Workload-Adaptive Scheduling for Efficient Use of Parallel File Systems in High-Performance Computing Clusters

Alexander V. Goponenko*, Benjamin A. Allan†, Jim M. Brandt‡, and Damian Dechev§
* § University of Central Florida, Department of Computer Science, Orlando, FL, USA
Email: *alexander.goponenko@ucf.edu, §damian.dechev@ucf.edu
† ‡ Sandia National Laboratories, Albuquerque, NM, USA
Email: †baallan@sandia.gov, ‡brandt@sandia.gov

***Abstract*—Whereas contentions within storage systems noticeably impact runtimes, shared bandwidth-type resources, such as Lustre, pose challenges for high-performance computing cluster schedulers. Additionally, accurately estimating job resource requirements, particularly related to I/O operations, remains a significant challenge for users. In response to these challenges, we have developed a prototype that facilitates I/O-aware scheduling in Slurm without imposing additional burdens on users. Accounting for the specific properties of this bandwidth-type resource, our system monitors real-time Lustre bandwidth utilization, estimates job I/O requirements, and dynamically adjusts to the demands placed on the file system. Our workload-adaptive scheduler aims to maintain the bandwidth utilization at a level that reflects the resource requirement of the job queue. We further enhance the efficacy of our approach by introducing a "two-group" approximation technique that ensures efficient performance regardless of the availability of zero-throughput jobs. We demonstrate effectiveness of our approach through evaluation on a real cluster.**

***Index Terms*—High-performance computing, parallel job scheduling, I/O-aware scheduling, multi-resource scheduling, Slurm, Lustre**

I. Introduction

High-Performance Computing (HPC) clusters are designed to handle intensive computations that are not feasible on a single computer. To fulfill this role, an HPC cluster contains many high-end computer servers called nodes and a large amount of other resources. Most jobs that run on an HPC cluster do not require all cluster resources. Therefore, HPC clusters typically process several jobs simultaneously. Often, the combined demand of the jobs submitted to a cluster surpasses the cluster's capacity. Schedulers negotiate conflicts between the demands and the available resources. An HPC cluster scheduler decides the order of execution of queued jobs and the allocation of the cluster resources to the running jobs. An efficient scheduler must ensure the fair use of the HPC cluster according to user policies and, at the same time, attain close to optimal performance and job throughput of the cluster. The growing use of high-intensity computing and its rising economic and environmental impact underscore the need for maximizing the utilization of HPC infrastructure [1], [2]. For instance, the Frontier system's construction cost is estimated to be around 400–600 million US dollars, with an operational energy consumption of 22 MW [3]. A 5% improvement in its performance could potentially allow for a 5% reduction in system size. This reduction could translate to savings of 20–30 million US dollars in construction costs and approximately 9.2 GW·h of energy annually. The annual energy savings could amount to 700 thousand US dollars (assuming a cost of $0.08 per kW·h [4]) or be sufficient to meet the energy needs of 400 people (based on a worldwide per capita energy consumption of 21 039 kW·h in 2022 [5]). Although Frontier represents one of the largest clusters, it comprises only a fraction of the HPC computing infrastructure (17% of TOP500 total performance [6]). Many HPC clusters are less efficient than Frontier, indicating that improving their performance could realize even greater benefits. Consequently, there is a constant demand for more efficient schedulers.

In the past, HPC schedulers primarily focused on managing compute nodes as if the nodes were the only cluster's resource. However, as jobs become more resource-intensive and HPC systems become more complex, other resource bottlenecks become prominent. As a result, the effectiveness of approaches initially designed with a single resource in mind deteriorates. For instance, I/O bandwidth is frequently becoming an increasingly scarce resource [7]–[10]. Contention within HPC storage systems leads to performance degradation. To mitigate I/O congestion and enhance cluster efficiency, schedulers must manage these bandwidth-type resources. Unlike compute

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC (NTESS), a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration (DOE/NNSA) under contract DE-NA0003525.

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00190

nodes, where resource management systems can appropriately restrict a job's usage to its dedicated share, the bandwidth-type resources, such as file systems and networks, cannot be exclusively reserved for a single job. A parallel file system, such as Lustre, will inevitably experience concurrent access from multiple jobs. This shared usage impacts the execution time of each job. Furthermore, an allocation of a specific portion of the file system bandwidth to a job may even prove counterproductive because HPC jobs I/O operations are bursty and unpredictable (Section II-B).

Advanced scheduling techniques encounter additional hurdles due to the need for detailed information regarding job resource requirements. Requests for job details inconvenience users and pose a potential barrier to the adoption of new scheduling methods. Furthermore, users often struggle to provide adequate estimations of their jobs' resource requirements. Thus, inaccuracies in user-provided job runtimes have a significant impact on scheduling quality [11]–[13]. Anticipating file system usage presents even greater challenges. Whereas some users can estimate the maximum total bytes written or read by their jobs, determining average I/O throughput can be cumbersome. Predicting file access patterns, peak rates, and their temporal occurrence within the job execution window is a nearly impossible task. Relying on inaccurate data jeopardizes scheduling efficiency.

A. Contributions

We have developed a prototype of a complete, functioning I/O-aware scheduling system for Slurm, a widely utilized HPC cluster resource manager [14]. The prototype introduces Lustre bandwidth as an additional resource in Slurm and implements full-featured multi-resource backfill scheduling.

Our scheduler accounts for the specific properties of the new bandwidth-type resource. The system measures real-time Lustre bandwidth utilization and autonomously determines additional job resource requirements the scheduler needs.

We also present the workload-adaptive scheduling that reduces I/O congestion by maintaining the bandwidth utilization at a level that matches the resource requirements of the job queue. We further enhance this approach via a "two-group" approximation technique, which attains efficient performance regardless of the availability of zero-throughput jobs.

Through evaluation conducted on a real HPC cluster, we demonstrate the effectiveness of our workload-adaptive I/O-aware scheduling approach. Our strategy reduces the file system congestion and surpasses not only the performance of the default Slurm scheduler but also the improvements attainable by the previously reported I/O-aware HPC cluster scheduling techniques.

Although our implementation specifically targets Slurm and Lustre, the concepts of bandwidth-type resource, workload-adaptive scheduling (Section VII), and "two-group" approximation (Section VII-A) that we introduce are widely applicable to other schedulers and I/O resources.

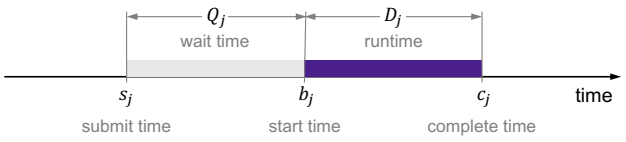


Fig. 1: Diagram of job lifetime showing the notation used herein.

II. Background

We will use the following notation in this paper, as described on the diagram that represents events happening to a job (Fig. 1). After a user submits a job j, at time s_j (called submit time or arrival time), the job may end up in a wait queue. After spending Q_j wait time in the queue, the job starts at time b_j and completes after additional D_j runtime at time c_j.

For simplicity, we discuss the implementation based on Slurm and Lustre, although the results apply to other resource management systems as well as other parallel file systems. Also for simplicity, we consider only two cluster resources: nodes (N nodes in total) and Lustre bandwidth. Each job is assumed to require a constant number of nodes to run n_j; the user provides this number during the job submission, along with a requested runtime limit L_j. The actual job runtime D_j, is not known until the job is completed. We denote the estimated runtime as d_j. The Lustre throughput of the job, averaged over its duration, is also unknown, depends on the cluster environment, and must be estimated. The estimated throughput is referred to as r_j.

A. Slurm Scheduling Algorithm

Slurm provides a backfill list scheduling algorithm, implemented as a plugin and enabled by default. Algorithm 1 delineates this scheduling algorithm. Conceptually, it is a hybrid backfill scheduling approach that can be adjusted via the value of $\mathrm{BackfillMax}$ to balance the fairness and the scheduling overhead. If $\mathrm{BackfillMax} = 1$, the algorithm becomes equivalent to EASY backfill [15]. However, the default Slurm setting is equivalent to $\mathrm{BackfillMax} = \infty$, which directs the scheduler to make reservations for all delayed jobs. The scheduler must therefore keep track of these reservations. Procedure `InitializeReservationTracker` instantiates a resource reservation tracker data structure and initializes it by reserving, for each running job j, the resources used by the job for the time the job is allowed to run, that is, the time interval $[b_j, b_j + L_j)$. Procedure `ReserveResources`(j, t) reserves the resources for job j starting at time t, also for the maximum time the job is allowed to run L_j. Procedure `EarliestStartTime`(j, t) returns the earliest time not later than t such that the resources required by job j are available for the duration L_j.

Slurm offers several methods for integrating various resources into the scheduling system [16]. The most suitable approach for managing the parallel file system resource is through the use of "licenses." Whereas Slurm's license management was previously basic, the release of version 22.05

Algorithm 1. Outline of Slurm scheduling backfill algorithm

Input: Running jobs $\mathcal{R}$, waiting jobs $\mathcal{Q}$, current time t_{now}
Options: Maximum number of backfill reservations to track BackfillMax

```
RT ← InitializeReservationTracker(R, Q)
Sort waiting jobs in Q.
BackfillCount ← 0
for j ∈ Q do
    t ← RT.EarliestStartTime(j, t_now)
    if t = t_now then
        Start job j.
        RT.ReserveResources(j, t_now)
    else
        if BackfillCount ≥ BackfillMax then
            Skip job j until the next scheduling round.
        else
            RT.ReserveResources(j, t)
            BackfillCount ← BackfillCount + 1
```

introduced the option to enable accurate license reservation tracking for delayed jobs [17]. However, if Lustre bandwidth is integrated into Slurm using existing functionality, the scheduler will require the users to specify the file system resource requirements for their jobs. This places an additional burden on the users and may encourage them to underestimate their job's file usage in an attempt to avoid potential scheduling delays caused by license shortages. The users would not face penalties for underestimating, as Slurm does not enforce license resource allocation. Furthermore, Algorithm 1 has a limited efficiency in cases of multiple resources: local depletion of one resource can cause under-utilization of the other resources [18].

B. Characteristics of HPC Jobs I/O Patterns

I/O patterns in HPC clusters are often characterized by burstiness [19]–[21]. Due to the periodic nature of scientific applications, characterized by computation phases followed by I/O phases, the bursts of I/O activity in HPC clusters tend to be lengthy [22]. Despite the adoption of buffers and other mechanisms to mitigate the negative impacts of these I/O bursts, the simultaneous occurrence of I/O phases from multiple jobs often leads to a slowdown in the execution of all affected jobs. Strategies that try to keep the average throughputs of all the jobs running below the available bandwidth (including the I/O-aware scheduling presented in Section VI) are less useful when there are bursts of I/O activity. The overlap of I/O bursts may still lead to periods of file system overload. Restricting jobs so that the sum of the maximum job throughputs always remains below the bandwidth is inefficient, resulting in either under-utilization of the cluster or in a cluster design with excessive over-provisioning of the file system bandwidth. Although scheduling jobs so that I/O phases do not overlap would help, predicting I/O patterns is challenging [23]. Therefore, alternative scheduling techniques that can minimize instances of I/O activity overlap, such as the workload-adaptive scheduling presented in Section VII, are preferable.

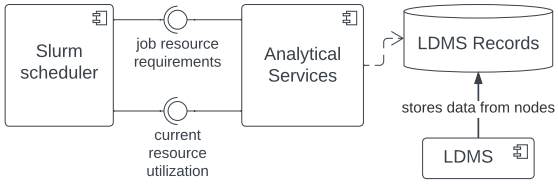


Fig. 2: Diagram of interactions between the components of the scheduling system.

III. Overview of Our Implementation

We developed our prototype by integrating three primary components: a scheduler, responsible for job scheduling and management; a performance monitoring tool that gathers resource usage data; and analytical services tasked with processing runtime data into real-time resource utilization and job resource requirements. Fig. 2 illustrates the schematic component diagram. The scheduler component consists of Slurm (with a small number of necessary modifications) and a scheduling plugin that is configurable to provide all the features described in this paper. The source code and details on the scheduler prototype are available at GitHub[1]. For performance monitoring, we employ LDMS, the Lightweight Distributed Metric Service [24], which collects various performance-related information from all compute nodes and stores it in a database accessible by the analytical services. Our database utilizes the Scalable Object Store (SOS), a component of LDMS.

The analytical services, the third component depicted in Fig. 2, fulfill Slurm's requests for information, with a primary focus on predicting job resource requirements. We compute these predictions as weighted averages of historical resource usage of similar jobs, using exponentially decaying weights to increase the contribution of recent jobs. When a job finishes, the scheduler notifies the analytical services, which will then retrieve LDMS records related to the job, compute the job's resource utilization, and reevaluate resource requirements for future requests. In the upcoming examples of workloads, identifying similar jobs and predicting job resource utilization pose no significant challenge. Consequently, our prototype does not explore advanced prediction techniques. It's worth emphasizing that various prediction methods discussed in existing literature [13], [25]–[29] can seamlessly integrate into our framework. In addition to estimating job resource requirements, the analytical services module computes the current total Lustre throughput, for robustness in scenarios where estimates are inaccurate or unavailable for certain jobs, as discussed in Section VI.

[1]https://github.com/algo74/slurm/tree/workload-adaptive-paper-2024

IV. Evaluation Methodology

In the remaining sections, we compare I/O-aware and workload-adaptive scheduling with the default Slurm scheduler and with each other. We demonstrate these scheduling techniques in application to the scheduling of simplified but representative workloads on a real HPC system (see Appendix for details).

Our experiments were conducted on Stria, a production cluster comprising 288 ARM64 nodes featuring 2.0 GHz Cavium Thunder-X2 processors with 2 sockets and 28 cores per socket. The cluster is equipped with a 2:1 oversubscribed Mellanox EDR Infiniband network [30]. Stria's Lustre 2.12.3 file system consists of two metadata servers, four object storage servers, and 56 Solid State Drive volumes with a total capacity of 383 TiB. This Lustre instance is not shared with other clusters. We allocated 16 nodes from Stria for our experiments. One node served as the control node, responsible for running our modified Slurm control instance and the other system modules. The remaining 15 nodes were utilized as compute nodes of our experiments.

Given the impractical length of time required to conduct scheduling experiments of workloads containing real HPC cluster applications, we opted for synthetic job workloads. We designed these workloads to reflect common characteristics of HPC I/O patterns and highlight various aspects of job scheduling in the context of parallel file system usage. However, we did not specifically tailor the workloads to maximize the observed differences between the evaluated methods. We configured each job in our workloads to require one node in order to streamline the analysis and avoid unnecessary complications associated with node scheduling. As a result, our experiments show no noticeable signs of backfill or node reservation for delayed jobs. Nevertheless, it's worth mentioning that our scheduler fully implements and handles these aspects of scheduling, similar to Slurm.

The first workload we analyze, which we refer to as "*Workload 1*," consists of two types of jobs: "*write×8*" and "*sleep*." A "*write×8*" job executes 8 parallel threads on one node, with each thread generating 10 GiB of data written to a randomly chosen Lustre storage volume (80 GiB total per job). Conversely, a "*sleep*" job remains idle for 600 s, also using one node. These jobs are organized within "*Workload 1*" into a pattern of 8 waves. Each wave comprises 30 "*write×8*" jobs and 60 "*sleep*" jobs, resulting in a total of 720 jobs in the entire workload. This workload mirrors the common scenario where users submit batches of similar jobs and reflects the predominance of write operations in scientific computation tasks [7]. The second workload, "*Workload 2*," shares similarities with "*Workload 1*" but features six distinct job types. It is described further in Section VII-A.

V. Default Slurm Backfill Scheduling

For comparison with the other algorithms, Fig. 3(a) presents the results of scheduling "*Workload 1*" using the default Slurm backfill scheduler. The top panel of Fig. 3(a) displays the Lustre throughput, while the bottom panel shows the node allocation. These results demonstrate that the default scheduler, which lacks awareness of the file system utilization, dispatches jobs in the order they appear in the queue. Consequently, periods of high Lustre utilization are followed by intervals of low utilization.

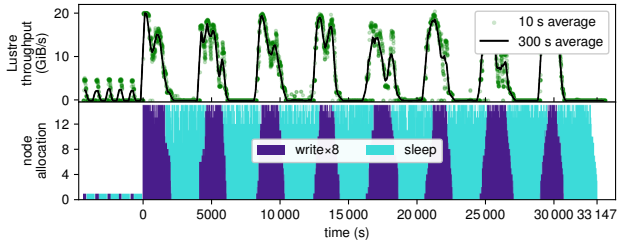
The Lustre throughput undergoes significant fluctuations even while the combination of the running jobs does not change, which is a common situation we observe in other instances of parallel file systems. The peak Lustre throughput is about 20 GiB/s. Fig. 4 indicates that as more "*write×8*" jobs are added to run simultaneously, the rate of increase in throughput diminishes. Although the "short-term" bandwidth is approximately 20 GiB/s, the "long-term" bandwidth may likely be located around 15 GiB/s; the throughput levels up near this threshold in Fig. 4.

Algorithm 2. Conceptual outline of I/O-aware scheduler's `InitializeReservationTracker`

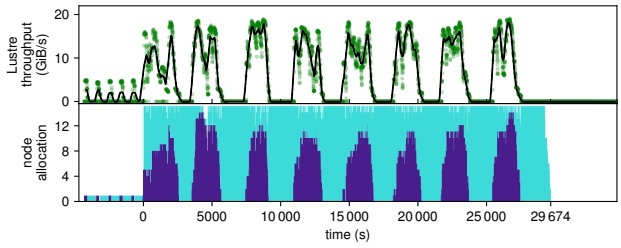
Input: Running jobs $\mathcal{R}$, waiting jobs $\mathcal{Q}$, current time t_{now}
Output: $\{\text{NT}, \text{LT}\}$
1 Obtain the latest values of r_j for $j \in \mathcal{R} \bigcup \mathcal{Q}$.
2 Obtain current Lustre throughput R_{now}.
3 Initialize a node tracker NT using the Slurm's standard method.
4 Create a new tracker LT to monitor Lustre throughput reservations.
5 **for** $j \in \mathcal{R}$ **do**
6 | Reserve r_j in LT for interval $[b_j, b_j + L_j)$.
7 **if** $\sum_{j \in \mathcal{R}} r_j < R_{\text{now}}$ **then**
8 | Reserve $R_{\text{now}} - \sum_{j \in \mathcal{R}} r_j$ in LT for $\left[t_{\text{now}}, \max_{j \in \mathcal{R}} (b_j + L_j)\right)$.

VI. I/O-aware Scheduling

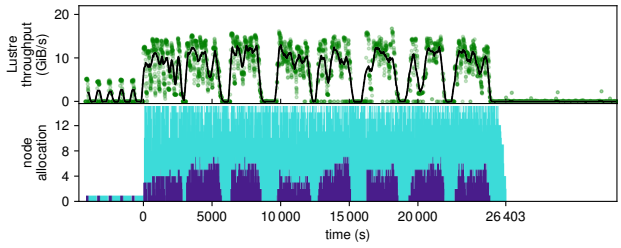
Our I/O-aware approach ensures that the estimated throughput within the schedule never exceeds the file system bandwidth. We implemented it by modifying the Slurm's backfill scheduling plugin, as well as making other adjustments to Slurm. Algorithm 2 outlines the procedure `InitializeReservationTracker` of the I/O-aware scheduler. Conceptually, this approach treats Lustre throughput as another "regular" cluster-wide resource, similar to Slurm's licenses. However, an important difference lies in employing predicted resource requirements, as well as measured current total resource utilization, both obtained from analytical services at the beginning of each scheduling round (Lines 1 and 2). Accounting for the most recent estimates not only alleviates the burden on users to provide the data but also helps address the inherent dependence of these job parameters on running conditions. By incorporating the measured total throughput, we compensate for potential underestimations of Lustre resource utilization. For instance, this approach helps mitigate possible file system overload caused by new jobs lacking historical data for predicting resource requirements, because we select the highest current bandwidth utilization among the values computed by the two methods, as described



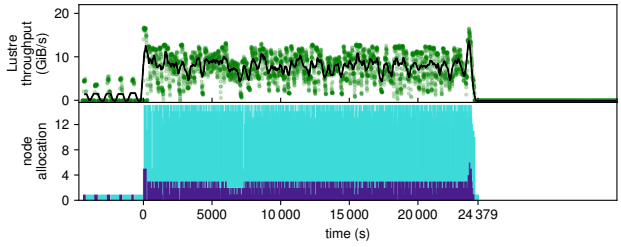
(a) Default Slurm scheduling



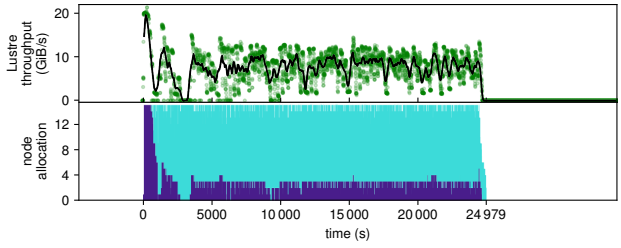
(b) I/O-aware with 20 GiB/s limit, "pre-trained"



(c) I/O-aware with 15 GiB/s limit, "pre-trained"



(d) Adaptive with 20 GiB/s limit, "pre-trained"



(e) Adaptive with 20 GiB/s limit, "untrained"

Fig. 3: Representative results of scheduling "*Workload 1*."

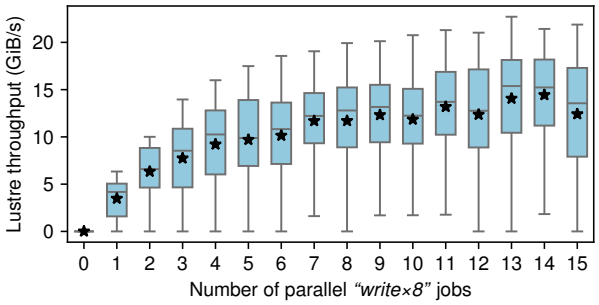


Fig. 4: Lustre total throughput as the number of concurrent "*write×8*" jobs varies from 0 to 15 (box plot).

in Line 7. The procedures `EarliestStartTime` (Algorithm 4) and `ReserveResources` (Algorithm 3) are fundamentally equivalent to other implementations of two-resource backfill scheduling, such as AI4IO Suite (Section VIII).

Algorithm 3. Conceptual outline of I/O-aware scheduler's `ReserveResources`

Input: Object $\{\mathrm{NT}, \mathrm{LT}\}$, job j, reservation start time t
1 Reserve n_j in NT for interval $[t, t + L_j)$.
2 Reserve r_j in LT for interval $[t, t + L_j)$.

Fig. 3(b) demonstrates the results of scheduling "*Workload 1*" using the I/O-aware scheduler with Lustre throughput limit of 20 GiB/s. The estimator is "pre-trained" by running jobs in isolation, to illustrate the scheduler performance with good data. At the beginning of the experiment, the scheduler dispatches fewer than six "*write×8*" jobs but later increases the number of simultaneous "*write×8*" jobs because the jobs' throughput estimate decreases due to the file system congestion. Running more "*write×8*" jobs intensifies the congestion, resulting in a further decrease in the estimate and, consequently, allowing even more "*write×8*" jobs to run. Eventually, this process stabilizes, although the number of concurrently running "*write×8*" jobs continues to fluctuate over time. Overall, the I/O-aware scheduler yields a 10% improvement of the total workload runtime (also referred to as makespan) compared to default Slurm scheduling [Fig. 3(a)].

Algorithm 4. Conceptual outline of I/O-aware scheduler's `EarliestStartTime`

Input: Object $\{\mathrm{NT}, \mathrm{LT}\}$, job j, initial reservation start time $t_{\min}$
Options: Total count of nodes N, throughput limit R_{limit}
Output: Earliest time all resources are available t
1 $t \leftarrow t_{\min}$
2 **repeat**
3 $\quad t_{\mathrm{NT}} \leftarrow$ earliest time not earlier than t when less than $N - n_j$ is reserved in NT for interval $[t, t + L_j)$
4 $\quad t \leftarrow$ earliest time not earlier than t_{NT} when less than $R_{\text{limit}} - r_j$ is reserved in LT for interval $[t, t + L_j)$
5 **until** $t = t_{\mathrm{NT}}$

A stricter throughput limit of 15 GiB/s leads to greater improvement in the total runtime of the workload. The outcome is presented in Fig. 3(c), indicating a considerable 20% improvement of the total runtime over the default Slurm scheduler. However, the reduced throughput limit may not yield benefits for every workload [31], which motivated our development of scheduling methods that dynamically reevaluate resource limits based on the job queue's composition, as described in the following section.

VII. Workload-adaptive Scheduling

Workload-adaptive scheduling aims to maintain a consistent I/O throughput level throughout the execution of the job queue [31]. This adaptive approach calculates the desired level from the estimated resource requirements of the jobs in the queue, hence its name. The target throughput is the one that allows all I/O operations ($\sum r_j d_j$ bytes in total) complete within the minimum time required to allocate the other resources—that is, the nodes in our scenario—to all jobs ($\sum n_j d_j / N$):

$$\widetilde{R} = \sum r_j d_j N \Big/ \sum n_j d_j \ . \tag{1}$$

Similar to the I/O-aware scheduler, the workload-adaptive scheduler prevents the total throughput from exceeding a given limit at any time by monitoring bandwidth utilization and reserving I/O throughput for delayed jobs. In addition, the workload-adaptive scheduler avoids scheduling I/O-heavy jobs during time intervals where the targeted throughput $\widetilde{R}$ has already been reached.

Fig. 3(d) displays the results of scheduling "*Workload 1*" with the workload-adaptive scheduling approach, following the same "pre-training" process as previously described. The scheduler converges to the optimal state where 2-3 "*write*×*8*" jobs run simultaneously, maintaining the total Lustre throughput close to 10 GiB/s. This approach results in 26% improvement of the total workload runtime over the default Slurm backfill scheduler.

The workload-adaptive scheduling converges to the optimal state even without prior information about the job resource requirements. Fig. 3(e) illustrates the scheduler's performance when it is not pre-trained by running jobs in isolation. Initially, having no estimates of the job resource requirements, the scheduler assumed zero Lustre throughput for all jobs. However, it promptly transitions from a scheduling approach similar to the default Slurm scheduler to one that maintains stable I/O throughput for the remainder of the experiment. Despite the initial less efficient scheduling, the total runtime was 5.5% shorter than that of the non-adaptive I/O-aware scheduler with the throughput limit of 15 GiB/s, and 25% shorter than the default Slurm scheduler.

A. Two-group Approximation

The "naïve" workload-adaptive scheduler, which simply refrains from scheduling jobs that utilize the file system once the targeted throughput is attained, exhibits effective scheduling for "*Workload 1*." This is because the workload contains a sufficient number of jobs that do not use the file system. Such a scheduler may succeed with real HPC cluster workloads because a significant proportion of HPC jobs exhibit minimal or no I/O activity [32]. Nevertheless, in a more general scenario this simplistic approach may result in idle nodes, causing performance degradation, when the number of non-I/O jobs in the queue is not sufficient to occupy the remaining nodes once the targeted throughput $\widetilde{R}$ is reached. To ensure efficiency, the algorithm must sustain a consistent throughput without rendering nodes idle.

We introduce a simplified approach, which we refer to as the "two-group" approximation algorithm, to address this non-trivial problem. Its primary concept involves categorizing waiting jobs into two groups according to their file system usage. Jobs with minimal usage are considered "zero jobs" and the remaining jobs are designated as "regular jobs":

$$\begin{aligned} \text{Zero Jobs} &= \{ j \in \mathcal{Q} : \ \ r_j \le n_j r^* \} \\ \text{Regular Jobs} &= \{ j \in \mathcal{Q} : \ \ r_j > n_j r^* \} . \end{aligned}$$

The threshold r^* defines the process of categorizing the jobs into the groups. Finding the value of the threshold that minimizes node idle time while maintaining the throughput sufficiently to optimize the performance is not straightforward because the optimized parameters depend on the threshold in a non-linear and possibly even non-monotonic way. One method for determining the value of this parameter is to consider the quality-of-service conditions. For example, in this study, we guarantee that at least half of the total node-hours in the waiting queue are not delayed as a result of efforts to regulate the file system throughput. In other words, we set threshold r^* in such a way that:

$$\sum_{j \,\in \text{ Zero Jobs}} n_j d_j \ge \sum_{j \,\in \text{ Regular Jobs}} n_j d_j \, . \tag{2}$$

After establishing the threshold, we identify "zero jobs" and assume they have no Lustre throughput. To keep the "two-group" approximation close on average to the original problem, we also modify other parameters. We compute the average load of the "zero jobs":

$$\overline{r_{\text{zero}}} = \sum_{j \,\in \text{ Zero Jobs}} r_j n_j d_j \Big/ \sum_{j \,\in \text{ Zero Jobs}} n_j d_j \ , \tag{3}$$

and recalculate the target load and job requirements for the "regular jobs":

$$\widetilde{R}\,' = \widetilde{R} - N\,\overline{r_{\text{zero}}} \, , \tag{4}$$

$$r_j{}' = \begin{cases} 0, & j \ \in \text{ Zero Jobs} \\ r_j - n_j\,\overline{r_{\text{zero}}}, & j \ \in \text{ Regular Jobs} \, . \end{cases} \tag{5}$$

This way, maintaining $\sum r_j{}'$ close to $\widetilde{R}\,'$ is equivalent, when time-averaged, to maintaining $\sum r_j$ close to $\widetilde{R}$.

The schematic of the workload-adaptive scheduling is summarized in Algorithms 5–7. Lines 3–5 of Algorithm 5 describe the computation of $\widetilde{R}$ following the ideas of (1) but accounting for both the jobs in the wait queue and the remaining portions

Algorithm 5. Conceptual outline of workload-adaptive scheduler's `InitializeReservationTracker`

Input: Running jobs $\mathcal{R}$, waiting jobs $\mathcal{Q}$, current time t_{now}
Output: $\{\text{RT}, \text{AT}, r^*, \overline{r_{\text{zero}}}, \widetilde{R}'\}$

1 Obtain the latest values of r_j and d_j for $j \in \mathcal{R} \bigcup \mathcal{Q}$.
2 Initialize an I/O-aware tracker RT using Algorithm 2.
3 $V_{\text{I/O}} \leftarrow \sum_{\substack{j \in \mathcal{R}, \\ t_{\text{now}} < b_j + d_j}} r_j \left(b_j + d_j - t_{\text{now}}\right) + \sum_{j \in \mathcal{Q}} r_j d_j$
4 $T_{\text{nodes}} \leftarrow \sum_{\substack{j \in \mathcal{R}, \\ t_{\text{now}} < b_j + d_j}} \frac{n_j}{N} \left(b_j + d_j - t_{\text{now}}\right) + \sum_{j \in \mathcal{Q}} \frac{n_j}{N} d_j$
5 $\widetilde{R} \leftarrow \frac{V_{\text{I/O}}}{T_{\text{nodes}}}$
6 Determine minimum threshold r^*, e.g. to satisfy (2).
7 Compute $\overline{r_{\text{zero}}}$ using (3).
8 Compute $\widetilde{R}'$ using (4).
9 Create a new tracker AT to monitor adjusted Lustre throughput reservations.
10 **for** $j \in \mathcal{R}$ **do**
11 | Reserve $r_j - n_j \overline{r_{\text{zero}}}$ in AT for interval $[b_j, b_j + L_j)$.

Algorithm 6. Conceptual outline of workload-adaptive scheduler's `ReserveResources`

Input: Object $\{\text{RT}, \text{AT}, r^*, \overline{r_{\text{zero}}}, \widetilde{R}'\}$, job j, reservation start time t

1 RT.`ReserveResources`(j, t)
2 **if** $r_j > n_j r^*$ **then** /* *"Regular job"* */
3 | Reserve $r_j - n_j \overline{r_{\text{zero}}}$ in AT for interval $[t, t + L_j)$.

of the running jobs. Lines 6–8 correspond to computation of the other parameters using (2)–(4). Lines 9–11 demonstrate initialization of an additional tracker used to determine when $\sum r_j{}'$ reaches $\widetilde{R}'$. "Regular jobs" are not scheduled to run during the time this target is reached (Lines 4–8 of Algorithm 7). Additionally, the scheduler controls, in the same manner as the I/O-aware scheduler in Section VI, that the utilization of the resources, including the file system throughput, does not exceed their limits.

In order to evaluate the effectiveness of this method, we conduct experiments involving the scheduling of a workload comprised of six different job types, each with its own file system usage. This workload, referred to as "*Workload 2*," is structured into waves, similar to "*Workload 1*" but totaling 5 waves in all. A wave is a sequence of phases, each phase is made up of jobs of the same type. Specifically, every wave starts with 30 "*write×8*" jobs, which are the same jobs used in "*Workload 1*," each job executing eight parallel threads and writing a total of 80 GiB per job. Following these jobs are 30 "*write×6*" jobs, which run six parallel threads on one node, with each thread also generating 10 GiB of data written to a randomly chosen Lustre volume (60 GiB per job). Subsequently, 30 "*write×4*" jobs are submitted (four threads, 40 GiB per job), followed by 70 "*write×2*" jobs (two threads, 20 GiB per job), and 120 "*write×1*" jobs (one thread writing 10 GiB to a randomly selected Lustre volume). Finally, the wave concludes with 30 "*sleep*" jobs, each idling for 600 s on one node, replicating the "*sleep*" jobs of "*Workload 1*." The workload contains 1550 jobs in total. To "pre-train" the estimator, representative jobs are run in isolation before the main stage of the experiment.

Algorithm 7. Conceptual outline of workload-adaptive scheduler's `EarliestStartTime`

Input: Object $\{\text{RT}, \text{AT}, r^*, \overline{r_{\text{zero}}}, \widetilde{R}'\}$, job j, initial reservation start time t_{min}
Output: Earliest time all resources are available t

1 **if** $r_j \leq n_j r^*$ **then** /* *"Zero job"* */
2 | $t \leftarrow$ RT.`EarliestStartTime`(j, t_{min})
3 **else** /* *"Regular job"* */
4 | $t \leftarrow t_{\text{min}}$
5 | **repeat**
6 | | $t_{\text{RT}} \leftarrow$ RT.`EarliestStartTime`(j, t)
7 | | $t \leftarrow$ earliest time not earlier than t_{RT} when no more than $\widetilde{R}'$ is reserved in AT for interval $[t, t + L_j)$
8 | **until** $t = t_{\text{RT}}$

As demonstrated in Fig. 5(a), the default Slurm scheduler dispatches the jobs of this workload in the order they are submitted.[2] The I/O-aware scheduler with a throughput limit of 20 GiB/s results in a very similar scheduling pattern [Fig. 5(b)], but nevertheless, manages to make the Lustre throughput somewhat more uniform and improve the total workload runtime. We repeated the experiments multiple times to account for the variability of the results. Fig. 6 summarizes the measured total runtime of the experiments. It illustrates a commonly observed high variability of parallel file system performance in HPC clusters [7], [33]. Because the runtime distribution is skewed, we use the median as the measure of central tendency. The median of the workload runtime achieved by the I/O-aware scheduler with the throughput limit of 20 GiB/s is 4% less than the runtime attained by the default Slurm scheduler. Decreasing the throughput limit to 15 GiB/s helps the I/O-aware scheduler further improve the workload runtime. The median runtime in this case is 7% less than the value obtained by the default Slurm scheduler (Fig. 6). However, with the throughput limit of 15 GiB/s, the I/O-aware scheduler quickly runs out of "*sleep*" jobs it uses to occupy the remaining nodes when the throughput limit is reached, causing idle compute nodes in the second half of the experiment [Fig. 5(c)]. For this workload, the total runtime still improves over the value obtained by the I/O-aware scheduler with the higher throughput limit. However, in other situations it could result in performance degradation.

The workload-adaptive scheduler using the throughput limit of 20 GiB/s is able to further improve the overall system performance. The median workload runtime decreases by 5% over the best I/O-aware scheduler's configuration and by 12% over the default Slurm scheduler (Fig. 6). These results demonstrate the effectiveness of our approach. Fig. 5(d) confirms that the workload-adaptive scheduling with the "two-group" approximation avoids idling nodes while keeping the file system throughput reasonably close to the target.

[2] Although our scheduler fully implements backfill scheduling, this experiment shows no noticeable signs of backfill because each job requires one node and no other resource.

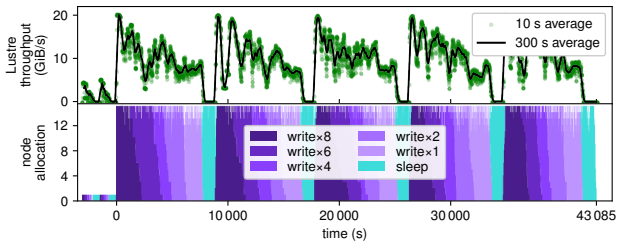
Interestingly, although the target throughput for "*Workload 2*" stays above 15 GiB/s throughout most of the experiment, the "two-group" approximation technique helps reduce the node idle time even when the scheduler's Lustre throughput limit is set to 15 GiB/s, as seen on Fig. 5(e). In this case, the median workload runtime of the workload-adaptive scheduler is about 3% less than the result of the I/O-aware scheduler with the same throughput limit.

VIII. Relevant Work

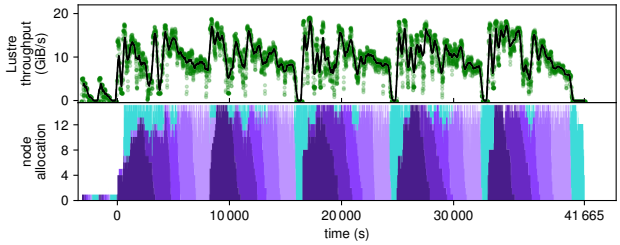
The conventional approach of constraining job scheduling so that the throughput does not surpass the bandwidth limit, which constitutes our I/O-aware scheduling, has been previously documented. Notably, AI4IO Suite [33] includes an I/O-aware scheduler and a resource utilization estimator. AI4IO also provides a "canary" application designed to detect intermittent file system degradation events. Although there are similarities with our approach, AI4IO employs Flux [34], an HPC management software that is still in the development stage. In contrast, our implementation is designed to operate with Slurm, a well-established and popular HPC cluster resource manager. Moreover, unlike our workload-adaptive scheduling method, AI4IO and other existing implementations of I/O-aware scheduling have no capability to mitigate I/O contention by attaining more uniform I/O throughput. Additionally, no other I/O-aware scheduler implements techniques such as our "two-group" approximation to reduce idle nodes.

Ideas similar to the workload-adaptive scheduling approach were presented in previous works that deal with process scheduling on multi-core systems. [35] proposed a technique to alleviate memory bus contention by regulating throughput to match the average bandwidth requirement. Furthermore, [36] introduced a concept of "fitness" of processes in order to balance the utilization of CPUs and memory bandwidth during scheduling, a goal akin to that of our "two-group" approximation method. Though these techniques are inspirational, they are not directly applicable to HPC job scheduling. Unlike HPC job scheduling, process scheduling only needs to decide which processes to run immediately without resource reservation or job backfilling considerations. Therefore, the problem of scheduling HPC jobs differs fundamentally from the problem of scheduling processes on CPU cores.

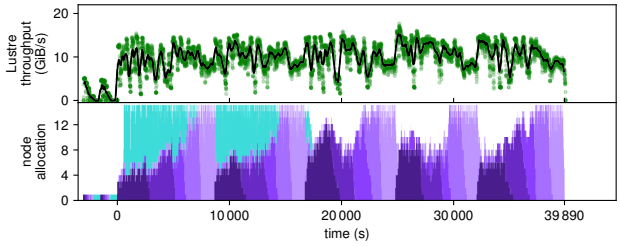
Several multi-resource scheduling techniques aimed at enhancing datacenter efficiency also do not account for resource reservations and backfilling. This relaxation transforms the scheduling into a vector bin packing problem. [37] demonstrated that best results in such scenarios are achieved by employing "dot-product" or "L2-norm" heuristics to choose the order of job resource allocation attempts. [38] implemented the "dot-product" heuristics in a Yarn scheduler referred to as "TETRIS." [39] suggest a three-stage scheduling algorithm that relies heavily on detailed information about arriving jobs, including the percentage of jobs and resource requirements of each job type. When such information is assumed to be available, the proposed algorithm outperforms "TETRIS." These techniques are also not applicable to HPC job scheduling,



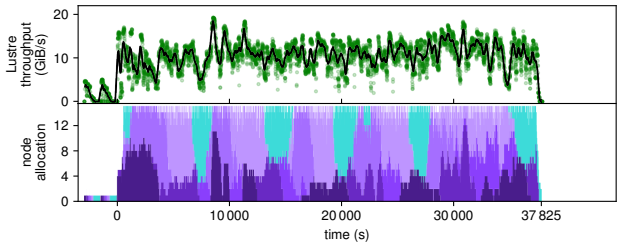
(a) Default Slurm scheduling



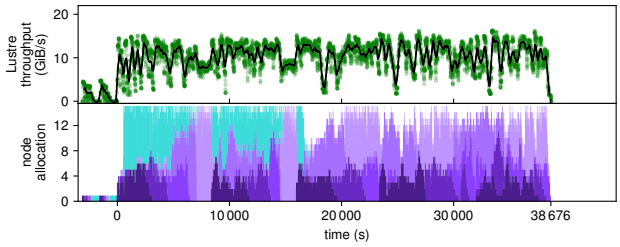
(b) I/O-aware with 20 GiB/s limit



(c) I/O-aware with 15 GiB/s limit



(d) Adaptive with 20 GiB/s limit



(e) Adaptive with 15 GiB/s limit

Fig. 5: Representative results of scheduling "*Workload 2*."

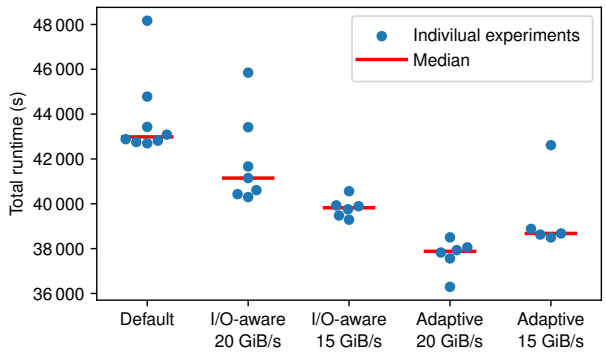


Fig. 6: Summary of results of scheduling "*Workload 2*" (swarm plot and median for each scheduler configuration).

which requires resource reservations and backfill to enforce job priorities.

[18] propose a scheduling approach for HPC clusters that defines a "window" of jobs at the head of the queue and schedules the jobs in the "window" without considering job order, effectively converting the scheduling problem into a vector bin packing one. The authors present a mechanism to prevent job starvation; however, they did not validate its effectiveness. Therefore, this approach may be less favorable among HPC cluster practitioners compared to the well-established backfill approach. Similarly, the "pack-scheduling" technique [18], [40], which may delay jobs while required resources are idle, may struggle to find acceptance within the HPC community.

[41] develop heuristics that improve the performance of EASY backfill algorithm ($\mathrm{BackfillMax} = 1$ as described in Section II-A) in the case of multiple resources. Although it could be an alternative to our "two-group" approximation, the approach is not applicable for $\mathrm{BackfillMax} > 1$ and considers only the case of a single machine. [42] evaluate heuristics for reducing I/O utilization conflicts, which could be an alternative to our workload-adaptive approach, but their work also considers EASY backfill algorithm only and is not applicable for $\mathrm{BackfillMax} > 1$. Furthermore, they evaluated their approach via simulations using their own runtime model and did not confirm the results with real-world experiments.

An orthogonal approach to our work deals with the scheduling of I/O calls of co-running jobs [8], [43]–[46]. This direction is thoroughly reviewed by [40]. Optimizing the I/O call scheduling reduces but cannot fully remove the effects of the file system congestion. Our workload-adaptive job scheduling is, therefore, complimentary to such studies, as it reduces the need for I/O call scheduling.

IX. Conclusions

Through a set of Slurm plugins and auxiliary modules, we have implemented I/O-aware scheduling in Slurm without imposing additional burdens on users. In addition to the straightforward I/O-aware scheduling, which imposes limits on the I/O throughput, our implementation introduces the workload-adaptive approach. This approach aims to maintain the throughput at a level computed based on the overall resource requirement of the job queue. Our workload-adaptive scheduler, requiring no manual tuning for the workload, usually outperforms both the default Slurm scheduler and the conventional I/O-aware scheduler. We showcase the advantages of our approach through an evaluation conducted on an actual HPC cluster system; similar improvements have been observed in experiments conducted through simulations [31], [47] and on a cluster of virtual machines [47]. The workload-adaptive scheduler is expected to enhance performance in all scenarios where the relationship between throughput and load is concave [31]. Fig. 4 illustrates such a relationship for the studied Lustre file system. A similar concave profile is anticipated for most parallel file systems.

We augment the workload-adaptive scheduling with our "two-group" approximation technique to eliminate the scheduler's reliance on zero-throughput jobs being available for scheduling on nodes that otherwise would remain idle once the target throughput is reached. The "two-group" approximation is based on treating a pre-defined fraction of the job queue as zero-throughput jobs. We demonstrate the efficacy of scheduling with the "two-group" approximation in scenarios where the workload contains few zero-throughput jobs.

In our evaluation, the workload-adaptive scheduler gains 12% to 26% in efficiency compared to the default Slurm scheduler and outperforms the I/O-aware scheduler by 5% to 8%. Whereas the improvements provided by the workload-adaptive scheduling may appear modest at times, their significance becomes apparent when considering the scale of the HPC computing industry. Since our workloads were not specifically designed to maximize the benefits of the proposed methods, the reported values serve as rough estimates of the potential advantages of workload-adaptive scheduling, rather than definitive limits. Real-world results could differ, especially because many HPC cluster file systems are over-provisioned to handle peak access rates. However, for workloads with little I/O bandwidth utilization, the proposed approach will closely resemble the standard Slurm behavior, thus causing no performance degradation. In the next iteration of our prototype, we aim to further reduce the overhead associated with scheduling minimally used resources. Additionally, we plan to expand the features for monitoring file system usage. With these enhancements, adopting the system will have no downside. Once it gains popularity, the data generated while using the system may inform the design of I/O components for future HPC clusters.

Acknowledgment

Some materials contained in this paper were previously presented in PhD thesis by Goponenko [47]. The authors thank Kenneth Lamar, Ramin Izadpanah, Christina Peterson, Zachary Painter, Benjamin Schwaller, and Omar Aaziz for their valuable suggestions. The works at the University of Central Florida were supported by contracts with Sandia National Laboratories.

References

[1] M. Etinski, J. Corbalan, J. Labarta, and M. Valero, "Linear programming based parallel job scheduling for power constrained systems," in *2011 International Conference on High Performance Computing Simulation*, (Istanbul, Turkey), pp. 72–80, IEEE, July 2011.

[2] N. Ensmenger, "The Environmental History of Computing," *Technology and Culture*, vol. 59, no. 4, pp. S7–S33, 2018.

[3] S. Atchley, C. Zimmer, J. Lange, D. Bernholdt, V. Melesse Vergara, T. Beck, M. Brim, R. Budiardja, S. Chandrasekaran, M. Eisenbach, T. Evans, M. Ezell, N. Frontiere, A. Georgiadou, J. Glenski, P. Grete, S. Hamilton, J. Holmen, A. Huebl, D. Jacobson, W. Joubert, K. Mcmahon, E. Merzari, S. Moore, A. Myers, S. Nichols, S. Oral, T. Papatheodore, D. Perez, D. M. Rogers, E. Schneider, J.-L. Vay, and P. K. Yeung, "Frontier: Exploring Exascale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, (New York, NY, USA), pp. 1–16, Association for Computing Machinery, Nov. 2023.

[4] U.S. Energy Information Administration, "Electric Power Monthly." https://www.eia.gov/electricity/monthly/epm_table_grapher.php.

[5] H. Ritchie, P. Rosado, and M. Roser, "Energy Production and Consumption," *Our World in Data*, Feb. 2024.

[6] TOP500.org, "Performance Development | TOP500." https://www.top500.org/statistics/perfdevel/, 2023.

[7] B. Xie, J. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, and N. Podhorszki, "Characterizing output bottlenecks in a supercomputer," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pp. 1–11, Nov. 2012.

[8] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir, "Scheduling the I/O of HPC Applications Under Congestion," in *2015 IEEE International Parallel and Distributed Processing Symposium*, pp. 1013–1022, May 2015.

[9] G. K. Lockwood, S. Snyder, T. Wang, S. Byna, P. Carns, and N. J. Wright, "A Year in the Life of a Parallel File System," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 931–943, Nov. 2018.

[10] S.-M. Tseng, B. Nicolae, F. Cappello, and A. Chandramowlishwaran, "Demystifying asynchronous I/O Interference in HPC applications," *The International Journal of High Performance Computing Applications*, vol. 35, pp. 391–412, July 2021.

[11] C. Galleguillos, A. Sîrbu, Z. Kiziltan, O. Babaoglu, A. Borghesi, and T. Bridi, "Data-Driven Job Dispatching in HPC Systems," in *Machine Learning, Optimization, and Big Data* (G. Nicosia, P. Pardalos, G. Giuffrida, and R. Umeton, eds.), Lecture Notes in Computer Science, (Cham), pp. 449–461, Springer International Publishing, 2018.

[12] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Modeling User Runtime Estimates," in *Job Scheduling Strategies for Parallel Processing* (D. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 1–35, Springer, 2005.

[13] E. Gaussier, D. Glesser, V. Reis, and D. Trystram, "Improving backfilling by using machine learning to predict running times," in *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, (Austin, TX, USA), pp. 1–10, IEEE, Nov. 2015.

[14] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job Scheduling Strategies for Parallel Processing* (D. Feitelson, L. Rudolph, and U. Schwiegelshohn, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 44–60, Springer, 2003.

[15] D. A. Lifka, "The ANL/IBM SP scheduling system," in *Job Scheduling Strategies for Parallel Processing* (D. G. Feitelson and L. Rudolph, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 295–303, Springer, 1995.

[16] SchedMD, "Slurm Workload Manager - Trackable RESources (TRES)." https://slurm.schedmd.com/tres.html, May 2018.

[17] SchedMD, "Release notes for Slurm version 22.05." https://slurm.schedmd.com/archive/slurm-22.05.0/news.html, May 2022.

[18] Y. Fan, Z. Lan, P. Rich, W. E. Allcock, M. E. Papka, B. Austin, and D. Paul, "Scheduling Beyond CPUs for HPC," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '19, (New York, NY, USA), pp. 97–108, Association for Computing Machinery, June 2019.

[19] Y. Kim, R. Gunasekaran, G. M. Shipman, D. A. Dillow, Z. Zhang, and B. W. Settlemyer, "Workload characterization of a leadership class storage cluster," in *2010 5th Petascale Data Storage Workshop (PDSW '10)*, pp. 1–5, Nov. 2010.

[20] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and Improving Computational Science Storage Access through Continuous Characterization," *ACM Transactions on Storage*, vol. 7, pp. 8:1–8:26, Oct. 2011.

[21] A. Kougkas, M. Dorier, R. Latham, R. Ross, and X.-H. Sun, "Leveraging burst buffer coordination to prevent I/O interference," in *2016 IEEE 12th International Conference on E-Science (e-Science)*, pp. 371–380, Oct. 2016.

[22] E. Rosti, G. Serazzi, E. Smirni, and M. S. Squillante, "The impact of I/O on program behavior and parallel scheduling," in *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '98/PERFORMANCE '98, (New York, NY, USA), pp. 56–65, Association for Computing Machinery, June 1998.

[23] E. Costa, T. Patel, B. Schwaller, J. M. Brandt, and D. Tiwari, "Systematically Inferring I/O Performance Variability by Examining Repetitive Job Behavior," in *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*, (St. Louis, MO, USA), pp. 1–15, IEEE, Nov. 2021.

[24] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden, M. Rajan, M. Showerman, J. Stevenson, N. Taerat, and T. Tucker, "The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications," in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 154–165, Nov. 2014.

[25] R. Gibbons, "A historical application profiler for use by parallel schedulers," in *Job Scheduling Strategies for Parallel Processing* (D. G. Feitelson and L. Rudolph, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 58–77, Springer, 1997.

[26] K. Lamar, A. Goponenko, O. Aaziz, B. A. Allan, J. M. Brandt, and D. Dechev, "Evaluating HPC Job Run Time Predictions Using Application Input Parameters," in *Proceedings of the 17th ACM International Conference on Distributed and Event-based Systems*, DEBS '23, (New York, NY, USA), pp. 127–138, Association for Computing Machinery, June 2023.

[27] M. R. Wyatt, S. Herbein, T. Gamblin, A. Moody, D. H. Ahn, and M. Taufer, "PRIONN: Predicting Runtime and IO using Neural Networks," in *Proceedings of the 47th International Conference on Parallel Processing*, ICPP 2018, (Eugene, OR, USA), pp. 1–12, Association for Computing Machinery, Aug. 2018.

[28] K. Lamar, A. Goponenko, C. Peterson, B. A. Allan, J. M. Brandt, and D. Dechev, "Backfilling HPC Jobs with a Multimodal-Aware Predictor," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, (Portland, OR, USA), pp. 618–622, IEEE, Sept. 2021.

[29] E. R. Rodrigues, R. L. F. Cunha, M. A. S. Netto, and M. Spriggs, "Helping HPC Users Specify Job Memory Requirements via Machine Learning," in *2016 Third International Workshop on HPC User Support Tools (HUST)*, pp. 6–13, Nov. 2016.

[30] B. A. Allan, M. Aguilar, B. Schwaller, and S. Langer, "LDMS Monitoring of EDR InfiniBand Networks," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 459–463, Sept. 2020.

[31] A. V. Goponenko, R. Izadpanah, J. M. Brandt, and D. Dechev, "Towards workload-adaptive scheduling for HPC clusters," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 449–453, Sept. 2020.

[32] J. P. White, A. D. Kofke, R. L. DeLeon, M. Innus, M. D. Jones, and T. R. Furlani, "Automatic Characterization of HPC Job Parallel Filesystem I/O Patterns," in *Proceedings of the Practice and Experience on Advanced Research Computing*, PEARC '18, (New York, NY, USA), pp. 1–8, Association for Computing Machinery, July 2018.

[33] M. R. Wyatt, S. Herbein, T. Gamblin, and M. Taufer, "AI4IO: A suite of AI-based tools for IO-aware scheduling," *The International Journal of High Performance Computing Applications*, vol. 36, pp. 370–387, May 2022.

[34] D. H. Ahn, N. Bass, A. Chu, J. Garlick, M. Grondona, S. Herbein, H. I. Ingólfsson, J. Koning, T. Patki, T. R. W. Scogland, B. Springmeyer, and M. Taufer, "Flux: Overcoming scheduling challenges for exascale

workflows," *Future Generation Computer Systems*, vol. 110, pp. 202–213, Sept. 2020.

[35] D. Xu, C. Wu, and P.-C. Yew, "On mitigating memory bandwidth contention through bandwidth-aware scheduling," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10, (Vienna, Austria), pp. 237–248, Association for Computing Machinery, Sept. 2010.

[36] C. D. Antonopoulos, D. S. Nikolopoulos, and T. S. Papatheodorou, "Scheduling Algorithms with Bus Bandwidth Considerations for SMPs," in *High-Performance Computing*, ch. 16, pp. 313–332, John Wiley & Sons, Ltd, 2005.

[37] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for Vector Bin Packing," tech. rep., Microsoft, Jan. 2011.

[38] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, (New York, NY, USA), pp. 455–466, Association for Computing Machinery, Aug. 2014.

[39] T. T. Tran, M. Padmanabhan, P. Y. Zhang, H. Li, D. G. Down, and J. C. Beck, "Multi-stage resource-aware scheduling for data centers with heterogeneous servers," *Journal of Scheduling*, vol. 21, pp. 251–267, Apr. 2018.

[40] E. Jeannot, G. Pallez, and N. Vidal, "IO-aware Job-Scheduling: Exploiting the Impacts of Workload Characterizations to select the Mapping Strategy," *The International Journal of High Performance Computing Applications*, p. 10943420231175854, May 2023.

[41] W. Leinberger, G. Karypis, and V. Kumar, "Job Scheduling in the presence of Multiple Resource Requirements," in *SC '99: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*, (Portland, Oregon, USA), pp. 47–47, IEEE, Nov. 1999.

[42] F. Guim, I. Rodero, and J. Corbalan, "The Resource Usage Aware Backfilling," in *Job Scheduling Strategies for Parallel Processing* (E. Frachtenberg and U. Schwiegelshohn, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 59–79, Springer, 2009.

[43] Z. Zhou, X. Yang, D. Zhao, P. Rich, W. Tang, J. Wang, and Z. Lan, "I/O-aware bandwidth allocation for petascale computing systems," *Parallel Computing*, vol. 58, pp. 107–116, Oct. 2016.

[44] E. Jeannot, G. Pallez, and N. Vidal, "Scheduling periodic I/O access with bi-colored chains: Models and algorithms," *Journal of Scheduling*, vol. 24, pp. 469–481, Oct. 2021.

[45] F. Boito, G. Pallez, L. Teylo, and N. Vidal, "IO-Sets: Simple and Efficient Approaches for I/O Bandwidth Management," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, pp. 2783–2796, Oct. 2023.

[46] Z. Zhou, X. Yang, D. Zhao, P. Rich, W. Tang, J. Wang, and Z. Lan, "I/O-Aware Batch Scheduling for Petascale Computing Systems," in *2015 IEEE International Conference on Cluster Computing*, pp. 254–263, Sept. 2015.

[47] A. V. Goponenko, *Objective-Driven Strategies for HPC Job Scheduling*. PhD thesis, University of Central Florida, Orlando, FL, USA, Aug. 2024.