Enhancing SLURM with Energy Consumption Control and Monitoring Features

Yiannis Georgiou
email: yiannis.georgiou@bull.net
Outline

- Introduction
  - Motivations
  - Summary of the new features

Measuring Energy Consumption
- Framework for Monitoring and Accounting
- RAPL Interfaces
- Issues and Limitations

Controlling Energy Consumption
- Frequency Scaling Implementation

Ongoing and Future work
Outline

- Introduction
  - Motivations
  - Summary of the new features

- Measuring Energy Consumption
  - Framework for Monitoring and Accounting
  - RAPL Interfaces
  - Issues and Limitations

- Controlling Energy Consumption
  - Frequency Scaling Implementation

- Ongoing and Future work
Motivations for Power Management

Why do we care?
• HPC systems use important amounts of power

The Goal
• Reduce Energy Consumption in HPC systems

What can we do?
• Resource and Job Management System holds a strategic place on the HPC software stack
  – Knowledge and control over Resources and Jobs
  – Merge of System and User needs and views
Existing energy saving mechanism in SLURM

- **System** side feature
- Framework for energy saving through *unutilized* nodes
  - Administrator configurable actions (hibernate, frequency scaling, power off, etc)
  - Automatic “Wake up” when jobs arrive

**Important Need:**

- Make energy saving a **User** concern:
  1. Monitor and report node and jobs energy consumption
  2. Control over the jobs energy usage
Summary of the new features

• **New framework** to allow **per job** energy consumption and **node** power monitoring
  - Different types of energy capturing mechanisms currently 2 plugins supported:
    • from **RAPL** Interfaces
    • Through **IPMI** protocol (still under development)

• **New parameter** (for srun) to allow **CPU frequency scaling** for job execution

• **Reporting** of step's average CPU frequency and energy consumption
Outline

- Introduction
  - Motivations
  - Summary of the new features

- Measuring Energy Consumption
  - Framework for Monitoring and Accounting
  - RAPL Interfaces
  - Issues and Limitations

- Controlling Energy Consumption
  - Frequency Scaling Implementation

- Ongoing and Future work
• Framework to support the **capturing** of power/energy consumption from the computing nodes
  
  – Captures and reports the **per node** power/energy consumption (scontrol show node)
  
  – Calculates the **per step** (job) energy consumption and stores on the DB along with the other execution characteristics (sstat / sacct)
Framework: Per node power/energy monitoring

• Framework launched from slurmctld as a background task (like scheduler, ping-nodes, healthcheck, etc)
• The update is executed periodically according to `EnergyAccountingNodeFreq` parameter of slurm.conf file
  – It makes a call to update_energy on every computing node through SLURM RPC
  – This function collects energy/power measures (RAPL, etc)
• The energy/power measures may be reported to the user as new information with `scontrol show node`

```
#slurm.conf
...
EnergyAccountingType=energy_accounting/rapl
EnergyAccountingNodeFreq=30
```
Architecture of the Framework - Node Monitoring

Energy Accounting Framework - Node Monitoring

- controller_background
- update_nodes_energy_data
  with period EnergyAccountingNodeFreq

slurmd

SLURM_RPC

slurmd

update_node_energy
  getjoules_rapl

slurmd

slurmd

slurmd

Server

Computing Nodes
Usage Power/energy Node Monitoring

[root@berlin19 georgioy]# scontrol show node
NodeName=berlin47 Arch=x86_64 CoresPerSocket=8
    CPUAlloc=0 CPUErr=0 CPUTot=32 Features=(null)
    Gres=(null)
    NodeAddr=berlin47 NodeHostName=berlin47
    OS=Linux RealMemory=1 Sockets=2 Boards=1
    State=IDLE ThreadsPerCore=2 TmpDisk=0 Weight=1
    BootTime=2012-09-28T11:04:32 SlurmdStartTime=2012-10-08T10:59:45
    CurrentWatts=33 LowestJoules=106789 ConsumedJoules=1652356
Measuring power and energy on node

• Interested in accurate internal node mechanisms that may give us power and energy measurements of the entire node or independent components of it
• Two solutions seemed adjusted
  – IPMI protocol collecting info from the BMC of the node
  – RAPL interfaces (Intel Sandy Bridge processors)
• IPMI may have overhead when contacting the BMC
• RAPL has less overhead higher accuracy and was simpler to support
• RAPL (Running Average Power Limit) are particular interfaces on Intel Sandy Bridge processors implemented to provide a mechanism for keeping the processors in a particular user-specified power envelope.

• RAPL Interfaces have **power capping** capabilities (not exploited yet through our developments)

• RAPL Interfaces can estimate current energy usage based on a software model driven by hardware performance counters, temperature and leakage models
• **PP0\_ENERGY**: energy used by “power plane 0” which includes all cores and caches of a socket

• **PP1\_ENERGY**: energy used by the ”uncores” (this may include on-chip Intel GPU)

• **PACKAGE\_ENERGY**: total energy consumed by entire package (PP0 + PP1)

• **DRAM\_ENERGY**: energy drawn by the memory controller inside the processor chip (the actual power fed into the main memory DIMMs is not included in the current measurement)
Intel indicates that power/energy readings are updated every millisecond (1kHz).

Rotem et al. experiments show good accuracy [1].

SLURM-RAPL Interfacing

- Access to RAPL data requires reading a CPU MSR register. This requires OS support.
- Linux supports an 'MSR' driver and access to the register can be made through `/dev/cpu/*/msr` with privileged read permissions.
- Inspired by PAPI support to RAPL Interfaces [3]

Advantages:
• No overhead during capturing power/energy data
• Measuring in energy units simplifies the process in calculating energy consumption per step/job basis
  – 2 values are sufficient (start/end of the job), no need of collecting big number of instant watts (IPMI case)
• Accuracy of Measurements
Disadvantages:
• Only processor and part of memory (DRAM) is supported (no motherboard, disk, external GPU, etc)
• Per socket granularity (not per core)
Framework: Per step energy consumption

• During job/step/task monitoring the energy accounting plugin is called by jobacct_gather
• Results aggregated per node considering the measurement of all packages (sockets + DRAM)
  – Initially the framework will give accurate results for **exclusive allocations** (whole nodes usage)
• One consideration in the beginning of the job and one in the end will be enough to have an accurate result
• Change in jobacct_gather to be certain that in the beginning and in the end of the job we collect results from RAPL Interfaces
Architecture of the RAPL Plugin

Energy Accounting Framework - Job Monitoring

**slurmctld**
- start job execution
- finish job execution
- aggregating results from nodes per step and store results in DB

**slurmd**
- start tasks monitoring
  - jobacct_gather with period jobacct_gatherFreq
  - update_task_getjoules_rapl

Client → slurmctld → slurmd → Database → Computing Nodes

srin, sailoc, sbatch
RAPL Plugin Usage for Job Energy Accounting

[root@berlin19 bin]# srun --resv-ports -N2 -n64 ./cg.C.64&
[1] 18060
[root@berlin19 bin]# sstat -j 58
    JobID   JobName ...   Elapsed ...   ConsumedEnergy
      58   cg.C.64 ...   00:04.000   1886
[root@berlin19 bin]# sacct -j 58
    JobID   JobName ...   Elapsed ...   ConsumedEnergy
      58   cg.C.64 ...   00:00:14   5033
Issues and Limitations

• Energy consumption results may be misleading in case of node sharing
  – Through RAPL possible to go up to socket granularity (not supported yet)

• RAPL does not give the energy consumption of the complete node but only that of CPU and DRAM (motherboard, network, GPU cannot be reported yet through these sensors)
  – GPU energy can be captured through NVIDIA Management Library (NVML)
Outline

- Introduction
  - Motivations
  - Summary of the new features
- Measuring Energy Consumption
  - Framework for Monitoring and Accounting
  - RAPL Interfaces
  - Issues and Limitations
- Controlling Energy Consumption
  - Frequency Scaling Implementation
- Ongoing and Future work
Controlling Energy Consumption

• Measuring Energy Consumption and reporting it per step/job level is an important step, but users should have means to influence it as well.

• How can we do it. Various ways:
  – CPU Frequency Scaling (Static or Dynamic)
  – CPU P/C States modifications (Static or Dynamic)
  – Activation/desactivation of CPU Turbo Mode
  – Memory?
Static Frequency Scaling with SLURM jobs

• Introduced **-cpu-freq** parameter in srun
• **Static** since it may not be changed during the execution (user does not usually have root access on the computing nodes)
• The user may ask either a particular value in kilohertz or use low/medium/high and the request will match the closest possible numerical value
Frequency Scaling Design/Configuration Details

- Implemented as a **request** that the job step initiated by this `srun` be run at the requested frequency (if possible), on the cpus selected for the step. Or else the job will be executed nonetheless without setting frequency.
- Tasks confinement to those cpus. If task confinement (i.e., `TaskPlugin=task/affinity` or `TaskPlugin=task/cgroup` with the "ConstrainCores" option) is not configured, this parameter will be ignored.
- Implemented through manipulation of the `/sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq` and governors drivers.
Calculating Step's Average Frequency

- Capturing tasks' CPU association and collecting the cpu time spending on a particular frequency.
- Calculate a weighted average of the number of cycles used time spent on a particular frequency for each CPU.
- Sum up the results for each CPU and divide with the total number of used cores to calculate the average frequency per step.
- Values in Average Frequency will be influenced by the CPU time utilization (in case of no CPU usage the average frequency of the CPUS among all of them will be returned).
Frequency Scaling Usage

[root@berlin47 georgioy]# cat
/sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
1200000
[root@berlin19 bin]# srun --cpu-freq=2700000 --resv-ports -N2 -n64 ./cg.C.64&
[1] 18060
[root@berlin47 georgioy]# cat
/sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
2700000
[root@berlin19 bin]# sacct -j 60
 JobID   JobName ... Elapsed ... ConsumedEnergy
    58   cg.C.64 ... 00:00:14    3065
[root@berlin19 bin]# sacct -j 58 -format=jobid,elapsed,aveCPUFreq,consumedenergy
 JobID    Elapsed  AveCPUFreq  ConsumedEnergy
------------ ---------- ---------- --------------
  66       00:00:49    2640340          19668
[root@berlin47 georgioy]# cat
/sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
1200000
Summary

The described features will appear on the stable release of 2.5 in November. They are currently under testing/debugging in the development track: 2.5.0-pre3 release - branch energy of the Schedmd git:

• **Monitoring** Energy consumption
  – Energy accounting framework
  – RAPL plugin for energy monitoring and job accounting

• **Controling** Energy consumption
  – Static CPU Frequency Scaling for jobs
### Usage Examples

- The new features may be used for job/application profiling in order to find the best trade-off between requested CPU Frequency and Execution Time which may defer depending on the codes of each job/application.
- Usage Examples with NAS Benchmarks execution (sp.C.64 and bt.C.64):

<table>
<thead>
<tr>
<th>AverageCPU Frequency</th>
<th>Elapsed Time</th>
<th>Consumed Energy(J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200000</td>
<td>00:01:35</td>
<td>19366</td>
</tr>
<tr>
<td>1396460</td>
<td>00:01:23</td>
<td>19018</td>
</tr>
<tr>
<td>1780477</td>
<td>00:01:09</td>
<td>19353</td>
</tr>
<tr>
<td>1996186</td>
<td>00:01:05</td>
<td>19817</td>
</tr>
<tr>
<td>2200000</td>
<td>00:01:02</td>
<td>20494</td>
</tr>
<tr>
<td>2362500</td>
<td>00:00:59</td>
<td>21408</td>
</tr>
<tr>
<td>2653125</td>
<td>00:00:56</td>
<td>23125</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AverageCPU Frequency</th>
<th>Elapsed Time</th>
<th>Consumed Energy(J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200000</td>
<td>00:01:35</td>
<td>17411</td>
</tr>
<tr>
<td>1386008</td>
<td>00:01:22</td>
<td>16990</td>
</tr>
<tr>
<td>1586343</td>
<td>00:01:64</td>
<td>16961</td>
</tr>
<tr>
<td>1783580</td>
<td>00:01:07</td>
<td>17119</td>
</tr>
<tr>
<td>1996875</td>
<td>00:00:58</td>
<td>17993</td>
</tr>
<tr>
<td>2231250</td>
<td>00:00:53</td>
<td>18531</td>
</tr>
<tr>
<td>2512500</td>
<td>00:00:49</td>
<td>19626</td>
</tr>
</tbody>
</table>
Introduction
- Motivations
- Summary of the new features

Measuring Energy Consumption
- Framework for Monitoring and Accounting
- RAPL Interfaces
- Issues and Limitations

Controlling Energy Consumption
- Frequency Scaling Implementation

Ongoing and Future work
Current Ongoing Works

Energy accounting framework and monitoring

- Support IPMI protocol with a new plugin: Energyaccounting/ipmi
  - Issues with kipmi0 thread overhead → seems to be resolved lately
  - More complex architecture because of the measurements in Watt → it can be addressed with:
    • averaging energy window
    • Or by keeping values in files or DB for later usage (reporting, profiling)
  - Evaluating which project would be better to integrate in SLURM: Ipmitool or freeipmi

- RAPL Interfaces plugin: Extend the current plugin to support the energy consumption upto socket granularity (supported by RAPL but not from the current SLURM-RAPL integration)
  - Jobs that allocate cores from different sockets on the same node would have different and accurate energy consumption
Energy aware scheduling

- Take into account power thresholds and job energy needs
  - The power thresholds may be set in the level of cluster, account and or user
  - The energy needs could be either defined in the beginning of the job (new srun/sbatch/salloc parameter), or estimated automatically from the jobs previous runs
- Take into account exterior environmental measurements (not only computing nodes energy but switches energy)
- Take into account node temperatures and placing jobs on cold spots rather than hot spots of the cluster
- External sensors plugin to collect energy data from external source
  - Less overhead
  - More information (energy of switches temperature of room, other components)
External Sensors Plugin Architecture

External Sensors Monitoring Framework

- collect_components_energy_info
- controller_background
- update_components_energy_data with period ExternalSensorsFreq

External Sensors Centralised Server

Server

Computing Nodes
People Involved

Research and Development:
- Dan Rusak (Bull, USA)
- Don Albert (Bull, USA)
- Martin Perry (Bull, USA)
- Yiannis Georgiou (Bull, France)
- Xavier Bru (Bull, France)

Design and Integration:
- Nancy Kritkausky (Bull, France)
- Moe Jette (SchedMD, USA)
- Danny Auble (SchedMD, USA)

Research and Design Ideas:
- Matthieu Hautreux (CEA, France)
- Francis Belot (CEA, France)