Improved system utilization respecting jobs needs under strict power budget

Dineshkumar Rajagopal,
Yiannis Georgiou,
David Glesser

R&D Bull
Introduction

- **Energy efficiency** major requirement in all levels of hardware and software design
- Additional constraints: **maximum power capping** or constant power consumption with only small perturbation.
  - If those constraints are met, a power supplier can often provide a significantly lower price, thus increasing the efficiency in terms of TCO.
  - If not there may be fines
Motivation for system-wide powercap

- Need for **centralized mechanism** to dynamically **adapt the instantaneous power** consumption of the whole platform
Motivation for system-wide powercap

- Need for **centralized mechanism** to dynamically **adapt the instantaneous power** consumption of the whole platform

![Diagram](image)

- Time
- Nodes
- Allowed max power
- Power
- Time

Nodes
First iteration
Power adaptive scheduling

Power adaptive scheduling is a feature within SLURM appeared in version 15.08
- Initial algorithms and prototype made by CEA in 2013
- A second prototype (extended version of the first) has been studied, experimented and published in [Georgiou et al. HPPAC-2015] by BULL + LIG

Final implementation (BULL) based upon the layouts framework and its API functions (CEA)
Power adaptive scheduling

The implementation as appeared in Slurm v15.08 has the following characteristics:

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

Power 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 \(-\text{--cpu-freq}\) allowed frequencies)
Layouts Framework

• Supercomputers become more complex structures
  • Not all the resources characteristics are taken into account by the RJMS:
    • Power Consumption per Component, Electrical Connections, Communications roles

• These characteristics can give valuable information that may be used to optimize automatic decisions:
  • Scheduling, Energy Efficiency, Scalability

Layouts framework appeared in Slurm v15.08

Slurm User Group presentation, Sep 2015:
Layouts Framework

- One Cluster
  - with multiple views (representing new characteristics) of the same resources
Slurm Layouts Framework
Example of Automatic Update

Power : autoupdate of values by inheritance

- GlobalWatts
  - CHILDREN: SUM
    - CurrentWatts
      - building1
        - GlobalWatts: 290000
      - room1
        - GlobalWatts: 450000
      - rack10
        - GlobalWatts: 19200
      - leaf100
        - CurrentWatts: 450
      - leaf101
        - CurrentWatts: 450
Slurm Layouts Framework
Example of Automatic Update

Power : autoupdate of values by inheritance

GlobalWatts
CHILDREN
SUM
CurrentWatts

building1
GlobalWatts
2900000

room1
GlobalWatts
450000

rack10
GlobalWatts
19200

leaf100
CurrentWatts
450

leaf101
CurrentWatts
1250
Slurm Layouts Framework
Example of Automatic Update

Power : autoupdate of values by inheritance
Static Power Layout Configuration

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

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

Entity=node0 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=node1 Type=...

Entity=core[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
Algorithm: DVFS Power Adaptive Scheduling

- Reductions through DVFS, idle and shut-down nodes (if power-save mode activated)
- Considering core level power consumption

Layouts = power/cpufreq

Respect DVFS (min-max) of each job

DVFS Power adaptive scheduling study

Yiannis Georgiou, David Glesser, Denis Trystram
Adaptive Resource and Job Management for limited power consumption
In proceedings of IPDPS-HPPAC 2015
Second iteration
Algorithm: CRAY Power Plugin

SLURMCTLD daemon main thread

- Infinite Loop with Balance Interval timer
- Get Current Power Value
- Adapt: Calculate new Powercap from current powercap and power values
- Set New Powercap Value

RPC Management Thread

CRAY Power Management Thread (Node Level)
Algorithm: Adaptive Power Management

Current Powercap = 160W, What would be new Powercap Value?

1. lower_threshold = 92% → 160 * 92 /100 = 147.2
2. upper_threshold = 96% → 160 * 96 /100 = 153.6
3. decrease_rate = 20% → 160 * (100-20) /100 = 128
4. increase_rate = 10% → 160 * (100+10)/100 = 176

<table>
<thead>
<tr>
<th>Sr No</th>
<th>Current Power</th>
<th>Threshold Calculation</th>
<th>New Powercap Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>140</td>
<td>140 &lt; 147.2</td>
<td>128</td>
</tr>
<tr>
<td>2</td>
<td>160</td>
<td>160 &gt; 153.6</td>
<td>176</td>
</tr>
<tr>
<td>3</td>
<td>150</td>
<td>between</td>
<td>150</td>
</tr>
</tbody>
</table>

Other possibilities and Complete details of algorithm explained in http://slurm.schedmd.com/power_mgmt.html
# CRAY vs DVFS Power Management in SLURM

<table>
<thead>
<tr>
<th>Behaviour</th>
<th>(2\textsuperscript{nd} iteration) CRAY</th>
<th>(1\textsuperscript{st} iteration) DVFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Calculation</td>
<td>Real power values</td>
<td>Theoretical power values</td>
</tr>
<tr>
<td>Powercapping Functionality</td>
<td>Separate Task to adapt PowerCap Frequently</td>
<td>Scheduler to adapt PowerCap</td>
</tr>
<tr>
<td>Powercap Guarantee</td>
<td>Yes real values and hardware powercap mechanism</td>
<td>No since based only on theoretical values</td>
</tr>
<tr>
<td>Power Congestion</td>
<td>When large number of application running, power congestion may be increased</td>
<td>Scheduler reduces Power Congestion by increasing Waiting time of the Jobs</td>
</tr>
<tr>
<td>Adaptive Behavior</td>
<td>Dynamic Power Allocation Unused Power shared with other jobs</td>
<td>Static power allocation Unused power not calculated/shared</td>
</tr>
<tr>
<td>Architecture</td>
<td>Only Cray platforms</td>
<td>All Linux platforms</td>
</tr>
<tr>
<td>Respecting application needs</td>
<td>System adaptation regardless application needs</td>
<td>Scheduler driven considering DVFS (min-max) given by job</td>
</tr>
</tbody>
</table>
Third iteration
New Approach based on Intel RAPL

Resolving Powercap guarantee Issue:
• **RAPL ensures hardware powercap guarantee** → Run cluster within the power budget

Resolving theoretical values inaccuracies:
• **RAPL provides a good estimate of socket power consumption** → Adapt layouts regularly to reflect real values

Resolving System Utilization Issue:
• **Adapt power based on real power consumption** (capture the application behavior) → Allow more jobs to take advantage of the unused power
RAPL Mechanism on Intel CPUs

- **RAPL** (Running Average Power Limit) are register interfaces for keeping the processors in a particular user-specified power envelope.
- Interfaces can estimate current energy usage based on a software model data collected represent energy in Joules.
- **Linux** supports an **MSR driver** and access to the register can be made through `/dev/cpu/*/msr` with priviledged read permissions.
- Access MSR features by external library **LibMSR** (developed by LLNL).
Algorithm: DVFS - RAPL Power Adaptive Scheduling (1)

1) Usual steps of DVFS power Adaptive Scheduling
Algorithm: DVFS - RAPL Power Adaptive Scheduling (2)

2) Layouts framework keep the new updated power information
3) A RAPL Power plugin sets the RAPL powercap based on the layouts power info and adapts the powercap as needed
4) The updated power kept on layouts for the new jobs
High Level Architecture

SLURMCTLD

Power/rapl → RAPL RPC Functionalities

RAPL RPC Req Message

SLURMD

LibMSR Library (RAPL Library)

Socket - 0
Power
Powercap

Socket - n
Power
Powercap

RPC call
Function call
# Power Management in SLURM

<table>
<thead>
<tr>
<th>Behaviour</th>
<th>CRAY</th>
<th>DVFS</th>
<th>RAPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Calculation</td>
<td></td>
<td></td>
<td><strong>Real</strong> power values</td>
</tr>
<tr>
<td>Powercapping Functionality</td>
<td></td>
<td></td>
<td>Both a <strong>Separate Task</strong> and the <strong>Scheduler</strong> to adapt PowerCap Frequently</td>
</tr>
<tr>
<td>Powercap Guarantee</td>
<td></td>
<td></td>
<td><strong>Yes</strong> real values and hardware powercap mechanism</td>
</tr>
<tr>
<td>Power Congestion</td>
<td></td>
<td></td>
<td>Scheduler reduces <strong>Power Congestion</strong> by increasing Waiting time of the Jobs</td>
</tr>
<tr>
<td>Adaptive Behavior</td>
<td></td>
<td></td>
<td><strong>Dynamic Power Allocation</strong> Unused Power shared with other jobs</td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
<td></td>
<td><strong>All Linux platforms</strong></td>
</tr>
<tr>
<td>Respecting application needs</td>
<td></td>
<td></td>
<td>Respect application needs based on prediction</td>
</tr>
</tbody>
</table>
Experiments
Experimental Evaluation

• Cluster Configuration
  • Intel Sandy Bridge Processor with RAPL, DVFS support
  • 3 nodes(2 sockets/node, 8 cores/socket and 65GB) cluster
  • Cluster’s powercap range is 400-1200 Watts
  • balance_interval=10, cap_watts=700, decrease_rate=20, increase_rate=10, lower_threshold=90, upper_threshold=96

• Experiment – I
  • Ensuring hardware Powercap by running HPLinpack application

• Experiment - II
  • System Utilization Experiment with lightESP benchmark workload

• Demo
Experiment I - Powercap Guarantee

• sbatch –N3 –exclusive --cpu-freq=2400000 ./HPLinpack.sh
• less HPLinpack.sh
  • #!/bin/sh
    #SBATCH -o ./hpl.out
    srun --cpu-freq=2400000 -n 48 ./xhpl
• less HPL.dat
  • 282256 Ns // More than 90% Memory was allocated
  • 1 # of NBs
  • 192 NBs
  • 0 PMAP process mapping (0=Row-,1=Column-major)
  • 1 # of process grids (P x Q)
  • 8 Ps
  • 6 Qs // P*Q = 48 tasks to run on 48 Cores
Cluster Powercap is Guaranteed by RAPL

Cluster Power consumption Comparison of two approaches to run Linpack MPI application on 3 Nodes (48 Cores) and Powercap=400W

RAPL sensors to collect Socket Power values
- RAPL Cluster Energy Consumption: 2124248.802277 Joules
- DVFS Cluster Energy Consumption: 2266496.720187 Joules
Experiment II - System Utilization

- lightESP to run ESP workload (25 jobs) with serial linpack application
- PowerPlugin=power/rapl
- Layouts=power/cpufreq
- cgroups enabled
- PowerParameters=balance_interval=10, cap_watts=700, decrease_rate=40, increase_rate=10, lower_threshold=92, upper_threshold=96, recent_job =20
DVFS based approach
RAPL based approach

RAPL based power management's System utilization for ESP workload of 25 jobs and SLURM upon 3 nodes (2 sockets/node and 8 cores/socket) cluster

System Utilization
Job Start Time Impulse

Number of Cores

Time (sec)
Algorithm Behavior: Based on RAPL Info's

- Launched **serial linpack** on one node
- RAPL Hardware information gathered in a **socket** to visualize the algorithm
Jobs Waiting Time (CDF)
Ongoing Works

• First implementation of the code is finished, working to provide the new RAPL based Power plugin for the upcoming Slurm version 17.02

• Evaluating and improving the scalability of the approach, especially that of layouts framework

• More experiments to be done with real traces and applications to validate the new code

• Active within the Energy and Power aware Job Scheduling and Resource Management team of Energy Efficiency HPC working group (https://eehpcwg.llnl.gov/) to exchange with the community and see what are the real use cases/needs of powercapping
Thank you for your attention

Questions ?
Discussion
Power Adaptive Scheduling VS Power Plugin

The Power adaptive scheduling and the Power Plugin logic (Cray) provide 2 different approaches for power capping:
- The first one is based on logical, static power values and theoretical calculations for altering scheduling to achieve a global power budget
- The former one based on the physical, real power values, and an integration to a hardware mechanism that will adapt each nodes power consumption to align to a global power budget

Disadvantages of each approach:
- Power adaptive scheduling is based on approximations so result may not be optimal either in system utilization or final power consumption
- PowerPlugin will change the node configuration of jobs without respecting their needs and depending on the executed application it will affect its turnaround time which may not be welcome

Our goal here is to take the best of the 2 worlds!