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.

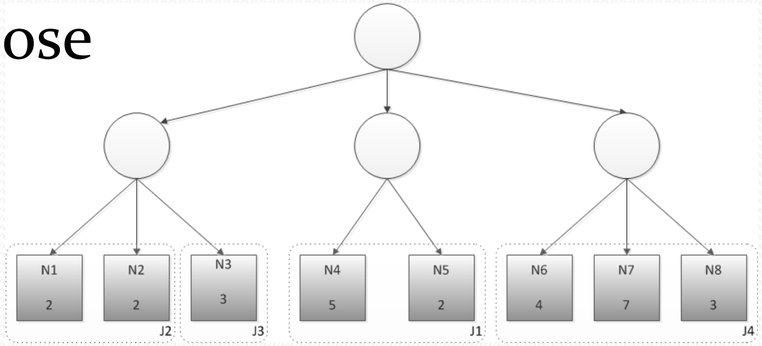
- Instead, consider multiple jobs at once, and try to allocate them in the optimal manner.

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

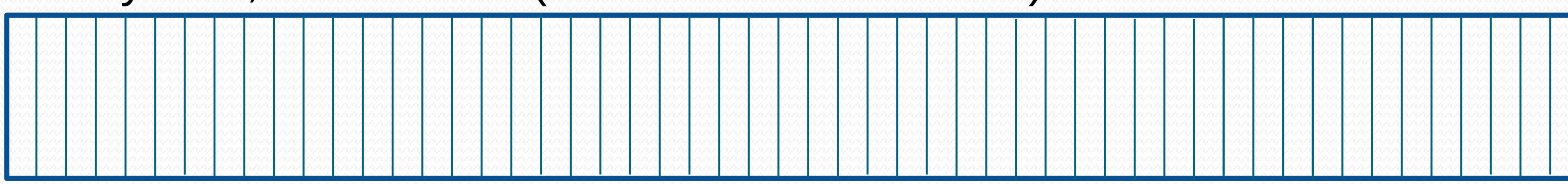


An Illustrative Example

J_1	J_2	J_3
4096 cores	2048 cores, 512 nodes	2048 cores, 512 nodes
	2 GPUs/node	2 GPUs/node
-n 4096	-N 512 –n 2048 –gres=gpu:2	-N 512 –n 2048 –gres=gpu:2

Priority ordered queue

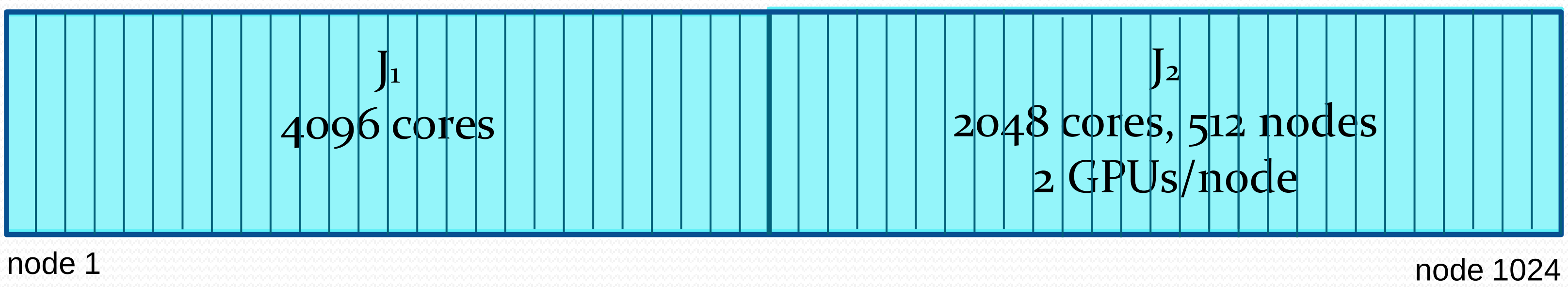
idle system, 1024 nodes (8 cores & 2GPUs/node)



node 1

node 1024

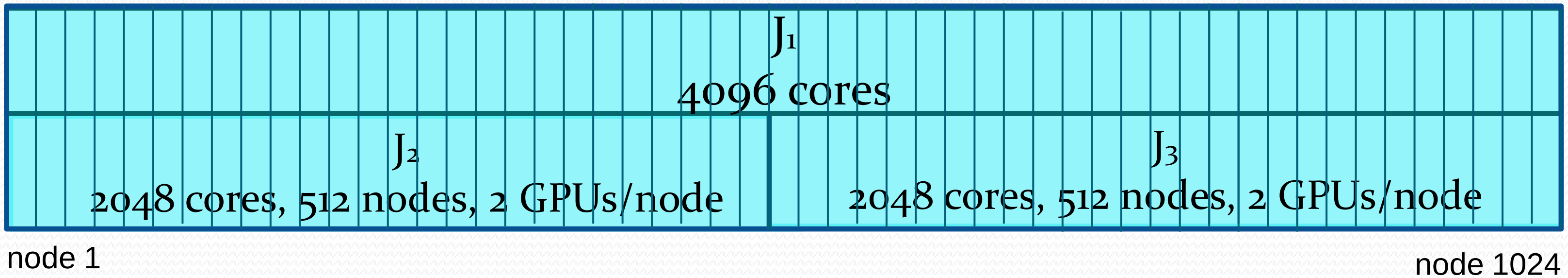
SLURM/Backfill allocation



- J_1 → nodes 1-512, 8 cores/node
- J_2 → nodes 513-1024, 4 cores/node, 2GPUs/node
- J_3 → waiting in queue

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

IPSched allocation



- J_1 → nodes 1-1024, 4 cores/node
- J_2 → nodes 1-512, 4 cores/node, 2GPUs/node
- J_3 → nodes 513-1024, 4 cores/node, 2GPUs/node

- All resources in all nodes are utilized.

IP formulation

$$max \sum p_j(s_j - c_j)$$

$$\sum_j^M x_{ij} \leq R_i \ \forall i \qquad (1)$$

$$\sum_i^N x_{ij} = r_j s_j \ \forall j \qquad (2)$$

$$\sum_j^M g_j t_{ij} \leq G_i \ \ \forall i \qquad (3)$$

$$c_j = \frac{\sum_i^N t_{ij}}{2N} \ \ \forall j \qquad (4)$$

$$N_{min,j} \leq 2Nc_j \leq \ N_{max,j} \ \ \forall j \qquad (5)$$

$$t_{ij} = \begin{cases} 1, & x_{ij} > 0 \\ 0, & x_{ij} = 0 \end{cases} \ \ \forall i,j \qquad (6)$$

Assumptions

- No preemption
- No topology
- Memory is not important

[1] Cplex Optimization, Inc, “Using the CPLEX Callable Library”. Incline Village, NV 89451-9436, 1989-1994.

Problem Size

Variable name	Number of variables
s_j	|N|
c_j	|N|
x_{ij}	|N| * |J|
t_{ij}	|N| * |J|
Total	**2 * |N| * (1 + |J|)**

Equation no	Number of constraints
1	|J|
2	|N|
3	|J|
4	|N|
5	2 * |J|
6	2 * |J| * |N|
Total	**2 * (|N| + 2 * |J| + |J|*|N|)**

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

[1] Cplex Optimization, Inc, “Using the CPLEX Callable Library”. Incline Village, NV 89451-9436, 1989-1994.

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

[2] A.T. Wong, L. Oliker,W.T.C. Kramer, T.L. Kaltz, D.H. Bailey, "ESP: A System Utilization Benchmark," in SC2000: High Performance Networking and Computing. Dallas Convention Center, Dallas, TX, USA, November 4–10, 2000, ACM, Ed., pp. 52–52, ACM Press and IEEE Computer Society Press.

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.

[3] Alejandro Lucero, «Simulation of batch scheduling using real production-ready software tools»

IPSCHED Results

Experiment	Waiting Time (hr) (mean ± std)	Slowdown Ratio (mean ± std)	Utilization (mean)
Backfill / Basic	1.60 ± 0.836	18.11 ± 25.49	0.90
IPSCHED / Basic	0.77 ± 1.257	9.95 ± 18.87	0.92
Backfill / Multifactor	2.42 ± 1.758	22.75 ± 22.02	0.89
IPSCHED / Multifactor	0.88 ± 1.223	10.75 ± 18.20	0.94

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 $\{c \in C \mid c \in B_j \ \ and \ \ n \in N_c\}$
A_n^{cpu} : number of available CPU cores on node n,
A_n^{gpu} : number of available GPUs on node n,
R_j^{cpu}: number of cores requested by job j,
R_j^{gpu}: number of gpus per node requested by job j,
R_j^{node}: number of nodes requested by job j,
R_j^{cpn}: 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.
α: 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

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

subject to constraints :

$$\sum_{c \in B_j} b_{jc} \leq 1 \; for \; each \; j \in J \quad (2)$$

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

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

$$\sum_{j \in J} u_{jn} + r_{jn} \leq A_n^{cpu} \; for \; each \; n \in N \quad (5)$$

$$\sum_{j \in J} u_{jn} \cdot R_j^{gpu} \leq A_n^{gpu} \; for \; each \; n \in N \quad (6)$$

$$0 \leq r_{jn} \leq u_{jn} \cdot min(A_n^{cpu} - 1, R_j^{cpu} - 1)$$
$$for \; each \; (j,n) \in J \times N \quad (7)$$

$$u_{jn} + r_{jn} = \sum_{c \in C_{jn}} b_{jc} \cdot R_j^{cpn}$$
$$for \; each \; (j,n) \in J \times N \; s.t.$$
$$R_j^{cpn} > 0 \; and \; C_{jn} \neq \emptyset \quad (8)$$

Problem Size

Variable name	Number of variables
b_{jc}	|B|
u_{jn}	|K|
r_{jn}	|K|
Total	**2|K| + |B|**

$$|K| = O(|B| * |N|)$$

Equation no	Number of constraints
2	|J|
3	|B|
4	|J|
5	|N|
6	|N|
7	-
8	|K|
Total	**2|N| + 2|J| + |K| + |B|**

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