POWER-AWARE JOB SCHEDULING
Maximizing Data Center Performance Under Strict Power Budget

Osman Sarood, Akhil Langer, Abhishek Gupta, Laxmikant Kale

Parallel Programming Laboratory
Department of Computer Science
University of Illinois at Urbana-Champaign

29th April 2014
Major Challenges to Achieve Exascale

- Energy and Power Challenge
- Memory and Storage Challenge
- Concurrency and Locality Challenge
- Resiliency Challenge

Major Challenges to Achieve Exascale

Power consumption for Top500

Data Center Power

How is data center power need calculated?
- using Thermal Design Power (TDP) of nodes

However, TDP is hardly reached!!

Solution
- constrain power consumption of nodes
- Overprovisioning - Use more nodes than conventional data center for the same power budget
Distribution of Node Power Consumption

Power distribution for BG/Q processor on Mira
- 76% by CPU/Memory
- No good mechanism for controlling other power domains

Pie Chart: Sean Wallace, Measuring Power Consumption on IBM Blue Gene/Q
Constraining CPU/Memory Power

**Intel Sandy Bridge**

- Running Average Power Limit (RAPL) library
  - measure and set CPU/memory power
Application Performance with Power

- Application performance does not improve proportionately with increase in power cap
- Better is to run on larger number of nodes each capped at lower power level

Performance of LULESH at different configurations

Configuration
\((n \times p_c, p_m)\)

- \(p_c\): CPU power cap
- \(P_m\): Memory power cap
Problem Statement

Maximizing Data Center Performance Under Strict Power Budget

Data center capabilities and job features
- Power capping ability
- Overprovisioning
- Moldability (Optional)
- Malleability (Optional)
  - Charm++
  - Dynamic MPI
Power Aware Resource Manager (PARM)

**Diagram:**

- **JOB PROFILER**
  - PASS MODEL

- **SCHEDULER**

- **JOB QUEUE**
  - SHRINK/EXPAND JOBS
  - APPLY POWER CAPS

- **EXECUTION FRAMEWORK**

**Flow:**

- TRIGGERS
  - JOB ARRIVAL
  - JOB TERMINATION
JOB PROFILER

- Measure job performance at various scales and cpu power caps

- Power Aware Strong Scaling (PASS) Model
  - Predict job performance at any \((n, p)\)
Power Aware Strong Scaling (PASS) Model

**Time vs Scale**

Downey’s strong scaling

\[ t = F(n, A, \sigma) \]

- \( n \): number of nodes
- \( A \): Average Parallelism
- \( \sigma \): duration of parallelism \( A \)

**Time vs Frequency**

\[ t(f) = \begin{cases} \frac{W_{cpu}}{f} + T_{mem}, & \text{for } f < f_h \\ \frac{T_h}{f}, & \text{for } f \geq f_h \end{cases} \]

- \( W_{cpu} \): CPU work
- \( T_{mem} \): memory work
- \( T_h \): minimum exec time

**Frequency vs Power**

\[ p = p_{core} + \sum_{i=1}^{3} g_i L_i + g_m M + p_{base} \]

- \( p_{core} \): core power
- \( g_i \): cost level I cache access
- \( L_i \): level I accesses
- \( g_m \): cost of mem access
- \( M \): mem accesses
- \( p_{base} \): idle power

Time as a function of power and number of nodes
Power Aware Resource Manager (PARM)

- JOB PROFILER
  - PASS MODEL
- SCHEDULER
- JOB QUEUE
- EXECUTION FRAMEWORK
  - SHRINK/EXPAND JOBS
  - APPLY POWER CAPS
- TRIGGERS
- JOB ARRIVAL
- JOB TERMINATION

3/4/15
Scheduler: Integer Linear Program Formulation

Objective Function

\[
\sum_{j \in J} \sum_{n \in N_j} \sum_{p \in P_j} w_j * s_{j,n,p} * x_{j,n,p}
\]

Select One Resource Combination Per Job

\[
\sum_{n \in N_j} \sum_{p \in P_j} x_{j,n,p} \leq 1 \quad \forall j \in I
\]

\[
\sum_{n \in N_j} \sum_{p \in P_j} x_{j,n,p} = 1 \quad \forall j \in I
\]

Bounding total nodes

\[
\sum_{j \in J} \sum_{p \in P_j} \sum_{n \in N_j} nx_{j,n,p} \leq N
\]

Bounding power consumption

\[
\sum_{j \in J} \sum_{n \in N_j} \sum_{p \in P_j} (n * (p + W_{base}))x_{j,n,p} \leq W_{max}
\]

Disable Malleability (Optional)

\[
\sum_{n \in N_j} \sum_{p \in P_j} nx_{j,n,p} = n_j \quad \forall j \in I
\]
Scheduler: Objective Function

- Maximizing throughput makes ILP optimization infeasible
- Maximize sum of power-aware speedup of selected jobs:

\[ s_{j,n,p} = \frac{t_{j,\min(N_j),\min(P_j)}}{t_{j,n,p}} \]
Power Aware Resource Manager (PARM)

- **JOB PROFILER**  
  PASS MODEL

- **SCHEDULER**

- **EXECUTION FRAMEWORK**
  - SHRINK/EXPAND JOBS
  - APPLY POWER CAPS

- **JOB QUEUE**

- **TRIGGERS**

- **JOB ARRIVAL**

- **JOB TERMINATION**
Experimental Setup

Applications
- Memory-intensive
  - Jacobi and Wave2D
- Computation-intensive
  - LeanMD
- Mixed
  - AMR and Lulesh

Job Dataset
- \(\beta\) corresponds to CPU sensitivity
- SetL: Mix of apps with average \(\beta=0.1\)
- SetH: Mix of apps with average \(\beta=0.27\)

Testbed
- 38-node Intel Sandy Bridge
- 6 physical cores, 16GB RAM
- Power capping using RAPL
- CPU power cap range [25-95]W

Power Budget
- CPU power levels={30, 32, 34, 39, 45, 55}W
- Node power consumption= 116W
- Power Budget = 3000W
- #nodes in traditional data center = 28
Estimating Performance using PASS

Model Parameters

<table>
<thead>
<tr>
<th>Application</th>
<th>a</th>
<th>b</th>
<th>pi</th>
<th>ph</th>
<th>β</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeanMD</td>
<td>1.65</td>
<td>7.74</td>
<td>30</td>
<td>54</td>
<td>0.40</td>
</tr>
<tr>
<td>AMR</td>
<td>2.45</td>
<td>6.57</td>
<td>32</td>
<td>54</td>
<td>0.33</td>
</tr>
<tr>
<td>Lulesh</td>
<td>2.63</td>
<td>8.36</td>
<td>32</td>
<td>54</td>
<td>0.30</td>
</tr>
<tr>
<td>Wave2D</td>
<td>3.00</td>
<td>10.23</td>
<td>32</td>
<td>42</td>
<td>0.16</td>
</tr>
<tr>
<td>Jacobi2D</td>
<td>1.54</td>
<td>10.13</td>
<td>32</td>
<td>37</td>
<td>0.08</td>
</tr>
</tbody>
</table>

The diagram shows the modeled (lines) and observed (markers) power-aware speedups for all applications with varying CPU power cap at 20 nodes. LeanMD has the highest power-aware speedup whereas Jacobi2D has the lowest. The actual power consumption of CPU and memory is capped in the range of 30W for LeanMD and 32W for all applications. The parameter $p_h$ (CPU base power) varies but is generally in the range of 30-54W. Higher value of $p_h$ means higher sensitivity to CPU power.
nodes to assign to each job, and 2) which nodes to assign
Hence, we consider boot times in our model.
would require dynamic process spawning when expanding.
residual processes after shrink. Hence, for more practical and
have recently proposed a new approach which eliminates the
where many jobs run simultaneously. Charm++ researchers
for small clusters as it starts processes on as many nodes
(§
inferior to both wSE and noSE, we concentrate on wSE and
baseline scheduling policy. Since noMM version of PARM was
shrinking and expanding jobs. We then give the experimental
gives us information about SLURM's scheduling decisions
lator [36] which is a wrapper around SLURM. This simulator
scheduling on very large machine, we use the SLURM simu-
Since it was practically infeasible for us to do actual job
real cluster, we now analyze its benefit on very large machines.
moldability and malleability features.
SetH (Figure 4) that the benefits of using PARM as compared to
flexibility to increase the number of nodes gives PARM higher
nodes at the cost of decreasing the CPU power are smaller. The
§
After experimentally showing the benefits of PARM on a
moldability features of Charm++.
Large Scale Projections

- SLURM simulator vs PARM simulator
- Modeling cost of shrinking and expansion of jobs
  - Boot times
  
  \[
  t_b \text{(in seconds)} = (n_t - n_f) \times 0.01904 + 72.73
  \]

- Communication cost for data transfer
  
  \[
  t_c = \frac{\left( \frac{m_j}{n_f} - \frac{m_j}{n_t} \right) \times n_f}{2 \times b \times n_f^{\frac{3}{2}}}
  \]

- Total cost
  
  \[
  t_{se} = t_c + t_b
  \]
Large Scale Projections
Experimental Setup

- **Job Datasets**
  - Intrepid job traces
  - 3 subsets: Set 1, Set 2, Set 3
  - 1000 jobs

- **Application Characteristics**
  - Model parameters chosen randomly from range defined by computationally and memory intensive apps

- **Node Range for Moldable/Malleable jobs**
  - min nodes = \( \theta \cdot \max(N) \)
  - \( \theta \in [0.2, 0.6] \)

- **Power Budget**
  - 40,960 nodes -> 4.75MW
  - CPU power levels
    - \( \{30, 33, 36, 44, 50, 60\} \)W
Large Scale Projections

Performance

Description
- **baseline**: SLURM scheduling
- **noSE**: with Moldability but no Malleability
- **wSE**: with Moldability and Malleability

- **Arrival times multiplied by γ**
- **Gives diversity in job arrival rates**

5.2X speedup with wSE!
Comparison with Naïve Overprovisioning

<table>
<thead>
<tr>
<th>CPU power cap (W)</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup of wSE over naive</td>
<td>4.32</td>
<td>1.86</td>
<td>2.33</td>
<td>5.25</td>
</tr>
<tr>
<td>Num. of nodes in naive strategy</td>
<td>55248</td>
<td>49493</td>
<td>44824</td>
<td>40960</td>
</tr>
</tbody>
</table>
Tradeoff between Throughput and Job Fairness

Objective function multiplier: $\omega^\alpha$
Conclusion

- Significant improvement in throughputs
  - Power-aware characteristics (PASS model)
  - CPU power capping
  - Overprovisioning
- Sophisticated ILP scheduling methodology useful for resource assignment
- Adaptive runtime system further increases benefits by allowing malleability
- Non-malleable jobs also benefit

Future Work

- Enable/disable caches
- Thermal constraints
  - To improve system reliability and improve cooling costs
- Rich support for user priorities
THANK YOU!

POWER-AWARE JOB SCHEDULING
Maximizing Data Center Performance Under Strict Power Budget

Osman Sarood, Akhil Langer, Abhishek Gupta, Laxmikant Kale

Parallel Programming Laboratory
Department of Computer Science
University of Illinois at Urbana-Champaign

29th April 2014