Exploring the Trade-off Space of Hierarchical Scheduling for Very Large HPC Centers

Stephen Herbein
University of Delaware
sherbein@udel.edu

Advisor: Dong H. Ahn
Lawrence Livermore National Laboratory
ahn1@llnl.gov

Advisor: Michela Taufer
University of Delaware
taufer@acm.org

ABSTRACT

Hierarchical batch scheduling has many advantages, but a lack of trade-off studies against traditional scheduling precludes the rapid emergence of effective solutions for very large HPC centers. Under the hierarchical scheme, any job can instantiate its own scheduler to schedule the sub-jobs with a distinct policy. While such scheduling parallelism and specialization are extremely attractive for the expanding scale and resource diversity of the centers, hierarchical resource partitioning can lead to poor resource utilization. In this poster, we explore the trade-off space between scheduling complexity and resource utilization under hierarchical scheduling. Our approach includes novel techniques to create a hierarchical workload automatically from a traditional HPC workload. Our preliminary results show that introducing only one additional scheduling level can reduce scheduling complexity by a factor of 3.58 while decrease resource utilization by up to 24%. Our study, therefore, suggests that poor utilization can be real under hierarchical scheduling and motivates dynamic scheduling as a complementary technique to combat the inherent issue.

1. HIERARCHICAL SCHEDULING

The path towards Exascale surely involves a large increase in the number of compute resources in a high performance computing (HPC) center along with the need for a scheduler to manage these resources. Several solutions to this problem exist for cloud data centers, but they lack scalability, strict policy enforcement, and multi-dimensional resource support required by HPC [1, 2]. Hierarchical scheduling provides high levels of parallelism and scalability while simultaneously ensuring strict enforcement of center policies and multi-dimensional constraints.

Flux is a resource and job management system currently being developed to enable hierarchical, multi-level scheduling for very large HPC centers. Each scheduler in the hierarchy is bound by two rules:

1. *Parent bounding rule*: a parent scheduler grants and confines the resource allocation of all of its children;
2. *Child empowerment rule*: within the bounds set by the parent, a child scheduler is delegated the ownership of the allocation and becomes solely responsible for the most efficient uses of its resources.

These rules ensure that even with multiple schedulers running in parallel, constraints are enforced and satisfied while also allowing for flexibility within each scheduler. Hierarchical scheduling can also scale to much higher numbers of resources and jobs than existing schedulers because each scheduler in the hierarchy has to manage only its immediate children, not all of the jobs across the entire center.

2. APPROACH

The advantages of hierarchical scheduling, however, can come at a significant cost: poor resource utilization. Thus, practical solutions require us to quantify this trade-off space, and if the utilization challenges are real, complementary techniques to address these challenges must be developed.

2.1 Realistic Hierarchical Workload Creation

Hierarchical scheduling is new for large HPC centers, and thus no hierarchical workload exists for use in our trade-off study. Thus, we use existing HPC workloads and aggregate multiple smaller jobs into large single jobs—i.e., hierarchical jobs—which can easily map to our hierarchical scheduling scheme. To automatically aggregate jobs from exiting workloads, we created a scheduling *pre-processor* that intercepts user job submissions and aggregates them. Jobs submitted within a short window of time and with similar characteristics are aggregated together into a larger job. If the aggregation rate is high, this can offer two main benefits. First, for each hierarchical job, a corresponding scheduler can be created, thus distributing the complexity of scheduling across the hierarchy. Second, many *flat* jobs from existing workloads can be easily and automatically aggregated to form hierarchical jobs, resulting in fewer jobs for the scheduler at the top of the hierarchy (top-level scheduler) to manage and thus improving scalability.

2.2 Job Aggregation Modes

Due to the constraints of existing HPC schedulers, hierarchical jobs must have their sizes set at submission time. We assess the effects of two methods for setting the size of hierarchical jobs. These two methods represent the two possible

extremes. The first method, which we refer to as *MinTime*, sets the size of jobs so that all of the sub-jobs can execute in parallel, thus minimizing the execution time of the hierarchical job. The second method, which we refer to as *MinRes*, sets the size of hierarchical jobs to the size of the largest sub-job, thus minimizing the resource footprint of the hierarchical job.

3. RESULTS

We evaluate these parameters of our hierarchical scheduling by extending the Flux scheduling emulator with our preprocessor for automatic job aggregation. As input, we feed the emulator with job traces from Rzmerl and Rzzeus, two clusters at Lawrence Livermore National Laboratory. The traces were collected over 2.5 months, and combined contain 32,046 jobs.

3.1 Scheduling Complexity

Scheduling complexity is a function of the number resources and jobs to be considered at each level in the hierarchy. When executing our hierarchical scheduling with job traces from Rzzeus, we observe a 3.58x reduction in the number of jobs sent to the top-level of the hierarchical scheduler, as seen in Table 1. This significantly reduces scheduling complexity at the top-level scheduler and ultimately increases the number of jobs that it can schedule over time. The job traces for Rzmerl experience a smaller reduction because Rzmerl has a lower job-submission rate; thus there is a lower probability that two jobs from the same user are submitted within the configured time interval.

Machine	Pre-Aggregation	Post-Aggregation	Ratio
Rzmerl	6,737 jobs	5,688 jobs	1.18
Rzzeus	25,309 jobs	7,073 jobs	3.58

Table 1: Effects of job aggregation on the number of jobs handled by the top-level scheduler

3.2 Resource Utilization

Our hierarchical scheduling decreases the number of jobs handled by the top-level hierarchical scheduler and thus improves scalability. At the same time, hierarchical job allocations associated to our scheduling can fragment the system's resources and lead to a decrease in resource utilization. We notice that a decrease in utilization occurs when a large job and a small job are aggregated together with *MinRes*. The two sub-jobs execute serially within the hierarchical job's allocation, and some of the resources are idle when the smaller job executes. In addition, a decrease in utilization occurs when a long-running job and a short-running job are aggregated together with *MinTime*. These sub-jobs execute in parallel within the hierarchical job's allocation, and some of the resources become idle when the short-running job terminates. We study the resource utilization for both conditions.

Figure 1 shows the resource utilization that we observe in our hierarchical scheduler emulations. We observe that *MinRes* aggregation creates hierarchical jobs that cause virtually no decrease in utilization.

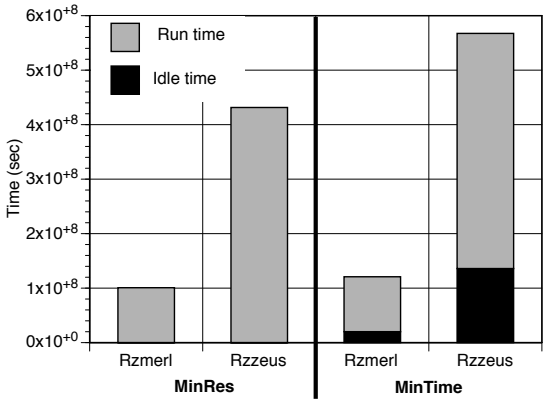

Figure 1: Effects of hierarchical scheduling on resource utilization caused by idle resources within hierarchical job allocations. The grey bars represent the time that resources within a hierarchical job spend executing a sub-job, while the black bars represent the time that resources within a hierarchical job spend idle.

4. CONCLUSION AND FUTURE WORK

Our results demonstrate that hierarchical scheduling offers a sizable reduction in scheduling complexity with a decrease in resource utilization between virtually none and up to 24%. They also show that larger and busier HPC centers can reduce their scheduling complexity with hierarchical scheduling with deeper levels. Our work motivates incorporation of dynamic scheduling into our hierarchical scheduling to address the resource utilization challenges we quantified in this study.

5. ACKNOWLEDGMENTS

Prepared by LLNL under Contract DE-AC52-07NA27344 (LLNL-ABS-675749). National Science Foundation CCF #1318445.

6. REFERENCES

[1] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308. USENIX Association, 2011.

[2] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *SIGOPS European Conference on Computer Systems (EuroSys)*, pages 351–364, Prague, Czech Republic, 2013.