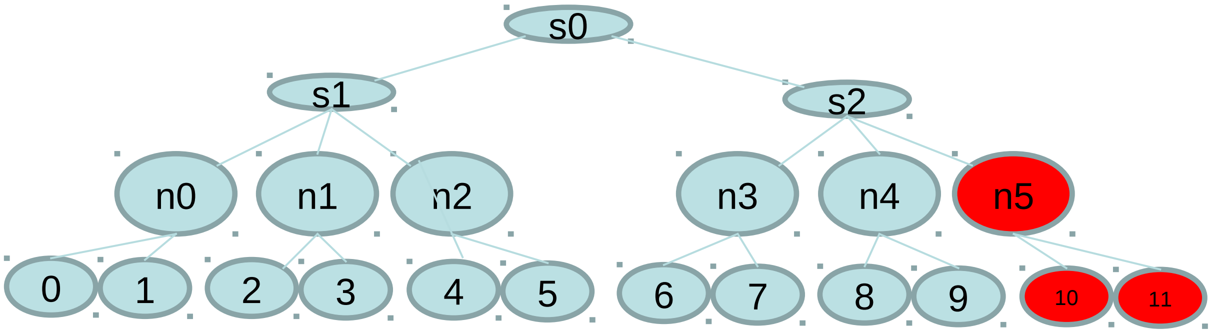
Topology-aware task placement

- Assume the following topology on a small cluster



Topology-aware task placement

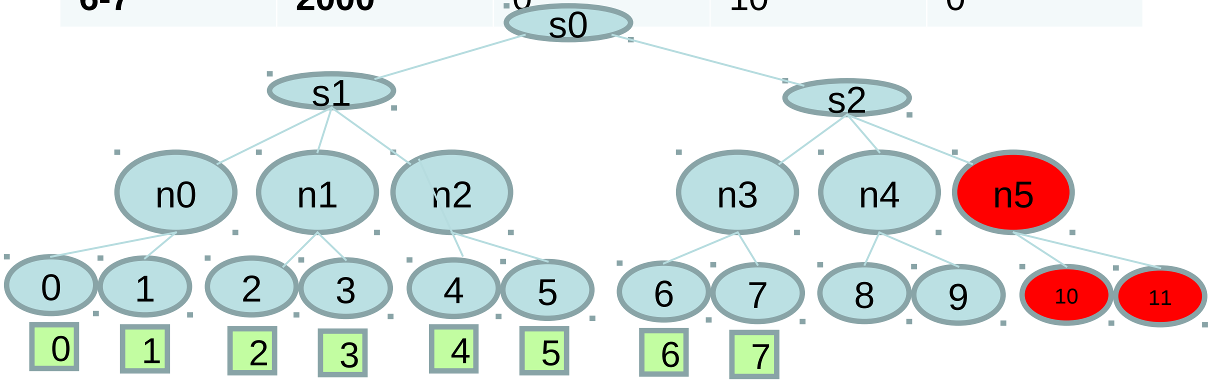
- Suppose that one job will allocate the 2 CPUs of node n5



Topology-aware task placement

- And another job demanding 8 CPUs of 4 nodes with the following communication matrix is submitted. Default Slurm will result into:

Proc.	0-1	2-3	4-5	6-7
0-1	0	20	0	**2000**
2-3	20	0	**1000**	0
4-5	0	**1000**	0	10
6-7	**2000**	0	10	0

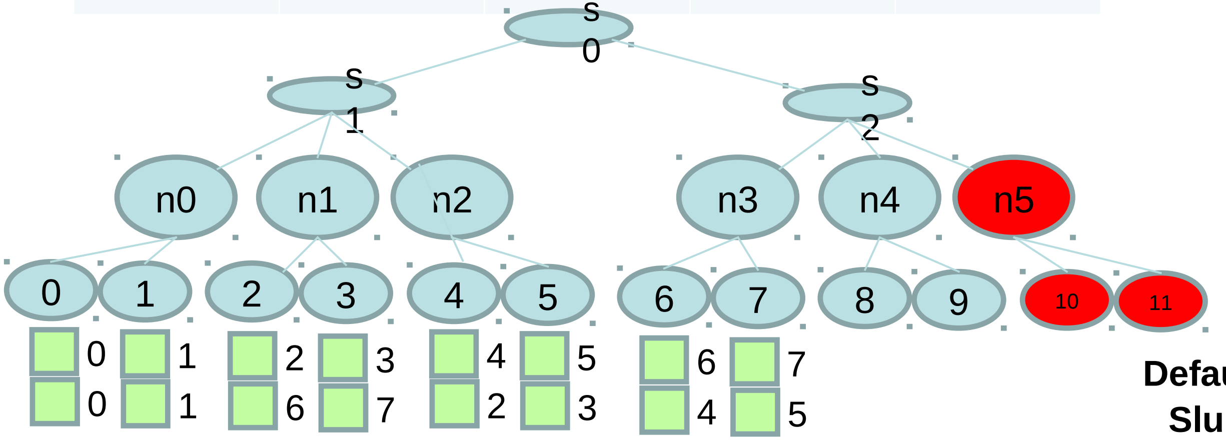


Default Slurm

Topology-aware task placement

- And another job demanding 8 CPUs of 4 nodes with the following communication matrix is submitted. Slurm then TM will result into:

Proc.	0-1	2-3	4-5	6-7
0-1	0	20	0	**2000**
2-3	20	0	**1000**	0
4-5	0	**1000**	0	10
6-7	**2000**	0	10	0



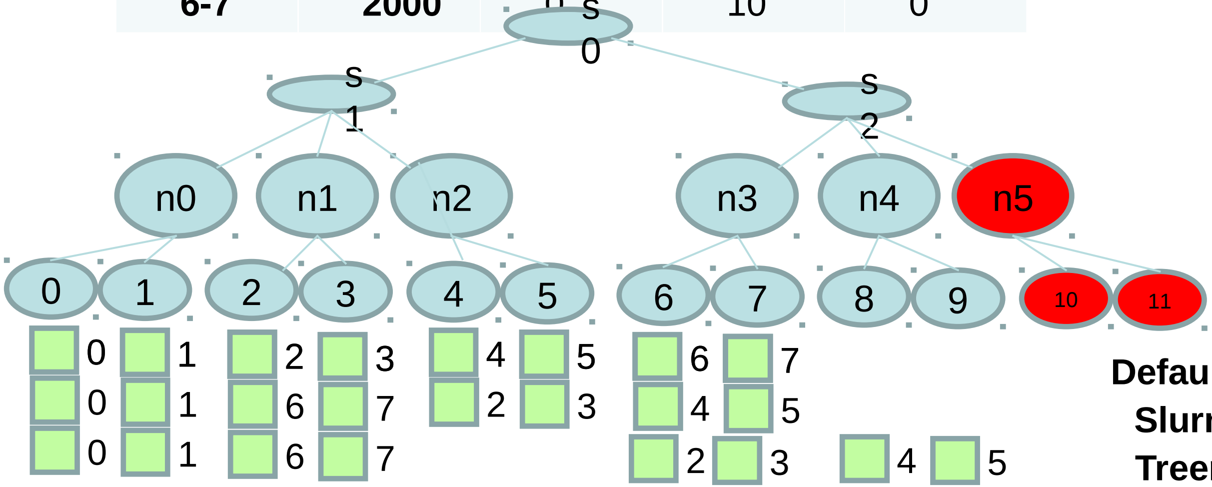
Default Slurm

Slurm then Treematch

Topology-aware task placement

- And another job demanding 8 CPUs of 4 nodes with the following communication matrix is submitted. TM within Slurm will result into:

Proc.	**0-1**	**2-3**	**4-5**	**6-7**
0-1	0	20	0	**2000**
2-3	20	0	**1000**	0
4-5	0	**1000**	0	10
6-7	**2000**	0	10	0



Default Slurm

Slurm then Treematch

Treematch within Slurm

Architecture Treematch within Slurm

- Implemented a new selection option for the select/cons_res plugin of Slurm
  - The communication matrix is provided at job submission time through a new distribution option

```
srun –m TREEMATCH=/comm/matrix/path
```

  - The topology as needed by Treematch is provided by a new parameter in the configuration file

```
#slurm.conf file
TreematchTopologyFile=/topology/file/path
```

  - The availability of resources is retrieved through the node and core bitmaps data structures
    - Slurm local CPU ids are translated to Treematch CPU ids in order to calculate the process permutation.
    - The selected list of CPUs as done by Treematch is then translated back to bitmaps for Slurm to use

Metrics Improvements: Application Performance and Job Execution Time

ryax technologies

Treematch with Slurm Integration

- Treematch-Slurm integration done by Adele Villiermet in the context of her PhD
- Validation and evaluation of this new functionality showed positive and promising results. This work has been published in:

- *Yiannis Georgiou, Emmanuel Jeannot, Guillaume Mercier, Adèle Villiermet: Topology-aware job mapping. IJHPCA 32(1): 14-27 (2018)*

Outline

1) Typical Job Scheduling and Allocation Mechanisms
2) Energy Reductions and Powercapping Techniques
3) Topology Aware Job Mapping
4) **Techniques for scheduling experimentation**

Experimenting with Scheduling Optimizations in Slurm

- Make use of real or synthetic workloads and
- either use the emulation technique of Slurm as used in [1]
- or use simulation techniques such as the one studied in [2] with Batsim and SimGrid. However developments are needed to integrate Slurm scheduler with Batsim

[1] Yiannis Georgiou, Matthieu Hautreux:
Evaluating Scalability and Efficiency of the Resource and Job Management System on Large HPC Clusters.
In proceedings of JSSPP 2012: 134-156

[2] Pierre-François Dutot, Michael Mercier, Millian Poquet, Olivier Richard:
Batsim: A Realistic Language-Independent Resources and Jobs Management Systems Simulator.
In proceedings of JSSPP 2016: 178-197

Activating emulation technique within SLURM

Multiple slurmd technique can be used to experiment with scheduling without using a real-scale cluster:

- 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:

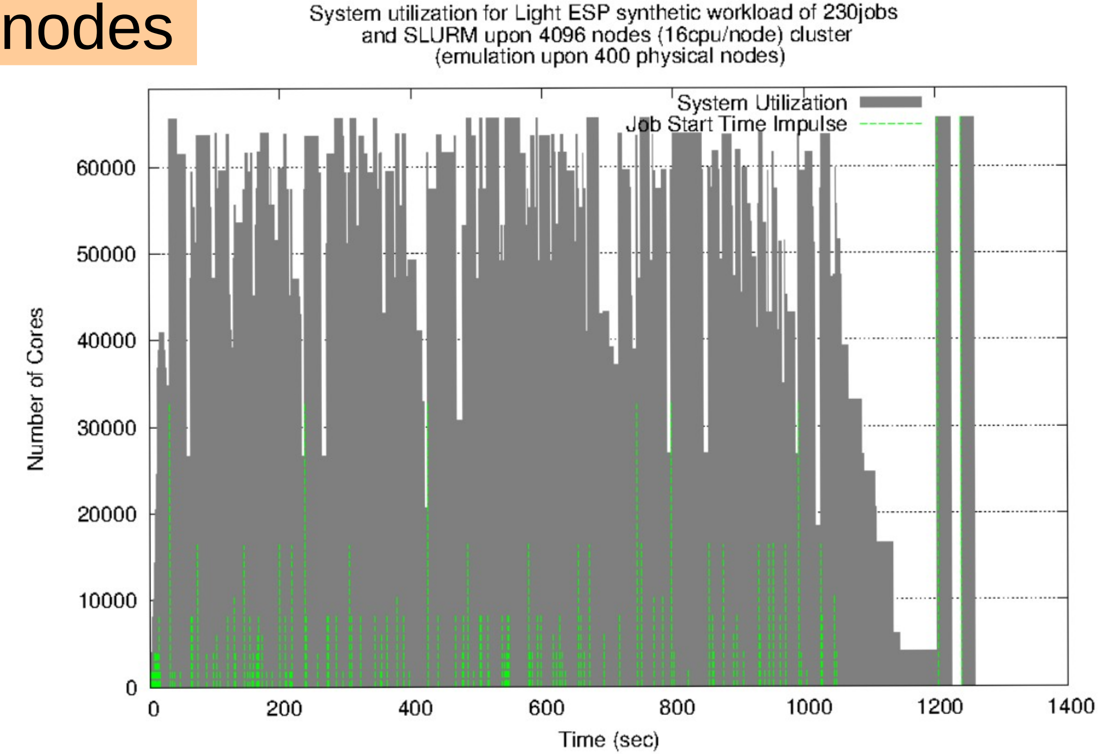
Activating emulation technique within SLURM

```
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

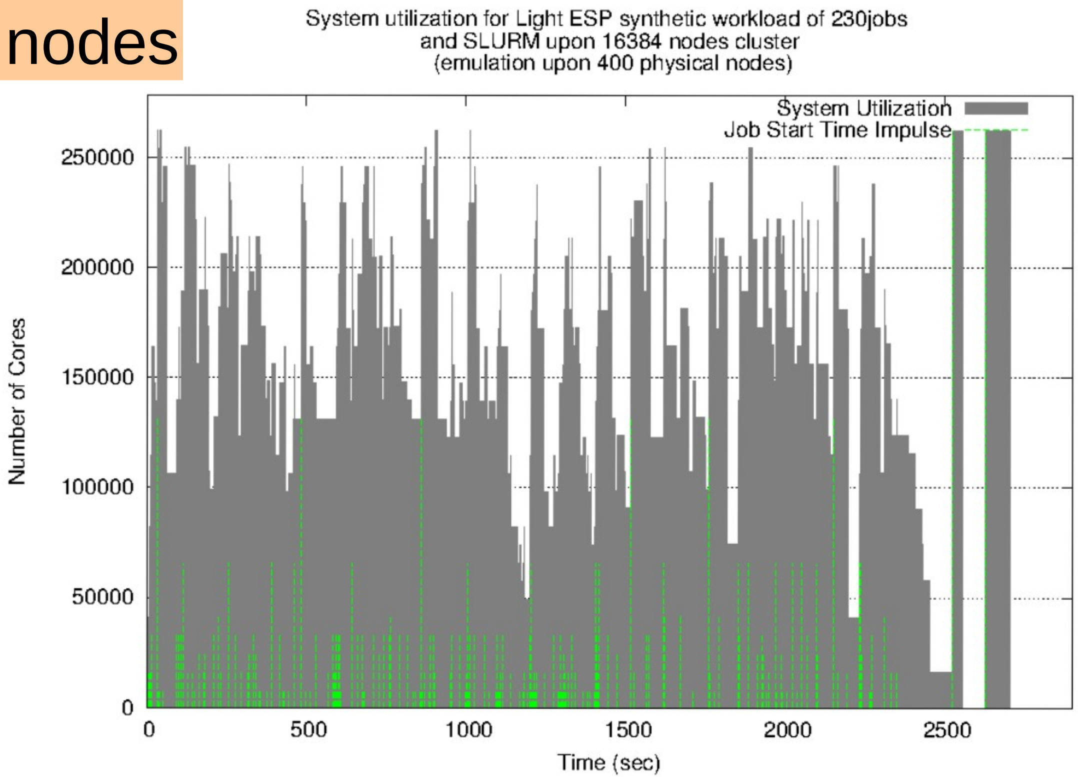
4096 emulated nodes
upon 400 physical nodes



Yiannis Georgiou, Matthieu Hautreux:
Evaluating Scalability and Efficiency of the Resource and Job Management System on Large HPC Clusters.
In proceedings of JSSPP 2012: 134-156

Examples of performance evaluation with emulation

16384 emulated nodes
upon 400 physical nodes



Yiannis Georgiou, Matthieu Hautreux:
Evaluating Scalability and Efficiency of the Resource and Job Management System on Large HPC Clusters.
In proceedings of JSSPP 2012: 134-156

ryax technologies

ryax technologies