The Journal of Supercomputing
https://doi.org/10.1007/s11227-019-03004-3

A hybrid scheduling platform: a runtime prediction reliability aware scheduling platform to improve HPC scheduling performance

Mina Naghshnejad[1] · **Mukesh Singhal**[1]

Abstract
The performance of scheduling algorithms for HPC jobs highly depends on the accuracy of job runtime values. Prior research has established that neither user-provided runtimes nor system-generated runtime predictions are accurate. We propose a new scheduling platform that performs well in spite of runtime uncertainties. The key observation that we use for building our platform is the fact that two important classes of scheduling strategies (backfilling and plan based) differ in terms of sensitivity to runtime accuracy. We first confirm this observation by performing trace-based simulations to characterize the sensitivity of different scheduling strategies to job runtime accuracy. We then apply gradient boosting tree regression as a meta-learning approach to estimate the reliability of the system-generated job runtimes. The estimated prediction reliability of job runtimes is then used to choose a specific class of scheduling algorithm. Our hybrid scheduling platform uses a plan-based scheduling strategy for jobs with high expected runtime accuracy and backfills the remaining jobs on top of the planned jobs. While resource sharing is used to minimize fragmentation of resources, a specific ratio of CPU cores is reserved for backfilling of less predictable jobs to avoid starvation of these jobs. This ratio is adapted dynamically based on the resource requirement ratio of predictable jobs among recently submitted jobs. We perform extensive trace-driven simulations on real-world production traces to show that our hybrid scheduling platform outperforms both pure backfilling and pure plan-based scheduling algorithms.

Keywords HPC scheduling · HPC cluster · Runtime prediction · Scheduling platform · Hybrid scheduling · Estimating prediction reliability · Meta-learning · Applied machine learning

✉ Mina Naghshnejad
mnaghshnejad@ucmerced.edu

Mukesh Singhal
msinghal@ucmerced.edu

[1] Electrical Engineering and Computer Science, University of California, Merced, CA 95343, USA

Published online: 28 September 2019

Real knowledge is to know the extent of one's ignorance.
Confucius.

1 Introduction

With ever increasing usage of HPC clusters by scientific researchers, the diversity of applications submitted to HPC clusters is non-deniable. This diversity requires smarter scheduling strategies to keep resource usage efficient while meeting the cluster customers' preferred performance goals. The management of job deployment and scheduling is performed by job management systems in HPC clusters. Job management systems for HPC clusters need to perform resource allocation and job scheduling cleverly to avoid capital and operational expenses for operating the cluster. Extensive research has been done in designing scheduling platforms with better performance and higher efficiency. However, most of these algorithms require an accurate value for job runtime. It is well known that user-provided runtime estimations are far from accurate [1–3]. Several systems-generated runtime prediction approaches based on the runtime of previously executed jobs have been proposed. However, due to the inherent uncertainties in job runtime values, perfect prediction of runtimes is not possible [4]. To the best of our knowledge, no reliability estimation for individual runtime predictions has been done in the past. Our studies on HPC trace data show that although prediction approaches predict runtimes for some jobs very accurately, they provide inaccurate predictions for a subset of submission jobs. The reliability of individual job predictions provides valuable information to be used by scheduling platforms. We use a supervised machine learning approach to estimate the reliability of each job runtime prediction. In other words, we determine if a runtime prediction for each new job is reliable using a machine learning model trained on previously completed jobs. We then use our estimations of the accuracy to design a scheduling platform that deploys an appropriate scheduling strategy based on the predictability of the runtime for each job. For that purpose, we first conduct experiments in multiple HPC trace datasets and measure the sensitivity of different scheduling algorithms to job runtime accuracy. Having done so, we design a reliability estimation approach to determine the accuracy estimates, which is a valuable input for our scheduling platform.

To be able to pick the best scheduling strategy based on the reliability of job runtime prediction, it is worthwhile to study the sensitivity of job scheduling policies to job runtime accuracy. There are two well-known groups of scheduling policies for HPC jobs: backfilling policies and plan-based policies. While backfilling tries to fill the free resources created by first come first served (FCFS) ordering jobs, plan-based scheduling routinely finds the optimal/near-optimal ordering of jobs to fill the resources. In backfilling, the initial ordering of jobs limits the feasibility of resources for later shorter jobs even though backfilling is far from optimal. The popularity of backfilling is due to its acceptable performance with inaccurate runtime prediction. However, with the advent of more accurate online prediction models, more plan-based approaches have been proposed [5]. One may assume that with more accurate prediction approaches, plan-based scheduling algorithms perform best. This is

not true. One issue about using machine learning approaches for runtime prediction is the fact that machine learning models are trained based on minimizing a loss function. The loss function often minimizes average error over all training points and does not necessarily perform well for all individual data points [6]. As a result, machine learning prediction approaches may not make accurate predictions for a large subset of submitted applications in an HPC cluster. The question is whether we can determine if a machine learning model accurately predicts the runtime for a specific newly submitted job. In Sect. 4.4.2, we see that we actually can have accurate estimations of runtime prediction accuracy values for job runtimes. We call this estimation the prediction reliability estimation following some previous works in machine learning literature [6]. Using the reliability estimation, we categorize jobs into two categories of *predictable* and *unpredictable* jobs and design our hybrid scheduling platform around these two categories of jobs.

1.1 Contributions

The research presented in this paper addressed three questions. (1) Do different scheduling strategies differ in sensitivity to prediction accuracy? (2) Can we estimate the prediction reliability of individual jobs? (3) Can we improve the performance and efficiency of scheduling HPC clusters with the knowledge of the answer to the first two questions? To answer the first question, we performed sensitivity experiments with scheduling strategies from the two broad groups of backfilling and plan-based. We observed that backfilling strategies are less sensitive to runtime accuracy. We also observed that the performance of plan-based strategies significantly improves with more accurate runtime values. We then studied prediction reliability and experimented existing approaches in machine learning literature for estimating prediction reliability for job runtime values. Based on our experiments, we designed a supervised model that estimates the accuracy of runtime using gradient boosting tree regression. The Pearson correlation for our estimated accuracies and the actual accuracy is 0.84 on our test data. We then designed a hybrid scheduling platform that picks a policy for scheduling each job based on the reliability of its runtime prediction. Our platform first schedules the predictable jobs using a plan-based scheduling strategy and then backfills the less predictable jobs on the top. To avoid starvation of resources, a dynamically adaptive portion of resources is reserved for backfilling jobs. However, backfilled jobs can use all the remaining available resources to avoid resource fragmentation. Our extensive trace-based simulations show that our platform outperforms each of the two strategies deployed individually in terms of performance and efficiency.

1.2 Paper organization

We start by presenting the problem and related background in Sect. 2. We review related work for HPC scheduling as well as prediction reliability estimation in machine learning literature in Sect. 3. We present our approach for reliability

estimation and hybrid scheduling platform in Sect. 4. We present our trace-based experiments with actual data in Sect. 5. We conclude our work in Sect. 6.

2 Background and problem description

In this paper, we are interested in a solution for improving the performance of scheduling jobs in HPC clusters in the existence of job runtime inaccuracies. The HPC applications are submitted to a central application management system. The job management system decides the allocation of resources to the applications. As the resources are finite, the applications may need to wait until they acquire resources. The users are often asked to provide running time and the required CPU and memory for their applications. It is well known that the users submit an overestimated runtime mostly because the application manager kills the applications if they take longer than the user-estimated runtime [7].

In order to better present the problem, we first define our notion of a job in HPC cluster: a *job* or *application* j is considered with specific submission time and resource requirement. At the time of submission, a value of runtime and resource requirement is submitted by the user. There are n independent jobs (indexed by integers), where application j has the following characteristic:

- Submission time: r_j
- Resource requirement: d_j
- Actual running time: p_j
- Requested running time: $\hat{p}_j$.
- Additional features (descriptors) including the user that submitted the application, the time of the day the application submitted, etc.

In Fig. 1, an application j is illustrated as a solid rectangle on the right, with length equal to actual runtime (p_j) and resource requirement equal to d_j. In the figure at left, the abstraction of resources in the HPC cluster is illustrated. The vertical dimension represents the total resources in the system, and the horizontal axis denotes time. In this section, we first present a background on scheduling approaches for HPC

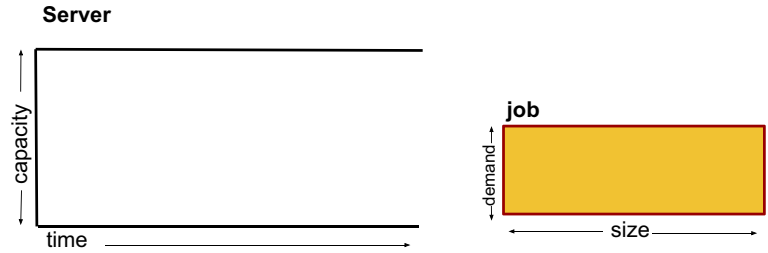

Fig. 1 Each job j is illustrated as a two-dimensional rectangle with height equal to its demand d_j and width equal to its size p_j. Each server has the unit capacity. Jobs can run simultaneously on each server as long as the total demand/height of running jobs do not exceed the server's capacity

clusters and introduce the issue of uncertainty in HPC workload traces. We then present the formulation of the problem we study in this paper.

2.1 Common scheduling algorithms for HPC workloads

FCFS is the most well-known scheduling algorithms for HPC jobs. FCFS schedules jobs in order of their submission. FCFS is a list scheduling algorithm that prioritizes jobs based on their submission time. In list scheduling algorithms, also known as queue-based scheduling algorithms, if there are enough available resources, resources are allocated to the submitted job and the job starts to process. Otherwise, the job is kept in a queue. Using FCFS does not consider the geometry of jobs to pack them tightly into resources. To improve the performance of FCFS, two strategies have been proposed: backfilling (FCFS-BF) [8] and plan-based scheduling [5] algorithms. In Fig. 2, an FCFS backfilling scheduling strategy is compared with a plan-based scheduling algorithm. While backfilling backfills the free resources available after FCFS scheduling with smaller jobs from the back of the queue, plan-based scheduling uses the runtime of jobs in the queue to find the near-optimal ordering jobs before assigning the jobs. As these two groups of scheduling algorithms are the building blocks of our hybrid scheduling platform, we elaborate their characteristics in the following subsections.

2.1.1 FCFS scheduling with backfilling

In FCFS with backfilling, jobs are prioritized based on their submission time to the system. Several variety of backfilling algorithms are proposed including EASY [9], conservative [8] and slack backfilling [10] algorithms. Although FCFS only relies on submission time to order the jobs, all these backfilling approaches rely on runtime

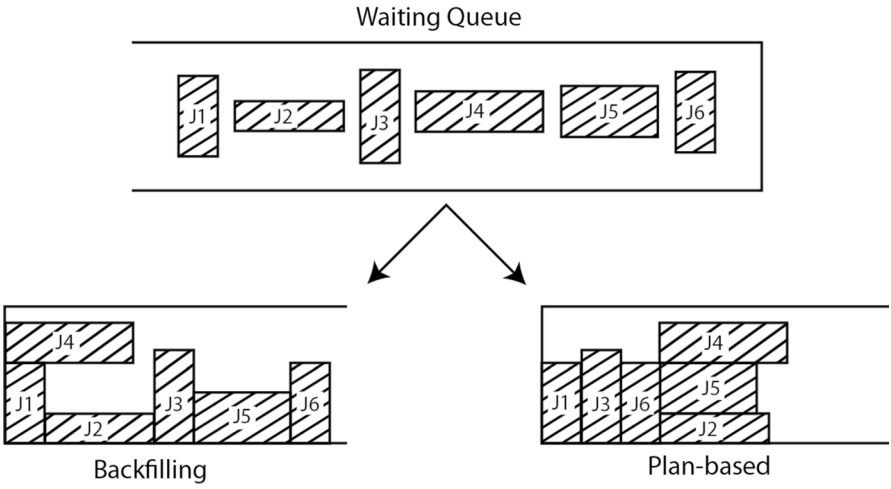

Fig. 2 The comparison of a plan-based scheduling algorithm (online-SJF) and a backfilling scheduling algorithm (FCFS-SJF) algorithms on an example of seven jobs

values to make the backfilling decision. Most of the resource management systems deployed in HPC clusters including SLURM [11], Cobalt [12], IBM LoadLeveler [9] use FCFS with backfilling. FCFS scheduling algorithms are used mainly due to their simplicity and scalability as well as stability to inaccurate input runtimes. The EASY-backfilling algorithm acts like a greedy first fit scheduler in the case that the next job in the queue has more resource demand than the available resources. It takes the first job from the back of the queue and fits into the available space.

One important observation is that as EASY-backfilling tries to backfill jobs greedily into available resources created by the FCFS ordering of jobs, its performance does not degrade substantially with inaccurate runtime estimates. On the other hand, the performance of the EASY does not improve substantially with more accurate runtime values. FCFS-SJF was proposed in [13] to use the application runtime for backfilling decision. In FCFS-SJF, the backfilled jobs are chosen in the order of increasing runtime and are commonly used in the works that propose more accurate prediction approaches to runtime prediction as SJF-BF is more sensitive to runtime prediction accuracy than EASY. We will also consider FCFS-SVF that orders jobs based on volume(multiplication of runtime and required CPU). FCFS-SJF and FCFS-SVF have some level of sensitivity to runtime accuracy, but still, have acceptable performance in the absence of accurate runtime prediction. In our experiments, we use FCFS-SJF and FCFS-SVF as representatives of FCFS with backfilling algorithms.

2.1.2 Plan-based scheduling algorithms

On the other side of the scheduling algorithms spectrum are the plan-based algorithms. Instead of deploying jobs immediately, plan-based approaches try to find a near-optimal ordering of jobs to optimize scheduling performance for a group of submitted jobs. The main issue with plan-based scheduling algorithms is the fact that their performance is highly sensitive to the accuracy of jobs' runtime predictions. As in real-world scenarios, user runtime estimates used for scheduling are not accurate; plan-based scheduling algorithms do not perform well. We study several plan-based and backfilling approaches and propose an adaptive hybrid scheduling platform. Our sensitivity analysis experiments in the next subsection show how plan-based work well with accurate and backfilling with inaccurate predictions. Plan-based scheduling algorithms try to search over the solution space to make the best scheduling decision for each job. As the problem is dynamic and jobs are submitted over time, the planning routine needs to be done periodically and based on the jobs already in the systems. Several plan-based scheduling policies have been proposed. Some policies propose a complete search over the solution space, and some propose local search to improve the computation overhead [5, 14, 15].

Although these methods claim to have better performance than backfilling scheduling methods, several issues make them less popular for cluster managers. First, for most of these approaches, the computational overhead for decision making makes them less favorable. Furthermore, they completely rely on job runtimes for their decisions and perform poorly if the runtimes are not accurate.

In this paper, for the choice of plan-based strategy in our hybrid scheduling platform, we use online priority-based scheduling algorithms. Online priority-based scheduling algorithms are the plan-based version of commonly used priority-based scheduling algorithms. Online priority-based scheduling algorithms have the high performance of plan-based scheduling algorithms on the availability of accurate runtime values while they have a low computational overhead. Shortest job first (SJF) algorithm is well known to have optimal performance in the offline case when one job is allowed to run at a time. In [16], authors proposed smallest volume first (SVF) as the two-dimensional extension of SJF that reaches near-optimal performance in the offline case. In comparison with other plan-based scheduling, including heuristic search algorithms, list-based scheduling algorithms are considerably faster. Online-SJF and online-SVF are dynamic. Similar to the other plan-based scheduling algorithms, new ordering is computed at the time of the system events. With using appropriate data structures like Heap, reordering the list takes constant time. Our experiments show that SVF outperforms SJF and other commonly used scheduling algorithms when accurate runtimes are provided. In this work, we use online-SVF as the plan-based scheduling algorithm and call it plan-based scheduling in the rest of this paper.

2.2 Sensitivity to job runtime accuracy

To study the sensitivity of backfilling and plan-based schedulers, we studied four traces from parallel workload dataset [17] in detail. These datasets contain traces of applications submitted to HPC clusters. An immediate observation, as noted by several previous works [13, 18], is inaccurate user estimates of job runtimes. In recent years, several prediction approaches are proposed [13, 19], and so some of them can predict application runtimes as accurate as 0.80. The question is how these more accurate predictions affect the scheduling performance for both customer satisfaction criterion (performance) and cluster provider criterion (utilization). To answer this question, we perturbed the available traces to provide traces with variable runtime prediction accuracy and simulated several scheduling algorithms discussed in the previous section to compare the scheduling algorithms regarding their sensitivity to accuracy. We also want to compare how different scheduling algorithms perform with more accurate runtimes. FCFS does not use runtime estimates and schedules jobs on the order of submission, so the average wait time of FCFS is not impacted by more accurate runtime prediction. The critical point to notice is the sensitivity of online-SJF and online-SVF to the accuracy of runtime. These two algorithms do not perform well with user-estimated runtimes, which are generally inaccurate.

Through our extensive trace-based simulations, we realized that plan-based scheduling algorithms are more sensitive to prediction accuracy than backfilling algorithms. The good news is that they outperform backfilling algorithms if an almost accurate prediction is available (accuracy more than 0.60). We perturbed the existing traces from ANL, LLNL, HPC2N and SDSC to build trace datasets with 0.20, 0.40, 0.60 and 0.80 runtime value accuracies and simulated backfilling and plan-based scheduling algorithms with the perturbed runtimes. The results of these

experiments are demonstrated in Fig. 3. We can see that for all four datasets, for accuracy level higher than a threshold (0.40–0.80), online-SVF outperforms all the other algorithms in terms of wait time.

2.3 Job runtime prediction reliability estimation

As discussed earlier, runtime values by users are far from accurate [13]. Several approaches have been proposed to generate more accurate runtime predictions based on available information about the job at the time of submission [20]. In [18], several time-series-based approaches are used to predict job runtimes of incoming applications. A machine learning approach was proposed in [19] where the authors propose an online learning model that makes a prediction for each newly submitted job and updates the prediction model with newly available runtime after completion of each job. However, to the best of our knowledge, there has not been any study about confidence or reliability of runtime predictions for individual jobs. In some other works, including [20] specific information about the application is used by a machine learning model to predict its performance in HPC cluster. In [21], authors use source code content to analyze the performance of the application on a simulated HPC infrastructure. In [22], authors propose state-space models to predict runtime of HPC applications. One important issue in using predicted runtimes is the fact that for some jobs, the prediction accuracy is low. Hence, there is the need

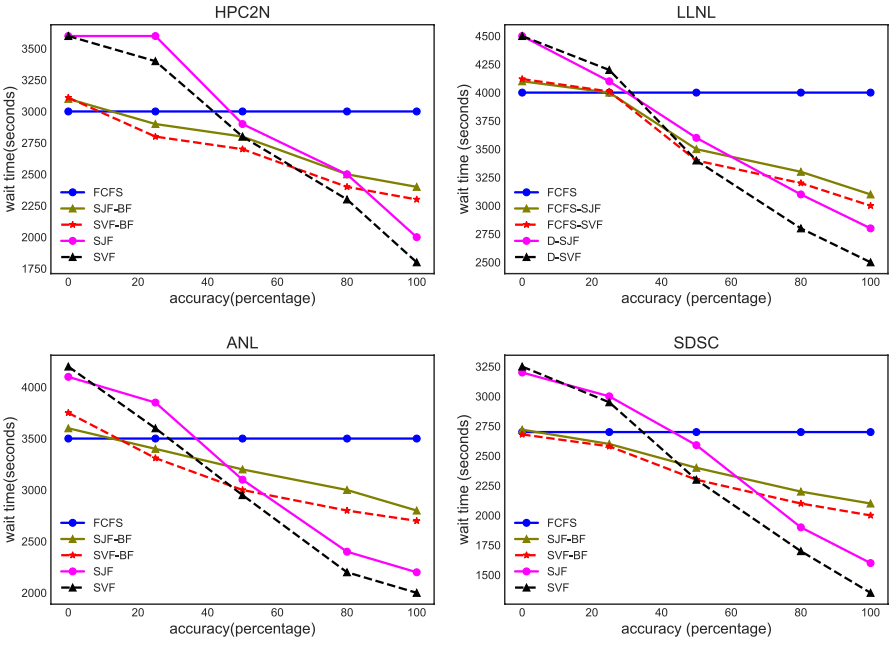

Fig. 3 Wait times of FCFS, FCFS-SJF, FCFS-SVF, online-SJF and online-SVF are plotted for traces with different accuracy levels

to measure the expected prediction accuracy for specific jobs that we denote as individual prediction reliability for job runtimes. Evaluating the quality of prediction accuracy is a challenging problem in machine learning [23]. Using the individual prediction accuracy metric (2) for previous jobs, we train a regression model to predict job runtime accuracy for the newly submitted jobs. When a job is submitted to the HPC cluster, information including the user id, time of submission, requested runtime, CPU and memory is available for each job; we extract useful features from job log. We explain the details of our approach to Sect. 4. The prediction accuracies estimated by our reliability estimation machine have a Pearson correlation of 0.84 with actual accuracies for HPC2N trace data.

2.4 Job runtime prediction reliability estimation

As discussed earlier, runtime values by users are far from accurate [13]. Several approaches have been proposed to generate more accurate runtime predictions based on available information about the job at the time of submission. In [18], several time-series-based approaches are used to predict job runtimes. A machine learning approach was proposed in [19] where the authors propose an online learning model that makes a prediction for each newly submitted job and updates the prediction model with newly available runtime after completion of each job. However, to the best of our knowledge, there has not been any study about confidence or reliability of runtime predictions for individual jobs. One important issue in using predicted runtimes is the fact that for some jobs, the prediction accuracy is low. Hence, there is the need to measure the expected prediction accuracy for specific jobs that we denote as individual prediction reliability for job runtimes. Evaluating the quality of prediction accuracy is a challenging problem in machine learning [23]. Using the individual prediction accuracy metric (2) for previous jobs, we train a regression model to predict job runtime accuracy for the newly submitted jobs. When a job is submitted to the HPC cluster, information including the user id, time of submission, requested runtime, CPU and memory is available for each job; we extract useful features from job log. We explain the details of our approach to Sect. 4. The prediction accuracies estimated by our reliability estimation machine have a Pearson correlation of 0.84 with actual accuracies for HPC2N trace data.

2.5 Formulation of the problem

The problem studied in this work is to execute a set of concurrent parallel jobs with rigid resource requirements on an HPC platform with m units of resources. The jobs are submitted over time in an online manner.

The resource requirement d_j of an application j is equivalent to the user-requested resource known at the time of submission. The actual value of runtime is only known a posteriori when the job completes. The problem we are trying to solve is how to schedule applications to achieve better performance and utilization, without knowing the accurate runtime values at submission time.

Although application runtime has some correlation with several features, as discussed in previous works [19], 100% accurate prediction of application runtimes is not possible [19]. Thus, we are looking for scheduling solutions that improve the performance and utilization over the currently used approaches when 100% accurate prediction is not available for all jobs in the system and more specifically are interested in a scheduling solution that:

- is online.
- has low overhead.
- performs well even with inaccurate runtime predictions.
- achieves high performance when accurate predictions are available.
- outperforms commonly used scheduling strategies in a real-world setting (accurate runtime is available for a subset of jobs).

3 Related work

The necessity of runtime prediction for parallel applications has been highlighted since the last years of twentieth century [24]. Different approaches have been proposed to predict HPC application runtimes with different machine learning methodology as well as prediction features inputs [13, 25]. Several time-series-based methodologies are proposed to use the information available about completed jobs in the system to predict the runtime for newly submitted jobs. They are mainly exponential smoothing and moving average methodologies that predict future values based on the recent runtime values. As noted by [18], these methods are not accurate and will not improve scheduling performance and utilization significantly. Some earlier works profile the existing applications and form a model similar to the regression decision tree to predict the runtime of the new application based on the most similar application profile [24, 26]. In [26], authors apply genetic algorithms to search among the history logs to find the most similar attributes. Wyatt II et al. [27] use neural networks to predict the runtime of HPC applications. Several statistical methods fit a distribution to previous data and predict the new job runtime with mean and confidence interval of the inferred distribution [13]. Nissimov and Feitelson [28] draw sixteen different distribution from previous data and design a hidden Markov model to transit along with these sixteen states. Some other researchers have considered additional features for each job in trace and performed predictions based on the similarity between these features of the jobs [29, 30]. Some more recent works focus on the specific family of scientific workflows and use machine learning for predicting runtime and resource usage for these applications [31]. Several works have proposed interrogating of codes to extract features for runtime prediction. This is not practical in many cases due to privacy considerations. Making an accurate prediction for application runtime is not easy. In fact, the scarcity of relevant training data makes the model building cumbersome. Several online prediction methods have been proposed that use the recently completed jobs in the same trace to strengthen the predicting power of their model [18, 25].

Some authors correctly identified the importance of avoiding overfitting the value of runtimes and considered approaches that favor under-prediction over over-prediction as over-predicted runtime values impose the expense of killing incomplete jobs [3, 19].

Similar to our work, some previous works considered inaccurate predictions. They proposed a solution to handle inaccuracies in predicted runtime values. Authors in [32] considered error margins for runtime prediction to be considered for runtime adjustment. In [33], authors developed an execution delay model for runtime prediction and designed an adaptive stochastic allocation strategy for production workload traces. Ilyushkin and Epema [34] study the effect of runtime inaccuracy and find out that the bounded slow-down affected by inaccurate runtimes in highly utilized clusters.

4 Proposed hybrid scheduling platform

The study of sensitivity in Sect. 2.2 gave us an intuition to design a hybrid scheduling platform. Since in real-world situations, accurate values of job runtimes are not available, we propose a hybrid scheduling platform that uses FCFS with backfilling for jobs with unpredictable runtime. At the time of job submission, the normalized regression model is used to predict the job runtime value in an online manner. Then, jobs are classified into two queues of *predictable* and *unpredictable* using our trained reliability estimation model. Our platform schedules jobs with higher expected runtime accuracy using plan-based scheduling. The platform then performs backfilling to schedule the jobs in the *unpredictable* queue on the remaining available resources. The *predictable* and *unpredictable* jobs are defined below.

Predictable jobs These are jobs with high prediction reliability. We characterize these jobs with an estimated runtime accuracy of 60% or more.

Unpredictable jobs These are jobs with low prediction reliability. We characterize these jobs with estimated runtime accuracy of less than 60%.

Our resource manager dynamically determines the resource quota for pure priority-based algorithm based on the ratio of predictable jobs.

4.1 Proposed design

In Sect. 2.2, we demonstrated how priority-based scheduling algorithms outperform backfilling algorithms when we have an almost accurate (accuracy of 60% and higher), prediction of job runtimes. We use this observation to design a hybrid scheduling algorithm. Similar to other algorithms discussed in Sect. 2.1, our algorithm is online, and scheduling takes place in rounds. Two different policies, SVF and FCFS-BF, are performed at each round. Using our reliability prediction model, we know for which subset of jobs we have more accurate runtime values. Our hybrid model applies plan-based scheduling for the subset of jobs and backfills the remaining jobs on the top. We

propose a hybrid scheduling algorithm that in a clever way amalgamates the backfilling algorithms and plan-based scheduling algorithms. With our hybrid design, we benefit from the effectiveness of backfilling scheduling approach when the runtime predictions are inaccurate, and we benefit from the near-optimal performance of plan-based scheduling when the predictions are more accurate. We determine the reliability of predicted runtimes using a supervised machine learning method of gradient boosting trees as presented in Sect. 4.4.2. The supervised regression outputs an estimation of the prediction accuracy. Our scheduling platform puts jobs in two queues of *predictable* and *unpredictable* based on their estimations of accuracy. For the jobs in *unpredictable* queue, the platform does not rely on system-generated predictions. The platform uses the runtimes requested by the user to backfill *unpredictable* jobs. Our hybrid scheduling platform assigns a subset of resources to plan-based scheduling to improve utilization of the resources and tops the remaining unused resources with backfilling. Our experiments prove the improved performance and CPU utilization of the proposed hybrid approach when compared to both pure plan-based and pure backfilling methodologies. Figure 4 presents the hybrid scheduling platform in detail. As demonstrated in Fig. 4, when a job is submitted to the cluster, its information is stored in the repository. ML-unit calculates the runtime prediction as well as estimated prediction accuracy. The hybridization parameter calculator calculates the hybridization parameter based on the estimated prediction accuracy values, as explained in Sect. 4.3. The hybrid central scheduler uses the predicted runtime to order *predictable* jobs on α portion of resources and backfills the *unpredictable* jobs on top. After each job completes on the HPC system, its actual runtime is recorded in the repository. As demonstrated in Fig. 4, our scheduling platform has four main components: ML-unit, hybridization parameter adjusting unit, centralized scheduler and repository. When a job is submitted to the system, its runtime and prediction accuracy estimation are calculated in the ML-unit. The values of runtime prediction and prediction accuracy estimation are used by the central scheduler to determine the deployment time of the new job in the HPC system. After the job completes running, information about actual runtimes is saved in the repository to be used by the ML-unit and hybridization parameter estimator unit. The *repository* contains the necessary data including prediction model parameters for the ML-unit as predictability estimation model. As a job completes running on HPC cluster, the runtime value is

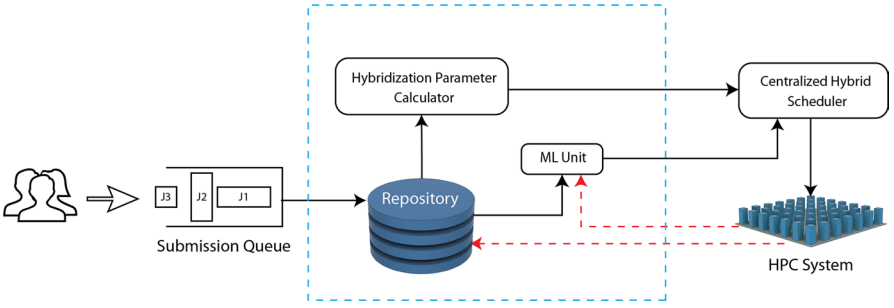

Fig. 4 Overview of the HS platform design

added to the repository. The *centralized scheduler*, the *hybridization parameter adjusting unit* and the ML-unit are elaborated in the following subsections.

4.2 The central scheduler

Using the input from ML-unit, jobs are partitioned into two queues: *predictable* and *unpredictable* jobs as defined in the beginning of current section. Jobs that are characterized as *predictable* by ML-unit are input to a new queue called *predictable*. As shown in Fig. 5, the hybrid scheduler is composed of two schedulers: plan-based scheduler and backfilling scheduler. At first, the central scheduler performs plan-based scheduling. It determines the starting time of each *predictable* job in the waiting queue following a plan-based scheduling algorithm, as explained in Sect. 2.1. For this plan-based scheduling, only a specific portion of resources using the specific portion of CPU cores determined by hybridization parameter unit is considered. Second, after the deployment plan of the predictable jobs is determined, FCFS with backfilling scheduler determines the starting time of *unpredictable* jobs on the remaining available resources. The complete schedule for all jobs is used to deploy jobs on the HPC cluster. This procedure is repeated at the time of each event in the computation cluster: job submission, job completion or job termination.

4.3 The hybridization parameter adjusting unit

The ratio of resources used by plan-based scheduling policy to schedule *predictable* jobs, α, is called hybridization parameter and will be updated by the hybridization parameter adjusting unit in the platform. The parameter adjusting unit updates the value of hybridization parameter, "α" based on the ratio of the sum of resource usage of predictable jobs to the total resource request all the jobs. The formula for updating α is presented below.

$$\alpha_{t+1} = \alpha_t * \frac{\sum_{i \in \text{predictable}} d_i}{\sum_{j \in \text{all jobs}} d_j} \tag{1}$$

where d_j is the resource requirement for the job j as defined in Sect. 2. The initial value of α denoted as α_0 is chosen through a grid search as illustrated in Fig. 13.

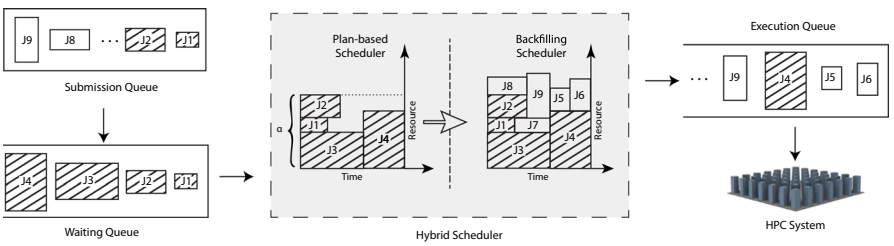

Fig. 5 Central scheduler design

4.4 The ML-unit

The ML-unit is responsible for predicting runtimes and estimating the accuracy of the predicted runtimes. The runtime prediction is performed using online normalized regression model [35, 36] similar to [19]. Another important component of ML-unit is runtime reliability estimator. This component determines how predictable the runtime of a given job is based on available features about the job at the time of submission. To determine the prediction reliability of jobs, we use a gradient boosting tree model. The model parameters are saved in the repository, and the model is rebuilt every 24 hours. One important function of the ML-unit is to determine if the jobs belong to a *predictable* or a *unpredictable* class of jobs. In order to study prediction accuracy for individual jobs, we need to use an appropriate metric to measure the accuracy of prediction for individual jobs.

Measure of prediction accuracy Common measures of accuracy in machine learning such as mean squared error (MSE) measure the accuracy of prediction globally. As we are interested in measuring and characterizing accuracy for each individual point in data set, we use a point-wise measure of accuracy that has been proposed in the literature of HPC job runtime prediction for HPC clusters [13, 18]. This metric measures accuracy of runtime prediction for each job by comparing the predicted job runtime value ($\hat{p}_j$) and actual job runtime (p_j) as presented in Eq. 2.

$$acc_j = \begin{cases} 1 & \hat{p}_j = p_j \\ \frac{\hat{p}_j}{p_j} & \hat{p}_j < p_j \\ \frac{p_j}{\hat{p}_j} & \hat{p}_j > p_j \end{cases} \tag{2}$$

4.4.1 Online job runtime prediction

As shown in Fig. 6, an online prediction module is in charge of predicting runtime values. For a newly submitted job, the online prediction module predicts the runtime. The model is tuned by using the actual runtime of completed jobs in the cluster. The goal of the online job runtime prediction module is to predict job runtimes in an online manner. As a job is submitted to the systems, a minimal set of features are extracted from the job description as well as resource management systems. These features are then used by an online job runtime prediction module to predict job runtimes. In our platform, we implemented an l_2 regularized polynomial model similar to

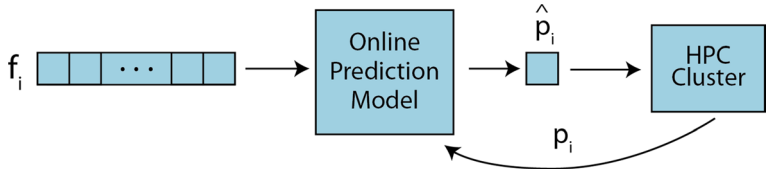

Fig. 6 Online learning module predicts runtime for the current job based on the feedback from previous jobs

[19]. New values of $\hat{p}_i$ are predicted using features available from $i-1$ completed jobs : $Z = \{z_{11}...z_{1i-1}, \ldots, z_{ki-1}\}$. In this work, we use online normalized regression model as proposed in [36] similar to [19]. Online normalized regression is an appropriate machine learning model for predicting runtimes as it is able to predict job runtimes with minimal information, is online and is robust to noise in input data [36]. We use online normalized regression with stochastic gradient descent to find a polynomial functions of 18 input features, as listed in Table 1, corresponding to $i-1$ previously completed jobs $Z = \{z_{11}...z_{1i-1}, ..., z_{ki-1}\}$. The model gets updated as more jobs are completed and the actual runtime values are available. The polynomial function to predict the runtime for a newly submitted job is of the form:

$$f(w, z) = w^T \phi(z). \tag{3}$$

To build a model that predicts the runtime values most accurately, a loss function is defined to penalize errors of the prediction using the training samples as in Eq. 5. The parameters of the model, w_is, are found by minimizing a loss function as in Eq. 4. The regularization term, $\lambda||w||$, is added to avoid bias in the model we are learning by avoiding extreme values of parameters. The minimization of regularized cumulative loss for up to i-th completed job is:

$$\arg_w \min \sum_{j=1}^{i} L\big(x_j, f(w, z), p_j\big) + \lambda||w||. \tag{4}$$

where λ is the regularization parameter and $f(w, z_j)$ is our prediction for runtime of job j. The loss function is defined as:

$$L(z_j, f(z_j), p_j) = \begin{cases} \lambda.f(z_j) - p_j)^2 & (z_j) \geq p_j \\ \lambda.(p_j - f(z_j))^2 & f(z_j) < p_j \end{cases} \tag{5}$$

To focus on the symmetric inaccuracy of prediction, we consider the symmetric loss function instead of asymmetric loss function in [19] and we penalize over-prediction and under-prediction equally.

4.4.2 A meta-learning approach to estimate runtime prediction accuracy

Meta-learning, or learning to learn, is the science of systematically observing how different machine learning approaches perform [37]. Here, we are interested in estimating the prediction accuracy of our online algorithm that predicts job runtime. To estimate the prediction accuracy of job runtimes (prediction reliability), we use a supervised regression model. The regression model predicts runtime accuracy of the newly submitted jobs using the features presented in Table 2 and accuracy of completed jobs as the target values. As shown in Fig. 7, feature vectors of completed jobs, $f_i = \{z_{1i}, \ldots, z_{ki}\}$, and their corresponding prediction accuracy, acc_i, are used to train a regression model. The regression model maps the features and the predicted runtime to accuracy values. We used gradient boosting tree regression [38] as it is known to find a nonlinear mapping from data to target values [39]. The trained prediction reliability estimation

Table 1 Features considered for online prediction of job runtime

Feature group	Feature names	Number of features
User-requested runtime	$\tilde{p}_j$ (reqTime)	1
Actual runtime and CPU for the previously completed jobs from the same user	last, beflast, beflast2, lastcpu	4
Maximum runtime and CPU for the previously completed jobs from the same user	maxrt, maxcpu	2
Seasonality features	tod1, tod2, dow1, dow2	4
Average and standard deviation of runtime and CPU of the jobs from the same user	meanrt, stdrt, meancpu, stdcpu	4
Number of jobs currently running	Num-current	1
Longest running time of the user's currently running job	Longest	1
Time elapsed since last job completion from the same user	Break-time	1

Table 2 Features considered for our prediction reliability estimation approach

Feature group	Feature names	Number of features
User-requested runtime	$\tilde{p}_j$ (reqTime)	1
Actual runtime and CPU for the previously completed jobs from the same user	last, beflast, beflast2, lastcpu	4
Maximum runtime and CPU for the previously completed jobs from the same user	maxrt, maxcpu	2
Seasonality features	tod1, tod2, dow1, dow2	4
Average and standard deviation of runtime and CPU of the jobs from the same user	meanrt, stdrt, meancpu, stdcpu	4
The number of completed jobs from the same user	prevuser	1

model is used to estimate the runtime prediction accuracy of the newly submitted jobs. In Fig. 8, the prediction reliability outputs accuracy estimation value for job i, $\hat{acc}_i$, using feature vectors $f_i = \{z_{1i}, \ldots, z_{ki}, \hat{p}_i\}$.

Gradient boosting tree regressor is a prediction approach that ensembles regression decision trees with gradient boosting [38]. In gradient boosting, a model is built in a stage-wise fashion. In each stage, the existing model is boosted by the optimization of an arbitrary differentiable loss function. In the gradient boosting algorithm, in each stage, a tree is built based on the residuals from the tree in the previous stage. Basically, in each stage, a new gradient tree is fitted into residual values. It is important to note that at each stage, randomized samples of training data are chosen without substitution to avoid the risk of overfitting [40]. Algorithm 1 shows the procedure to find gradient boosting regression tree predictor $\tilde{meta}$ for input F. The algorithm has training set of k-dimensional input features of size N denoted as $F = \{f_1, f_2, \ldots, f_N\}$ and their corresponding target accuracy values $ACC = \{acc_1, \ldots, acc_N\}$. As the first step, a decision tree $meta_0(F)$ is fitted to F and ACC. Namely, the features to branch upon at each level of the tree as well as the threshold to make the decision are found by solving an optimization problem [41]. Using the trained decision tree, give us the ability to find a decision region of feature space for a new data point. The predicted target value for a new data point is predicted by averaging the runtime of the applications in the same subregion. Then, in each stage, the gradient of the residuals of predictions are calculated, and a regression tree is fitted to this gradient to find R_{jm} decision regions. After calculating the multiplier for each gradient residual subtree and adding the weighted sum of subtrees to the previous regression tree, the stage is complete. The routine is repeated for a tunable number of stages.

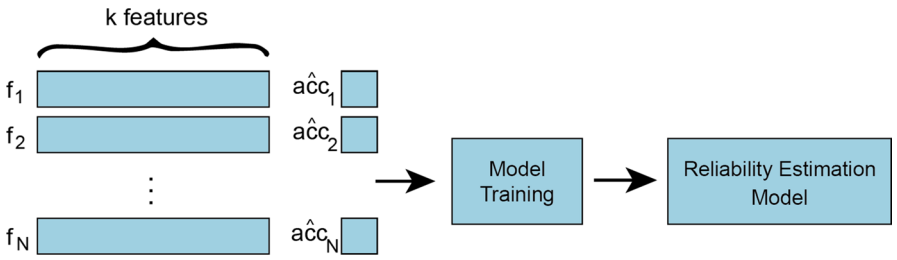

Fig. 7 Prediction reliability estimation machine is trained using features of completed jobs $f_i = \{z_{1i}, \ldots, z_{ki}\}$ and their corresponding prediction accuracy values acc_i

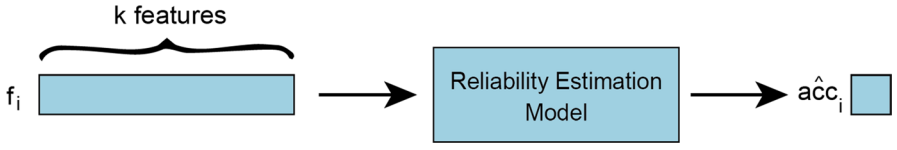

Fig. 8 The trained reliability estimation machine is used to predict the accuracy for newly submitted jobs

Algorithm 1 Gradient Boosting Tree as the Meta Learning Approach

Initialize $meta_0 = arg\min \sum_{i=1}^{N} L(acc_i, \gamma)$
for $m = 1$ to M **do**
  for $i = 1, .., N$ **do**
    compute $r_{im} = -[\frac{\partial Lacc_i meta(f_i)}{\partial meta(f_(x_i)}]_{f=f_{m-1}}$
  end for
  fit a regression tree to the target $r_i m$
  for $j = 1, 2, ...m$ **do**
    compute $\gamma_{jm} = arg\min_{\gamma} \sum_{x_j \in R_{jm}} (acc_i meta_{m-1}(f_{m-1}(x_i) + \gamma)$
  end for
  update $meta_m(x) = meta_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
end for
output $\tilde{meta}(x) = meta_M(x)$

In Fig. 9, the relative importance of the most important features is presented. The most important feature is assumed as 100%, and all other feature importance factors are scaled to [0, 100]. The value of runtime for the previously completed job from the same user has the most influence on the predictability of the runtime for the current job. We see that the requested runtime by users and the runtime for the job before the last job from the same user are the second and third important features. The features names are presented in Table 2.

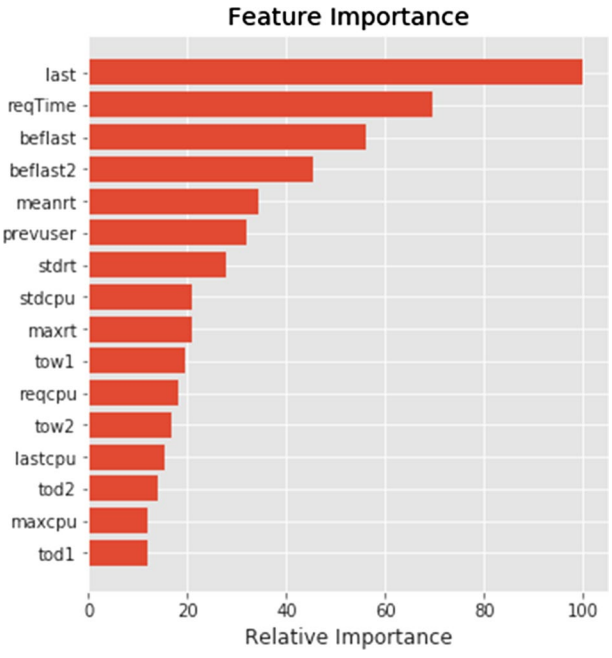

Fig. 9 Feature importances for meta-learning model are shown

Correlation of estimated accuracy values with the actual accuracy values has been used to measure the goodness of fit for meta-learning approaches in the literature [42]. In Table 3, the correlation of actual runtime accuracies and estimated accuracies are compared with CNK [42–44] and decision tree regressor. We observe that gradient boosting tree meta-learning estimation of accuracy has the highest correlation with actual prediction accuracy among our experimented machine learning algorithms.

5 Evaluating the performance of our proposed hybrid scheduling platform

The use of benchmark datasets rather than in-house datasets provides repeatability of experiments and eases comparison of the results of the researches. One challenge in studying HPC applications and their scheduling is the scarcity of public datasets [45]. During the previous two decades, some national laboratories have published the traces of their high-performance computing clusters. A collection of these datasets are cleaned and benchmarked as parallel workload dataset [17]. As parallel workload dataset [17] is the current benchmark for HPC scheduling [16, 46, 47] and runtime prediction [48], we use three traces from this dataset. They were consistent with our problem statement in terms of inputs we require for each application and the specification of the scheduling problem. More specifically, our chosen datasets were collected from non-preemptive scheduling clusters, which is consistent with our problem statement. In addition, the set of available features we require for our ML-unit is provided in these datasets. Additionally, these datasets were commonly used in published research work and facilitate the comparison of results. HPC2N is a trace log containing three and a half years worth of accounting records from the High-Performance Computing Center North (HPC2N) in Sweden. LLNL ATLAS is a trace log from ATLAS cluster in Lawrence Livermore National Lab, and ANL Intrepid is from Intrepid cluster in Argonne National Lab. SDSC trace is from the SDSC Blue Horizon in San Diego Super Computer.

5.1 Event-driven simulation

We simulate the scheduling of traces using the open-source event-driven simulation package, Alea2 [49]. Alea2 simulator is based on the GridSim simulation toolkit [50]. Alea2 extends Gridsim to provide a simulation environment that supports simulation of varying job scheduling problems [19, 49]. Alea2 uses a centralized job

Table 3 Correlation of estimated accuracies with actual accuracies for gradient boosting tree is compared with decision tree and CNK

	Gradient boosting tree	Decision tree	CNK
HPC2N	0.84	0.77	0.44
SDSC	0.81	0.75	0.53
ANL	0.79	0.68	0.47
LLNL	0.78	0.66	0.50

scheduler which uses scheduling techniques for schedule generation. For priority-based scheduling algorithms, Alea2 central scheduler, jobs are submitted according to their submission time in the trace. An ordered queue of jobs is maintained and is updated in the event of job submission or job termination. Once started, jobs run to completion, implying that the central scheduler is not allowed to preempt or migrate the tasks. We have added an online regression method as a machine learning-based prediction module and gradient boosting tree as job runtime prediction accuracy estimation to the Alea2 package. At the time of each job submission, the prediction module calculates the prediction of runtime and returns the runtime value to the central scheduler.

5.2 Comparison with existing scheduling approaches

The aim of this section is to compare the performance and efficiency of our proposed hybrid scheduling algorithm with common scheduling algorithms in a real-world scenario. For this purpose, we implemented a hybrid scheduling platform, as described in Sect. 4. To implement the platform, we extended the existing prediction module in Alea2 with implementing the online normalized regression as described in [19] as well as a module for performing reliability estimation with gradient boosting tree regression. A thin module of hybridization parameter calculator was also added to Alea2 repository. In the scheduling module, hybrid scheduler as well as FCFS backfilling with SVF and online-SVF was added.

In the performance and efficiency comparison experiments, hybrid scheduling algorithm was compared with two backfilling approaches as well as two plan-based scheduling algorithms on production traces. *FCFS-BF1* denotes backfilling with SJF as used by [13, 19]. *FCFS-BF2* is similar to *FCFS-BF1* but uses smallest volume first (SVF) rule to choose jobs to backfill. *Plan-based1* is the dynamic version of SJF as described in Sect. 2.1. *Plan-based2* is dynamic version of SVF [16].

Waiting time and bounded slow-down are among the most used criteria for evaluating the performance of the scheduling algorithm, and utilization is a criterion to evaluate the resource usage efficiency of the scheduling algorithm [13]. The criteria are listed below.

Bounded slow-down The slow-down of a job is the maximum ratio of job response time on the loaded system to its runtime on a dedicated system. To calculate slow-down, we calculate the ratio of response time plus wait time to the response time. The slow-down value is often more than one. The lower the value of the slow-down in a clustering environment indicates better scheduling of the applications. As using slow-down formula, the short jobs with near-zero response time will have a high value of slow-down with a small wait time, Tsafrir et al. [13] have proposed bounded slow-down. Bounded slow-down substitutes the runtime in the dividend by the maximum of a constant value, τ and the response time.

$$\text{blsd} = \max_{j \in J} \left(\frac{\text{wait}_j + p_j}{\max\left(p_j, \tau\right)} \right) \tag{6}$$

Average wait time The average wait time measures the average of the time between job submission and the job start time on the system overall jobs.

Utilization The ratio of total node hours used by the scheduling algorithm to the total nod hours elapsed from the time the first job was submitted to the system.

We used hybrid platform to schedule jobs of four different traces from parallel workload dataset and compared the three criteria explained above with two backfilling and two plan-based scheduling approaches: FCFS and backfilling algorithms, FCFS-BF1 and FCFS-BF2, as well as plan-based scheduling algorithms, Plan-based1 and Plan-based2. The results of these experiments are shown in Figs. 10, 11 and 12. In all these figures, the hybrid scheduling platform is denoted as HS.

In the first set of experiments, we compared the wait times. In Fig. 10, wait times of our hybrid platform is compared with the two plan-based and backfilling algorithms. Plan-based schedulers perform slightly better than backfilling algorithms, but our hybrid platform performs significantly better than both categories of schedulers (about 20%).

In Fig. 11, bounded slow-down improvement of hybrid platform is compared with backfilling and plan-based approaches. We can see that the slow-down is considerably lower than the plan-based and backfilling schedulers. In fact, we can see that the hybrid platform's bounded slow-down is more than 50% lower than backfilling and plan-based methods. The improved bounded slow-down and average wait time in the hybrid platform are because the hybrid platform uses plan-based scheduling for jobs identified as predictable and backfills the jobs identified as unpredictable on remaining available resources. This curated choice of scheduling algorithm results in lower slow-down and average wait time and better performance.

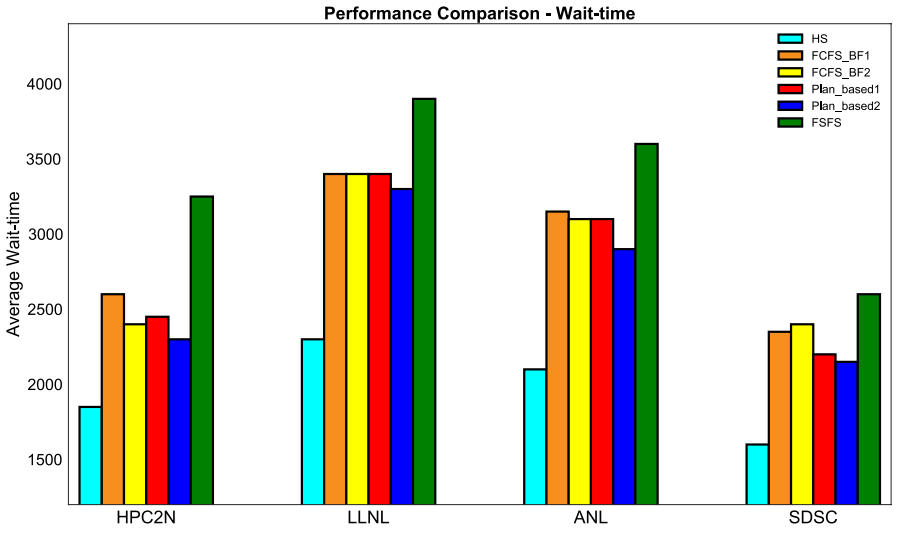

Fig. 10 Average wait time values of HS is compared to FCFS, SVF-BF, SJF-BF, SVF and SJF. HS has the lowest average wait time

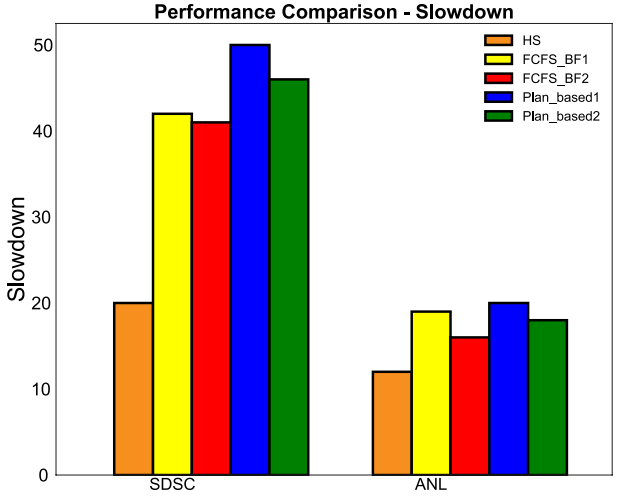

Fig. 11 Bounded-slow-down values of HS is compared with SVF-BF, SJF-BF, SVF and SJF. HS has the lowest bounded slow-down

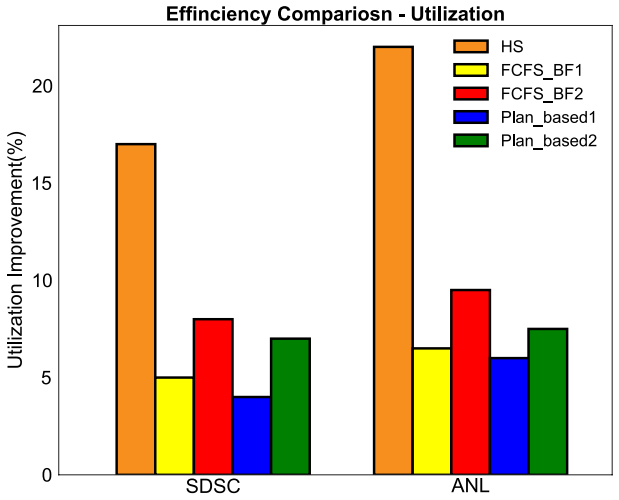

Fig. 12 Utilization percentage of HS is compared with SVF-BF, SJF-BF, SVF and SJF. HS has the highest utilization

In Fig. 12, the utilization improvement of the hybrid platform is compared with backfilling and plan-based approaches. The utilization is reported as an improvement over the simple SJF scheduling. We can see in Fig. 12 that utilization improvement is also significantly greater than the backfilling and plan-based approaches. The better utilization of hybrid also shows that using predictability information about the jobs and choosing the scheduling algorithm accordingly leads to better packing of

jobs into resources than completely relying on predicted runtimes in plan-based algorithms or ignoring job runtimes in backfilling algorithms.

5.3 The effect of clairvoyance on hybrid scheduler

In Table 4, we compare the bounded slow-down of hybrid platform with best performing backfilling and plan-based algorithms both in the clairvoyant case where accurate runtime values are available at the time of job submission as well as in the non-clairvoyant case where runtimes are predicted with system-generated approaches. In the clairvoyant scenario, as expected, plan-based scheduling has lower bounded slow-down than backfilling scheduling. This is because plan-based scheduling determines the order of jobs based on their runtime values. When the accurate values of runtime are available, plan-based scheduling performs best. However, in the more realistic situations, where runtime values are predicted using machine learning approaches and our proposed accuracy estimation is used to decide about the scheduling approach of each job, our proposed hybrid platform has the lowest bounded slow-down.

5.4 Parameter selection for hybrid scheduling

As discussed in Sect. 4, the portion that we allocate for predictable jobs (α_0) needs to be determined. To study the effect of choosing different values of α_0, we repeated our simulations of hybrid platform scheduling with different values of α_0 on HPC2N trace and measured the values of average wait time. Our simulations showed that with all values of α_0 between 0.3 and 0.8 hybrid scheduling outperformed backfilling and plan-based scheduling algorithms. In Fig. 13, we have plotted average wait times for different values of α_0. We can see that the initialization of $\alpha_0 = 0.6$ resulted in the best performance. We repeated the simulations for 20 times to ensure the reliability of our results.

6 Conclusion

The goal of this paper was to design a scheduling platform to handle inaccuracies in workload runtimes. We designed a novel scheduling platform that hybridizes two popular classes of scheduling algorithms, namely backfilling and plan-based

Table 4 Bounded-slow-down comparison with Clairvoyant problem setting

	Clairvoyant		Non-clairvoyant		
	Backfilling	Plan-based	Backfilling	Plan-based	Hybrid platform
SDSC	40	20	50	45	25
ANL	24	11	24	22	14

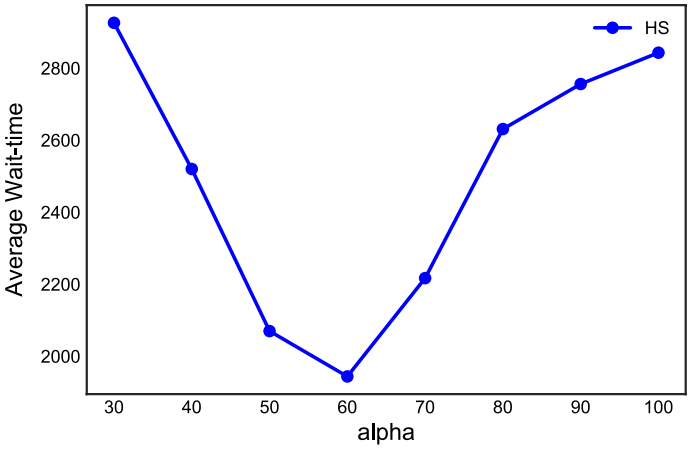

Fig. 13 Comparing average wait time of hybrid platform on HPC2N dataset with various initial alpha (α_0). It can be seen that with $\alpha_0 = 0.6$ hybrid scheduling platform has its best average wait time

scheduling. Our design was motivated by the difference in sensitivity of these two classes of scheduling algorithms to runtime prediction accuracy. Our platform is designed based on a deep understanding of the characteristics of plan-based and backfilling scheduling algorithms. Our hybrid scheduling platform uses the estimation of predictability of HPC application runtimes to choose an appropriate scheduling algorithm for each specific application. Our proposed platform avoids resource fragmentation by adaptively and dynamically changing the portion of computation resources that plan-based scheduler uses. This portion is calculated based on the resource requirement ratio of recently submitted jobs with reliable runtime prediction. Our meta-learning solution uses the power to predict the level of uncertainty of job runtimes to better schedule HPC applications on available computation resources. Our extensive trace-based experiments show significant improvement in the performance and utilization of HPC workload traces.

While our present work serves as a demonstration of using prediction reliability to improve performance and utilization of HPC scheduling platforms, there are several directions for taking our idea further. A natural direction is to implement the proposed platform in scheduling packages such as Cobalt [12]. Another direction is to extend the proposed approaches to predict reliability for resource consumption of data center workloads.

References

1. Lee CB, Schwartzman Y, Hardy J, Snavely A (2004) Are user runtime estimates inherently inaccurate? In: Workshop on Job Scheduling Strategies for Parallel Processing. Springer, pp 253–263
2. Tang W, Lan Z, Desai N, Buettner D (2009) Fault-aware, utility-based job scheduling on blue, gene/p systems. In: 2009 IEEE International Conference on Cluster Computing and Workshops. IEEE, pp 1–10

3. Fan Y, Rich P, Allcock WE, Papka ME, Lan Z (2017) Trade-off between prediction accuracy and underestimation rate in job runtime estimates. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, pp 530–540
4. Tchernykh A, Schwiegelsohn U, Alexandrov V, Talbi E-G (2015) Towards understanding uncertainty in cloud computing resource provisioning. Procedia Comput Sci 51:1772–1781
5. Zheng X, Zhou Z, Yang X, Lan Z, Wang J (2016) Exploring plan-based scheduling for large-scale computing systems. In: 2016 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, pp 259–268
6. Bosnić Z, Kononenko I (2008) Estimation of individual prediction reliability using the local sensitivity analysis. Appl Intell 29(3):187–203
7. Hovestadt M, Kao O, Keller A, Streit A (2003) Scheduling in HPC resource management systems: queuing vs. planning. In: Workshop on Job Scheduling Strategies for Parallel Processing. Springer, pp 1–20
8. Srinivasan S, Kettimuthu R, Subramani V, Sadayappan P (2002) Characterization of backfilling strategies for parallel job scheduling. In: International Conference on Parallel Processing Workshops, 2002. Proceedings. IEEE, pp 514–519
9. Skovira J, Chan W, Zhou H, Lifka D (1996) The easy loadleveler API project. In: Workshop on Job Scheduling Strategies for Parallel Processing. Springer, pp 41–47
10. Talby D, Feitelson DG (1999) Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling. In: 13th International and 10th Symposium on Parallel and Distributed Processing Parallel Processing, 1999. 1999 IPPS/SPDP. Proceedings. IEEE, pp 513–517
11. Yoo AB, Jette MA, Grondona M (2003) Slurm: simple linux utility for resource management. In: Workshop on Job Scheduling Strategies for Parallel Processing. Springer, pp 44–60
12. Desai N (2005) Cobalt: an open source platform for HPC system software research. In: Edinburgh BG/L System Software Workshop, pp 803–820
13. Tsafrir D, Etsion Y, Feitelson DG (2007) Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans Parallel Distrib Syst 18(6):789–803
14. Xhafa F, Carretero J, Dorronsoro B, Alba E (2012) A tabu search algorithm for scheduling independent jobs in computational grids. Comput Inform 28(2):237–250
15. Klusáček D, Chlumskỳ V, Rudová H (2015) Planning and optimization in torque resource manager. In: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing. ACM, pp 203–206
16. Im S, Naghshnejad M, Singhal M (2016) Scheduling jobs with non-uniform demands on multiple servers without interruption. In: IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications, April, pp 1–9
17. Parallel workloads archive. http://cs.huji.ac.il/labs/parallel/workloads. Accessed 2015-07-01
18. Sonmez O, Yigitbasi N, Iosup A, Epema D (2009) Trace-based evaluation of job runtime and queue wait time predictions in grids. In: Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing. ACM, pp 111–120
19. Gaussier E, Glesser D, Reis V, Trystram D (2015) Improving backfilling by using machine learning to predict running times. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, p 64
20. Bhimani J, Mi N, Leeser M, Yang Z (2017) Fim: performance prediction for parallel computation in iterative data processing applications. In: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD). IEEE, pp 359–366
21. Obaida MA, Liu J, Chennupati G, Santhi N, Eidenbenz S (2018) Parallel application performance prediction using analysis based models and HPC simulations. In: Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation. ACM, pp 49–59
22. Naghshnejad M, Singhal M (2018) Adaptive online runtime prediction to improve HPC applications latency in cloud. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). IEEE, pp 762–769
23. Lakshminarayanan B, Pritzel A, Blundell C (2017) Simple and scalable predictive uncertainty estimation using deep ensembles. In: Advances in Neural Information Processing Systems, pp 6402–6413
24. Gibbons R (1997) A historical application profiler for use by parallel schedulers. In: Workshop on Job Scheduling Strategies for Parallel Processing. Springer, pp 58–77

25. Tang W, Desai N, Buettner D, Lan Z (2010) Analyzing and adjusting user runtime estimates to improve job scheduling on the blue gene/p. In: 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS). IEEE, pp 1–11
26. Smith W, Foster I, Taylor V (1998) Predicting application run times using historical information. In: Feitelson D G, Rudolph L (eds) Job scheduling strategies for parallel processing. Springer, Berlin, pp 122–142
27. Wyatt II MR, Herbein S, Gamblin T, Moody A, Ahn DH, Taufer M (2018) PRIONN: predicting runtime and IO using neural networks. In: Proceedings of the 47th International Conference on Parallel Processing. ACM, p 46
28. Nissimov A, Feitelson DG (2007) Probabilistic backfilling. In: Workshop on Job Scheduling Strategies for Parallel Processing. Springer, pp 102–115
29. Mendes CL, Reed DA (1998) Integrated compilation and scalability analysis for parallel systems. In: Proceedings. 1998 International Conference on Parallel Architectures and Compilation Techniques, 1998. IEEE, pp 385–392
30. Schopf JM, Berman F (1999) Using stochastic intervals to predict application behavior on contended resources. In: Proceedings. Fourth International Symposium on Parallel Architectures, Algorithms, and Networks, 1999 (I-SPAN'99). IEEE, pp 344–349
31. Matsunaga A, Fortes JA (2010) On the use of machine learning to predict the time and resources consumed by applications. In: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. IEEE Computer Society, pp 495–504
32. Dimitriadou S, Karatza H (2010) Job scheduling in a distributed system using backfilling with inaccurate runtime computations. In: 2010 International Conference on Complex, Intelligent and Software Intensive Systems (CISIS). IEEE, pp 329–336
33. Ramírez-Velarde R, Tchernykh A, Barba-Jimenez C, Hirales-Carbajal A, Nolazco-Flores J (2017) Adaptive resource allocation with job runtime uncertainty. J Grid Comput 15(4):415–434
34. Ilyushkin A, Epema D (2018) The impact of task runtime estimate accuracy on scheduling workloads of workflows. In: 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). IEEE, pp 331–341
35. Hazan E et al (2016) Introduction to online convex optimization. Found Trends Optim 2(3–4):157–325
36. Ross S, Mineiro P, Langford J (2013) Normalized online learning. arXiv:1305.6646
37. Vanschoren J (2019) Meta-learning. In: Hutter F, Kotthoff L, Vanschoren J (eds) Automated machine learning: methods, systems, challenges. Springer, Cham, pp 35–61. https://doi.org/10.1007/978-3-030-05318-5_2
38. Friedman JH (2001) Greedy function approximation: a gradient boosting machine. Ann Stat 29(5):1189–1232
39. Chen T, Guestrin C (2016) Xgboost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp 785–794
40. Friedman JH (2002) Stochastic gradient boosting. Comput Stat Data Anal 38(4):367–378
41. Hastie T, Tibshirani R, Friedman JH (2009) The elements of statistical learning: data mining, inference, and prediction, 2nd edn. Springer series in statistics. Springer, Berlin
42. Bosnić Z, Kononenko I (2008) Comparison of approaches for estimating reliability of individual regression predictions. Data Knowl Eng 67(3):504–516
43. Amvrosiadis G, Park JW, Ganger GR, Gibson GA, Baseman E, DeBardeleben N (2018) On the diversity of cluster workloads and its impact on research results. In: 2018 USENIX Annual Technical Conference (USENIX ATC 18), pp 533–546
44. Feitelson DG (2015) From repeatability to reproducibility and corroboration. ACM SIGOPS Oper Syst Rev 49(1):3–11
45. Mann ZÁ (2015) Allocation of virtual machines in cloud data centersa survey of problem models and optimization algorithms. ACM Comput Surv: CSUR 48(1):11
46. Pinedo ML (2012) Scheduling: theory, algorithms, and systems. Springer, Berlin
47. Augonnet C, Thibault S, Namyst R, Wacrenier P-A (2011) Starpu: a unified platform for task scheduling on heterogeneous multicore architectures. Concurr Comput Pract Exp 23(2):187–198
48. Witt C, Bux M, Gusew W, Leser U (2019) Predictive performance modeling for distributed batch processing using black box monitoring and machine learning. Inf Syst

49. Klusáček D, Rudová H (2010) Alea 2: job scheduling simulator. In: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), p 61
50. Buyya R, Murshed M (2002) Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurr Comput Pract Exp 14(13–15):1175–1220