Job Grouping Based Intelligent Resource Prediction Framework

Beste Oztop[1][0009−0007−3731−8425], Benjamin Schwaller[2][0000−0002−9282−8977], Vitus J. Leung[2][0009−0008−3950−1626], Jim Brandt[2][0000−0002−8605−5795], Brian Kulis[1][0000−0002−1704−3838], Manuel Egele[1][0000−0001−5038−2682], and Ayse K. Coskun[1][0000−0002−6554−088X]

[1] Boston University, Boston MA 02215, USA
{boztop,bkulis,megele,acoskun}@bu.edu
[2] Sandia National Laboratories, Albuquerque, NM, USA
{bschwal,vjleung,brandt}@sandia.gov

Abstract. In High-Performance Computing (HPC) systems, users estimate the resources they need for job submissions based on their best knowledge. However, underestimating the required execution time, number of processors, or memory size can lead to early job terminations. On the other hand, overestimating resource requests leads to inefficiencies in job backfilling, wasted compute power, unused memory, and poor job scheduling, ultimately reducing the overall system efficiency. As we enter the exascale era, we want to utilize resources more efficiently than ever. Existing schedulers lack mechanisms that predict the resource requirements of batch jobs. To address this challenge, we design a data-driven recommendation framework that leverages historical job information to predict three key parameters for batch jobs: the execution time, the maximum memory size, and the maximum number of CPU cores required. In contrast to existing machine learning-based resource prediction methods, we introduce an online resource suggestion framework that considers both underestimates and overestimates in the batch jobs' resource provisioning. Our framework outperforms the baseline method with no grouping mechanism by achieving over 98% success in eliminating underpredictions and reducing the amount of overpredictions.

Keywords: HPC Resource Management · Recommendation Framework · Data-driven Resource Prediction

---

This work has been partially funded by Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under Contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

1 Introduction

High-Performance Computing (HPC) systems host and run many scientific applications from domains like genomics, climate modeling, and computational physics [21,15,11]. Users of these large-scale systems provide resource requests through batch job submission scripts. They request the necessary resources for their applications, such as maximum wallclock time, number of processors, and maximum memory size, based on their best knowledge [19]. This user-based request for batch job resources often leads to underprediction or overprediction problems. From the resource utilization perspective, underprediction refers to requesting fewer resources than needed, leading to early job terminations, out-of-memory, and insufficient processor errors. One of the resource utilization logs we analyze in this work shows that a total of 7.46% of the jobs failed due to out-of-memory and out-of-time errors. Overprediction of resources, on the other hand, results in increased job wait times, idle memory and computational resources, reduced quality of services for users, and overall decreased system efficiency [16]. Thonglek et al. demonstrate the overprediction and waste of resources in the Google Borg Cluster and reveal that users requested extra numbers of CPUs and larger memory sizes for their jobs over 95% of the time, which leads to these valuable resources sitting idle [25].

Predicting the resource requirements of batch jobs to avoid these problems has been an ongoing machine learning (ML) problem in the HPC domain [24,18]. A resource predictor is successful if it accurately estimates the job runtime and the required resources for successful execution. In this work, we design a resource recommendation framework that predicts the resource requirements of batch jobs and gives suggestions to the users before the jobs are run on the HPC system. Figure 1 presents the overview of our framework.

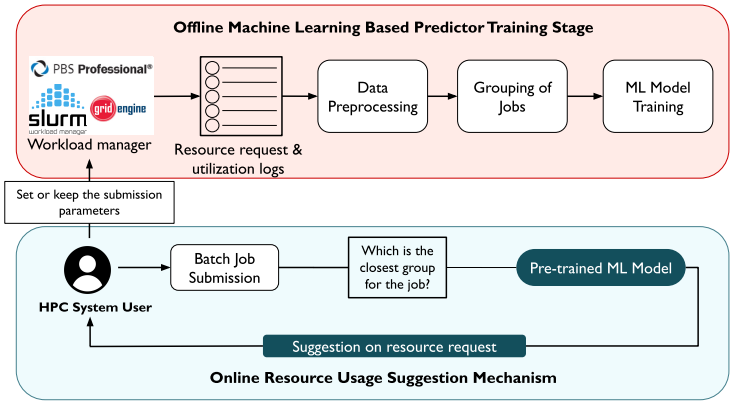

Fig. 1. Overview of the proposed resource recommendation framework.

In this framework, we group the historical workload logs based on their submission parameters and train individual ML models for these clusters. We determine the number of clusters for each dataset through our experiments. After the training, we find the cluster most similar to each testing data point and make predictions on resource requirements. Finally, we evaluate the framework's performance with historical resource utilization datasets. The contributions of this work are as follows:

- We design a job grouping-based intelligent prediction framework for HPC batch job resource provisioning to reduce overpredictions and minimize underpredictions[1].
- We analyze and compare ML models' resource usage prediction accuracy using real-world workload manager logs from 5 different HPC systems to develop the recommendation framework.
- We evaluate how our framework reduces resource waste by minimizing overprediction, comparing its effectiveness against baseline methods.

Our work demonstrates the advantages of job grouping and machine learning for resource prediction in HPC batch jobs. Our recommendation framework can reduce the early job terminations due to execution time underprediction down to 0.89%. Additionally, experimental results show a substantial decrease in the execution time overestimation and unnecessary memory allocation.

Section 2 provides the state-of-the-art resource prediction in the HPC domain. In Section 3, we explain the design principles and our motivation for building the recommendation framework. The details about the datasets we use in the experimental procedure, the baseline methods, and the performance metrics are in Section 4. In Section 5, we compare our framework's performance with the baseline methods and evaluate its success. Finally, we conclude our paper and give ideas for future work in Section 6.

2 Related Work

Existing workload managers and schedulers like SLURM [4], Altair PBS [2], and Sun Grid Engine [1] allow users to request resources in a fine-grained manner. They accept job submissions, then queue and schedule the batch jobs based on their scheduling policies. However, they assign resources to the batch jobs based on user requests and do not provide resource usage predictions or suggestions for batch job submissions. User-dependent resource requests can result in early job terminations or idle resources due to users' under- or overprediction. To prevent out-of-time, out-of-memory, and other resource shortage problems, accurately predicting batch job resource requirements has been an ongoing interest in the cloud and HPC domains. One existing work for cloud systems resource management scales resources proactively by considering memory and CPU-related

[1] Our implementation is available at: https://github.com/peaclab/intelligent-resource-allocation

anomalies at runtime [14]. Furthermore, Zhang et al. [27] propose using reinforcement learning to predict the resource requirements of microservices. In HPC systems, on the other hand, previous work on resource prediction for batch jobs depends on historical log data from various workload managers. In their recent work, Menear et al. [18] develop an execution time predictor for batch jobs using the XGBoost [8] model. Regression models like XGBoost are common choices in predicting resource requirements based on historical data. However, neural networks can also solve resource prediction problems since they can capture complex relationships between resource logs and utilization statistics. For example, by analyzing the Google Cluster Dataset from 2019 [26], Thonglek et al. [25] introduce a time series prediction model based on Long Short-Term Memory (LSTM) networks [13]. This model uses submission and utilization parameters to predict CPU and memory utilization percentages for Google cluster jobs.

To predict multiple types of resources with a single model, Tanash et al. [24] propose a regression-based ML model. The maximum memory size and execution time of jobs running on SLURM-operated systems are the target variables. They use specific user accounts' resource usage information to build the model and try to optimize RMSE and R^2 values while reducing the wait time of jobs. They achieve 86% accuracy in predicting the execution time and memory requirements of jobs at the same time. They use resource utilization information as training features, which limits the framework from making predictions at the submission time. In other words, despite their high accuracy and low RMSE values, existing methods do not offer recommendations to users before job submission. Since we aim to make predictions before jobs are executed, we cannot rely on resource utilization information while building our framework.

Existing work considers execution time prediction separately from CPU and memory requirements. For example, one study addresses job queue time prediction resulting from requesting more wallclock time than necessary [20]. Another study focuses on idle resources and the inefficiencies in the system due to the overprediction of the number of CPU cores and maximum memory size [25]. In our work, we build a framework that is able to predict execution times, the required number of processors, and the maximum memory size of jobs.

Additionally, as we demonstrate in Section 5 of our work, regression-based models tend to underpredict execution times and require additional strategies to solve this problem. Contrary to the existing work, our approach simultaneously tackles underprediction and overprediction problems by proposing overprediction-aware buffering and resampling mechanisms, and a recommendation framework that helps HPC system users optimize their batch job submissions.

3 Machine Learning Based Resource Prediction

Our framework's main goal is to accurately predict the execution time and the required CPU and memory resources at the submission time. Figure 2 overviews our offline ML model training and testing mechanisms. In this section, we justify some of our important design choices.

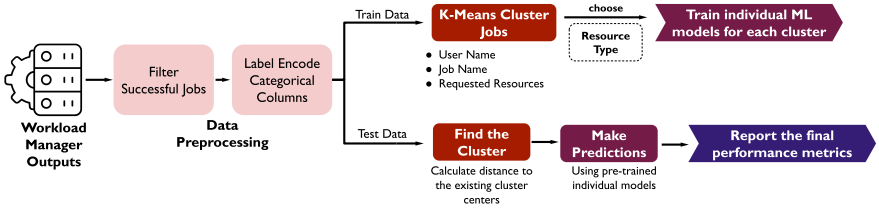

Fig. 2. Offline training methodology of our proposed prediction framework.

3.1 Data Preprocessing and Grouping

The datasets we use in this study contain information on successfully executed and failed batch job submissions. Since our main goal is to make accurate predictions, we only use data related to the submission and resource usage of the completed jobs in the training stage. Therefore, the first phase of data preprocessing is to filter out jobs that failed to execute successfully. Then, we apply label encoding to categorical columns, such as user name, job name, and other string-like parameters. Note that we exclude the job ID column that has repeating values and provides inaccurate job resource utilization similarities.

In the job grouping stage, we use the features from the workload managers which are available before the job execution starts. Some of the features we use include the user's name, job name, job ID, account name, hard resource requests (e.g., the number of processors, maximum memory size requests from batch job scripts), and wallclock limits (i.e., maximum allowed runtime the user sets). For batch job information grouping across datasets, we apply K-means clustering from the scikit-learn library [22]. We determine the optimal number of clusters for each experimental dataset using the **elbow method**. To apply the elbow method, we calculate the Within-Cluster Sum of Squares (WCSS) score for varying numbers of clusters (k) with the following formula:

$$\text{WCSS} = \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \mu_k\|^2, \tag{1}$$

where:

- K is the total number of clusters,
- C_k is the set of all data points in the k-th cluster,
- x_i is a single data point that belongs to the cluster C_k,
- μ_k is the centroid of the k-th cluster,
- $\|x_i - \mu_k\|^2$ is the squared Euclidean distance between the data point x_i and the centroid μ_k.

The WCSS score shows the distances of data points from the centroids of their respective clusters. A smaller WCSS indicates that the data points are closer to their centroids and clusters contain more similar batch job information. After calculating the WCSS values, we observe the elbow point for each dataset and

determine the optimal number of clusters. We show the elbow method results in Section 5.1.

3.2 Machine Learning Model Selection and Training

In this section, we explain our recommendation framework's offline model training stage design choices. The framework provides continuous numerical predictions on resource usage and execution time. We use XGBoost and Random Forest linear regression models for model training since they offer low computational overhead and high accuracy in predicting resource utilization for HPC applications [18]. Their design allows us to make continuous predictions based on selected training features, which are submission parameters from the corresponding workload manager. We do not include resource usage information in the model training feature set.

LSTM and other neural network models also offer advantages and high prediction accuracy in time-series prediction problems. Thonglek et al. propose an LSTM-based resource predictor that consistently predicts values lower than the user's requests while avoiding underprediction of CPU and memory utilization [25]. However, in this approach, the input features for the LSTM model include CPU and memory utilization. Since we focus on a **recommendation** framework that can accurately predict resource requirements of jobs **before** they run on a given HPC system, we cannot rely on or use resource usage features that are unavailable before job execution. Therefore, in this work, we implement different regression models, compare their performance, and show their efficiency in resource prediction while leaving the implementation of neural networks for future work.

To build our recommendation system, we train resource prediction models with an offline training principle. While using the XGBoost model, we note that there might be negative predictions since the original model with default parameters does not force the predictions to be positive. Since the target variables we want to predict are resource utilization values, we apply an additional logarithmic transform before training the model to prevent any negative execution time and resource utilization predictions. Furthermore, we experiment with different time windows and train durations, and the results of our experiments for training and testing data splits are in Section 5.2. The ML models we train with historical logs from workload managers build the recommendation framework's offline training stage.

3.3 Handling Underpredictions

One of our primary goals with this work is to prevent early batch job terminations. Regression models require addressing the problem of underprediction for the requested wallclock time, number of processors, and memory allocated to each job, as we introduce in the Related Work section.

In our proposed methodology, we group similar jobs based on their submission parameters in the training set and train separate models for each cluster.

However, when we apply these models to new jobs in the test set, they tend to underestimate resource usage compared to the actual values. We analyze this underprediction problem in detail through our experiments in Section 5. To mitigate this problem, we present two strategies in this section.

The first solution we offer is to add a buffer to the ML model predictions. In their recent work, Dai et al. [10] scale the execution time predictions by a predefined value. In contrast, Cui et al. [9] add an offset value based on the training dataset characteristics. Since we aim to limit overpredictions while mitigating the underpredictions, choosing a buffer value that reduces the underpredictions without increasing the overpredictions by a great amount is important. As a result, we decide to use twice the standard deviation of underpredictions within each cluster as our buffer value. First, we separate a validation set among all clusters in the training stage. Then, we train individual models with training sets and test these models on validation sets to calculate the buffer value. An illustration of our buffer calculation is in Figure 3. At the test stage, we observe the effectiveness of our framework by shifting the predictions in the test dataset with the buffer value. We use the following formula for the 2σ calculation.

$$2\sigma = 2\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}, \tag{2}$$

where:

- n is the number of job information within the cluster validation set,
- x_i is the resource usage of job i,
- μ is the mean resource usage within the cluster validation set.

During the experimental procedure, we observe that the clusters formed in each training dataset do not clearly separate jobs based on resource utilization. This is expected, as we do not consider the resource usage information while forming the clusters. If the amount of batch jobs with high resource usage or execution time is low for a dataset, the clusters we create do not represent these jobs well. In other words, the number of long-running jobs with high CPU or memory resource usage in the clusters can be substantially smaller than the number of short-running jobs with smaller resource needs.

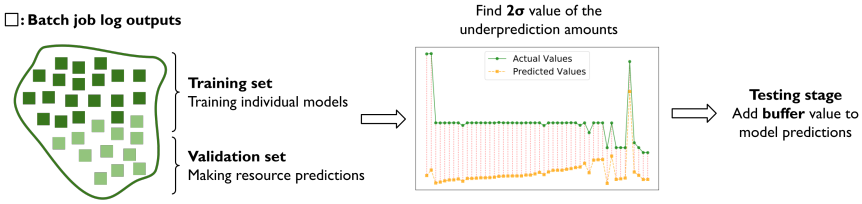

Fig. 3. This is a representative figure for buffer value calculation per cluster. Each square represents a data point with different batch job information.

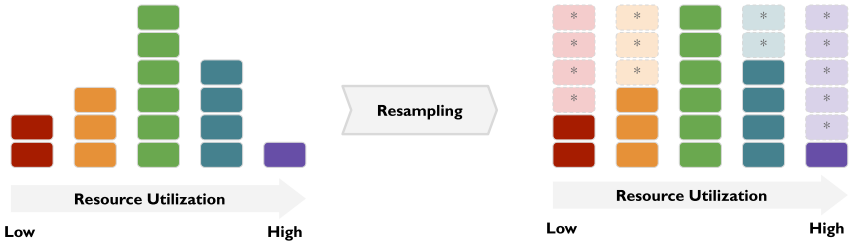

Fig. 4. This figure illustrates the resampling method. Each rectangle represents a data point corresponding to a different batch job information within a training dataset cluster.

To alter this nonuniform distribution in training data and possibly overcome the underprediction problem, we introduce a second strategy to resample less frequent batch jobs in the training clusters. For each cluster, we determine the minimum and maximum resource usage values for CPU and memory utilization, and the execution time. Based on these values, we create equally spaced bins and group the jobs in a sense. Then, we find the bin with the maximum number of samples and resample other bins to reach this maximum value. We aim to have representative job information in the training set and ensure that it sufficiently reflects the diversity of resource usage within each cluster. We visualize this approach in Figure 4. In our experimental procedure, we investigate the results of the execution time, maximum memory size, and maximum number of processor predictions of batch jobs with and without these two different strategies. We refer to the buffer value as 2σ and indicate the resampling method whenever applicable.

4 Experimental Methodology

This section explains the datasets and features we use for ML model training, the baseline methods we compare our method with, and the performance metrics we choose to test the effectiveness of our recommendation framework.

4.1 Datasets

We evaluate our approach using real system workload manager outputs from different schedulers, such as SLURM and Sun Grid Engine. In this subsection, we describe the datasets we use in our study. These datasets include information on both successfully executed jobs and failed jobs, and possibly along with the failure reasons. In this work, we use only the successful job data in model training to predict the target resources as accurately as possible. Table 1 summarizes the features we use to train each dataset's ML models.

Sandia Dataset We analyze and make resource predictions on a resource utilization dataset from Sandia National Laboratories. The dataset duration is from May to October 2024. There are 313,615 completed jobs from 614 different users, and the execution time, mean, and maximum CPU and memory utilization of batch jobs are available.

NREL Eagle Dataset The National Renewable Energy Laboratory (NREL) provides publicly available data on resource requests and utilization for over 11 million jobs executed on the Eagle supercomputer [12]. Eagle served as NREL's primary HPC system from 2018 to 2024, hosting 2,000 nodes and providing a total computing capacity of 8 petaflops per second [3]. This dataset includes job information from November 2018 to February 2023, collected by SLURM, and submitted by 936 users, with user and project account names included. This dataset contains 67% successful jobs, and 7.28% of the total jobs failed due to timeout, which emphasizes the need for accurate execution time prediction.

M100 CINECA Dataset Borghesi et al. [6] provide a 31-month-long resource utilization data using a SLURM plugin from Marconi-100 (M100) supercomputer located at the CINECA datacenter. M100 supercomputer hosts 347,776 cores and provides up to 21.64 petaflops per second [23]. The dataset is organized in months [7], and we test our approach by using a subset of it from December 2021, providing 79,173 successful jobs from 402 users. The failed jobs we exclude from this work have 6.63% timeout errors and 1.23% out-of-memory errors.

Fugaku Dataset Fugaku supercomputer, located at the RIKEN Center for Computational Science (R-CCS) in Kobe, Japan, comprises 7,630,848 cores and provides a total computing capacity of 442.01 petaflops per second. Fugaku is ranked as the sixth supercomputer system to reach exascale computing in 2024 [23]. A three-year resource utilization dataset (March 2021- April 2024) from Fugaku is publicly available [5]. In our work, we utilize a subset of this dataset containing 338,796 completed jobs, submitted by 483 users in April 2024.

Boston University Shared Computing Cluster (BU SCC) Dataset The BU Shared Computing Cluster (SCC) is a Linux-based system with over 28,000 CPU cores, 400 GPU nodes, and 14 petabytes of research data storage. Boston University's Research Computing Services (RCS) manages the cluster and the workload manager is the Sun Grid Engine (SGE). SGE stores the historical job submissions and resource usage data, known as "accounting information". In this work, we utilize the accounting data from 2023 containing 11,831,831 successful jobs.

4.2 Baseline Methods

One of the fundamental baselines for resource prediction and allocation in HPC systems we consider is the user-requested values. In the datasets we use, as Table 1 summarizes, we have access to the execution time, peak memory, and CPU request information except for the Sandia Dataset. For the remaining datasets,

Table 1. Selected Training Features, Present at Submission Time

Feature Name	Sandia	Eagle	M100	Fugaku	BU SCC
User Name/ID	✓	✓	✓	✓	✓
Job Name	✓	✓		✓	✓
Project Account Name/ID		✓	✓		✓
Partition		✓	✓		
Priority			✓		
Working Directory	✓				
Requested Wallclock Time		✓	✓	✓	✓
Requested Memory Size		✓		✓	
Requested Processors		✓		✓	✓
Requested Number of Nodes		✓		✓	
Requested GPUs		✓			
Requested Frequency				✓	

the first baseline for comparison and improvement is the resource requests by users at the time of submission, which we refer to as **User Requested**. Additionally, we compare our framework's results with the method Menear et al. [18] propose, where the authors use an XGBoost model to predict the execution time of jobs. The model characteristics and hyperparameters are publicly available [17]. In our experimental procedure, we refer to this second baseline method as **Single XGB**.

4.3 Performance Metrics

Selecting performance metrics that suit our framework's goals is essential for a more effective comparison and better interpretability. In this work, we measure the framework's success based on two main objectives: preventing underpredictions and reducing the amount of overpredictions. Classical error metrics, such as Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), do not specifically reflect the underpredictions or the amount of overpredictions. These metrics compare each prediction point only with the true values and report the statistical error. To gain more insights into our results, we need to use alternative metrics to better assess the success of our framework.

For our framework's first objective, we report the ratio of jobs with underpredictions to the total number of jobs, as Dai et al. [10] suggest. Ideally, this ratio should be equal to 0 where none of the jobs experience early job terminations. We define this metric as the **Underprediction Ratio (UR)** and calculate it using the following formula:

$$\text{Underprediction Ratio (UR)} = 100 \times \frac{\text{Number of Jobs with Underprediction}}{\text{Total Number of Jobs}} \tag{3}$$

While this metric shows how many jobs can execute successfully using our framework, it does not reflect the amount of underpredictions. We include the

MAE distribution and median error of resource predictions to support the UR results in Section 5.3. Our motivation is to quantify the prediction errors since an underprediction of 1 hour is more significant than 1 minute of underprediction.

Our second goal, reducing overpredictions, requires observing the user-requested values, actual resource usage, and the predictions our framework makes. Ideally, our framework's predictions should not be much larger than the actual values, especially compared to the user-requested values. While we report the total amount of overprediction for different resource types in the experimental procedure, we want to observe the changes in resource request distribution without using aggregated values. Consequently, we use the following formula for the **overestimation factor** that Menear et al. [18] introduce.

$$\text{Overestimation Factor (OF)} = \frac{\text{Requested Resource of Predicted Utilization}}{\text{Actual Resource Utilization}} \tag{4}$$

For example, consider a user who requests 32 CPU cores for two jobs. One job requires only 2 cores, resulting in an OF of 16; the other requires 8 cores, giving an OF of 4. If our framework predicts the number of processors as 4 and 16 cores for these jobs, the OF becomes 2 in both cases. However, there is more improvement in predictions for the first job, where the OF drops from 16 to 2. Therefore, observing the OF distribution of user-requested values and framework predictions on the test set shows our framework's success. We calculate this metric for both users' resource requests and ML model predictions for maximum memory size and number of processors. Then we visualize the results using density plots in Section 5.4 to evaluate how effectively our framework reduces overpredictions.

5 Results and Discussion

In this section, the goal is to show our framework's success as a recommendation system. Through our experiments, we find the optimal number of clusters for job grouping and choose the window sizes for model training. We present the inference experiment results of our framework's predictions on execution time, maximum memory size, and number of processors for different datasets by comparing them with the baseline methods. Note that we represent the method we offer without any underprediction prevention mechanism as 'Clustering', use 'XGB' and 'RF' as abbreviated versions of the XGBoost and Random Forest models, and indicate the buffer and resampling methods as '2σ' and 'Resampling'.

5.1 Finding Optimal Number of Clusters

As we explain in section 3.2, we calculate and plot the WCSS scores for varying numbers of clusters for each experimental dataset. We select the elbow point from the WCSS plots in Figure 5 and determine the number of clusters to create

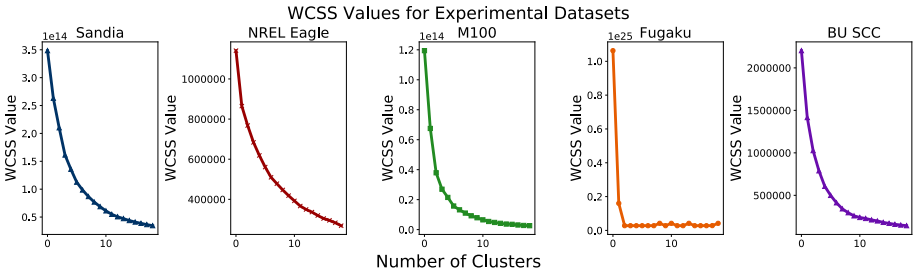

Fig. 5. The WCSS value per cluster

for each dataset. At this stage, for the Fugaku and M100 datasets, increasing the number of clusters further than 4 and 6 may result in empty data frames. This indicates that the natural grouping for these smaller datasets, compared to Sandia, Eagle, and BU SCC, should have fewer clusters. Also note that the K-Means algorithm selects the initial centroids randomly, and we initialize the same centroids in repeating experiments by using the same random state.

As a result, we choose 18 for the Sandia, Eagle, and BU SCC datasets, 4 for the Fugaku dataset, and 6 for the M100 CINECA dataset as the number of clusters in the experimental procedure.

5.2 Evaluating the Effect of Window Size on Underpredictions

The experimental datasets in this work contain batch job logs from a wide range of durations, from 1 month to almost 5 years. The train and test time split for these datasets significantly changes the predictions and the framework's success. To decide on an optimal train and test data split, we conduct execution time prediction experiments with varying training windows. The window size is the duration of a subset we train and test our models with. The update interval refers to the amount of time shift we apply for each window to cover all data points. For each window, we use the last 20% of batch jobs for testing. We separate 40% of the training data as the validation set to calculate the buffer values. For each dataset, we use the update interval and window size pairs in Table 2 where the units are the number of days.

We determine the window size for each dataset by analyzing the UR, along with the mean and standard deviation of underprediction amounts across different window sizes. Note that the underprediction amount is the total difference

Table 2. Window Size and Update Interval Pairs for Train-Test Split Experiment

Parameters	**Sandia**	**Eagle**	**M100**	**Fugaku**	**BU SCC**
Window Size	7, 14, 30	7, 14, 30, 60, 90	3, 7, 14	3, 7, 14	7, 14, 30, 60, 90
Update Interval	7, 7, 30	7, 7, 30, 30, 30	3, 7, 7	3, 7, 7	7, 14, 30, 30, 30

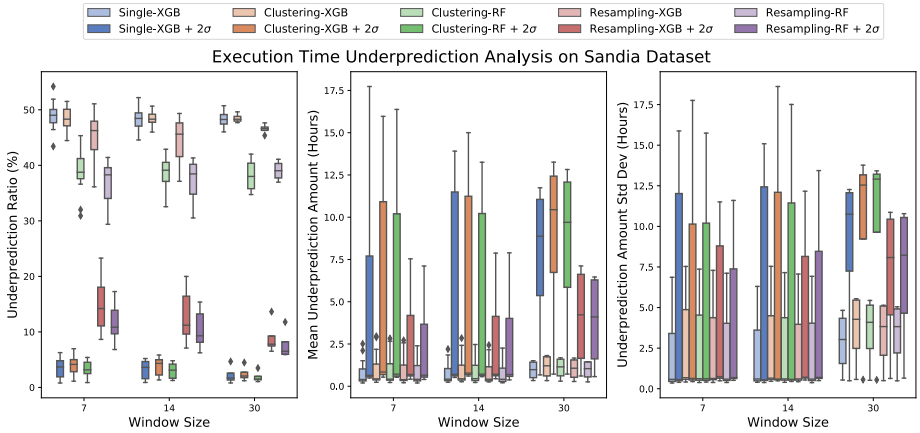

Fig. 6. Window size experiment results for Sandia Dataset execution time prediction.

between actual and predicted execution times. We create bar plots for the separate datasets' window size experiment results. We include all of the models we test and the underprediction prevention mechanisms in these plots. For example, we present the Sandia dataset execution time prediction results in Figure 6. We observe that the standard deviation and the mean underprediction amount of models with buffer addition are larger than the no buffer cases. This results from the decrease in the number of underpredicted test data points with the buffer addition.

As a result of this experiment, we determine the optimal window size and update interval pairs as follows: [7,7] days for the Sandia, [30,30] days for Eagle, [3,3] days for M100 and Fugaku, and [60,30] days for the BU SCC datasets. We generally observe that looking at a smaller window for execution time prediction gives better performance. The next sections provide more insight into the results for these best-performing window size values.

5.3 Execution Time Prediction

In this section, we compare the results of our framework on execution time prediction with the baseline methods using the performance metrics we select. We treat execution time prediction as a regression problem and test two strategies we develop to mitigate the underprediction problem. We shift the windows for each dataset's optimal window size to cover the entire duration. Therefore, the underprediction ratios in Table 3 show the results from the full datasets using our training strategy.

The execution time prediction experiment outcomes show that clustering similar jobs before model training can improve the UR value up to 12.08% in the BU SCC dataset. The baseline method underpredicts more than the Clustering-RF

Table 3. Predicted Execution Time (Underprediction Ratio Results).

Method	**Sandia**	**Eagle**	**M100**	**Fugaku**	**BU SCC**
Single-XGB	48.59%	50.16%	48.59%	46.91%	49.87%
Clustering-XGB	48.45%	49.31%	49.90%	45.78%	49.58%
Clustering-RF	38.22%	40.89%	46.31%	40.71%	37.79%
Resampling-XGB	46.65%	42.35%	48.21%	40.73%	47.07%
Resampling-RF	39.20%	37.66%	45.68%	38.58%	37.04%
Single-XGB + 2σ	2.01%	1.37%	2.10%	1.89%	0.88%
Clustering-XGB + 2σ	2.28%	3.80%	2.24%	2.61%	1.24%
Clustering-RF + 2σ	1.77%	2.86%	2.19%	2.29%	0.89%
Resampling-XGB + 2σ	8.52%	4.70%	5.17%	3.60%	2.06%
Resampling-RF + 2σ	7.39%	4.65%	4.75%	3.57%	1.82%

model for all datasets. However, to provide better recommendations to users, our framework should have UR values close to zero in the practical case. Therefore, we consult the underprediction handling strategies we introduce in Section 3.4. The resampling strategy provides improvement up to 9.39% in the Sandia dataset, when the offline training model is Random Forest. Across datasets, we observe the smallest UR values resulting from adding a buffer value of 2σ to the baseline model.

Figure 7 provides an MAE density plot for the Eagle dataset to analyze the prediction errors in execution time further. Although the baseline method with a buffer value performs the best in terms of UR, we observe that the resampling strategy improves the MAE distribution and reduces the median error in execution time compared to the Single-XGB and Clustering-XGB methods. We observe a similar trend for other experimental datasets, which can motivate users to select the clustering and resampling strategy instead of training a single model to reduce the MAE for different job durations.

For our second objective, reducing the overestimation of execution time, we need to observe the predictions with a buffer value addition. As the first step, we visualize the user requests of execution time and the Resampling-RF method

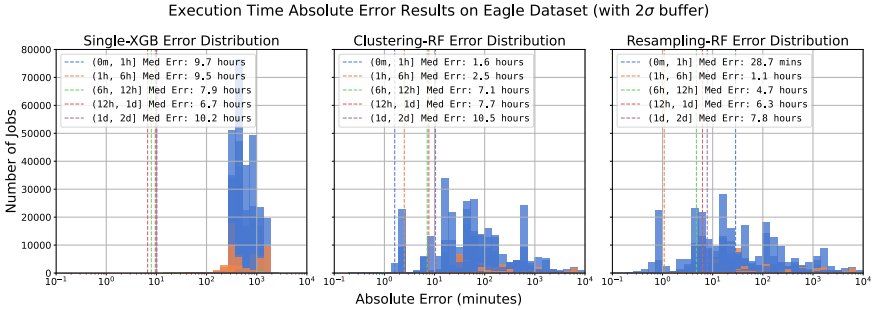

Fig. 7. Mean absolute error distribution in execution time prediction for Eagle Dataset.

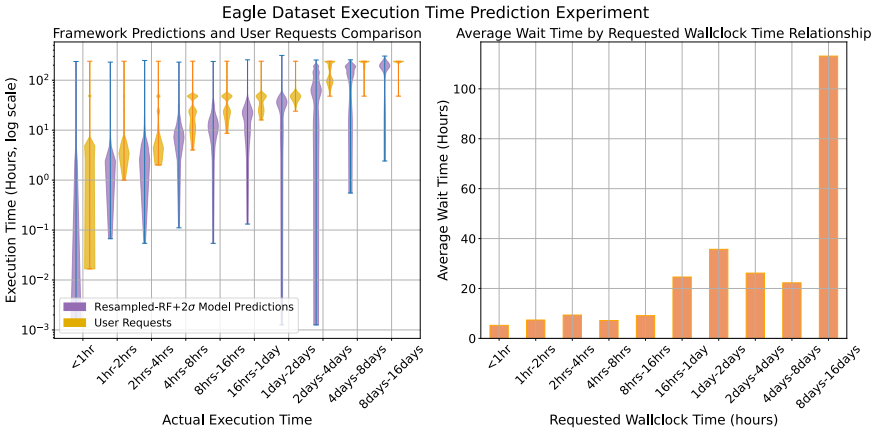

Fig. 8. Framework execution time predictions comparison with the user requests for Eagle Dataset.

with buffer value addition predictions in Figure 8. Through this analysis, we observe that our framework can predict smaller execution time regions than the user requests, for ranges of actual execution time of jobs. In the practical case, this can result in shorter job waiting times since the wallclock time request and waiting time are positively correlated, as Figure 8 shows. Hence, reducing the wallclock time requests can improve the mean waiting time and increase the quality of service for HPC system users.

In figure 9 we provide the total amount of overestimation compared to the user requests. This figure illustrates the advantages of using a machine learning model to predict the execution time of jobs without relying on the user's manual selections. Our framework reduces the overestimation in total execution time by applying regression models for all datasets. The clustering method with 2σ buffer and resampling strategies gives the biggest improvement in the execution time overpredictions, which reaches a 13.60 times reduction for the M100 dataset

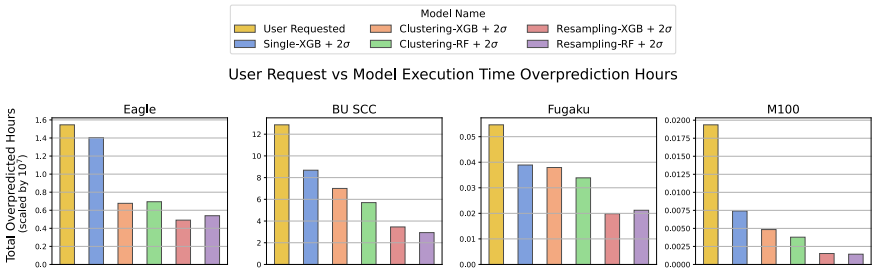

Fig. 9. Total amount of execution time overprediction comparison for user requested values with machine learning model predictions.

with the Resampled RF model. We also note that the differences between XGB and RF models are very subtle with a buffer value addition for the execution time prediction, both in terms of UR values and the overprediction results.

In light of the execution time experiments for varying workload manager outputs, we conclude that resampling the less frequent batch job information in the machine learning model training stage and applying a buffer value to the predictions can reduce the number of underpredictions below 2% of total jobs while keeping the overpredictions under control.

5.4 Predicting Resource Requirements of Batch Jobs

We extend our framework to predict additional resource types, along with execution time, to provide HPC users with extensive job resource utilization recommendations. The first resource type we make predictions on is the maximum memory size requirement of a given batch job. In the Fugaku dataset, we have access to the requested maximum memory size and the utilization values, while the Sandia dataset does not provide user requests. Therefore, we perform resource prediction experiments on both datasets and illustrate the improvement in users' resource selection only with the Fugaku dataset.

Our job grouping-based framework achieves a 2.59% underprediction ratio for the maximum memory size prediction in the Sandia dataset, outperforming the baseline, which underpredicts the memory resources for 48.31% of total test jobs. The reduction in the underprediction ratio is down to 5.50% in the Fugaku dataset. Table 4 presents our results of underprediction prevention mechanisms along with the baseline and clustering models.

Moreover, in Figure 10, we observe the amount of user-selected memory size requests compared to our framework predictions. Fugaku system users without the aid of any prediction framework overestimate the maximum memory size of their jobs by 10^{10} with a density centering around 0.4. This system contains jobs with thousands of nodes allocated to them; hence, the maximum memory size requests reach up to 10^{12} bytes. Although we add the 2σ buffer value to our predictions, the framework still reduces the overestimation factor to the

Table 4. Predicted Peak Memory Utilization (Underprediction Ratio Results).

Method	**Sandia**	**Fugaku**
Single-XGB	48.31%	49.87%
Clustering-XGB	48.45%	47.58%
Clustering-RF	37.32%	46.10%
Resampling-XGB	46.87%	46.38%
Resampling-RF	37.94%	45.20%
Single-XGB + 2σ	2.48%	3.04%
Clustering-XGB + 2σ	3.22%	5.48%
Clustering-RF + 2σ	2.59%	5.50%
Resampling-XGB + 2σ	12.22%	6.72%
Resampling-RF + 2σ	9.74%	6.70%

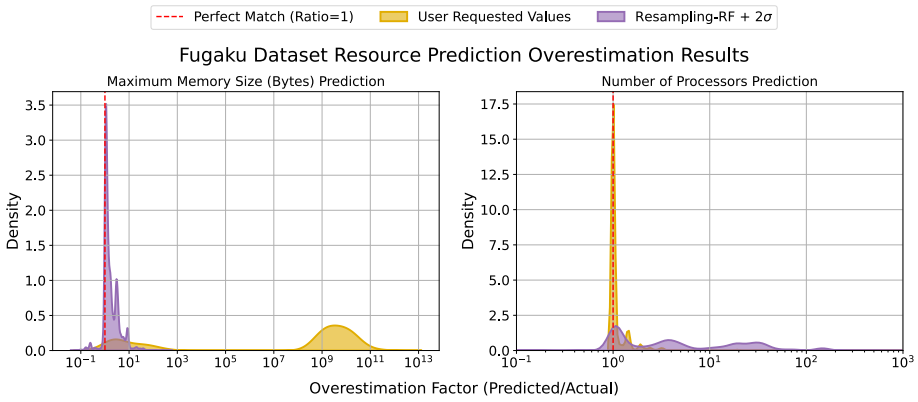

Fig. 10. Overestimation factor density plots for maximum memory size and the number of processors prediction in the Fugaku dataset.

$10^0 - 10^2$ range. This result shows the effectiveness of our framework in reducing overestimations and its usability in a real HPC system where users tend to request more memory resources for batch job submissions than they need.

To support the use case of applying the framework for other resource types, we also conduct experiments on the number of processor utilization of batch jobs in the Sandia and Fugaku datasets. Similar to the maximum memory size utilization, the Fugaku dataset includes both requested and utilized number of processors information, while the Sandia dataset provides only utilization information. For the execution time and memory size, we use regression models and directly predict continuous values. However, since the number of cores can only take integer values, we round the predictions of the regression models to the closest larger integer value for this prediction problem.

With the same training and testing methodology as in previous sections, we achieve an underprediction ratio of 11.05% in the Fugaku dataset using only the Clustering-RF method without adding the 2σ buffer and outperform the baseline method that underpredicts the number of cores for 25.78% of the jobs in the test dataset. In the Sandia dataset predictions, adding the buffer value results in UR ratio reduction down to 3.79% with the baseline method as visible in Table 5. Testing the success of our framework in terms of the number of processor predictions supports our claim of using this methodology as a recommendation system.

However, a tradeoff exists between preventing the underestimations of a given resource type and limiting the amount of overestimations. This phenomenon is visible from the number of processor prediction results in Figure 10. Observing the user-requested number of cores and the machine learning model predictions with buffer values shows us that models overpredict the number of cores more than the users. Unlike the maximum memory size prediction, users tend to re-

Table 5. Predicted Maximum Number of Core Usage (Underprediction Ratio Results).

Method	**Sandia**	**Fugaku**
Single-XGB	37.42%	25.78%
Clustering-XGB	28.75%	16.57%
Clustering-RF	23.81%	11.05%
Resampling-XGB	31.88%	19.90%
Resampling-RF	22.45%	12.99%
Single-XGB + 2σ	3.79%	0.27%
Clustering-XGB + 2σ	4.74%	1.39%
Clustering-RF + 2σ	4.09%	1.29%
Resampling-XGB + 2σ	14.35%	2.03%
Resampling-RF + 2σ	10.32%	2.41%

quest fewer resources compared to our framework, which shows us the variance in user requests. While the manual selection of resources gives satisfying results for the number of core requests per batch jobs, the maximum memory size prediction needs the help of our framework.

6 Conclusion and Future Work

This paper proposes an intelligent, job grouping-based resource recommendation framework for HPC system users. Users can select the resource type for the framework to predict and the underprediction prevention mechanisms they choose to use.

Using this framework, we reduce the underprediction of execution time, memory, and CPU resources to less than 2% and outperform the baseline method. The framework also reduces the amount of overestimation in batch job resources and possibly will decrease the user waiting time. The historical workload manager logs we include in this work are real system datasets, and we observe that waiting time is correlated with the users' resource requests. Therefore, HPC users can benefit from this recommendation system by predicting their jobs' resource requirements before submitting them to the workload manager, using historical resource utilization datasets of their own.

As future work, we aim to use a real workload manager simulator or a real production system supercomputer to show our framework's success in reducing the early terminations of jobs, longer queue times, and waste of valuable computing and memory resources. To extend this work, we plan to investigate the same batch jobs' resource utilization variations and their relationship with the resource contention in the system.

Acknowledgments. The authors are pleased to acknowledge that the computational work reported on in this paper was performed on the Shared Computing Cluster which is administered by Boston University's Research Computing Services. www.bu.edu/tech/support/research/.

References

1. Altair Grid Engine Homepage, https://altair.com/grid-engine
2. Altair PBS Professional Homepage, https://altair.com/pbs-professional
3. National Renewable Energy Laboratory (NREL): HPC User Facility, https://www.nrel.gov/computational-science/hpc-user-facility.html
4. SLURM Workload Manager Documentation, https://slurm.schedmd.com
5. Antici, F., Bartolini, A., Domke, J., Kiziltan, Z., Yamamoto, K.: F-DATA: A Fugaku Workload Dataset for Job-centric Predictive Modelling in HPC Systems (Jun 2024). https://doi.org/10.5281/zenodo.11467483
6. Borghesi, A., Di Santi, C., Molan, M., Ardebili, M.S., Mauri, A., Guarrasi, M., Galetti, D., Cestari, M., Barchi, F., Benini, L., et al.: M100 ExaData: a data collection campaign on the CINECA's Marconi100 Tier-0 supercomputer. Scientific Data **10**(1), 288 (2023)
7. Borghesi, A., Santi, C.D., Molan, M., Ardebili, M.S., Mauri, A., Guarrasi, M., Galetti, D., Cestari, M., Barchi, F., Benini, L., Beneventi, F., Bartolini, A.: M100 dataset 5: from 22-01 to 22-02 (Jan 2023). https://doi.org/10.5281/zenodo.7589942
8. Chen, T., Guestrin, C.: XGBoost: A Scalable Tree Boosting System. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 785–794. KDD '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939785
9. Cui, H., Takahashi, K., Shimomura, Y., Takizawa, H.: Clustering Based Job Runtime Prediction for Backfilling Using Classification. In: Workshop on Job Scheduling Strategies for Parallel Processing. pp. 40–59. Springer (2024)
10. Dai, Y., Dong, Y., Lu, K., Wang, R., Zhang, W., Chen, J., Shao, M., Wang, Z.: Towards Scalable Resource Management for Supercomputers. In: SC22: International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–15 (2022). https://doi.org/10.1109/SC41404.2022.00029
11. Dongarra, J., Keyes, D.: The co-evolution of computational physics and high-performance computing. Nature Reviews Physics **6**(10), 621–627 (2024)
12. Duplyakin, D., Menear, K.: NREL Eagle supercomputer jobs (Feb 2023), https://data.openei.org/submissions/5860
13. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Computation **9**(8), 1735–1780 (11 1997). https://doi.org/10.1162/neco.1997.9.8.1735
14. Kardani-Moghaddam, S., Buyya, R., Ramamohanarao, K.: ADRL: A Hybrid Anomaly-Aware Deep Reinforcement Learning-Based Resource Scaling in Clouds. IEEE Transactions on Parallel and Distributed Systems **32**(3), 514–526 (2021). https://doi.org/10.1109/TPDS.2020.3025914
15. Kurth, T., Treichler, S., Romero, J., Mudigonda, M., Luehr, N., Phillips, E., Mahesh, A., Matheson, M., Deslippe, J., Fatica, M., Prabhat, P., Houston, M.: Exascale Deep Learning for Climate Analytics. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 649–660 (2018). https://doi.org/10.1109/SC.2018.00054
16. Lamar, K., Goponenko, A., Peterson, C., Allan, B.A., Brandt, J.M., Dechev, D.: Backfilling HPC Jobs with a Multimodal-Aware Predictor. In: 2021 IEEE International Conference on Cluster Computing (CLUSTER). pp. 618–622 (2021). https://doi.org/10.1109/Cluster48925.2021.00093
17. Menear, K., Duplyakin, D.: Eagle Jobs (2024), https://github.com/NREL/eagle-jobs/tree/master

18. Menear, K., Nag, A., Perr-Sauer, J., Lunacek, M., Potter, K., Duplyakin, D.: Mastering HPC Runtime Prediction: From Observing Patterns to a Methodological Approach. In: Practice and Experience in Advanced Research Computing 2023: Computing for the Common Good. p. 75–85. PEARC '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3569951.3593598
19. Newaz, M.N., Mollah, M.A.: Memory Usage Prediction of HPC Workloads Using Feature Engineering and Machine Learning. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region. p. 64–74. HPCAsia '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3578178.3578241
20. Pal, A., Malakar, P.: An Integrated Job Monitor, Analyzer and Predictor. In: 2021 IEEE International Conference on Cluster Computing (CLUSTER). pp. 609–617 (2021). https://doi.org/10.1109/Cluster48925.2021.00091
21. Patil, S., Devrani, R., J, B.: The Potential of HPC in Enhancing AI-Driven Comparative Genomics Studies. In: 2024 IEEE 13th International Conference on Communication Systems and Network Technologies (CSNT). pp. 1053–1059 (2024). https://doi.org/10.1109/CSNT60213.2024.10545899
22. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine Learning in Python. the Journal of Machine Learning Research **12**, 2825–2830 (2011)
23. Strohmaier, E., Dongarra, J., Simon, H., Meuer, M.: Top500 List (2024), https://www.top500.org
24. Tanash, M., Yang, H., Andresen, D., Hsu, W.: Ensemble Prediction of Job Resources to Improve System Performance for Slurm-Based HPC Systems. In: Practice and Experience in Advanced Research Computing, pp. 1–8 (2021)
25. Thonglek, K., Ichikawa, K., Takahashi, K., Iida, H., Nakasan, C.: Improving Resource Utilization in Data Centers using an LSTM-based Prediction Model. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER). pp. 1–8 (2019). https://doi.org/10.1109/CLUSTER.2019.8891022
26. Wilkes, J.: Yet more Google Compute Cluster Trace Data (2020), https://ai.googleblog.com/2020/04/yet-more-google-compute-cluster-trace.html
27. Zhang, Y., Hua, W., Zhou, Z., Suh, G.E., Delimitrou, C.: Sinan: ML-based and QoS-aware resource management for cloud microservices. In: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. p. 167–181. ASPLOS '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3445814.3446693