ITEA

INFORMATION TECHNOLOGY
FOR EUROPEAN ADVANCEMENT

PERFCLOUD

//H4H (ITEA2 09011)

Optimize HPC Applications on Heterogeneous Architectures

Deliverable: D<5.2.3.3>

Prototype of an algorithm to allocate resources based on energy consumption

Version:	V1.1
Date:	14/10/2013
Authors:	CEA / M. Hautreux
Status:	Final
Visibility:	Public

HISTORY

Document version #	Date	Remarks	Author
V1.0	08/10/2013		Hautreux
V1.1	14/10/2013	Modifications following Benoit Pradelle review.	Hautreux

TABLE OF CONTENTS

1. Executive Summary

The last decades have shown an ever growing requirement in terms of computing and storage resources. This tendency recently puts the pressure on the ability to efficiently manage the power required to operate the vast amount of electrical components associated with state-of-the-art computing and data centers.

This deliverable presents the work performed on the resource and job management system SLURM in order to provide a power capping mechanism enabling to limit the power budget available to a computing cluster among time.

A working prototype providing the capability to autonomously adapt the executed workload to the available or planned budget is detailed. Limitations and future works are presented and proposed as following work of this task of the PerfCloud project.

2. Objectives

The main objectives of the software prototype of *dynamic power capping* within SLURM are to allow the definition of a power budget for a cluster as well as its variations among time. By variations, we mean the ability to plan additional *power cuts* in a defined budget in order to further reduce the amount of available power to the execution nodes at some points in time.

Use cases of the *dynamic power capping* are the ability to limit the instantaneous power consumption of a cluster in order to cope with power supplier restrictions or to avoid using too much power in a fluctuating power prices system when the price of the energy is too high to afford a fully used system.

In order to stay in the defined or planned budget, the resource manager will have to arbitrate the computing resources usage to enforce the targeted threshold. To do that, the resource manager will compute the current maximum amount of required power, based on the executing workload, and will only start jobs for which the selected resources usage will not prevent the system from respecting the target.

3. Background

SLURM is a widely used open-source resource manager and batch scheduler. It enables to efficiently manage large workloads on large systems, among the largest currently in production. It is currently available in the BULL software stack as installed on the *PerfCloud* hardware prototype *Cirrus*. SLURM offers a large variety of features but provides only a limited support of energy saving mechanism within a cluster.

A *power saving* logic [PSV] enables to define a set of nodes, scripts, timeouts and ratios, enabling to perform automated *suspend/resume* operations on idle nodes not yet necessary. In spite of being interesting to reduce the total power bill of a system, this mechanism does not enable to enforce the respect of a particular power cap. Furthermore, on systems where shutting down nodes on demand is not possible or simply not desired, this logic has a limited benefit in terms of real power savings.

BULL has recently enhanced the power awareness of SLURM by introducing the capability to regularly capture the instantaneous consumed power of nodes. This information is used to estimate the amounts of energy required to execute the different jobs [ENG]. Coupled with the introduction of a DVFS logic, enabling to modify the maximum frequencies of the cores involved in a parallel execution, this new feature helps to identify the behavior of applications in terms of power consumption when varying the frequency. Thus, users can experiment different frequency values to evaluate the behavior of their jobs and optimize their energy efficiency. This feature is an important milestone in the power awareness road map of SLURM but does not help to work in a limited power budget.

SLURM offers a mechanism called advanced reservations [RES]. This mechanism enables to plan reservations of resources, whether real computing nodes or virtual resources like ISV licenses, in order to avoid their usages or restrict them to a predefined set of users on particular time slots. Although being commonly used to anticipate full or partial maintenance or simply to dedicate resources to training sessions, this mechanism is not of any help when it comes to plan *power cuts*.

Introducing the *dynamic power capping*, as explained in the objectives of this deliverable, therefore requires to implement a new power capping feature and to adapt the product reservation logic.

4. Solution design

SLURM, in a nutshell, is designed as a client-server distributed application : a centralized server daemon *slurmctld*, also known as the *controller*, communicates with a client daemon *slurmd* running on each computing node. Users can request the controller for resources to execute interactive or batch applications, referred as *jobs*. The controller dispatches the jobs on the available resources, whether full nodes or partial nodes, according to a configurable set of rules.

To achieve the targeted goal of *dynamic power capping*, a new parameter, *PowerCap* is added to the controller 's set of states. It represents the allowed power budget in watts of the cluster.

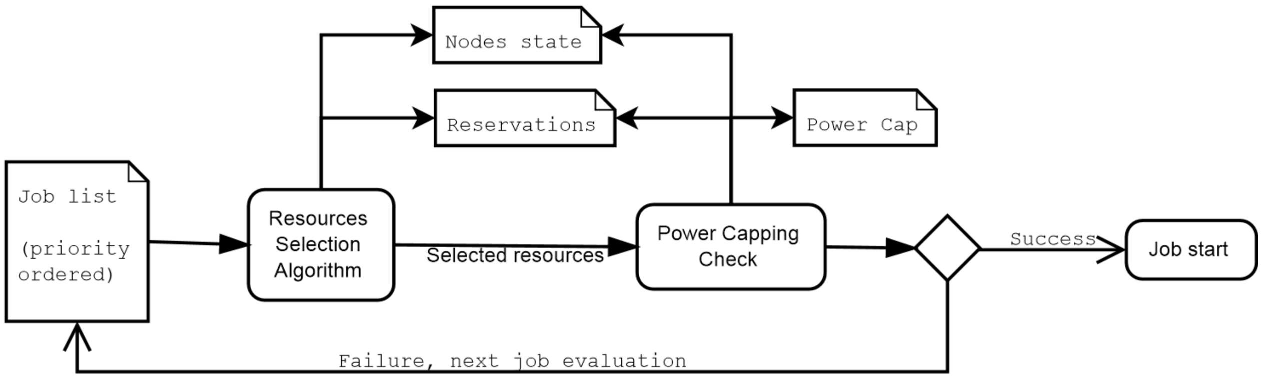
To compute the maximum power amount required to operate a cluster, new parameters are associated to the compute nodes definition to provide the different amounts of watts required to operate them at different states. Thus, the *IdleWatts*, *MaxWatts*, *PowerSaveWatts* and *DownWatts* will respectively correspond to the amounts of watts required to operate a node in idle, fully used, power saved and down states. The down state corresponds to the state the controller uses to characterize a node not being currently accessible within SLURM.

While computing the instantaneous maximum amount of power of the cluster, the controller will use the known states of the nodes in order to sum up the individual maximum amounts of watts and produce a single power value for the whole cluster.

An additional parameter *PowerCapPriority*, when set to the zero value, enables to specify nodes for which the power capping logic must always consider the *MaxWatts* as the current maximum amount of power required. Thus, these nodes will always have their power requirement guarantee and will always be available for executions.

SLURM reservation characteristics are extended by a new *Watts* parameter in order to specify a particular amount of power reserved for a particular time slot: this enables to define *power reservations* corresponding to the previously discussed power cuts. The power reservations will thus enable to dynamically cap the amount of available power among time.

When evaluating the impact of the start of a pending job, the controller will temporary alter the states of the candidate nodes, compute the resultant threshold and compare it with the defined and planned caps. By planned caps, we mean the values corresponding to the current cap of the cluster minus the different power reservations encountered during the estimated execution time of the job.



Schematic view of the power capping mechanism

In case of power budget overflow, the evaluated job will stay pending and the next one of the list will be tried instead. Thus, only the first n jobs of the pending list enabling to respect the budget will be executed. As soon as jobs finish, associated nodes may return to the idle state, resulting in a new power capacity available for other pending jobs.

Note that the current prototype does not make any difference in power requirements whether nodes are fully or partially used. The evaluation of new jobs only filling partially used nodes will always pass the power capping criteria as no extra power will be required. As a result, the scheduler will tend to fill the compute nodes up to the targeted power budget.

The *PowerSaveWatts* is an important parameter in the implemented dynamic power capping logic. By default, it corresponds to the same amount of watts as the *IdleWatts* parameter. This means that the scheduler does not know if a configured power saving logic has an effect on the power consumption nodes in the power saved state. In that case, the only lever of the capping logic is to let nodes idle (or equivalent) to respect the cap, resulting in a potentially large amount of unused active nodes.

However, when the dynamic power capping is used in combination with an efficient power saving configuration (shutdown or deep-sleep) and the *PowerSaveWatts* value is set to the real amounts of watts required by the power saved nodes, the reclaimed power can be used to keep a larger number of nodes usable. In such a configuration, the system should tend to use the optimum number of nodes over a power capped period.

5. Solution usage

This section presents different utilization examples of the implemented dynamic power capping mechanism to better understand its logic, benefits and possibilities.

In the different examples, an emulated cluster is used as the back-end platform running the modified version of SLURM 2.6.2 (latest stable branch at the time of writing). The emulated mode consists of running a modified nested SLURM cluster inside an allocation of resources in an existing SLURM cluster. All the SLURM components required to operate a real cluster are present in the emulated instance and communicate with each other, ensuring the validity of the results in a real scenario.

The emulated cluster is made of 257 nodes with 32 cores per node, 256 being part of the main partition. The power capping parameters are configured as follows:

- PowerCap = INFINITE : meaning that the power capping logic is enabled but without any restriction on the available power budget.
- IdleWatts = 450 : the amount of watts considered for nodes in idle state
- MaxWatts = 950 : the amount of watts considered for nodes used by jobs
- DownWatts = undefined : meaning that we use the *MaxWatts* value by default
- PowerSaveWatts = undefined : meaning that we use the *IdleWatts* value by default

The following examples will precise the configuration changes when necessary.

1. Example 1

The first example illustrates the monitoring of the power capping logic counters.

```
[mat@leaf0 utilit]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
physical     up   infinite       1  idle leaf0
virtual*     up   infinite     256  idle leaf[1000-1255]
[mat@leaf0 utilit]$ scontrol show powercap
MinWatts=116150  CurrentWatts=116150  PowerCap=INFINITE  AdjustedMaxWatts=244150
MaxWatts=244150
[mat@leaf0 utilit]$
```

While no nodes are used, the current maximum amount of watts required, provided by the *CurrentWatts* parameter, equals the minimum amount *MinWatts*. In our case, it represents 257 * 450 = 116,15 kW.

While *MaxWatts* represents the maximum amount of watts required to operate the cluster having all the nodes busy, the *AdjustedMaxWatts* is the maximum amount of watts required to operate the cluster, taking into consideration down and power saved nodes. As no nodes are in these two states in our first example, the two values are equal.

Once a job is started, the values are updated to reflect the current usage of the power, as illustrated below.

```
[mat@leaf0 utilit]$ srun -n 10 -N 10 sleep 100 >/dev/null &
[1] 23819
[mat@leaf0 utilit]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
physical     up   infinite      1   idle leaf0
virtual*     up   infinite     10    mix leaf[1000-1009]
virtual*     up   infinite    246   idle leaf[1010-1255]
[mat@leaf0 utilit]$ scontrol show powercap
MinWatts=116150  CurrentWatts=121150  PowerCap=INFINITE  AdjustedMaxWatts=244150
MaxWatts=244150
[mat@leaf0 utilit]$
```

Setting a power cap below the required amount of 121,15 kW then prevents from executing an equivalent job, unless the number of involved nodes is reduced :

```
[mat@leaf0 utilit]$ scontrol update powercap=121000
[1]+  Done                    srun -n 10 -N 10 sleep 100 > /dev/null
[mat@leaf0 utilit]$ scontrol show powercap
MinWatts=116150  CurrentWatts=116150  PowerCap=121000  AdjustedMaxWatts=244150
MaxWatts=244150
[mat@leaf0 utilit]$ srun -n 10 -N 10 sleep 100 >/dev/null &
[1] 25975
srun: Required power not available now
srun: job 5 queued and waiting for resources
[mat@leaf0 utilit]$ srun -n 10 -N 9 sleep 100 >/dev/null &
[2] 25978
[mat@leaf0 utilit]$ squeue
       JOBID PARTITION     NAME     USER ST     TIME  NODES NODELIST(REASON)
           5   virtual    sleep        mat PD   0:00     10 (PowerNotAvail)
           6   virtual    sleep        mat  R   0:05      9 leaf[1000-1008]
[mat@leaf0 utilit]$ scontrol show powercap
MinWatts=116150  CurrentWatts=120650  PowerCap=121000  AdjustedMaxWatts=244150
MaxWatts=244150
[mat@leaf0 utilit]$ squeue
       JOBID PARTITION     NAME     USER ST     TIME  NODES NODELIST(REASON)
           5   virtual    sleep        mat PD   0:00     10 (PowerNotAvail)
[2]+  Done                    srun -n 10 -N 9 sleep 100 > /dev/null
[mat@leaf0 utilit]$ scontrol update powercap=INFINITE
[mat@leaf0 utilit]$ srun: job 5 has been allocated resources
```

```
[mat@leaf0 utilit]$ squeue
       JOBID PARTITION     NAME     USER ST     TIME  NODES NODELIST(REASON)
           5   virtual    sleep      mat  R     0:14     10 leaf[1000-1009]
```

2. Example 2

The second example illustrates the usage of the power saving mechanism with the *PowerSaveWatts* parameter of the nodes to optimize the number of available nodes.

The configuration is modified to reduce the *PowerSaveWatts* parameter to 5 Watts, the supposed amount of power required to enable the BMC of the nodes when the nodes are shutdown.

```
[mat@leaf0 utilit]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
physical     up   infinite      1   idle leaf0
virtual*     up   infinite    256   idle leaf[1000-1255]
[mat@leaf0 utilit]$ scontrol show powercap
MinWatts=2230  CurrentWatts=116150  PowerCap=INFINITE  AdjustedMaxWatts=244150
MaxWatts=244150
[mat@leaf0 utilit]$ scontrol update powercap=121000
[mat@leaf0 utilit]$ scontrol show powercap
MinWatts=2230   CurrentWatts=116150   PowerCap=121000   AdjustedMaxWatts=244150
MaxWatts=244150
[mat@leaf0 utilit]$ srun -n 10 -N 10 sleep 100 >/dev/null &
[1] 1629
srun: Required power not available now
srun: job 2 queued and waiting for resources
[mat@leaf0 utilit]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
physical     up   infinite      1   idle leaf0
virtual*     up   infinite    256   idle leaf[1000-1255]
[mat@leaf0 utilit]$ # waiting for nodes  to be shutdown
[mat@leaf0 utilit]$ srun: job 2 has been allocated resources
[mat@leaf0 utilit]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
physical     up   infinite      1   idle leaf0
virtual*     up   infinite    107  idle~ leaf[1010-1116]
virtual*     up   infinite     10    mix leaf[1000-1009]
virtual*     up   infinite    139   idle leaf[1117-1255]
[mat@leaf0 utilit]$ scontrol show powercap
MinWatts=2230   CurrentWatts=73535   PowerCap=121000   AdjustedMaxWatts=143035
MaxWatts=244150
```

```
[mat@leaf0 utilit]$
```

The listing illustrates that a job is first blocked, waiting for sufficient available power and then started as soon as a first set of nodes is taken into account by the power saving mechanism. At that moment, enough power was reclaimed from formerly idle nodes and starting the job is possible.

Note that once nodes are reclaimed by the power saving logic, the power capping will prevent them from being used again, ensuring that the remaining number of idle nodes is the optimal number..

3. Example 3

The last example illustrates the usage of a power reservation in order to define in advance a power cut. The SLURM configuration used in this example is the default configuration for ease of understanding. However, the power saving logic could be used in combination too.

```
[mat@leaf0 utilit]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
physical     up   infinite      1   idle leaf0
virtual*     up   infinite    256   idle leaf[1000-1255]
[mat@leaf0 utilit]$ scontrol show powercap
MinWatts=116150  CurrentWatts=116150  PowerCap=INFINITE  AdjustedMaxWatts=244150
MaxWatts=244150
[mat@leaf0     utilit]$     scontrol     create     res     FLAG=LICENSE_ONLY
starttime=now+20minutes duration=3 Watts=123150 Users=root
Reservation created: root_1
[mat@leaf0 utilit]$ scontrol show res
ReservationName=root_1      StartTime=2013-10-02T23:57:26      EndTime=2013-10-
03T00:00:26 Duration=00:03:00
   Nodes=    NodeCnt=0    CoreCnt=0    Features=(null)    PartitionName=virtual
Flags=LICENSE_ONLY
   Users=root Accounts=(null) Licenses=(null) Watts=123150 State=INACTIVE
[mat@leaf0 utilit]$ scontrol show powercap
MinWatts=116150  CurrentWatts=116150  PowerCap=INFINITE  AdjustedMaxWatts=244150
MaxWatts=244150
[mat@leaf0 utilit]$ srun --immediate -t 40 -n 10 -N 10 sleep 100 >/dev/null
srun: Force Terminated job 2
srun: error: Unable to allocate resources: Required power at least partially
reserved
[mat@leaf0 utilit]$ srun --immediate -t 15 -n 10 -N 10 sleep 60   &
[1] 10135
[mat@leaf0 utilit]$ scontrol show powercap
```

```
MinWatts=116150  CurrentWatts=121150  PowerCap=INFINITE  AdjustedMaxWatts=244150
MaxWatts=244150
[1]+  Exit 1                    srun --immediate -t 20 -n 10 -N 10 sleep 60
[mat@leaf0 utilit]$ srun -t 20 -n 10 -N 10 sleep 60 &
[1] 12270
[mat@leaf0 utilit]$ srun: Required power at least partially reserved
srun: job 12 queued and waiting for resources
[mat@leaf0 utilit]$ srun --immediate -t 30 -n 10 -N 9 true >/dev/null
[mat@leaf0 utilit]$ scontrol delete reservation=root_1
[mat@leaf0 utilit]$ srun: job 12 has been allocated resources
[1]+  Done                      srun -t 20 -n 10 -N 10 sleep 60
[mat@leaf0 utilit]$
```

The listing shows that jobs finishing before the start of a limiting power reservation are executed properly. However, jobs overlapping the reservation time and requiring more power than available are blocked. Those requiring an allowed amount of power are started as usual.

As soon as the reservation is removed, the blocked jobs are started if the power is then available.

6. Conclusion and future works

A first software prototype for *dynamic power capping* within SLURM is now operational. It should help to autonomously enslave the load of clusters with the power supply capacity of the computing and data centers.

It is noteworthy that the current power capping logic is a conservative approach. It needs the definition of empirical values associated with the different amounts of power required by the nodes in their various states. These empirical values need to be estimated based on hardware characteristics and fully-loaded stress-tests. Optimizations could be performed and result in the autonomous definition of some of these values to adapt the internal capping counters to the real power usage of the resources.

The prototype still needs further evaluations and enhancements to optimize its behavior with real heterogeneous workloads. Indeed, applying a new power constraint in SLURM showed that some scheduling behaviors, like the back-fill mechanism, are modified in depth and need to be adapted. For example, the current prototype tends to follow a *first-fit* scheduling mechanism as soon as the power is capped. This ensures a good utilization of the power resource but potentially delays large jobs having higher priorities but not being eligible because of insufficient available power.

The scheduling logic of SLURM was not altered to search for the best trade-off between resources and power when selecting resources. The introduced mechanism only considers the selected resources and validates that the associated power requirements can be satisfied. Properly supporting multi-objectives scheduling in SLURM requires complex additional adaptations.

Additional enhancements in the power capping mechanism are planned. The current logic does not take into account the possibilities offered to users of using DVFS to limit the amount of power required per node. Handling this feature properly, it could be possible to further optimize the number of usable nodes reclaiming the extra free power of nodes used at lower frequencies. Hardware capabilities to specify a power cap per node, like the one offered by the HP Proliant servers [HPP] or using the Intel Intelligent Power Node Manager [ITL], could be used in combination with the allowed amounts of watts to enforce the amount of watts usable per node running at intermediate levels of power.

These possibilities will be studied in the following months. The most interesting and applicable ones will be introduced in the next version of the prototype. The result of these studies will be detailed in the next deliverable D5-2.3.5 planned for June 2014.

7. Abbreviations and acronyms

BMC	Baseboard Management Card, the embedded hardware in charge among others of powering on/off the node
DVFS	Dynamic Voltage/Frequency Scaling : mechanism enabling to reduce the voltage or frequency of an electrical component in order to reduce its consumption and/or temperature
ISV	Independent Software Vendor

8. References

[PSV] http://slurm.schedmd.com/power_save.html

[RES] http://slurm.schedmd.com/reservations.html

[ENG] http://slurm.schedmd.com/acct_gather_energy_plugins.html

[HPP] http://h20000.www2.hp.com/bc/docs/support/SupportManual/c01549455/c01549455.pdf

[ITL] https://communities.intel.com/docs/DOC-4766