Integer Programming Based Heterogeneous CPU-GPU Clusters

Seren Soner, Can Özturan
Boğaziçi University
10/10/2012
Outline

- Motivation
- Challenges
- Co-Allocation Approach
- Integer Programming Based Scheduler
  - Formulation
  - Implementation details
  - ESP benchmark
  - Results
- Auction Based Scheduler
  - Formulation
  - Bid generation
Motivation

• Job schedulers schedule jobs in a sequential fashion.
• Not considering other jobs in the queue may cause unnecessary waiting.

• Instead, consider multiple jobs at once, and try to allocate them in the optimal manner.
Co-allocation Based Approach

- The problem of allocating multiple resources (whether of the same type or different types) simultaneously to jobs is known as co-allocation.
- This problem also appears as auction problem in the e-commerce area where auctioneers submit bids for purchasing a bundle of items (of the same type or different types).
- Algorithms developed in the literature for auctions can be made use of in job scheduling also.
- Repeatedly take a collection of jobs from the front of the job queue (i.e. a window of jobs) and solve co-allocation problem.
Challenges

- **Scalability**: Massive number of resources and large number of jobs with different resource requirements and priorities (i.e. massive number of variables)
- **GPU awareness**: GPU resources are appearing on supercomputers in different configurations.
- **Topology awareness**: Mapping of an application to the resources in close vicinity on the topology
An Illustrative Example

- **J₁**: 4096 cores, -n 4096
- **J₂**: 2048 cores, 512 nodes, 2 GPUs/node, -N 512 -n 2048 -gres=gpu:2
- **J₃**: 2048 cores, 512 nodes, 2 GPUs/node, -N 512 -n 2048 -gres=gpu:2

Priority ordered queue

idle system, 1024 nodes (8 cores & 2 GPUs/node)

node 1  node 1024
SLURM/Backfill allocation

- **J₁**: 4096 cores
  - nodes 1-512, 8 cores/node
- **J₂**: 2048 cores, 512 nodes
  - 2 GPUs/node
  - nodes 513-1024, 4 cores/node
- **J₃**: waiting in queue

- GPUs in nodes 1-512 are unutilized.
- 4 cores/node in nodes 513-1024 are unutilized.
### IPSched allocation

<table>
<thead>
<tr>
<th>Job</th>
<th>Nodes</th>
<th>Cores</th>
<th>GPUs per Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>J&lt;sub&gt;1&lt;/sub&gt;</td>
<td>1-1024</td>
<td>4096</td>
<td>2 per node</td>
</tr>
<tr>
<td>J&lt;sub&gt;2&lt;/sub&gt;</td>
<td>1-512</td>
<td>2048</td>
<td>4 per node, 2 GPUs per node</td>
</tr>
<tr>
<td>J&lt;sub&gt;3&lt;/sub&gt;</td>
<td>513-1024</td>
<td>2048</td>
<td>4 per node, 2 GPUs per node</td>
</tr>
</tbody>
</table>

- **J<sub>1</sub>** → nodes 1-1024, 4 cores/node
- **J<sub>2</sub>** → nodes 1-512, 4 cores/node, 2 GPUs/node
- **J<sub>3</sub>** → nodes 513-1024, 4 cores/node, 2 GPUs/node

- All resources in all nodes are utilized.
IP formulation

\[
\begin{align*}
\max \sum p_j (s_j - c_j) \\
\sum_j^M x_{ij} &\leq R_i \quad \forall i \\
\sum_i^N x_{ij} & = r_j s_j \quad \forall j \\
\sum_j^M g_j t_{ij} & \leq G_i \quad \forall i \\
c_j &= \frac{\sum_i^N t_{ij}}{2N} \quad \forall j \\
N_{\min,j} &\leq 2N c_j \leq N_{\max,j} \quad \forall j \\
t_{ij} &= \begin{cases} 
1, & x_{ij} > 0 \\
0, & x_{ij} = 0 
\end{cases} \quad \forall i, j
\end{align*}
\]
Assumptions

- No preemption
- No topology
- Memory is not important

## Problem Size

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Number of variables</th>
<th>Equation no</th>
<th>Number of constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_j$</td>
<td>$</td>
<td>N</td>
<td>$</td>
</tr>
<tr>
<td>$c_j$</td>
<td>$</td>
<td>N</td>
<td>$</td>
</tr>
<tr>
<td>$x_{ij}$</td>
<td>$</td>
<td>N</td>
<td>\times</td>
</tr>
<tr>
<td>$t_{ij}$</td>
<td>$</td>
<td>N</td>
<td>\times</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>$2 \times</td>
<td>N</td>
<td>\times (1 +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>$2 \times</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td>$2 \times (</td>
</tr>
</tbody>
</table>
Implementation Details

- Plug-in runs on `slurmctld`
- The scheduler runs at most every 4 seconds
- Collects information about nodes and jobs at each step
- Solve IP problem using CPLEX [1] in pre-determined time (3 seconds)
- Allocate jobs
- Create and solve the problem again

Implementation Details (cont’d)

- Scheduler at the SLURM core code has been removed, we want IPSched to schedule all the jobs.
- A new select plugin has been designed, similar to cons_res. Schedules the jobs to the resources that IPSched requests.
- Minor addition in order to retrieve the number of available GPUs at nodes.
Algorithm

Create job window, size <= MAX_JOB_COUNT
From each job in window, collect
  a. priority \( (p_j) \)
  b. CPU request \( (r_j) \)
  c. GPU request \( (g_j) \)
  d. Node request \( (N_{j,min} - N_{j,max}) \)
From each node, collect
  a. number of available CPU’s
  b. number of available GPU’s,
Form the IP problem
Solve the IP problem and get \( s_j \) and \( x_{ij} \) values.
For jobs with \( s_j = 1 \), set job’s process layout matrix and start the job by:
  a. For each node \( i \), assign processors on that node according to \( x_{ij} \)
  b. Start the job, no more node selection algorithm is necessary.
ESP benchmark [4]

- Consists of various job sizes
- 230 jobs in one set
- Execution times fixed
- Each job duplicated
  - One copy requests CPU only
  - One copy requests CPU + 2 GPUs/node

Emulation settings

- Real time emulation
- 1024 nodes, each with 8 cores and 2 GPUs
- IP solution time is 4 seconds
- Up to 200 jobs in window
- Priority settings
  - Multifactor (age factor = size factor)
  - Basic
- Backfill and IPSCHED comparison
- Ran this on a machine with 9 nodes (2x Intel X5670, 48 GB memory). One node dedicated to slurmctld, all other nodes running 128 slurmd.
Why not SLURM Simulator?

- Alejandro Lucero has coded a SLURM simulator [3].
- Works well for comparing different fairshare, priority decisions etc.

- Would not be useful for our simulation, since the governing issue for our simulation is not the job execution itself, but the solution of the IP problem.

# IPSCHED Results

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Waiting Time (hr) (mean ± std)</th>
<th>Slowdown Ratio (mean ± std)</th>
<th>Utilization (mean)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backfill / Basic</td>
<td>1.60 ± 0.836</td>
<td>18.11 ± 25.49</td>
<td>0.90</td>
</tr>
<tr>
<td>IPSCHED / Basic</td>
<td>0.77 ± 1.257</td>
<td>9.95 ± 18.87</td>
<td>0.92</td>
</tr>
<tr>
<td>Backfill / Multifactor</td>
<td>2.42 ± 1.758</td>
<td>22.75 ± 22.02</td>
<td>0.89</td>
</tr>
<tr>
<td>IPSCHED / Multifactor</td>
<td>0.88 ± 1.223</td>
<td>10.75 ± 18.20</td>
<td>0.94</td>
</tr>
</tbody>
</table>
Topology problems

- IPSched was not good enough in terms of topology.
- The allocation showed that there was room for improvement in SLURM’s approach, but did not consider topology at all.
- Came up with another approach, a more complex one.

- Please note that AUCSCHED is still under progress, formulation and implementation details may be subject to change.
AUCSCHED Formulation

\( J \): set of jobs that are in the window: \( J = \{j_1, \ldots, j_{|J|}\} \),
\( P_j \): priority of job \( j \),
\( N \): set of nodes: \( N = \{n_1, \ldots, n_{|N|}\} \),
\( C \): set of bid classes: \( C = \{c_1, \ldots, c_{|C|}\} \),
\( N_c \): set of nodes making up a class \( c \),
\( K \): union of all \( C_jn \) sets, i.e. \( K = \bigcup_{j \in J, c \in B_j, n \in N_c} C_jn \).
\( B \): set of all bids, \( B = \{b_1, \ldots, b_{|B|}\} \),
\( B_j \): set of bid classes on which job \( j \) bids, i.e. \( B_j \subseteq C \),
\( C_jn \): the set \( \{c \in C \mid c \in B_j \text{ and } n \in N_c\} \)
\( A_{\text{cpu}}^{n} \): number of available CPU cores on node \( n \),
\( A_{\text{gpu}}^{n} \): number of available GPUs on node \( n \),
\( R_{\text{cpu}}^{j} \): number of cores requested by job \( j \),
\( R_{\text{gpu}}^{j} \): number of gpus per node requested by job \( j \),
\( R_{\text{node}}^{j} \): number of nodes requested by job \( j \),
\( R_{\text{cpn}}^{j} \): number of cores per node requested by job \( j \). If not specified, this parameter gets a value of 0.
\( F_{jc} \): preference value of bid \( c \) of job \( j \), ranging between 0 and 1. All bids have a preference value, closer to 1 if they are allocated better, 0 if they are fragmentation is high.
\( \alpha \): reciprocal of minimum priority difference between jobs in \( J \)

\( b_{jc} \): binary variable for a bid on class \( c \) of job \( j \),
\( u_{jn} \): binary variable indicating whether node \( n \) is allocated to job \( j \)
\( r_{jn} \): non-negative integer variable giving the remaining number of cores allocated to job \( j \) on node \( n \) (i.e. at most one less than the total number allocated on a node).
AUCSCHED Formulation

\begin{equation}
\text{Maximize} \quad \sum_{j \in J} \sum_{c \in B_j} (P_j + \alpha \cdot F_{jc}) \cdot b_{jc} \quad (1)
\end{equation}

subject to constraints:

\begin{equation}
\sum_{c \in B_j} b_{jc} \leq 1 \quad \text{for each } j \in J \quad (2)
\end{equation}

\begin{equation}
\sum_{n \in N_c} u_{jn} = b_{jc} \cdot R_j^\text{node} \\
\quad \text{for each } (j, c) \in J \times C \text{ s.t. } c \in B_j \quad (3)
\end{equation}

\begin{equation}
\sum_{n \in N_c} u_{jn} + r_{jn} = R_j^\text{cpu} \cdot \sum_{c \in B_j} b_{jc} \quad \text{for each } j \in J \quad (4)
\end{equation}

\begin{equation}
\sum_{j \in J} u_{jn} + r_{jn} \leq A_n^\text{cpu} \quad \text{for each } n \in N \quad (5)
\end{equation}

\begin{equation}
\sum_{j \in J} u_{jn} \cdot R_j^\text{gpu} \leq A_n^\text{gpu} \quad \text{for each } n \in N \quad (6)
\end{equation}

\begin{equation}
0 \leq r_{jn} \leq u_{jn} \cdot min(A^\text{cpu}_n - 1, R^\text{cpu}_j - 1) \\
\quad \text{for each } (j, n) \in J \times N \quad (7)
\end{equation}

\begin{equation}
\begin{aligned}
& u_{jn} + r_{jn} = \sum_{c \in C_{jn}} b_{jc} \cdot R_j^\text{epn} \\
& \text{for each } (j, n) \in J \times N \text{ s.t. } R_j^\text{epn} > 0 \text{ and } C_{jn} \neq \emptyset
\end{aligned} \quad (8)
\end{equation}
Problem Size

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Number of variables</th>
<th>Equation no</th>
<th>Number of constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_{jc}$</td>
<td>$</td>
<td>B</td>
<td>$</td>
</tr>
<tr>
<td>$u_{jn}$</td>
<td>$</td>
<td>K</td>
<td>$</td>
</tr>
<tr>
<td>$r_{jn}$</td>
<td>$</td>
<td>K</td>
<td>$</td>
</tr>
<tr>
<td>Total</td>
<td>$2</td>
<td>K</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>$</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>2$</td>
<td>N</td>
</tr>
</tbody>
</table>
Bid Generation

- Choose «nodeset»s so that
  - They fit the job’s needs
  - They are «less fragmented»
  - Give different preference values according to fragmentation
- This time the IP variables are not nodes themselves, but the bids – therefore nodesets.
- While generating the bids, all types of constraints can be checked (nodelist, exclude nodes, generic resources, licenses)
Bid Generation

- Choose bids so they do not overlap (as distinct as possible)
- Generate up to $MAXBIDPERJOB$ bids for each job
- Generate up to $MAXBID$ in total
AUCSCHED results

- Utilization in PWA too low
- We created our own workload – instead of only 14 type of jobs, job size, request, execution times are random (similar to a real workload).
- Work is still in progress, however preliminary results show that we can reach better utilization values compared to SLURM/Backfilling.
- Fragmentation problem is decreased, but is still around 10-20% higher than that of SLURM.
Conclusions & Future work

- Shows better results in terms of metrics
- Not applicable to everybody due to usage of CPLEX (not free for commercial licenses)

- Formulate a heuristic working in polynomial time
- Implement other constraints to bid generation (currently only gres is implemented)
Acknowledgments

• PRACE 1IP project
  grant agreement RI-261557

• PRACE 2IP project
  grant agreement FP7-283493

• Matthieu Heatroux for discussions
• Alejandro Lucero for help with the simulator