# Exploiting Multi-Resource Scheduling for HPC

Yuping Fan
Illinois Institute of Technology
yfan22@hawk.iit.edu

Zhiling Lan (Advisor)
Illinois Institute of Technology
lan@iit.edu

## 1 INTRODUCTION

High-performance computing (HPC) is undergoing significant changes. Next generation HPC systems are equipped with diverse global and local resources (Figure 1), such as I/O burst buffer resources, GPUs, memory resources (e.g., on-chip and off-chip RAM, external RAM), network resources, and possibly other resources. HPC job scheduler plays a crucial role in efficient use of resources. However, traditional job schedulers are single-objective and fail to efficient use of other resources. In our previous work, we present a job scheduling framework named BBSched [3] to schedule CPUs and burst buffers. As we are heading toward exascale computing, a variety of heterogeneous resources are deployed in HPC systems. In this poster, we extend BBSched for allocating and managing multiple resources beyond CPUs and burst buffers. We formulate multi-resource scheduling as a general multi-objective optimization (MOO) problem, present a heuristic method to solve the NP-hard MOO problem, and provide a preliminary evaluation of the multi-resource scheduling design for scheduling up to ten distinct global/local resources. The proposed multi-resource scheduling design is intended to enhance Cobalt [2], a production batch scheduler and resource manager deployed on HPC systems at Argonne Leadership Computing Facility (ALCF).

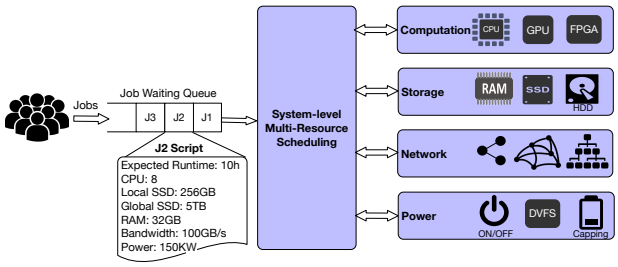


**Figure 1: HPC job scheduling problem involves in multiple resources.**

## 2 METHODOLOGY

Figure 2 depicts a high-level overview of our multi-resource job scheduling framework named MultiSched.

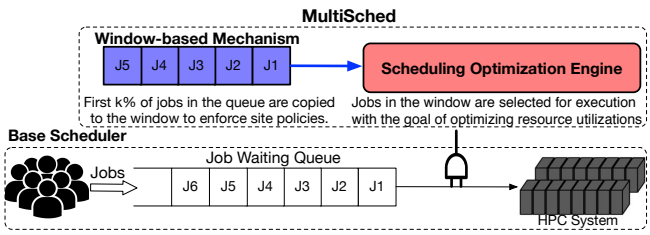


**Figure 2: Multi-resource scheduling design. The framework is composed of two key components: window and scheduling optimization engine.**

### 2.1 Window-based Mechanism

The purpose of window-based mechanism is to balance the goals of enforce site policies and optimize resource utilization. In BBSched, we set the static window size (e.g., 20). However, job queue length changes frequently. It is typically longer during workdays and it shorter during weekends. Hence, we use a knob to dynamically adjust window size based on job queue length. The knob $k$ takes values in the range of [0%,100%] and presents what percentage of jobs in the queue are entered in the window. $k = 0\%$ means jobs are executed based in the order of the site policy; $k = 100\%$ means all jobs in the queue are considered in optimizing resource utilization.

### 2.2 Scheduling Optimization

Scheduling optimization process selects jobs from the window for execution so that utilizations of multiple resources are maximized. We extend MOO formulation in BBSched to maximize more types of resources. Assume a system has $N$ nodes and $R1, \ldots, Rn$ amount of other schedulable resources (e.g., GPU, burst buffer, memory, network bandwidth, power, etc.). Upon a scheduling invocation, the amount of nodes and other resources being used are $N_{used}$ and $R1_{used}, \ldots, Rn_{used}$ respectively. Suppose $J = J_1, \ldots, J_w$ is a set of $w$ jobs in the window: job $J_i$ requests $n_i$ nodes and $r1_i, \ldots, rn_i$ of other resources. We transform the multi-resource scheduling problem into the following MOO problem:

$$\max_{1 \le i \le w} \sum n_i * x_i$$
$$\max_{1 \le i \le w} \sum r1_i * x_i$$
$$\ldots$$
$$\max_{1 \le i \le w} \sum rn_i * x_i$$

subject to

$$x_i \in \{0, 1\},$$
$$\sum n_i * x_i \le N - N_{used},$$
$$\sum r1_i * x_i \le R1 - R1_{used},$$
$$\ldots,$$
$$\sum rn_i * x_i \le Rn - Rn_{used}.$$

Because this MOO problem is NP-hard, we explore genetic algorithm to rapidly approximate the true Pareto set (optimal solutions). However, as we increase the number of scheduled resources, it becomes harder to find solutions that are close to the true Pareto set in the limited scheduling timeframe (30 seconds). In order to address this challenge, instead of considering all the jobs in the window, we first exclude the jobs that cannot fit into the available resources (shown in Figure 3). Because the genetic algorithm is a stochastic process which randomly generate, crossover and mutate solutions, this preprocessing step greatly reduces the chances of

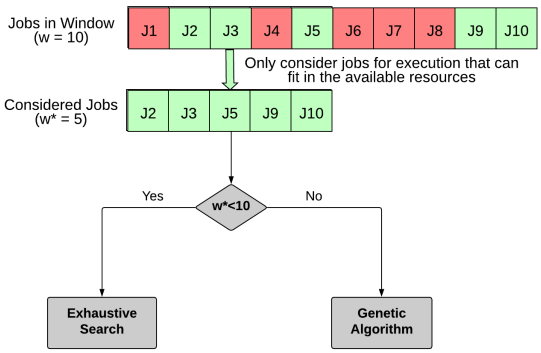


**Figure 3: Improvements made on optimization engine.**

the genetic algorithm hit the invalid solutions. In our experiments, we find that on average, more than half of jobs in the window are excluded from the genetic algorithm. Therefore, MultiSched can reduce the problem size from $2^w$ to $2^{\frac{w}{2}}$,

To further improve the quality of solutions, as shown in Figure 3, if less than 10 jobs are considered, we use the exhaustive search instead of genetic algorithm to find the optimal solutions. When the problem size is small, the exhaustive search and genetic algorithm spend similar amount of time, and therefore, we prefer to use exhaustive search to ensure that we find all optimal solutions.

## 3 PRELIMINARY RESULTS

We evaluate our design through trace-based simulation [1][4][5] using the half-year trace from Theta [2]. In addition to CPU resource, we consider five other schedulable resources denoted as $R1 - R5$. The requests for $R1 - R5$ resources were randomly generated following normal distribution. We evaluate and compare the five existing scheduling methods with MultiSched: FCFS, Weighted, Constrained, Bin Packing, and BBSched. FCFS is the Naive method which allocates jobs in the order of their arrival time until the job at the front of the queue cannot fit into the available resources. Weighted maximizes the weighted sum of utilizations. Constrained method maximizes the CPU utilization. Bin packing method iteratively selects big jobs from the queue. BBSched is scheduling method we used in our previous work. For more details about the existing methods, please refer to our previous paper [3]. MultiSched is used to denote our improved design.

Figure 4 compares the resource utilization using different methods. Clearly, MultiSched achieves the best performance. The sensitive analysis on MultiSched's knob ($k$) shows that increasing $k$ can improve the resource utilization at the cost of less preservation on site policies (Figure 5). Figure 6 presents the time-to-solution of BBSched and MultiSched. When we increase the number of scheduled resources, average scheduling time increases. MultiSched has slower increase rate compared with BBSched. MultiSched's scheduling time is less than 0.6 seconds when we increase the number of resources to 10. The typical HPC systems require job scheduler to be responded in 30 seconds, which makes MultiSched practical to be used on real HPC systems.

## 4 SUMMARY

In this poster, we present a proof-of-concept design of extending our prior work named BBSched for multi-resource scheduling in HPC. The goal is to enhance the existing job scheduler such as Cobalt used at ALCF for scheduling multiple resources beyond CPUs and burst buffers. Our preliminary study indicates that our design achieves higher resource utilization with less overhead.

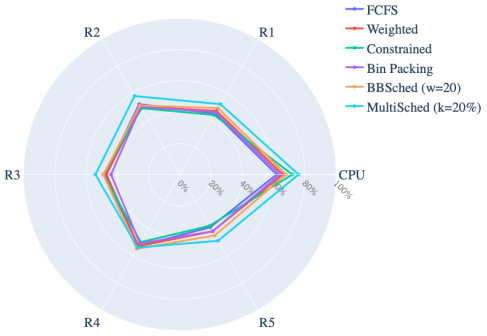


**Figure 4: Resource utilization using different methods.**

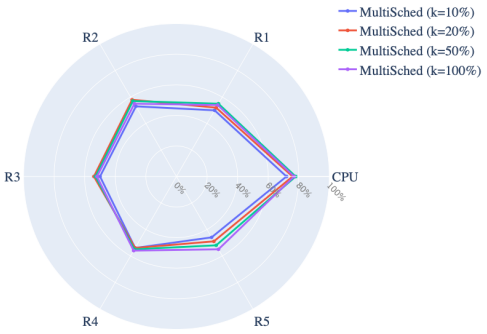


**Figure 5: Resource utilization using different knob values.**

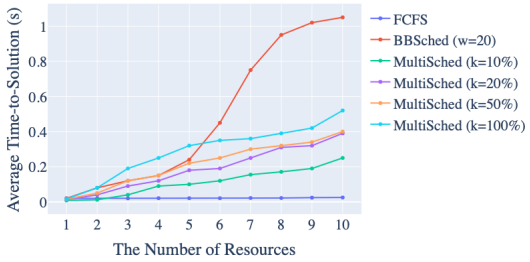


**Figure 6: Average scheduling time using different methods.**

## ACKNOWLEDGMENTS

This work is supported in part by US National Science Foundation grants CNS-1717763, CCF-1422009, CCF-1618776, and the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357.

## REFERENCES

[1] CQSim. http://bluesky.cs.iit.edu/cqsim/
[2] W. Allcock, P. Rich, Y. Fan, and Z. Lan. 2017. Experience and Practice of Batch Scheduling on Leadership Supercomputers at Argonne. In *Job Scheduling Strategies for Parallel Processing (JSSPP)*.
[3] Y. Fan, Z. Lan, P. Rich, W. E. Allcock, M. E. Papka, B. Austin, and D. Paul. 2019. Scheduling Beyond CPUs for HPC. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*.
[4] Y. Fan, P. Rich, W. Allcock, M. Papka, and Z. Lan. 2017. Trade-Off Between Prediction Accuracy and Underestimation Rate in Job Runtime Estimates. In *CLUSTER*. 530–540.
[5] B. Li, S. Chunduri, K. Harms, Y. Fan, and Z. Lan. 2019. The Effect of System Utilization on Application Performance Variability. In *Proceedings of the 9th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS)*.