Trade-off between Prediction Accuracy and Underestimation Rate in Job Runtime Estimates

Yuping Fan, Zhiling Lan
Illinois Institute of Technology
Chicago, IL, USA
yfan22@hawk.iit.edu, lan@iit.edu

Paul Rich, William E. Allcock, Michael E. Papka
Argonne National Laboratory
Argonne, IL, USA
richp@anl.gov, allcock@anl.gov, papka@anl.gov

***Abstract*—Job runtime estimates provided by users are widely acknowledged to be overestimated and runtime overestimation can greatly degrade job scheduling performance. Previous studies focus on improving accuracy of job runtime estimates by reducing runtime overestimation, but fail to address the underestimation problem (i.e., the underestimation of job runtimes). Using an underestimated runtime is catastrophic to a job as the job will be killed by the scheduler before completion. We argue that both the improvement of runtime accuracy and the reduction of underestimation rate are equally important. To address this problem, we propose an online runtime adjustment framework called TRIP. TRIP explores the data censoring capability of the Tobit model to improve prediction accuracy while keeping a low underestimation rate of job runtimes. TRIP can be used as a plugin to job scheduler for improving job runtime estimates and hence boosting job scheduling performance. Preliminary results demonstrate that TRIP is capable of achieving high accuracy of 80% and low underestimation rate of 5%. This is significant as compared to other well-known machine learning methods such as SVM, Random Forest, and Last-2 which result in a high underestimation rate (20%-50%). Our experiments further quantify the amount of scheduling performance gain achieved by the use of TRIP.**

***Keywords*-job scheduling; runtime prediction; Blue Gene systems**

I. Introduction

The insatiable demand for computing resources in science and engineering has driven the development of ever-growing supercomputers in the field of high-performance computing (HPC). In order to harness the full potential of extreme scale systems, effectively allocating parallel jobs to available resources (i.e., *batch scheduling*) is of paramount importance. Job scheduler periodically checks the status of job queues and available resources, and determines the order in which the jobs in the queues will be executed [34]. This is done by maintaining job priority in accordance with a site policy. The scheduler decides when and where to execute jobs [34]. In order to improve system utilization, job scheduling commonly adopts some backfilling strategies [13].

Typically, users submit their jobs to a queue managed by job scheduler through scripts. A job script contains all the information necessary to run the job such as job size, job runtime estimate, and job name. Here, *job runtime estimate* is a crucial information supplied by a user regarding the amount of time needed for job execution. Job runtime estimate is used by the scheduler for decision making including job prioritizing and backfilling. *Job prioritizing* policies order jobs based on job attributes, such as job arrival time, job runtime estimate and/or job size. For instance, the shortest job first policy (SJF) orders jobs by job runtime estimates in an increasing order. The longest job first policy (LJF) orders jobs by job runtime estimates in a decreasing order. The scheduling policy deployed at ALCF[1] uses job runtime estimates for job ordering [26]. Moreover, job runtime estimate is crucial for backfilling decision making. *Backfilling* allows lower-priority jobs to run ahead of higher-priority jobs if the lower priority jobs do not delay the higher-priority jobs. The selection of queued jobs for backfilling requires the knowledge of job runtime estimates of the running jobs and the queued jobs [24].

Unfortunately, job runtime estimates supplied by users are widely acknowledged to be overestimated [10] [11] [17] [24]. For example, Cirne and Berman showed that 50% to 60% of jobs use less than 20% of user-supplied job runtime estimates [10]. There are two major reasons for exaggerating runtime estimates. One is the fear of jobs being killed if the user-supplied job runtime estimate is less than the actual job runtime. The other is the lack of knowledge of jobs and/or the potential benefits of providing an accurate runtime estimates, such as reducing job wait time and improving system utilization [15].

Considerable research has been conducted to improve the accuracy of job runtime estimates. For instance, Tang et al. presented a simple strategy to improve runtime estimates by multiplying a user-supplied runtime estimate with an adjusting parameter. They showed that the adjusted runtimes can achieve up to 20% reduction in average wait time and slowdown [25]. Gaussier et al. used a polynomial model to enhance the accuracy of runtime estimates [19]. Tsafrir et al. improved runtime estimates by using the average of two last actual runtimes of the user as a prediction (Last-2) [17].

Although the existing studies are capable of improving runtime prediction accuracy, they all overlook an important aspect of job runtime estimate, that is, underestimation rate. A job is underestimated if its runtime estimate is less than the actual time. Using an underestimated runtime estimate is *catastrophic* to a job as the job will be killed by the job scheduler when the actual runtime exceeds the runtime estimation. We argue

[1]ALCF stands for Argonne Leadership Computing Facility.

that *both the improvement of runtime accuracy and the reduction of underestimation rate are important to job scheduling*. Unfortunately, these two are conflicting goals. Improving the accuracy of runtime estimates of overestimated jobs could potentially increase the chance of underestimation. As we will illustrate later, although Last-2 [17] could greatly improve runtime prediction accuracy, the percentage of underestimated jobs is as high as 47%, which means about half of the jobs exit system abnormally and all those underestimated jobs needed to be rerun.

To address this problem, in this work we explore the Tobit model to make job runtime estimation. The Tobit model is a censored regression model designed to estimate relationships between variables when there is either left- or right-censoring in the latent variable (i.e., data censoring) [28]. Contrary to other machine learning techniques such as Support Vector Machine (SVM) and Random Forest, censored regression models are capable of censoring data without producing bias estimation [29]. This salient feature enables us to set the lower bound of runtime prediction and thus to alleviate the problem of underestimation in job runtime prediction. Note that the idea of data censoring cannot be adopted by other machine learning techniques, such as SVM, and Random Forest, because using censored data in these methods typically yield biased estimation toward the values of data censoring [30] [31], rather than the actual job runtime.

Specifically, in this work we develop a new runtime adjustment framework named *TRIP (Tobit RuntIme Prediction)* for improving prediction accuracy and reducing underestimation rate of job runtime estimates. TRIP consists of *a repository* for archiving historical job information and *a runtime adjuster* for predicting job runtime. It can be easily incorporated in existing job schedulers to make online job runtime prediction. For each incoming job, TRIP first retrieves related historical jobs from the repository and determines whether to adjust the user-supplied runtime of the incoming job. If TRIP decides to make an adjustment, it estimates job runtime by using an enhanced Tobit model on the extracted historical jobs. The overhead of TRIP is very lower (less than 1.0 second in our experiments). In TRIP, we enhance the standard Tobit model by adding elastic net regularization. The standard Tobit model requires independent features; however, here we only have a limited number of features and these features are not independent. The enhanced Tobit model is capable of automating feature selection and making prediction even in the presence of highly correlated features.

We evaluate TRIP by means of production workload traces. We first evaluate TRIP in terms of prediction accuracy and underestimation rate. As compared to other well-known machine learning methods such as SVM, Random Forest, and Last-2, TRIP is the only one that is capable of reducing underestimation rate while improving prediction accuracy. While all other methods are able to improve runtime prediction, they achieve this at the cost of high underestimation rate. Specifically, although Random Forest could improve an accuracy of 80%, the underestimation rate is as high as 32% (meaning one-third of the jobs will be killed and rerun). On the other hand, TRIP can achieve an accuracy of over 80% with underestimation rate of less than 5%.

We further perform trace-based simulations to quantify the scheduling performance gain by using TRIP. We assess the impact of improved runtime estimates with two scheduling policies, namely the widely used FCFS (First Come First Serve) with EASY backfilling [33] and the scheduling policy deployed at ALCF that favors large and old jobs [26]. Our results clearly show that TRIP is capable of greatly improving scheduling performance by up to 45%. While other machine learning methods such as SVM, Random Forest, and Last-2 improve some scheduling metrics (e.g., average job wait time and average bounded slowdown), they fail to improve system utilization due to the high underestimation rate.

The rest of this paper is organized as follows. We start by introducing background in Section II, including the Tobit model and job scheduling on HPC. Section III presents our analysis of two production workload traces. Section IV describes our TRIP design. The experimental results are presented in Section V. We discuss the related studies in Section VI. Finally, we conclude the paper in Section VII.

II. Background

A. Tobit Model

The Tobit model was first proposed by James Tobin in 1958 [28]. The standard Tobit model was created to analyze household expenditure on durable goods with the constraint that the expenditure cannot be negative. The Tobit model differs from other regression models in several ways. One of the most important features of the Tobit model is data censoring. The household expenditure can be equal to or higher than the threshold (zero). In the case that prediction of household expenditure is below the threshold, the values are censored, which means the values are set to the threshold. Another feature is that instead of using the least square approach to estimate parameters, the Tobit model utilizes maximum likelihood estimator. Amemiya has proved that maximum likelihood method can avoid bias estimation compared with the least square method [29].

Variations of the Tobit model were developed and they change where and when data censoring occurs. Amemiya classifies the variations into five categories (Tobit type I to Tobit type V) [29]. We explore Tobit type I in this study. Figure 1 gives the meanings of notations that will be used for the rest of the paper. X is a $m * n$ matrix, where m is the number of features and n is the number of instances. For each feature j, there is a corresponding parameter β^j. Parameters of all m features form a parameter vector $\vec{\beta} =< \beta^1, ..., \beta^j, ..., \beta^m >$. Similarly, we use $X_i (i = 1, ..., n)$ to represent an instance. For each instance i, the latent variable y_i^* linearly depends on X_i with an error term u_i. The latent variable y_i^* can be written as:

$$y_i^* = \underbrace{\beta^1 x_i^1 + ... + \beta^j x_i^j + \beta^m x_i^m}_{X_i \vec{\beta}} + u_i \tag{1}$$

The latent variable y_i^* is observed if $y_i^* > y_L$ and is not observed if $y_i^* \leqslant y_L$. Then the observed variable y_i is defined as:

$$y_i = \begin{cases} y_i^* & \text{if } y_i^* > y_L \\ y_L & \text{if } y_i^* \leqslant y_L \end{cases} \tag{2}$$

y_L is a threshold. Equation (2) is called data censoring. If the latent variable y_i^* is less than this threshold, the observed variable y_i is set to y_L.

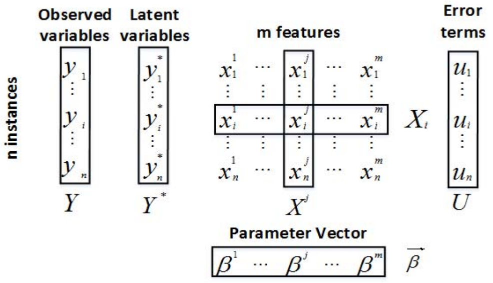


Fig. 1: Notations used to present the Tobit model. There are in total n instances and m features. Take instance i for example. A latent variable y_i^* depends on a vector of features $X_i =< x_i^1, ..., x_i^j, ..x_i^m >$ with the error term u_i. $\vec{\beta} =< \beta^1, ...\beta^j, ...\beta^m >$ is a paramter vector corresponding to m features.

The Tobit model utilizes maximum likelihood estimation to learn parameter $\vec{\beta}$ and σ. By converting to logarithm likelihood function, the objective function to learn $\vec{\beta}$ and σ becomes:

$$\underset{\vec{\beta},\sigma}{\operatorname{argmax}} \sum_{y_i > y_L} \log(\frac{1}{\sigma}\phi(\frac{y_i - X_i\vec{\beta}}{\sigma})) + \sum_{y_i = y_L} \log(1 - \Phi(\frac{X_i\vec{\beta} - y_L}{\sigma})) \tag{3}$$

Once $\vec{\beta}$ and σ are learned from Equation (3), prediction can be made from Equation (1) and Equation (2).

B. Job Scheduling on HPC

Job scheduling is responsible for determining the order in which jobs will be executed [34]. When submitting a job, the user is required to provide two basic information about the job:

- Number of compute nodes required by the job (i.e., job size)
- Runtime estimate of the job

The number of compute nodes is usually a rigid requirement. *Job runtime estimate* is the upper limit of the job runtime, which means the job will be killed if its actual runtime is greater than the estimate. The job scheduler determines where and when to execute the jobs. Once a job is submitted, the underlying job scheduler sorts all the jobs in the wait queue based on a job prioritizing policy. In the past, a number of job prioritizing policies have been proposed, and one of the widely used policy is FCFS (first come first served), which sorts jobs in the order of job arrivals [13]. At Argonne, a scheduling policy named WFP is adopted [25]. WFP determines job priority according to $(\frac{t_{queue}}{t_{supplied}})^3 * n_i$, where t_{queue}, $t_{supplied}$, and n_i denote job wait time, user-supplied job runtime estimate, and job size, respectively. Upon a job completion, the scheduler typically records the job along with a number of its attributes (e.g., user name, project name, job name, job submission time, job start and end time, etc.) in a workload log. Table I illustrates an example of a workload trace.

Despite of different types of job prioritizing policies, job scheduling often suffers from fragmentation, where free resources cannot meet the requirement of the next queued job and therefore remain idle until additional resources become available. *Backfilling* is a commonly used approach to enhance job scheduling by improving system utilization, where subsequent jobs are moved ahead to utilize free resources [13]. EASY backfilling is a widely used strategy, which allows short jobs to skip ahead under the condition that they do not delay the job at the head of the queue [33]. For both job prioritizing and backfilling, job runtime estimate plays a critical role.

III. Analysis of Job Runtime Estimates

We analyze workload traces from two supercomputers at ALCF [3]. One is the 48-rack IBM Blue Gene/Q machine named Mira [1] [4]. This system comprises 786,432 processors, and 768 terabytes of memory and features with 5D torus interconnect. The other is the 40-rack IBM Blue Gene/P system named Intrepid [2]. This system comprises 40,960 quad-core nodes, with 163,849 cores and uses 3D torus interconnect. Table II summarizes both logs.

We make two important observations from the trace analysis. First, many users tend to submit jobs frequently on these supercomputers. On average, each user on Mira submits 162 jobs in one year and each user on Intrepid submits 292 jobs in the nine-month period. Figure 2 summarizes the top 30 users on Mira and Intrepid. The top user on Mira submits 6500+ jobs in one year, while the top user on Intrepid submits 7500+ jobs in nine months. At a supercomputing center like XSEDE [6] or DOE [7] leadership computing facilities like ALCF, HPC resources access is provided through allocation. Each allocation is associated with a group of users working on a common project, typically in a specific field such as computation fluid dynamics, cosmology, molecular dynamics, etc. As such, HPC jobs are repetitious and have distinct characteristics in terms of their resource requirements. For example, one project aims to test the effectiveness of a newly developed model for weather forecasting. This requires the user(s) to submit the application repetitively with different input climate data [5]. Such a repetitive nature of HPC jobs enables us to leverage machine learning methods to make runtime prediction.

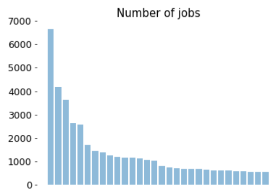
Second, user-supplied runtime estimates are highly inaccurate. Here, inaccurate runtime estimation either means overestimation or underestimation. Figure 3 shows the distributions of the runtime estimation accuracy (R) on Mira and Intrepid. Most of the jobs are overestimated with an R value less than 1. On Mira, 25% of the jobs have an R value under 0.25 and 40% of the jobs use less than 50% of their requested time.

TABLE I: Example of a workload trace

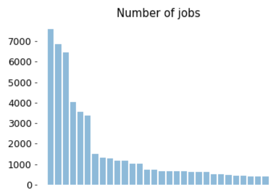
Job Number	Submit Time	Start Time	End Time	Requested number of nodes	User-supplied runtime estimates	Used number of nodes	User name	Project name	Job Name	Exit Status
1	1/1/2014, 0:00:00	1/1/2014, 14:00:09	1/2/2014, 1:13:42	49152	64800	49152	userA	projectA	a	0
2	1/1/2014, 0:20:00	1/1/2014, 14:00:17	1/2/2014, 00:34:52	512	38000	512	userB	projectA	b	1

TABLE II: Workload traces from Mira and Intrepid

Traces	Jobs	Users	Time Period
Mira	78,915	487	Jan/2014 - Dec/2014
Intrepid	68,936	236	Jan/2009 - Sep/2009

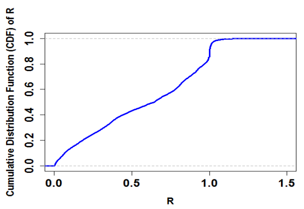


(a) Mira

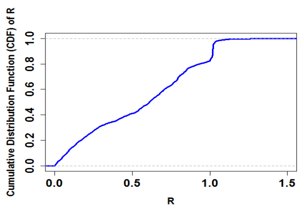


(b) Intrepid

Fig. 2: Number of jobs of the top 30 users on Mira and Intrepid.



(a) CDF of R on Mira



(b) CDF of R on Intrepid

Fig. 3: Cumulative Distribution Function (CDF) of R on Mira and Intrepid. R is defined as $\frac{t_{actual}}{t_{supplied}}$.

The user-supplied job runtime estimates on Intrepid are even worse. On Intrepid, 25% of the jobs have an R value under 0.2216 and 50% of jobs use less than 61% of their requested time. Both Mira and Intrepid logs have approximately 15% of jobs are underestimated with an R value over 1. Typically, underestimated jobs would be killed by the scheduler. The recording of these underestimated jobs was caused by the fact that the systems took some time to release the resources.

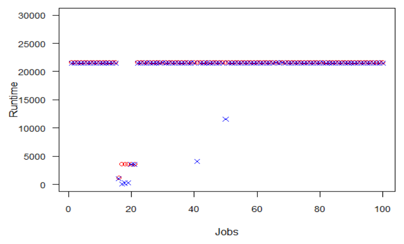
We further analyze the user-supplied runtime estimates per user. We find that although jobs, in general, are highly inaccurate, not all users have the same level of inaccuracy. Broadly speaking, users can be classified into three groups based on the accuracy of their supplied runtimes: *highly accurate, moderately accurate, highly inaccurate*. Figure 4 depicts three typical users from these groups. The user shown in Figure 4a provides highly accurate estimates of his/her jobs. On the other hand, the users shown in Figure 4b and 4c were not able to provide a good estimate of their job runtimes. The user depicted in Figure 4b tried to give a good estimate of his/her jobs by varying runtime estimates, whereas the user shown in Figure 4c always gave a fixed runtime estimate for all jobs. While we desire more users like the one shown in Figure 4a, we find that the majority of HPC users are like the ones shown in Figure 4b and 4c. Specifically, we find that more than 80% of the users at ALCF behave like the users shown in Figure 4b or 4c.

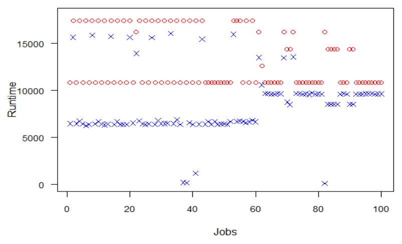
IV. TRIP Design

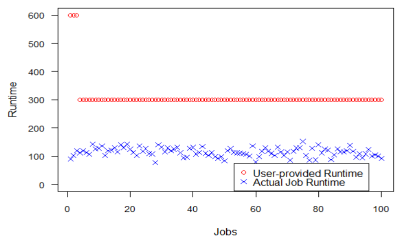
In HPC, highly overestimation of job runtime estimates is observed by many research works. Existing work mainly focused on reducing the runtime overestimation, while overlooking a critical feature of job runtime estimates, that is, the catastrophic consequence of underestimated job runtimes. A job will be killed by the scheduler when its actual runtime exceeds its runtime estimate. The design of TRIP is to reduce overestimation as well as to minimize underestimation. A high level overview of TRIP is presented in Figure 5.

Our design consists of two main components: a runtime adjuster and a job repository. Upon each incoming job, the adjuster first retrieves jobs with the same *user name, project name, and job name* from the repository. The information of these historical jobs is used to predict the runtime of the incoming job. The repository records jobs executed in the system. For each job, the repository records not only job submission attributes (e.g., job number, user name, project name, job name, job size, user-supplied runtime estimate, etc.), but also the attributes listed in Table III.

Figure 6 presents the flow of TRIP runtime adjustment method. For each incoming job, TRIP first extracts job attributes: user name, project name, and job name. It then retrieves historical jobs matching these job attributes from the repository. If the number of the retrieved historical jobs is less than a threshold, user-supplied runtime estimate is directly forwarded to the job scheduler. Otherwise, TRIP checks whether the average accuracy of the historical jobs is satisfactory (e.g., greater than a threshold). If yes, user-supplied runtime estimate is directly forwarded to the scheduler. If the historical jobs do not satisfy the threshold of the average accuracy, the enhanced Tobit model described in Section IV-A is invoked to make an adjustment of user-supplied runtime, denoted as t_{adjust}, which will be fed to the scheduler.


(a) Highly accurate estimates


(b) Moderately accurate estimates



(c) Highly inaccurate estimates

Fig. 4: Three typical user submission behaviors. In each plot, we show user-supplied runtimes and actual runtimes of one user.

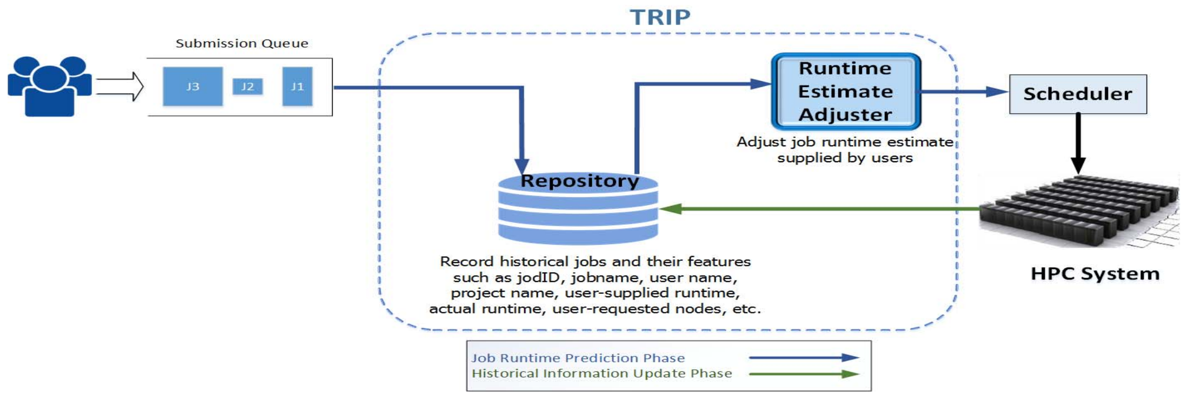

TRIP
Submission Queue
J3
J2
J1
Runtime Estimate Adjuster
Adjust job runtime estimate supplied by users
Scheduler
Repository
Record historical jobs and their features such as jodID, jobname, user name, project name, user-supplied runtime, actual runtime, user-requested nodes, etc.
HPC System
Job Runtime Prediction Phase
Historical Information Update Phase

Fig. 5: Overview of our TRIP design.

TABLE III: The features used for runtime prediction. Here, a class contains jobs with the same user name, project name, and job name.

Feature	Description
t_{last1}	the actual runtime of the last job of the same class
t_{last2}	the actual runtime of the second-to-last job of the same class
$t_{supplied}$	user-supplied job runtime estimate
$n_{supplied}$	the number of nodes requested by the user
$A_{average}$	the average accuracy of the historical jobs of the same class
A_{max}	the maximum accuracy of the historical jobs of the same class
$t_{longest}$	the longest actual runtime of the historical jobs of the same class
$t_{longest10}$	the longest actual runtime of the ten last jobs of the same class
$t_{average}$	the average actual runtime of the historical jobs of the same class
$t_{average10}$	the average actual runtime of ten last jobs of the same class
$t_{percentile25}$	the actual runtime of the 25th percentile historical jobs of the same class

A. Enhanced Tobit Model

TRIP explores the data censoring capability of the Tobit model to improve prediction accuracy as well as to reduce underestimation rate. The standard Tobit model described in Section II-A assumes that all features are independent. This requirement is impossible to achieve in practice due to the fact

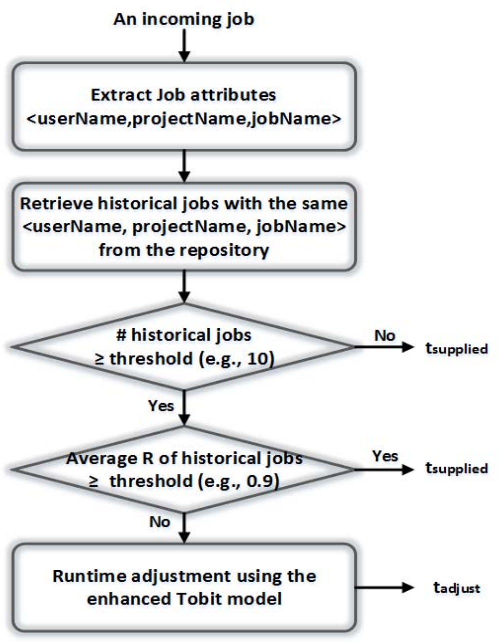

An incoming job
Extract Job attributes <userName,projectName,jobName>
Retrieve historical jobs with the same <userName, projectName, jobName> from the repository
# historical jobs ≥ threshold (e.g., 10)
No
tsupplied
Yes
Average R of historical jobs ≥ threshold (e.g., 0.9)
Yes
tsupplied
No
Runtime adjustment using the enhanced Tobit model
tadjust

Fig. 6: TRIP runtime adjustment method.

that limited features can be extracted from job logs.

To overcome this limitation, we make two enhancements. First, rather than purely based on the job attributes retrieved from job submission script, we include a number of other job attributes as listed in Table III. Second, in order to address the dependency of job features, we enhance the standard Tobit model by adding an elastic net regularization [36]. Elastic net regularization is a linear combination of L1 and L2 regularization, where L1 regularization performs feature selection and L2 regularization delivers stable results even when training a group of highly correlated features [37] [38]. Specifically, we replace the Equation (3) by the following formula to learn the parameter $\vec{\beta}$ and σ:

$$\underset{\vec{\beta},\sigma}{\text{argmax}} \sum_{y_i > y_L} \log(\frac{1}{\sigma}\phi(\frac{y_i - X_i\vec{\beta}}{\sigma})) + \sum_{y_i = y_L} \log(1 - \Phi(\frac{X_i\vec{\beta} - y_L}{\sigma})) \underbrace{\underbrace{-\lambda_1||\vec{\beta}||_1}_{L1} \underbrace{-\lambda_2||\vec{\beta}||_2}_{L2}}_{ElasticNet} \quad (4)$$

Here, y_L is set to the minimal of the actual job runtimes among all the retrieved historical jobs. X_i denotes the vector of attributes described in Table III and y_i is the actual runtime of historical job i.

We adopt stochastic gradient ascent to calculate the parameters (i.e., $\vec{\beta}$ and σ). Stochastic gradient ascent approximates the maximum iteratively [39]. It is considered as an efficient and fast converging method, because it approximates the maximum rather than computes the maximum. In our experiments, it usually takes less than 1.0 second to converge on a laptop.

Note that regularization parameters (i.e., λ_1 and λ_2) in Equation (4) is determined off-line through cross validation on a training dataset. For each incoming job being fed to the enhanced Tobit model, $\vec{\beta}$ and σ are learned online. Then, Equation (1) and (2) are used to make online job runtime prediction.

V. Experiments

To comprehensively assess the gain associated with using TRIP, we conduct two sets of experiments. Recall that, in section III, our design intends to alleviate overestimation and underestimation, whereas existing methods only focus on overestimation. Hence, in the first set of experiments, we focus on answering the question: *can our design outperform existing methods in terms improving accuracy and lowering the underestimation rate?* In particular, we compare TRIP with existing methods in terms of prediction accuracy and underestimation rate.

The ultimate purpose of improving job runtime estimates is to boost the performance of job scheduling. Therefore, we conduct the second set of experiments to answer the question: *to what extend could the improvement of job runtime estimates boost job scheduling?* To answer this question, we perform a series of trace-based simulations using the predictions made by our method and existing methods. We then compare the results in terms of job wait time, bounded slowdown, and system utilization.

The production traces listed in Table II are used in our experiments. To mimic an online operation, *we select the regularization strength (i.e. λ_1 and λ_2) by using 10-fold cross validation on the first month of jobs from each log, and use the rest of the log for online learning and prediction*. Hence, the results shown in the rest of the paper are calculated based on all the jobs in a log except for the jobs in the first month.

A. Prediction Accuracy and Underestimation Rate

The first set of experiments is intended to answer the first question, namely prediction accuracy and underestimation rate of TRIP as compared to other methods. We compare five sets of job runtime estimates:

1) The runtime estimates supplied by users.
2) The prediction results obtained by using the Last-2 method [17]. In this method, job runtime prediction is the average of a user's two last actual runtimes.
3) The prediction results obtained by applying Support Vector Machine (SVM) [40]. SVM is well-known supervised learning method used for both classification and regression. Here, we use SVM for regression.
4) The prediction results obtained by applying Random Forest [41]. A random forest is an ensemble learning method which constructs a number of decision trees on various sub-samples of the dataset and uses the average to improve the prediction accuracy.
5) The prediction results obtained by our TRIP design.

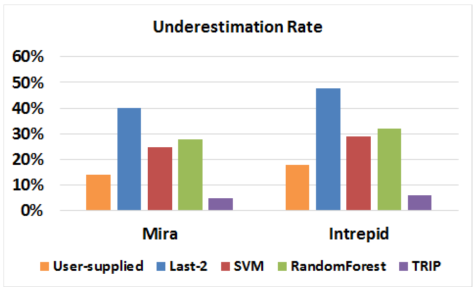
Note that the prediction processes of SVM and Random Forest are similar to TRIP, except that either SVM or Random Forest model replaces the enhanced Tobit model in Figure 6. Additionally, they all use the same features listed in Table III to predict actual job runtimes.

As mentioned in Section I, our design goal is to improve average accuracy as well as to reduce underestimation rate. Hence, the following two metrics are used for evaluation:

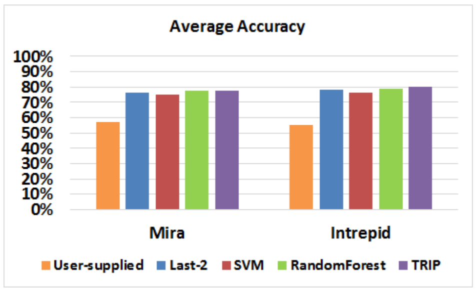
- **Underestimation rate**: We define underestimation rate (R_{under}) as the ratio of the number of underestimated jobs (n_{under}) to the total number of jobs (n_{total}) in a trace. Obviously, the lower the rate is, the better the runtime prediction is.
- **Average Accuracy**: Following the literature [17], we define the accuracy of job runtime estimate as:

$$A = \begin{cases} 1 & \text{if } t_{supplied} = t_{actual} \\ t_{supplied}/t_{actual} & \text{if } t_{supplied} < t_{actual} \\ t_{actual}/t_{supplied} & \text{otherwise} \end{cases} \quad (5)$$

Here, t_{actual} and $t_{supplied}$ denote actual job runtime and user-supplied job runtime respectively. This formula insures accuracy always lies between 0 and 1. The greater A means the job runtime estimate is closer to the actual job runtime. Average accuracy is calculated as an average amount across all jobs in a log.



(a) Underestimation Rate R_{under}. Clearly, the lower the rate is, the better the prediction is.



(b) Average Accuracy A

Fig. 7: Comparison of runtime estimates by applying different methods.

Figure 7 presents the prediction results, in terms of underestimation rate and average accuracy, obtained by different methods on both workloads. Clearly, our method outperforms the others in terms of both metrics. As shown in Figure 7a, in comparison to 15%-18% underestimation rate of the user-supplied runtime estimates, our method reduces the underestimation rate to 5%-8%, while Last-2, SVM and Random Forest increase underestimation rate, up to 47%, 29% and 31% respectively. In Figure 7b, compared with the average accuracy of user-supplied runtime estimates (55%-58%), all methods have the significantly higher average accuracy (75%-80%).

By combining the results presented in Figure 7a and 7b, we observe that although Last-2, SVM and Random Forest enhance prediction accuracy, their prediction accuracy improvement comes at the cost of increasing underestimation rate. TRIP is the only method that can reduce underestimation rate as well as improve runtime prediction accuracy. We believe this is due to the unique feature of Tobit, that is, the left data censoring capability (Equation(2)).

B. Impact on Job Scheduling

The second set of experiments is to answer the second question, i.e., to quantify the impact of our design on job scheduling. We conduct extensive trace-based simulations by means of the workload traces using the open-source simulator CQSim [9]. Both FCFS/EASY backfilling and WFP/EASY backfilling (i.e., the scheduling policy adopted at ALCF) are evaluated in this study. By using different scheduling policies as well as different workload traces, we intend to quantify the benefits of our design across different systems and across different scheduling policies. Three metrics are used to assess the impact of improved job estimates on job scheduling:

- **Average Job Wait Time**. Job wait time measures the time period between job submission to job start on a system.
- **Average Bounded Slowdown**. The slowdown of a job is the ratio of job response time to its actual runtime. However, this metric overemphasizes the importance of extremely short jobs. Feitlson et al. have proposed the bounded slowdown [32]:

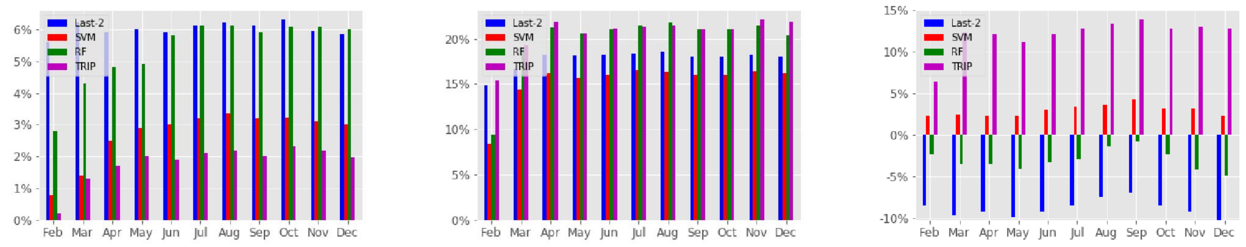
$$bounded\_slowdown = max(\frac{t_{wait} + t_{actual}}{max(t_{actual}, \tau)}, 1) \quad (6)$$

where t_{wait}, t_{actual} denotes job wait time and actual job runtime respectively. τ is a constant used to prevent the impact of extremely short jobs. We set τ to 60 seconds in our experiments. Average bounded slowdown is computed across all the jobs in a trace.
- **System Utilization**. It is the ratio of the node-hours used for running jobs to the total elapsed node-hours of a system.

Note that the first two metrics are from *user's point of view*, whereas system utilization is a metric from *the system's perspective*. Together, these metrics are used to assess the impact on scheduling from both user's and system's perspectives.

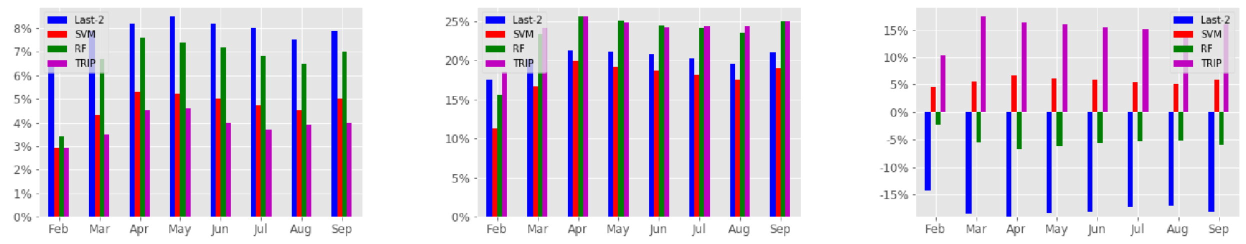
Similar to the results presented in the previous subsection, we compare scheduling performance using five sets of job runtime estimates. In the rest of the paper, we present the *relative improvement* over the baseline (i.e., the results obtained by using user-supplied runtimes).

Figure 8 and 9 present scheduling results using different runtime estimates where FCFS/EASY is adopted, and Figure 10 and 11 plot scheduling results using different runtime estimates where WFP/EASY is used. We make four key observations. First, our method outperforms Last-2, SVM and Random Forest. While other methods are able to improve some scheduling metrics, such as average job wait time and average bounded slowdown, they deliver negative or minor improvement in system utilization. Recall that our method is the only method reduce the underestimation rate as shown in Figure 7. Underestimated jobs are killed before completion, and therefore the other methods decrease the total used core hours. The more underestimated jobs the greater chance of empty job queue, which reduces system utilization. This also explains why the Last-2 method makes a greater reduction in job wait time. Due to the high underestimation rate of the Last-2 method, running jobs, on average, need fewer core hours to complete, and thus jobs in the wait queue need to wait less time. On the other hand, TRIP increases the total used hours, because fewer jobs are underestimated. Therefore,



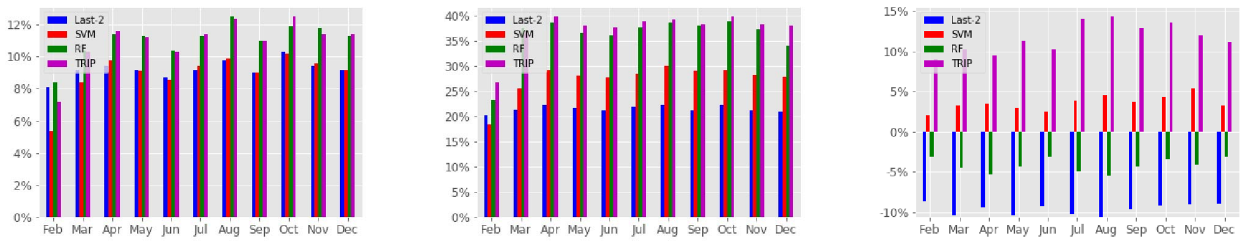
(a) Average Job Wait Time (b) Average Bounded Slowdown (c) System Utilization

Fig. 8: Scheduling performance on Mira by using different runtime estimates, and the baseline is obtained by using the original user-supplied runtime estimates. Here FCFS with EASY backfilling is used.



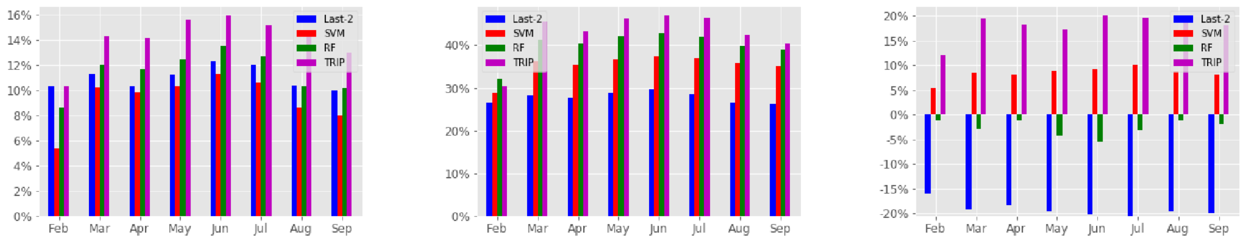
(a) Average Job Wait Time (b) Average Bounded Slowdown (c) System Utilization

Fig. 9: Scheduling performance on Intrepid by using different runtime estimates, and the baseline is obtained by using the original user-supplied runtime estimates. Here FCFS with EASY backfilling is used.



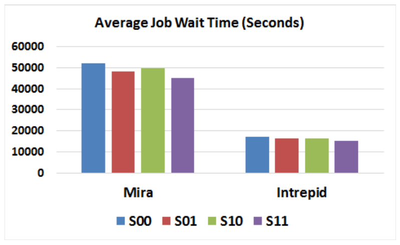
(a) Average Job Wait Time (b) Average Bounded Slowdown (c) System Utilization

Fig. 10: Scheduling performance on Mira by using different runtime estimates, and the baseline is obtained by using the original user-supplied runtime estimates. Here WFP with EASY backfilling is used.

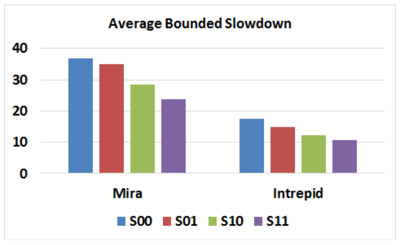


(a) Average Job Wait Time (b) Average Bounded Slowdown (c) System Utilization

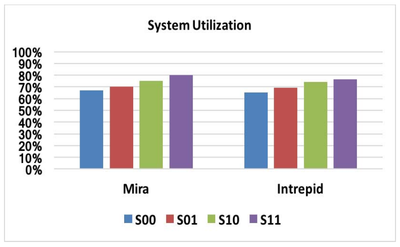
Fig. 11: Scheduling performance on Intrepid by using different runtime estimates, and the baseline is obtained by using the original user-supplied runtime estimates. Here WFP with EASY backfilling is used.



(a) Average Job Wait Time



(b) Average Bounded Slowdown



(c) System Utilization

Fig. 12: Comparison of scheduling performance metrics by using selected methods shown in Table IV. Here, WFP with EASY backfilling is used.

TRIP is capable of improving system utilization by increasing used core hours.

Second, using improved runtime prediction on the Intrepid trace has a more obvious effect compared with that on Mira. This is due to the trace characteristic. 77% of the jobs on Intrepid are adjusted, whereas 67% of the jobs on Mira are adjusted. Therefore, the more percentage of jobs on Intrepid can be predicted. Additionally, using our method and Last-2, SVM, Random Forest on Intrepid makes the greater improvement in average accuracy compared with Mira. This result suggests that we can enhance job scheduling performance by improving runtime estimation as much as possible.

Third, all four methods need less than one month to reach their best performance. Initially, the repository has very limited job information. Last-2 need two historical jobs of a user to make runtime prediction, while other methods need ten. That is why we can see that Last-2 method has good performance in terms of job wait time and bounded slowdown in the first month. Other methods quickly catch up after the first month and keep relatively higher improvement in bounded slowdown.

Forth, WFP/EASY benefits more from the use of accurate runtime estimates than FCFS/EASY. TRIP reduces average bounded slowdown by up to 45% using WFP/EASY, and by less than 25% using FCFS/EASY. Using TRIP with WFP/EASY can improve system utilization by up to 20%, while the improvement is 16% for FCFS/ EASY. This is due to the fact that WFP/EASY uses runtime estimates for both job prioritizing and backfilling, whereas FCFS/EASY only uses runtime estimates for backfilling. With respect to WFP/EASY, an overestimated job could obtain a higher priority thus leading to less wait time and slowdown by improving prediction accuracy. For FCFS/EASY, the adjusted runtime estimates only affect backfilling. The job priority in FCFS/EASY is determined by job arrival times, and therefore job prioritizing remains the same for FCFS/EASY no matter whether we use the original runtime estimates or improved runtime estimates. Overestimated short jobs can benefit from more accurate runtime estimates, because they obtain more opportunities to be backfilled. TRIP offers more accurate runtime prediction for short jobs, and therefore it achieves greater improvement in bounded slowdown, while less improvement in job wait time compared with other methods when using FCFS/EASY.

TABLE IV: Selective Methods

Name	Backfilling	Job Prioritizing
S00	$t_{supplied}$	$t_{supplied}$
S01	$t_{supplied}$	t_{adjust}
S10	t_{adjust}	$t_{supplied}$
S11	t_{adjust}	t_{adjust}

The results in Figure 8, 9, 10 and 11 suggest that both job prioritizing and backfilling benefit from more accurate runtime estimates. One interesting question is that which one benefits more from the improved runtime estimation. To answer this question, we evaluate the impact of adjusted runtime estimates on different aspects of scheduling (i.e., job prioritizing and backfilling) using TRIP. Toward this end, we design four selective methods for WFP/EASY scheduling, each using different runtime estimates for job prioritizing and backfilling (see Table IV). For example, S00 means using user-supplied job runtime estimates for both prioritizing and backfilling. S10 means using adjusted runtime estimates for backfilling, while S01 means using adjusted runtime estimates for job prioritizing.

As shown in Figure 12, more accurate runtime estimates have positive effects on both job prioritizing and backfilling. S10 makes a greater improvement in bounded slowdown and system utilization compared with S01, which implies backfilling is the main reason for the performance enhancement. Additionally, the improvement made by S11 compared with S00 is not simply by adding the improvement from S01 and S10. Both S01 and S11 prioritize jobs using our adjusted runtime estimates, while S10 and S00 use the user-supplied runtime estimates to prioritize jobs. Because the backfilling decision is made after job prioritizing, job priorities influence backfilling. Therefore, the jobs that can be backfilled in S10 and S11 may not be the same. S10 uses the user-supplied runtime estimates for prioritizing and uses the adjusted runtime estimates for backfilling potentially allowing more jobs to be backfilled.

VI. Related Work

A. *Inaccuracy of Job Runtime Estimates*

Users often overestimate their runtimes. Cirne and Berman reported that in four different traces, 50% to 60% of jobs use less than 20% of their supplied times [10]. Similarly, Ward, Mahood, and West studied a workload trace from a Cray T3E and found that on average the jobs only used 29% of their supplied times [11]. Chiang et al. also found that 35% of the jobs use less than 10% of their supplied times [12]. Other workload analyses lead to the similar results [13] [14].

B. Predicting Job Runtimes

Historical workload logs are considered especially useful for improving the accuracy of job runtime estimates. A number of efforts have been devoted to improving runtime accuracy by utilizing historical information. The Last-2 scheme is one of the simplest and widely used strategies [17]. It takes the average of the latest two actual runtimes from the same user as the prediction. While the method is simple, it is far superior to runtime estimates supplied by users and is capable of doubling accuracy [17]. Gibbons attempted to categorize jobs by predefined job attributes and made predictions based on different job categories [16]. Smith et al. improved the categorization of jobs [20] [21] [22]. Jobs are no longer categorized by predefined characteristics, rather templates are used to categorize jobs. Templates, sets of job attributes, are automatically generated by a genetic algorithm and then jobs are assigned to a set of categories generated from templates. This work takes advantage of job similarity in one group and uses the mean of historical jobs runtimes in one group for the job runtime prediction. Gaussier et al. applied the polynomial model to predict runtimes, which was shown to improve several workload traces from the Parallel Workloads Archive [27] [19].

In contrast to existing studies which solely focus on high prediction accuracy, our study aims at high prediction accuracy as well as low underestimation rate. Lowering underestimation rate is crucial for job scheduling as a job will be killed if its actual runtime is greater than its estimated runtime. To the best of our knowledge, this is the first study which targets at both prediction accuracy and underestimation rate.

C. Impact of Adjusted Runtime Estimates on Job Scheduling

There are several studies attempting to improve scheduling performance by adjusting job runtime estimates. Some studies indicate that more accurate runtime estimates have minimal impact on system performance [15] [22] [23]. However, Chiang et al. have shown that significant improvement in system performance can be achieved by using more accurate runtimes [12]. In addition, Zhang et al. presented that although the improvement in overall system performance may not be obvious, more accurate runtime estimates can substantially improve individual job performance [18]. In our prior study, we found that more accurate runtime estimates could reduce average slowdown time by up to 55% depending on scheduling policy [25]. Gaussier et al. showed that the prediction made by their approach improves the average bounded slowdown by 28% [19]. In this study, we used the production workload traces from different systems and provided in-depth analysis of runtime impact on different aspects of scheduling (e.g., job prioritizing and backfilling). We believe the results presented in this work provide useful insights for the HPC community.

VII. Conclusion

In this work, we have presented TRIP to improve job runtime estimates for HPC. In contrast to existing studies solely emphasizing on improving prediction accuracy by reducing runtime overestimation, our design stresses the importance of both prediction accuracy and underestimation rate. Our design is based on an in-depth analysis of the production workload traces collected from two leadership computing machines, namely the 40,960-node Intrepid machine and the 49,152-node Mira machine at ALCF. The analysis on both logs has illustrated that user-supplied runtime estimates suffer from overestimation and underestimation. In order to improve prediction accuracy and decrease underestimation rate, TRIP has explored the data censoring of the Tobit model, a unique feature which cannot be adopted by other machine learning methods, and have enhanced the Tobit model to address the problem of feature dependence in predicting job runtimes. The overhead of TRIP is very low and it takes, on average, less than 1 second to learn and predict job runtime of one incoming job. The extensive trace-based experiments have clearly demonstrated that TRIP is capable of improving an accuracy to over 80% as well as reducing an underestimation rate to about 5%, whereas other machine learning methods, i.e. SVM, Random Forest, and Last-2, improve their prediction accuracy at the cost of increasing underestimation rate. Moreover, our in-depth analysis of runtime impact on scheduling has indicated that the use of our design can greatly improve scheduling performance. TRIP enhances the system utilization by up to 20%, whereas other methods make negative or minor improvement in this metric. The amount of performance gain depends on the workload as well as the underlying scheduling policy. For instance, more accurate runtime estimates have a greater impact on WFP/EASY in comparison with FCFS/EASY. A key reason is that WFP/EASY uses runtime estimates for both job prioritizing and backfilling, whereas FCFS/EASY only uses runtime estimates for backfilling.

While the present work serves as a basis demonstration of the value of using TRIP to improve job runtime estimates, there are several avenues for taking these ideas further. A natural direction to extend our current work is to deploy the proposed framework as a software component for various scheduling packages such as Cobalt and Slurm [8] [35]. Another interesting avenue is to include more features as inputs to the predictive model.

Acknowledgment

The work at the Illinois Institute of Technology is supported in part by US National Science Foundation grants CNS-1320125 and CCF-1422009. The work at the Argonne Leadership Computing Facility is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

References

[1] "Mira". [Online]. Available: https://www.alcf.anl.gov/mira

[2] "Intrepid". [Online]. Available: https://www.alcf.anl.gov/intrepid

[3] "Argonne Leadership Computing Facility (ALCF)". [Online]. Available: https://www.alcf.anl.gov

[4] "IBM Redbooks, IBM System Blue Gene Solution: Blue Gene/Q System Administration". Vervante, 2013.

[5] "Mira Early Science Program". [Online]. Available: https://www.alcf.anl.gov/programs/esp-mira

[6] "Extreme Science and Engineering Discovery Environment (XSEDE)". [Online]. Available: https://www.xsede.org/

[7] "Department of Energy (DOE)". [Online]. https://energy.gov/

[8] "Cobalt Project". [Online]. Available: http://trac.mcs.anl.gov/projects/cobalt

[9] "CQSim: An Event-driven Simulator". [Online]. Available: http://bluesky.cs.iit.edu/cqsim.

[10] W. Cirne and F. Berman, "A Comprehensive Model of the Supercomputer Workload", Proc. of IEEE International Workshop on Workload Characterization, 2001.

[11] W. Ward, C. Mahood, and J. West, "Scheduling Jobs on Parallel Systems using a Relaxed Backfill Strategy", Proc. of Job Scheduling Strategies for Parallel Processing, 2002.

[12] S.-H. Chiang, A. Arpaci-Dusseau, and M. Vernon, "The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance", Proc. of Job Scheduling Strategies for Parallel Processing, 2002.

[13] A. Mu'alem and D. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling", IEEE Transactions on Parallel and Distributed Systems 12(6), 529-543, 2001.

[14] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Characterization of Backfilling Strategies for Parallel Job Scheduling", Proc. of the International Conference on Parallel Processing Workshops, 2002.

[15] C. Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "Are User Runtime Estimates Inherently Inaccurate?", Proc. of Job Scheduling Strategies for Parallel Processing, 2004.

[16] R. Gibbons. "A Historical Application Profiler for Use by Parallel Schedulers", in Job Scheduling Strategies for Parallel Processing. 1997.

[17] D. Tsafrir, Y. Etsion, and D. Feitelson, "Backfilling using System-generated Predictions Rather Than User Runtime Estimates", IEEE Transactions on Parallel and Distributed Systems 18(6), 789-803, 2007.

[18] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam, "Improving Parallel Job Scheduling by Combining Gang Scheduling and Backfilling Techniques", Proc. of IEEE International Parallel and Distributed Processing Symposium, 2000.

[19] E. Gaussier, D. Glesser, V. Reis, D. Trystram. "Improving Backfilling by using Machine Learning to Predict Running Times", Proc. of IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, 2015.

[20] W. Smith, "Prediction Services for Distributed Computing", Proc. of IEEE International Parallel and Distributed Processing Symposium, 2007.

[21] W. Smith, I. Foster, and V. Taylor, "Predicting Application Runtimes with Historical Information", Journal of Parallel and Distributed Computing 64(9), 1007-1016, 2004.

[22] W. Smith, V. Taylor, and I. Foster, "Using Run-time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance", Proc. of Job Scheduling Strategies for Parallel Processing, 1999.

[23] D. Zotkin and P. Keleher, "Job-length Estimation and Performance in Backfilling Schedulers, Proc. of IEEE International Symposium on High Performance Distributed Computing, 1999.

[24] W. Tang, Z. Lan, N. Desai, and D. Buettner, "Fault-aware, Utility-based Job Scheduling on Blue Gene/P Systems", Proc. of IEEE International Conference on Cluster Computing, 2009.

[25] W. Tang, N. Desai, D. Buettner, Z. Lan, "Analyzing and Adjusting User Runtime Estimates to Improve Job Scheduling on the Blue Gene/P", Proc. of IEEE International Parallel and Distributed Processing Symposium, 2010.

[26] W. Allcock, P. Rich, Y. Fan, and Z. Lan, "Experience and Practice of Batch Scheduling on Leadership Supercomputers at Argonne", Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), 2017.

[27] "Parallel Workloads Archive". [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/

[28] J. Tobit, "Estimation of Relationships for Limited Dependent Variables", Econometrica 26 (1): 24-36, 1958.

[29] T. Amemiya, "Tobit Models", Advanced Econometrics, pp. 360-411, 1985.

[30] W. Greene, "Censored Data and Truncated Distributions, In Mills, T.C., Patterson, K. (eds.), Palgrave Handbook of Econometrics, Volume 1: Econometric Theory, Palgrave Macmillan, Hampshire, 2005.

[31] A. Henningsen, "Estimating Censored Regression Models in R using the censReg Package, [Online] Available: cran.r project.org/web/packages/censReg/vignettes/censReg.pdf.

[32] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and Practice in Parallel Job Scheduling", Job Scheduling Strategies for Parallel Processing, pp. 1-34, 1997.

[33] D. Lifka, "The ANL/IBM SP Scheduling System", In Job Scheduling Strategies for Parallel Processing, pp. 295-303, Springer-Verlag, 1995.

[34] S. Iqbal, R. Gupta, and Y. Fang, "Planning Considerations for Job Scheduling in HPC Clusters", Dell Power Solutions, February 2005.

[35] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple Linux Utility for Resource Management, in Job Scheduling Strategies for Parallel Processing. Springer, 2003, pp. 44-60.

[36] H. Zou and T. Hastie, "Regularization and Variable Selection via the Elastic Net", Journal of the Royal Statistical Society, 2005.

[37] R. Tibshirani, "Regression Shrinkage and Selection via the Lasso", Journal of the Royal Statistical Society, 1996.

[38] A. Y. Ng, "Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance", Proc. of ICML, 2004.

[39] L. Bottou, "Stochastic Learning", In Advanced lectures on machine learning, Springer, pp. 146-168, 2004.

[40] C. Cortes, V. Vapnik, "Support-vector networks", Machine Learning, Vo. 20, no. 3, pp 273-297, 1995.

[41] L. Breiman, "Random forests", Machine Learning, vol. 45, no. 1, pp.5-32, 2001.