Scheduling Beyond CPUs for HPC

Fan Y, Lan Z, Rich P, et al.
HPDC 19

Zhouxin Xue, Xuanhang Diao

School of Biomedical Engineering,
ShanghaiTech University

14, April

Pareto set

A **Pareto set** is a set of optimal solutions, where no objective can be improved without worsening another objective.

Burst Buffer

A **Burst Buffer** is an intermediate storage layer positioned between compute nodes and parallel file systems (PFS) in high-performance computing (HPC) systems.

- Absorb the bursty I/O data generated by data-intensive applications.
- Built from solid-state drives (SSDs).
- Can be either attached to compute nodes as local resources or configured as global resources shared by compute nodes.

Multi-Resource Scheduling

- HPC systems are equipped with diverse global and local resources.
- HPC job scheduler plays a crucial role in efficient use of resources.

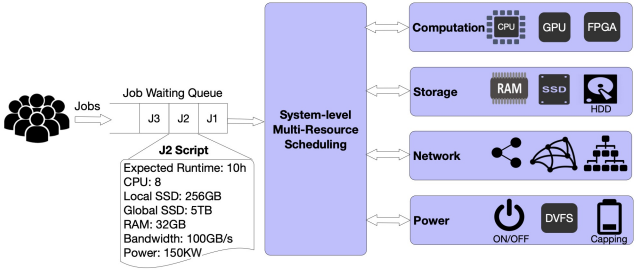


Figure 1: HPC job scheduling problem involves in multiple resources.

Limitations of Existing Scheduling Methods

Existing methods often overlook alternative solutions or optimal resource combinations, leading to under-utilization of resources or poor application performance.

- Naive method
- Constrained method
- Weighted method
- Bin packing method

Goal

The Motivation of the this paper is to improve overall resource utilization and reduce job wait time in HPC systems.

- providing rapid scheduling decisions
- minimizing the impact on site policies
- ensuring extensibility to accommodate emerging resources

Window-based Scheduling

The idea of window-based scheduling as shown is that the first w jobs in the job waiting queue get copied into the window.
This allows our BBsched to consider the site policies and keep the order of the base scheduler as similar as possible.

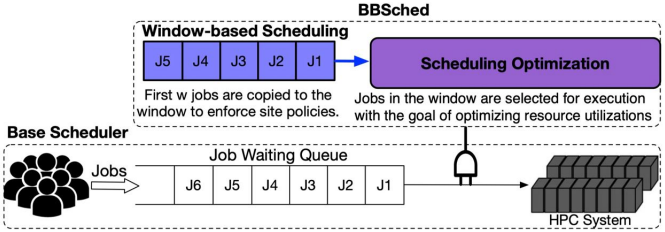


Figure 2: The overview of BBsched.

MOO Solver

Suppose a system has N nodes and burst buffers of total B GB. The amounts of nodes and burst buffers being used are N_{used} and B_{used} respectively. Suppose $J = \{J_1, ..., J_w\}$is a set of w jobs in the scheduling window. Job J_i requiring n_i nodes and b_iGB of burst buffers.

The scheduling problem can be transformed into the following MOO: to determine a finite set of Pareto solutions X; each Pareto solution $x \in X$ is represented by a binary vector $x = [x_1, ..., x_w]$, such that $x_i = 1$ if J_i is selected to execute and $x_i = 0$ otherwise.

Pareto solution optimizes the following two objectives:
(1) maximize node utilization: $f_1(\boldsymbol{x}) = \sum_{i=1}^{w} n_i \times x_w$
(2) maximize burst buffer utilization: $f_2(\boldsymbol{x}) = \sum_{i=1}^{w} b_i \times x_i$
Formally, the problem can be formulated as:

$$
\begin{array}{ll}
\max & (f_1(\boldsymbol{x}), f_2(\boldsymbol{x})) \\
\text{s.t.} & \sum_{i=1}^{w} n_i \times x_i \leq N - N_{\text{used}}\ , \quad x_i \in \{0,1\} \\
& \sum_{i=1}^{w} b_i \times x_i \leq B - B_{\text{used}}\ , \quad x_i \in \{0,1\}
\end{array}
$$

MOO Solver

Because this MOO problem is NP-hard, this paper uses a genetic algorithm to approximate the Pareto set (optimal solutions).

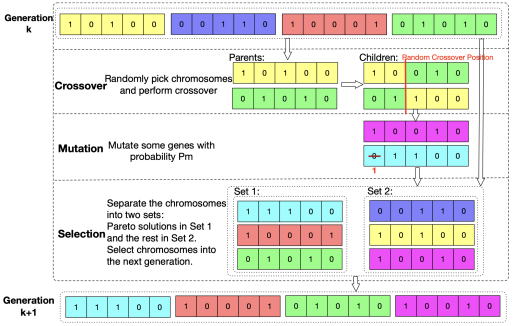


Figure 3: MOO solver maintains a population of candidate solutions (4 chromosomes). A chromosome consists of 5 genes, where each gene represents the selection of the job at a specific location in the window and encodes as a binary number: 1 (selected) or 0 (not selected).

Decision making

The output of the solver is a Pareto set, and a decision maker needs to select one preferred solution.

Different HPC facilities may have different site policies and scheduling priorities.

System managers may use a site-specific metric for selecting a preferred solution out of the Pareto set.

Workloads

Simulation using real workload traces collected from production systems.

	Cori	Theta
Location	NERSC	ALCF
Scheduler	Slurm	Cobalt
System Types	Capacity computing	Capability computing
Compute Nodes	12,076 (2,388 Haswell; 9,688 KNL)	4,392 (4,392 KNL)
Aggregated Memory	1,304.5TB	913.5TB
Shared Burst Buffer	1.8PB	1.26PB (projected)
Trace Period	Apr. 2018 - Jul. 2018	Jan. 2018 - May. 2018
Number of Jobs	2,607,054	70,507
BB Data Source	Slurm log	Darshan log
BB Range	[1GB, 165TB]	[1GB, 285TB]

Figure 4: Overview of Cori and Theta workloads

- Theta is currently not deployed with any shared burst buffer (set the more than 1GB of transferred data as the corresponding job's burst buffer request)

Workload Settings

Workload	percentage of jobs requesting BB	assigned burst buffer request
S1	50%	$\geq$ 5TB
S2	75%	$\geq$ 20TB
S3	50%	$\geq$ 5TB
S4	75%	$\geq$ 20TB

- One issue with both traces is that original burst buffers are not heavily used, and expecting significant increases in requests for burst buffers.
- the assigned burst buffer request is randomly selected from the original burst buffer requests.

Workload Settings

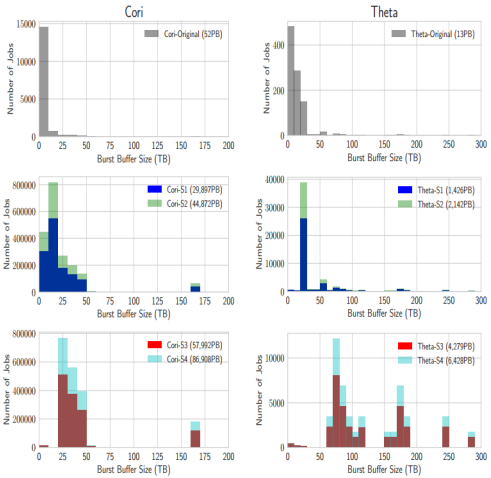


Figure 5: S3 and S4 workloads have larger burst buffer requests than S1 and S2. S1 and S2 have similar distributions, but more jobs in S2 request burst buffers

Metrics

The first two metrics are system-level performance metrics, whereas the last two are user-level performance metrics

1. Node usage: running time/allocate time
2. Burst buffer usage: using time/allocate time
3. Job wait time: interval between job submission to job start time
4. Job slowdown: (job runtime + wait time)/actual runtime

Methods

Eight multi-resource scheduling methods:

1. Baseline
2. Weighted: weights of node 50% - weight of BB 50%
3. Weighted_CPU: weights of node 80% - weight of BB 20%
4. Weighted_BB: weights of node 20% - weight of BB 80%
5. Constrained_CPU: maximize node utilization
6. Constrained_BB:maximize burst buffer utilization
7. Bin_Packing:compute a dot product between the vector of available and requested resources for jobs, then allocate jobs with highest alignment score recursively
8. BBSched: window size to 20, the number of generation to 500, the population size to 20, and the mutation probability to 0.05%

Overview of Result

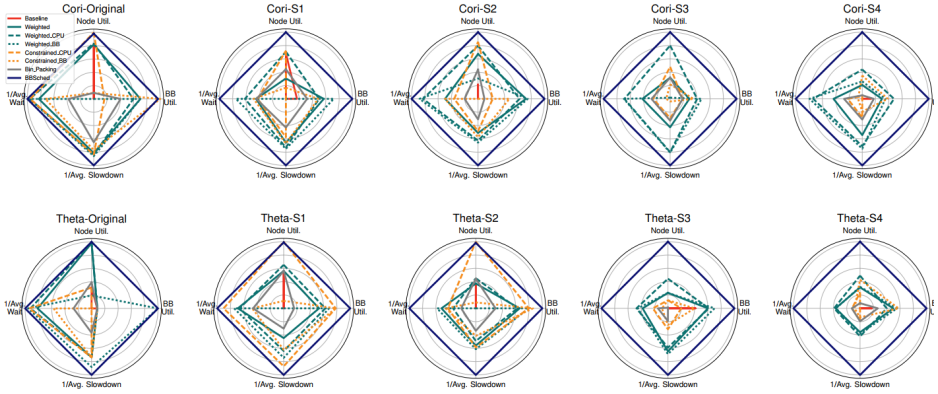
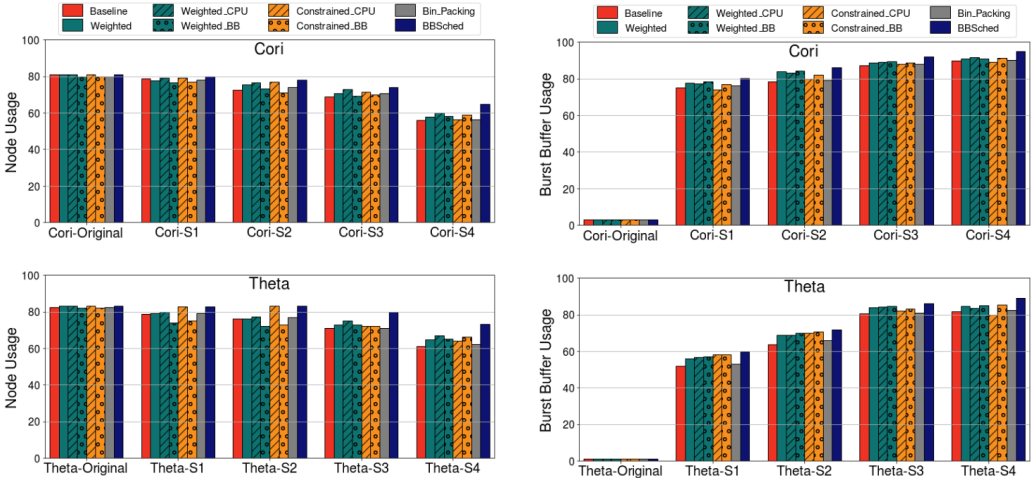


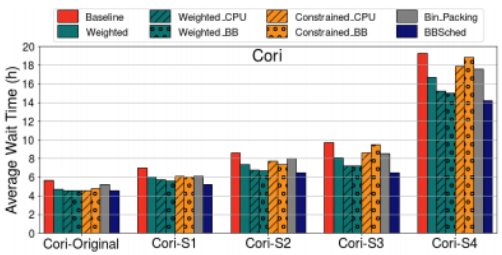
Figure 6: Kiviat graphs: Cori traces (top) and Theta traces (bottom).The larger the area is, the better the overall performance is.

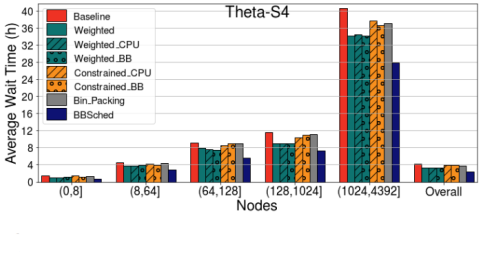
Nodes and Buffer Usage

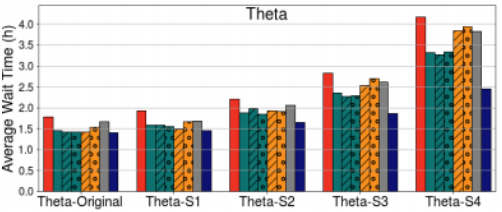


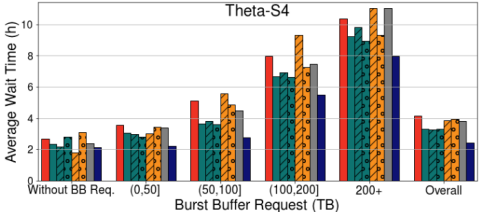
- Node usage and BB usage are almost negatively correlated.
- Where's the sweet point?

Wait Time









- BBSched achieves the most significant reductions on average job wait time, the most significant gain comes from small jobs.

Sensitivity Analysis

Metrics \ Window Size	10	20	50
CPU Usage	60.18%	64.90%	65.06%
	67.12%	73.29%	74.34%
Burst Buffer Usage	92.53%	94.74%	94.65%
	84.23%	89.54%	89.63%
Average Job Wait Time (s)	55,732	51,028	50,871
	10,402	8,847	8,792
Average Slowdown	162.37	154.43	153.20
	8.93	8.16	8.08

Figure 7: BBSched performance under different window sizes. There are two numbers per cell: the top is for Cori-S4 and the bottom is for Theta-S4

Window sizes between 10 to 30 have no significant effect. But a range of 10 to 20 is best.

Incorporate Additional Resources

Present a case study to illustrate that BBSched can be easily extended to incorporate additional schedulable resources. Add two additional objectives

- maximize local SSD utilization
- minimize wasted local SSD

For hardware configuration, assume 50% of nodes in the system are equipped with 128 GB local SSDs, the rest of the nodes are equipped with 256 GB local SSDs.
Generate three workloads (S5-S7) based on Cori-S2 and Theta-S2 by creating job's local SSD requests

Workload	0-128G SSD request	129-256G SSD request
S5	80%	20%
S6	50%	50%
S7	20%	80%

Weighted method aims to maximize the equally weighted sum of node, burst buffer, local SSD utilization, and negative percentage of wasted SSD. Constrained_SSD method aims to maximize local SSD utilization under the constraints of the other resources.

More Complex Optimization Problems

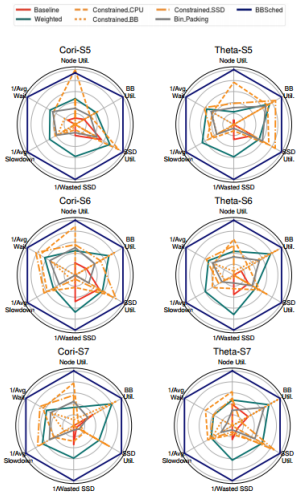


Figure 8: BBSched achieves the best overall performance on all workloads

Conclusion

- Multi-resource scheduling problem can be formulated into a MOO problem and is rapidly solved by a multi-objective genetic algorithm.
- The extensive trace-based simulations demonstrate BBSched outperforms the existing methods in terms of both system-level and user-level metrics(41% over naive method, 33% over bin packing method, 35% over constrained methods, and 20% over weighted methods).
- Can be extended to incorporate other resources.
- Needs test on real systems.