Journal of Big Data

RESEARCH Open Access

Design and implementation of I/O performance prediction scheme on HPC systems through large-scale log analysis

Sunggon Kim[1], Alex Sim[2], Kesheng Wu[2], Suren Byna[2,4] and Yongseok Son[3*]

*Correspondence:
sysganda@cau.ac.kr

[1] Seoul National University of Science and Technology, Seoul, Republic of Korea
[2] Lawrence Berkeley National Laboratory, Berkeley, USA
[3] Chung-Ang University, Seoul, Republic of Korea
[4] The Ohio State University, Columbus, USA

Abstract

Large-scale high performance computing (HPC) systems typically consist of many thousands of CPUs and storage units used by hundreds to thousands of users simultaneously. Applications from large numbers of users have diverse characteristics, such as varying computation, communication, memory, and I/O intensity. A good understanding of the performance characteristics of each user application is important for job scheduling and resource provisioning. Among these performance characteristics, I/O performance is becoming increasingly important as data sizes rapidly increase and large-scale applications, such as simulation and model training, are widely adopted. However, predicting I/O performance is difficult because I/O systems are shared among all users and involve many layers of software and hardware stack, including the application, network interconnect, operating system, file system, and storage devices. Furthermore, updates to these layers and changes in system management policy can significantly alter the I/O behavior of applications and the entire system. To improve the prediction of the I/O performance on HPC systems, we propose integrating information from several different system logs and developing a regression-based approach to predict the I/O performance. Our proposed scheme can dynamically select the most relevant features from the log entries using various feature selection algorithms and scoring functions, and can automatically select the regression algorithm with the best accuracy for the prediction task. The evaluation results show that our proposed scheme can predict the write performance with up to 90% prediction accuracy and the read performance with up to 99% prediction accuracy using the real logs from the Cori supercomputer system at NERSC.

Keywords: High performance computing, Distributed file system, Performance modeling

Introduction

Due to the vast data produced by traditional HPC applications and recent machine learning and big data applications, the I/O performance on HPC systems has a significant impact on the overall performance. Understanding the I/O performance and predicting it on HPC systems paves the path to optimizing applications. Accurately predicting the I/O performance of HPC jobs also allows the systems to better allocate

CPU, I/O, and networking resources. It enables computer centers to better provision resources when they purchase systems. In this paper, we focus on understanding the performance characteristics of the jobs running on these HPC systems, in particular, the I/O performance that is known to be difficult to predict.

The I/O performance on HPC systems primarily depends on I/O libraries and the underlying parallel file systems, such as Lustre [38] and IBM's SpectrumScale (previously known as GPFS) [36]. Parallel file systems allow parallel access to a large number of I/O servers. However, this introduces a new challenge in managing a large number of separate storage servers and providing consistency of the file system. An application might experience poor I/O performance in many ways, for example, heavy metadata accesses, unexpected data traffic from other applications, or network traffic passing through portions of the I/O data path. Thus, it is important to understand the application performance in large HPC environments and orchestrate them from the perspective of efficiency and priority.

Moreover, HPC systems are continuously monitored by various tools. For example, Slurm workload manager [51] records the progress of each job, Lustre Monitoring Tool (LMT) [46] logs file system activities, and darshan [42] monitors the I/O activities. These tools continuously collect the system status, such as CPU usage and the application I/O behavior, such as access pattern. However, these systems produce log files that are separate from each other. There is no easy way to combine them to provide coherent information to understand application behaviors.

Many studies have been conducted to understand the I/O performance using HPC system logs. Lockwood et al. [23] ran I/O-intensive scientific benchmarks and studied the logs to determine various factors that affect the performance. Using application I/O and scheduler logs, TOKIO [22] provides a comprehensive graphical display that helps users understand the I/O behavior. Various efforts have been made to predict the I/O performance. For instance, I/O performance models [3, 4] based on specific applications estimate I/O performance tuning parameters. In contrast to these efforts, in this paper, we analyze a large number of logs in a large HPC system (Cori at NERSC), finding key features that impact the I/O performance and predicting the I/O performance using the features.

In this article, we analyze the system and I/O logs from a large production system and proposed a performance prediction scheme to predict the I/O performance of HPC applications. Our paper aims to answer two research questions: (1) Is it possible to capture a significant correlation between the I/O performance and I/O characteristics of an application and system? (2) If so, is it possible to devise a model that covers all applications in a complex HPC system without a large overhead?. Our analysis results demonstrate that the I/O performance of applications is affected not only by the application behavior but also by the file system behavior. In addition, the correlation between features and the I/O performance changes according to the I/O intensity of applications, and it is important to dynamically identify relative features. Through the analysis, our paper presents a deep understanding of I/O systems in a large HPC

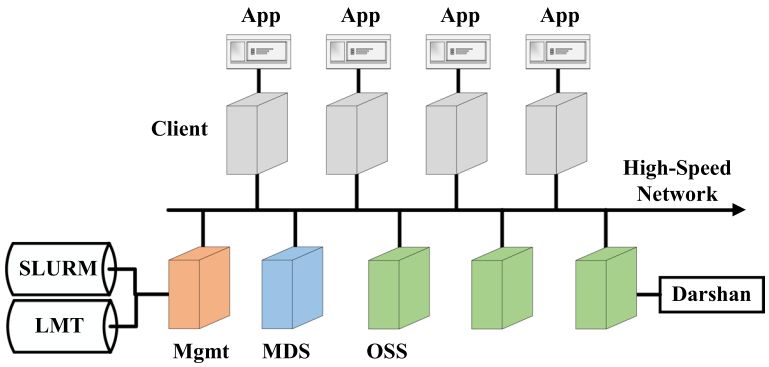


Fig. 1 Overall architecture of Cori supercomputer

environment and helps both system administrators and users understand the system and coordinate multiple I/O intensive applications to avoid performance degradation.

With the analysis findings, we proposed a prediction scheme which selects the important features from the system logs using various feature selection algorithms and uses combinations of various regression algorithms to predict various system metrics such as the I/O performance and runtime. Our evaluation showed that various feature selection and regression algorithms can predict the performance with an accuracy of up to 90% for write throughput, 99% for the read throughput, and 95% for runtime. In our previous work [15], we focused on predicting the I/O performance using various regression algorithms. This article extends our scheme by using various feature selection algorithms and scoring functions to improve prediction accuracy. Our analysis shows that using various feature selection and scoring functions can help to improve the prediction accuracy of medium and low I/O-intensity applications that have less correlation. In addition, we further evaluated our scheme and presented results for read performance and runtime prediction.

HPC Environment and tools

In Fig. 1, we present the architecture of Cori supercomputer at NERSC with compute nodes, network interconnects (e.g. Infiniband), and a storage system. Cori supercomputer at NERSC is based on Lustre parallel file system [38]. Lustre contains Metadata Servers (MDS) to manage file operations, such as file create, modify, and permission operations. MDS is responsible for maintaining a global and consistent view of the file system, it handles every metadata operation. Lustre's Object Storage Servers (OSS) are responsible for storing and retrieving user data. Cori is equipped with 248 OSSs and each OSS is connected to a single Object Storage Target (OST), which is a set of HDDs grouped by RAID. The computation and storage systems are connected via Infiniband [35], providing a fast storage area network between the computation server and storage server.

As the example HPC system is a complicated system designed for multiple users, it is important for the administrators to monitor the system status and analyze possible bottleneck while providing the service. To do this, there are many analysis tools that

continuously monitor various system statuses. Slurm workload manager [51] allocates compute nodes and processes on HPC systems. It stores a complete history of jobs, usernames, numbers of processes, and other information as logs using MySQL database. Darshan I/O characterization tool [6] is an I/O characterization tool that collectes I/O information of a job in memory and stores the information as a file in Lustre file system. It stores I/O characteristics such as the number of bytes written, a histogram of the request size, the time spent in I/O operations, and more. As Darshan stores the detailed I/O information for each application execution, it is a crucial tool for users to analyze the I/O behavior of applications and understand the bottlenecks that are orthogonal to other application behavior. While Darshan is focused on a single application, Lustre monitoring tool (LMT) [46] is designed to monitor file system activities. It collects information, such as MDS CPU usage, MDS operations per second, OSS write throughput, and more in five-second intervals and stores the information using MySQL database. For example, LMT is used to determine the file system performance after a major update to the system by analyzing any abnormal MDS or OSS usage. This information can provide insight into the abnormal behavior of the file system. These tools are widely deployed at production HPC sites, and previous analyses have shown that they have negligible runtime overhead [23, 41, 46]. Therefore, utilizing these logs can help understand I/O characteristics without introducing additional overhead to the system.

I/O Performance prediction method

Integrated database for system logs

Slurm, Darshan, and LMT logs have different fields and are stored in different locations. To use them, we first store them in an integrated SQLite database to collect, store, and access information during the prediction phase. We build the database using Darshan log entries, which create an I/O log for each program execution. We use the JobID as a unique key in the database. We process Darshan logs using the Darshan-parser [6] tool that parses a Darshan compressed format to a text file format. We then read the output file and extract the I/O related information such as the application execution start time (StartTime), runtime (RunTime), and total bytes written by the application (TotalBytesWritten). We use the JobID from the Darshan log to search the entry in the Slurm database to extract the information from a Slurm log. As Slurm stores the history of jobs using MySQL, we perform a simple select operation using the JobID and store the fetched information to the integrated database. In contrast to Darshan and Slurm, LMT continuously collects the information using time information. Thus, it is impossible to collect information using the JobID. To overcome this issue, we use the StartTime and RunTime acquired from the Darshan log to select the time interval related to the application. Using TOKIO tool [22], we extract the file system usage from the StartTime and end time (StartTime + RunTime) of the application. This process of building the database can be done daily, prior to application execution. Based on our analysis, processing all the log information during a 4-month period with a single server took 51 min and 27 s, and the size of the resulting database was 1.6 GB. Our proposed scheme can

effectively collect and access the I/O behavior of applications and the file system by continuously updating the integrated database with information collected from these three logs.

Selecting features

Since the prediction must be made at the start of the execution, only limited information is available For example, although LMT collects the file system information continuously in 5-second intervals, information collected before the application execution can be irrelevant. Similar to LMT, Darshan collects information at the termination of the application. While Darshan intercepts the I/O requests during the execution, it reports the information at the end of the runtime. Thus, the information from LMT and Darshan is only available after the application execution.

In contrast to LMT and Darshan, Slurm and Lustre require users to specify the resources at the start of execution. For example, when executing an application, a user must specify the resources, such as the number of processes and nodes. In addition to Slurm, users must specify the stripe size and count of the output directory on Lustre before the application execution using a Lustre command. Darshan logs from the previous executions of the same application can be used for analysis. This information is important because many HPC applications, especially I/O-intensive applications, have similar I/O behavior such as access patterns and request size. In addition to the information from the same environment, if the application history with the identical application name exists, we can use that information for the prediction. An execution history with an identical application name indicates that the application is executed multiple times with similar I/O characteristics. For example, out of the 3,543,538 studied application executions, only 2039 distinct application names exist, suggesting that only a small set of applications are executed multiple times in the example system. This finding allows our scheme to use the information, such as the access pattern and request size, from the previous executions to predict the performance of a target application,

In addition to the information availability, determining which information is important when predicting application I/O performance is also important [16, 30]. To dynamically adjust the features based on the characteristics of applications, we use various feature selection algorithms with different scoring functions at the start of the prediction to determine the optimal set of features rather than using a set of predefined features. We use the univariate feature selection functions for the feature selection function because they are widely used and exhibit good performance with relatively low overhead. Since our scheme predicts the target metric with all combinations and selects the combination with the highest prediction accuracy, it is important to have a low overhead. Out of many univariate feature selection functions, we utilize K-Best (KBest), false positive rate (Fpr), false discovery rate (Fdr), and family wise error (Fwe). For scoring functions of feature selection functions, we use univariate linear regression test (f_regression), mutual information (mutual_info), and chi-squared test (chi2). Our evaluation results indicate that the prediction accuracy depends on the feature selection algorithm and scoring function. ven if the identical feature selection algorithm was used, the selected features can be different based on the scoring functions. Thus, it is important to determine the

Table 1 List prediction models used in the paper

Algorithm	Description
Linear	Linear model based on distance
Polynomial	Polynomial model based on distance
K-nearest neighbors	Model based on distance and group of adjacent data sets
Gradient boosting random forest (GBDT)	Model based on decision trees and boosting for combining trees
Random forest (RF)	Model based on decision trees and bagging for combining trees
Multilayers precentron (MLP)	Feedforward artificial neural network
Convolutional neural network (CNN)	Artificial neural network based on convolution

Table 2 Hyperparameters of the model

Name	Description
Linear	None
Polynomial	degree=2
KNN	n_neighbors=2
RF	random_state_42, n_estimators=1000
GBDT	loss=ls, n_estimators=1000, max_depth=3, subsample=1.0, random_state=42
MLP	2 layers, num_nerons=256, activation=relu, loss=mse, optimizer=adam
CNN	num_filters=256, size_filter=3X3, activation=relu, inputshape=X_test.columns, loss=mse, optimizer=RMSprop(0.01)

optimal features using a feature selection algorithm and scoring function based on the correlation between the features and target metric.

Regression algorithms

With the information from the logs, we use various regression algorithms to predict the I/O performance of applications. We have selected several regression algorithms with distinct characteristics, including statistical, instance-based, tree-based, and neural network-based regression algorithms. Table 1 shows the list of regression algorithms and Table 2 shows the hyperparameters used for each algorithm. For statistical regressions, we used linear and polynomial regression methods. While statistical regressions are simple and fast, the prediction accuracy may suffer when the data records do not fit in a linear or polynomial equation. For instance-based machine learning regression, we use the K-nearest neighbors (KNN) [9]. For tree-based machine learning regressions, we use gradient boosting random forest (GBDT) [10, 11], and random forest (RF) [21] algorithms. Finally, we used the multilayered perceptron (MLP) [32] and convolutional neural network (CNN) [13, 18, 53] regression algorithms for neural network-based machine learning regressions. We used various regression algorithms as the accuracy of algorithms can be different based on the correlation between the features and the target metrics. Thus, it is important to evaluate each algorithm and dynamically select the algorithm when target applications, features, and target metrics change. Our evaluation result shows that the accuracy of the algorithm is dependent on the information. Thus, rather than statically choosing a single algorithm, our proposed scheme predicts I/O

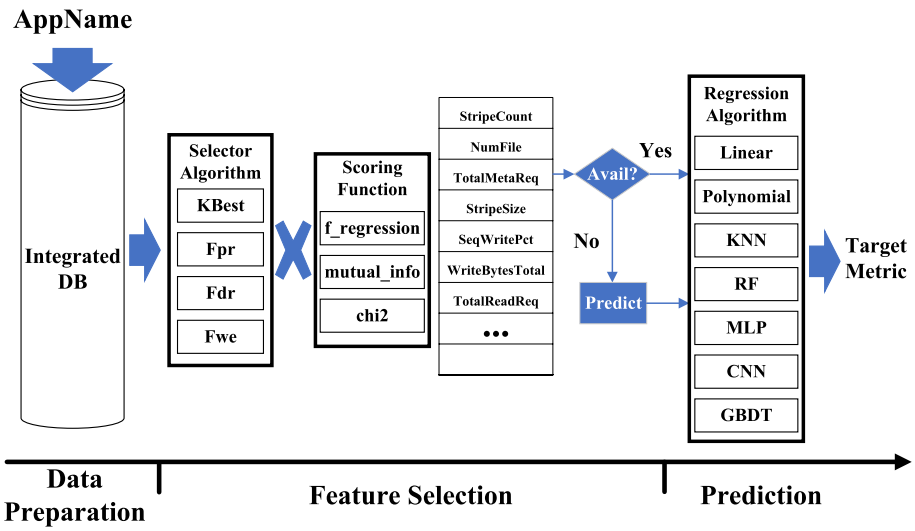


Fig. 2 Overall procedure of the proposed scheme

performance using various combinations of regression algorithms that exhibit different characteristics.

I/O Prediction algorithm

To predict the I/O performance using the existing system logs, we prepare the data, select the important features, and perform prediction using the data. Figure 2 shows the overall procedure of the proposed scheme. As shown in the figure, there are three main procedures in the proposed scheme: Data preparation, Feature Selection, and Prediction. We first extract the necessary data from the integrated database, perform feature selection using various selector algorithms and scoring functions, and finally perform prediction using the extracted data and selected important features that are correlated to the target metric.

PROCEDURE 1 Overall procedure of data preparation.

```
Target = [StartTime, ProgName, UserName, GroupID]
TargetFeature = Read/Write Throughput
DB = integrated database
ApplicationFeatures = [Features from Darshan logs]
FileSystemFeatures = [Features from LMT logs]
If Select Target.ProgName From DB != []
      //Previous records of the target application exists
      AvailableFeatures = [Features from Slurm and Darshan logs]
      UnavailableFeatures = [Features from LMT logs]
Else
      //No previous record of the target application
      AvailableFeatures = [Features from Slurm logs]
      UnavailableFeatures = [Features from LMT and Darshan logs]
```

PROCEDURE 2 Overall procedure offeature selection.

```
ScoringFuction = [f_regression, mutual_info, chi2]
SelectorAlgorithms = [KBest, Fpr, Fdr, Fwe]
Function SelectFeatures (TargetFeature, selector)
    If UnavailableFeatures.find(TargetFeature) == false
        HistoryRecords = SELECT * FROM DB
    Else
        HistoryRecords = SELECT AvailableFeatures FROM DB
    SelectedFeatures = selector(TargetFeature, HistoryRecords)
   /* Sort and Select the correlated features */
    X_train = SELECT (SelectedFeatures) FROM DB
    Y_train = SELECT (Target) FROM DB
    For FeatureValue in SelectedFeatures
        If FeatureValue != Available and FeatureValue.Feature == Application-
Features
            MissingFeatureValue = SELECT FeatureValue.Feature FROM DB
WHERE ProgName == Target.ProgName AND UserName == Target.UserName
AND GroupID == Target.GroupID ORDER BY StartTime
            If MissingFeatureValue == [] ///No records
                /* Relax the select condition until the record exist */
            FeatureValue = Value from the most recent execution
        If FeatureValue != Available and FeatureValue.Feature == FileSystem-
Features
            MissingFeatureValue = Prediction(X_train, Y_train, X_test)
            TargetRecords.replace(FeatureValue, MissingFeatureValue)
    X_test = TargetRecords
    retrun X_test
```

Data preperation

Procedure 1 shows the overall procedure of data preparation. At the start of the procedure, we initialize the values (e.g. TargetProgName) from the target application and feature. This is to select only relevant features and data. Then, we check whether the target application was previously executed by searching for the application name in the database. This is to determine which feature values are available. If the previous record of the application exists (i.e. a recurring application), we can utilize the information from the previous executions. Thus, we set the available feature as the features from Slurm and Darshan logs, and the unavailable features as the features from LMT log. However, if there is no previous record of the application, then there is no information from previous executions. Thus, we set the available features as the features from Slurm log, and the available features as the features from Darshan and LMT logs.

PROCEDURE 3 Overall procedure of prediction.

```
RegressionAlgorithms = [Linear, Polynomial, KNN, RF, MLP, CNN, GBDT]
Function Prediction (X_train, Y_train, X_test)
    Prediction, Coefficient = 0
    For Algorithm in RegressionAlgorithms
        Algorithm.fit(X_train, Y_train)
        TempPrediction = Algorithm.predict(X_test)
        TempCoefficient = Algorithm.predict(X_test).Coefficient
        If TempCoefficient > Coefficient
            Prediction = TempPrediction
    return Prediction
```

Feature selection

In the feature selection function, our proposed scheme selects the features that are correlated to the target feature (TargetFeature) that needs to be predicted. Procedure 2 shows the feature selection procedure of the proposed scheme. To select features, we must first determine the target feature. The target feature can be either the target metric such as read/write throughput or an unavailable feature such as write time and the number of write requests. In the case of predicting a target metric, we select all features that exist in our integrated database as possible features (HistoryRecords) that can have a correlation with the target feature. In the case of predicting an unavailable feature, we select only features that are available (AvailableFeatures) as possible features.

With the records selected from the features (HistoryRecords), we select features that have a high correlation with the target feature. To dynamically select the best feature selection algorithm with the best scoring function, our proposed scheme uses various feature selection algorithms (SelectorAlgorithms) and scoring functions (ScoringFunction) to predict the target feature and use the combination with the highest prediction accuracy. Feature selection algorithms determine the features that are highly correlated to the target feature. Scoring functions are used by the feature selection algorithm to evaluate the correlation between each possible feature and the target feature.

In terms of the equation and implementation of each scoring function, we used the implementation of the scoring function from scikit-learn [34]. For f_regreesion scoring function, we used the following equation from pearson's correlation [5].

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2(y_i - \bar{y})^2}} \tag{1}$$

For mutual_info, we used the following equation [17].

$$I(x, y) = H(x) - H(x|y) = H(y) - H(y|x) \tag{2}$$

For chi2, we used the following equation [12]

$$\tilde{\chi}^2 = \frac{1}{d}\sum_{k=1}^{n}\frac{(O_k - E_k)^2}{E_k} \tag{3}$$

Based on our experiments, features with the highest scoring value do not yield the highest performance. In addition, the final prediction accuracy of the target metric is highly affected by the feature selection algorithm, prediction algorithm, correlation, and application type. To overcome this issue, we perform the prediction using all combinations of features selected by feature selection algorithms and scoring functions and determine the prediction accuracy using the r-squared value. While mathematical modeling can reduce the search space of optimal combinations of various algorithms, we take a brute force grid search approach as the computational complexity is 2.08 s for predicting 153 different executions with various algorithms. Thus, our proposed scheme can determine the best combination of feature selection algorithm and scoring function by employing various feature selection algorithms and scoring functions, and choose the combination with the highest prediction accuracy. We return the selected features that have a strong correlation with the target feature.

Then, our scheme prepares two types of data: data for building a model (training phase) and data for the prediction (testing phase). In the training phase, records from the previous executions are used to build a model for the selected features and target feature. To do this, we select all records of the selected features (X_train) from the feature selection function and target feature (Y_train). In the testing phase, records of the target applications are used as input for the model to determine the value of the target feature. For unavailable features related to application behavior, we use the values from the most recent execution in the case of recurring applications to overcome this issue. We first select records that share the application name, user name, and group ID. If no records that share all information, we relax the select conditions to find records that share any of the three conditions. Finally, we select the record from the most recent execution to replace the unavailable feature value. We do this because the application behavior can change due to the application library and system update. By using the record from the latest execution, our proposed scheme can reflect the changed application behavior. For unavailable features related to file system behavior, we use the predicted file system values by performing the prediction using the available information to predict the file system behavior. Our scheme uses values from the current execution (from Slurm) and the most recent execution (from Darshan) to predict the unavailable feature. Thus, by using the features from the recent execution of an identical application and predicting unknown file system features, our scheme can select the important feature and prepare the data of selected features.

Prediction

With the prepared data, we make the prediction using a predefined set of regression algorithms (RegressionAlgorithms). The prediction procedure is identical in all six regression algorithms. We first build a model using the information on selected features (X_train) and the target feature (Y_train) from the database. With the model based on the history of all applications, we predict the target feature using the model and information from the selected feature (X_test) related to the target application. We repeat this

process for all combinations of feature selection algorithms, scoring fictions, and regression algorithms, and determine the coefficient of determination for all the combinations. Finally, by comparing the coefficient of determination, we select the combination with the highest accuracy and return the prediction

To automatically and dynamically select the features and prediction algorithm, we evaluate the algorithms and build a new model when there is a possible I/O pattern change. If there is a possible I/O pattern change due to a software/hardware update, a user can perform the proposed procedure to select new features and build a new model that reflects the new correlation between features and the target metric. When choosing a feature selection algorithm, scoring function, and regression algorithm, our proposed scheme chooses an algorithm that shows the highest prediction accuracy. If the change from software/hardware update is dominant and the existing system logs degrade the accuracy of the prediction, a user can adjust the training data to include more recent execution logs and build a model that reflects recent changes. Thus, our proposed scheme can dynamically and automatically select the prediction algorithms by performing predictions with various algorithms and selecting the algorithm with the highest accuracy. We utilized the R-squared metric to measure the accuracy of our model. We chose R-squared over other indicators, such as mean squared error (MSE), because I/O characteristics can be highly diverse even within a single HPC system, especially when a large number of I/O servers are used. Consequently, a few instances with unique I/O behavior could have a significant impact on the MSE. To minimize the impact of such cases, we utilized R-squared as the accuracy metric and adjusted our algorithms based on the resulting accuracy.

For the implementation, we used Python to build our proposed prediction algorithm. For the database operations, we used the *sqlite3 DB-API* module provided in Python. For the feature selection algorithm and scoring functions, we used the algorithm provided by Pandas dataframe and scikit-learn library [5, 27]. In the case of the regression algorithms, we used the *scikit-learn* library [34] with except for the CNN using *TensorFlow* [1, 8].

Analysis of I/O logs

This section presents our analysis results on the I/O behavior of HPC applications and the correlation between their I/O characteristics and performance. We used 3,543,538 application logs from October 2017 to January 2018, acquired from the Cori system. This number does not represent distinct applications; it represents the jobs dispatched to the system. For example, if an application is executed on 10 occasions, it creates 10 different logs. The average run time of the executions was 1015.6 s and the average number of nodes 1.2 with an average number of processes of 85.1. In terms of I/O, the average read and written bytes were 283.5 GBs, and the average I/O rate was 14GB/s.

Application I/O characteristics

While the I/O performance of an application is affected by many factors, the I/O behavior of the application itself is one of the critical factors in both small- and large-scale systems [19, 29, 39]. We conducted an analysis using the information acquired by Slurm and Darshan to determine the effect of the I/O behavior on the performance. Table 3

Table 3 Information from Slurm and Darshan logs. (S) - SLURM, (D) - Darshan

Name	Description
ProgName	Name of the program (S)
UserName	Name of the user (S)
GroupID	Group ID of the user (S)
NumProcs	Number of processes (S)
NumNodes	Number of computation nodes (S, D)
StripeCount	Number of OSS used by the write bursts of the application (D)
StripeSize	Amount of data written to an OSS per request (D)
NumFile	Number of files used by the application (D)
Seq[Read/Write]Pct	Percentage of sequential read/write requests (D)
Consec[Read/Write]Pct	Percentage of consecutive read/write requests (D)
[Read/Write]Less1M	Number of read/write requests less than 1M (D)
TotalMetaReq	Number of metadata requests (D)
Total[Read/Write]Req	Number of read/write requests (D)
[Read/Write]BytesTotal	Total bytes read/written by the application (D)
[Read/Write]Throughput	Read/Write throughput by the application (D)

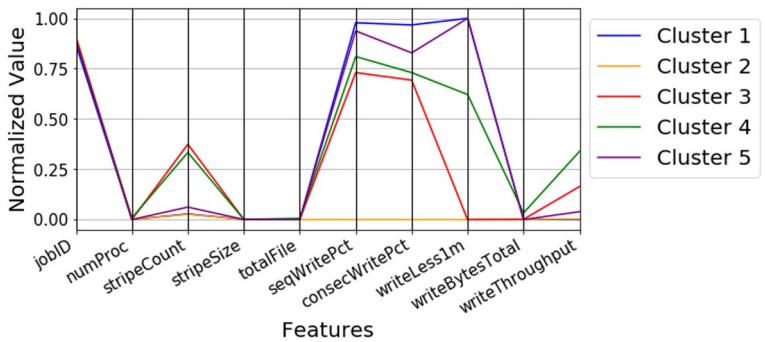


Fig. 3 Correlation between I/O behavior metrics and WriteThroughput

presents the information extracted from Slurm and Darshan logs. The top four features in the table are from Slurm logs, and the other features are from Darshan logs.

To find the correlation between the I/O behavior and write throughput, we analyzed the correlation using metrics presented in Table 3. By clustering the execution history, we aimed to create five clusters with distinct I/O behaviors and check the correlation between a set of I/O behavior and write throughput. We used the Gaussian mixture model [44] from the Scikit-learn python library [34] to build clusters and set the number of clusters as five. Also, to reduce the difference in the unit for different metrics, we used a minmax scaler and scaled the values between 0 and 1.

Figure 3 shows clusters that are clustered using the clustering algorithm and their write throughput. We used a hyperparameter of five for the number of clusters. Each line denotes the cluster formed by the clustering algorithm. There are 20.1%, 37.9%, 4.7%, 4.1% and 33.2% of executions in cluster 1, 2, 3, 4, and 5, respectively. As shown in the figure, the I/O behaviors of each cluster are relatively similar. This shows that the I/O behaviors of applications are not distinct enough to form clusters with distinct I/O behaviors. However, although the clusters exhibit similar I/O behaviors, Clusters 3 and 4 have higher write throughput than the other clusters (1, 2, and 5). This result suggests

Table 4 Information from LMT logs

Name	Description
mdsCPU[Mean/95]	Mean and 95th percentile of the CPU usage of a MDS server during application runtime (percentage)
mdsOPS[Mean/95]	Mean and 95th percentile of the operations per second of a MDS server during application runtime (ops)
ossCPU[Mean/95]	Mean and 95th percentile of the CPU usage of OSSs during application runtime (percentage)
ossWrite[Mean/95]	Mean and 95th percentile of the write throughput of OSSs during application runtime (GB/s)
ossRead[Mean/95]	Mean and 95th percentile of the read throughput of OSSs during application runtime (GB/s)

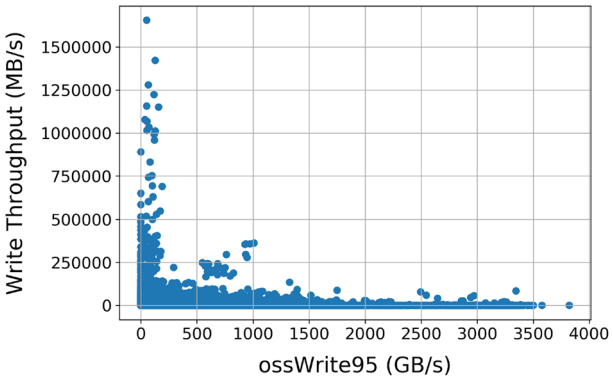


Fig. 4 Correlation between ossWrite95 and write throughput

that, in addition to considering application behavior, the impact of other applications and file system activity must be taken into account to accurately predict I/O performance in HPC environments [15].

File system activities

The analysis results from the previous subsection suggest that an analysis of the file system activities is needed to understand the I/O performance, in addition to the application behavior itself. For example, when the application is scheduled during a busy interval, such as a massive backup period, the application can exhibit very unusual performance [23]. Thus, we focused on the LMT, which collects the Lustre file system information in 5-second intervals. Table 4 shows the list of information extracted from the LMT logs. As LMT collects the file system status continuously, we first collect the start time and run time of the job from the Darshan log and selected information from LMT log during the application run time.

Figure 4 shows the correlation between the file system activity (ossWrite95) and write throughput. As shown in the figure there is no correlation between the OSS write activity and write throughput. However, we focus on the linear data points starting from 0 to 1000. These linear data points suggest that in certain types of applications, as the write activity of the file system increases, the write performance of the application increases as well. Thus, while not all the HPC applications follow an identical correlation with

write throughput, in certain applications, the general activity of the file system can have a strong correlation with write throughput.

I/O-intensive applications

To further investigate the analysis results presented above, we focused on the correlation between OSS write activity and write throughput. From Fig. 4, we observed that there can be a correlation between two metrics in a certain group of applications. By analyzing the write activity of the file system when an intensive I/O application is running, we found that I/O-intensive applications can have a dominant effect on the file system. When an application has a very low I/O activity, it is hard to investigate the I/O behavior of the application from the perspective of the file system due to its low impact on the file system. Thus, the correlation between file system activity and application write throughput is more significant when the write activity of a certain application dominates the file system activity [15].

To find the applications that induce heave I/O activity, we extracted the list of OSTs used by an application. Although the Lustre usage is not available for all logs, applications compiled with the Lustre information flag have Lustre usage information such as which OSTs are used by the application. We used 10,641 executions that were compiled with Lustre information flag and performed more than 10GB of I/O operations to select I/O-intensive applications. Note that while the Lustre information flag had to be included manually at the time, Cori now compiles all the applications with Lustre information flag automatically, and the Lustre information is available for all execution logs. With the list of used OSTs, we propose a new metric called WriteBytesPct and ReadBytesPct.

$$WriteBytesPct = \frac{WriteBytes_{UsedOSTs}}{WriteBytes_{AllOSTs}} \tag{4}$$

$$ReadBytesPct = \frac{ReadBytes_{UsedOSTs}}{ReadBytes_{AllOSTs}} \tag{5}$$

Equations 4 and 5 shows how we calculate WriteBytesPct and ReadBytesPct, respectively. As shown in the equation, WriteBytesPct represents the number of written bytes in OSTs used by a certain application over the number of written bytes in all OSTs in the file system. While OSTs can be shared by many applications, it is the closest estimation with available logs and an I/O-intensive application should have a high WriteBytesPct. Similar to WrtieBytesPct, an application execution entry with high ReadBytesPct represents that the application is an I/O-intensive application. Thus, we utilize this metric to select applications that have high I/O intensity in terms of the effect on the file system. To calculate these values, we first extracted the list of OSTs used by the application and the start and end time of the application from the darshan log. Then, we collected the written and read bytes of used OSTs and all OSTs during the application run time from the LMT logs. Finally, we calculated WriteBytesPct and ReadBytesPct using the collected write and read throughput.

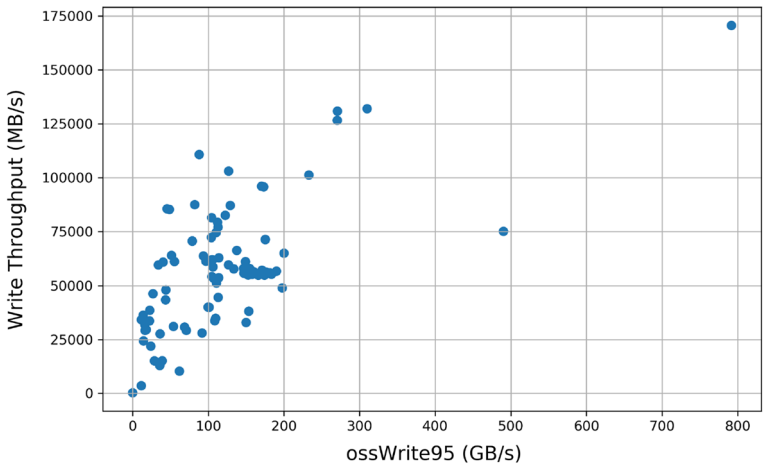


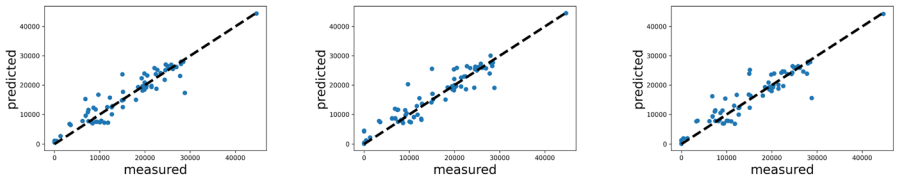
Fig. 5 Correlation between ossWrite95 and write throughput on high I/O intensity applications

With the new metric (WriteBytesPct), we select high I/O-intensity applications and find the correlation between the OSS write activity and write throughput. We selected high I/O-intensity applications with a WriteBytesPct of 80% or higher, which suggests that the application accounts for more than 80% of the file system activity, to find applications with a strong influence on the file system. Figure 5 presents the correlation between the OSS write activity and application write throughput for high I/O-intensity applications. As illustrated in the figure, the application performance strongly correlates with the file system activity because the file system performance is the application performance when an application dominates the I/O activity of the file system.

Based on our analysis, we made two significant observations on the I/O characteristics of HPC applications. First, no single I/O characteristic is dominant enough to observe a clear correlation with the write throughput, demonstrating that HPC systems have a very complex I/O performance model. Second, a strong correlation exists between OST usage and application write performance in I/O-intensive applications. These observations suggest that, while the correlation between the I/O characteristics and write throughput is complex, in specific scenarios where I/O performance is dominant in the overall application performance, the I/O characteristics can accurately predict the I/O performance of HPC applications.

Evaluation

For the evaluation, we used identical logs from the analysis section. We divided logs into training and test data using dates. We used logs from October 1 to November 30, 2017 for training data and logs from December 1, 2017, to January 31, 2018 for test data. The training data selected from October 1 to November 30 was 87% and the test data from December 1 to January 31 was 13%. By dividing the data based on date, we were able to demonstrate the ability to predict future performance using past logs. We validate our proposed scheme by predicting the performance of the test data, visualizing the predicted performance alongside the actual recorded performance, and presenting the R-squared value for each model. We used the 10,641 execution logs that have Lustre



(a) KBest selector with mutual information, linear, and random forest regression.

(b) KBest selector with chi-square, linear, and random forest regression.

(c) Fpr selector with mutual information, polynomial, and random forest regression.

Fig. 6 Prediction results of the write throughput (WriteThroughput) on high I/O-intensity applications

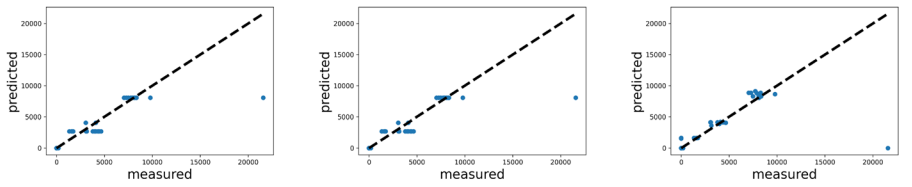
information and performed more than 10GB of I/O operations.[1] Out of I/O operations performed on-site, more than 95% of I/O operation was included in the selected logs. In terms of I/O intensity, the percentage of high, medium, and low I/O-intensity applications was 3.2%, 3.6%, and 93.2%, respectively. To standardize the values with different units, we used a standard scaler.

High I/O-intensity applications: We first predicted write and read throughput using the application with high I/O-intensity. These applications have WriteBytesPct and ReadBytesPct values of over 80%.

Figure 6 shows the graphs of prediction results of write throughput using different combinations of feature selection algorithms with scoring functions and regression algorithms (Intermediate evaluation results to predict the unknown feature value are presented in our previous paper [15]). Table 5 shows the list of features that were selected by the algorithms that yield the highest accuracy. As shown in the first row of the table, features such as ossWriteMean, writeBytesTotal, and TotalStatReq were selected, suggesting that global file system status and total written bytes are important features. This is because high-I/O intensity applications dominate the write activity of the global file system. Combinations listed in Fig. 6a, b, and c have coefficients of determination of 0.90, 0.88, and 0.88, respectively. In terms of comparison with existing schemes, the coefficients of determination using a single model of linear regression, K-best, and random forest are 0.29, 0.71, and 0.71, respectively. In terms of computational complexity, when using the combinations of algorithms listed in Fig. 6a, feature selection took 0.16 s, model training took 1.37 s, and prediction took 0.1 s. The entire process of model training and prediction, involving 153 executions with high I/O intensity, was completed in 2.08 s. Fig. 6a, b and c have data points close to the dotted black line, which represents 100% accuracy. These accuracy results suggest that the write throughput can be predicted accurately in high I/O-intensity applications, which is in line with our observations from the analysis. In conclusion, features from high I/O-intensity applications have a strong correlation with the write throughput, our proposed scheme can accurately predict the performance.

Figure 7 shows the prediction results of the read throughput (ReadThroughput) on high I/O-intensity applications. During the feature selection phase, features such as ossReadMean, TotalOpenReq, and totalMetaReq were selected, suggesting that global file system status and total written bytes are important features. Similar to write

[1] While we cannot reveal the name of applications due to the Lab's policy, many widely used representative HPC applications such as HACC I/O, GTC, and Vasp are included in the evaluation process.



(a) KBest selector with chi-square, polynomial, and k-nearest neighbor regression.

(b) KBest with chi-square, linear, and k-nearest neighbor regression.

(c) KBest with mutual information, polynomial, and k-nearest neighbors.

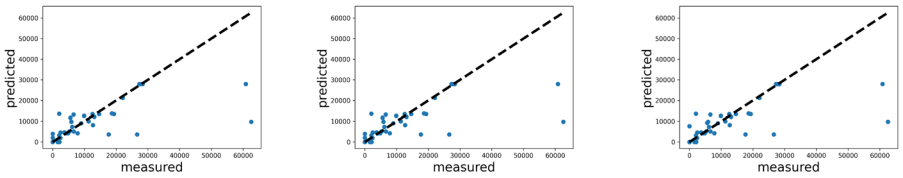
Fig. 7 Prediction results of the read throughput (ReadThroughput) on high I/O-intensity applications

Table 5 Selected features for different I/O-intensity and target metrics

Type	Target	Features
High I/O	Write	ossWriteMean, writeBytesTotal, TotalStatReq
High I/O	Read	ossReadMean, TotalOpenReq, totalMetaReq
Medium I/O	Write	ossWriteMean, writeLess1m, consecWritePct
Medium I/O	Read	NumOST, ReadBytesTotal, TotalMetaReq
Low I/O	Write	ConsecReadpct, totalIOReq, WriteBytesTotal
Low I/O	Read	ossRead95, numOST, seqReadPct
Quantum chemistry	Write	WriteBytesTotal, ReadBytesTotal, TotalMetaReq
Biology 1	Write	WriteLess1m, TotalIOReq, Numfile
Biology 2	Write	NumFile, TotalWriteReq, SeqWritePct

throughput prediction, high-I/O intensity applications dominate the read activity of global file systems. In contrast, metadata operations such as TotalOpenReq had more impact on read performance compared to the write performance. To reflect these I/O characteristics, metadata related features were selected during the feature selection phase. The combinations of the feature selection algorithm, scoring functions and regression algorithms listed in Fig. 7a, b, and c have the coefficient of determination of 0.81, 0.81, and 0.57, respectively. Compared with the prediction results of write throughput, prediction results of read throughput are less accurate because the read throughput variance of high I/O-intensive applications is higher. While the prediction results of an application execution at 20,000 on the x-axis are inaccurate, most of the prediction results represented as blue dots are placed near the black line, suggesting that the predictions are generally accurate. For example, the combination listed in Fig. 7c presents accurate prediction results except for the one prediction near 20,000, which suggests that, while the general prediction accuracy is good compared with the other combinations listed in Fig. 7a and b, the single inaccurate prediction result lowers the coefficient of determination. In terms of the feature selection and regression algorithm, algorithms that use distance metrics showed good performance prediction results. The prediction results can be grouped as presented in Fig. 7 near 50 and 100 on the x-axis. Thus, these prediction results indicate that our scheme can accurately predict both write and read throughput when applications share similar I/O characteristics.

Medium I/O-intensity applications: We evaluated our scheme by predicting the write and read throughput of applications with less I/O activity. To do this, we used applications that have WriteBytesPct and ReadBytesPct values of between 30% and 80%, which

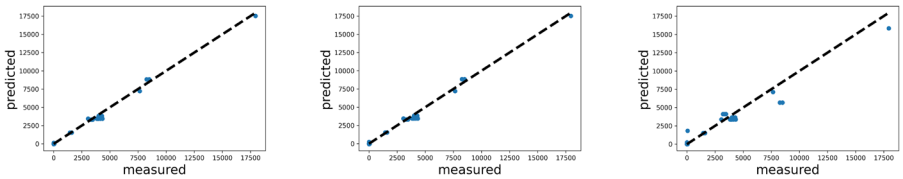


(a) Fdr with chi-square, random forest, and k-nearest neighbor regression

(b) Fpr with chi-square, random forest, and k-nearest neighbor regression

(c) KBest selector with mutual info, random forest, and k-nearest neighbor regression

Fig. 8 Prediction results of the write throughput (WriteThroughput) on medium I/O-intensity applications



(a) KBest selector with mutual info and gradient boost regression

(b) KBest selector with chi-square and gradient boost regression

(c) KBest selector with chi-square and random forest regression

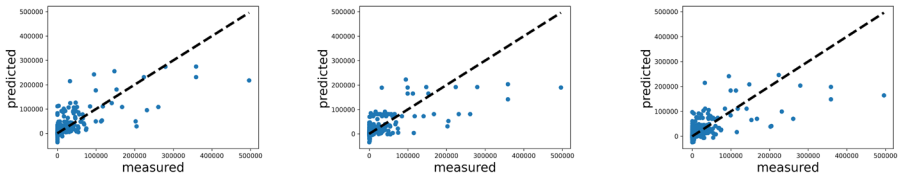
Fig. 9 Prediction results of the read throughput (ReadThroughput) on medium I/O-intensity applications

suggests that OSTs used by applications were generating 30% to 80% of the file system activity.

Figure 8 shows the write throughput prediction graph of the top three combinations of algorithms with the highest coefficient of determination. The coefficient of determination for Fig. 8a, b and c is 0.56, 0.56, and 0.55, respectively. The overall accuracy decreased compared with the evaluation with high I/O-intensity applications because the features used to predict write throughput had less correlation than that of high I/O-intensity applications. While high I/O-intensity applications have similar I/O behavior and effect on the file system, medium I/O-intensity applications have diverse characteristics. Thus, although the regression algorithms are identical, the overall accuracy decreased significantly.

As shown in the figures, the prediction results of two application executions at 60,000 on the x-axis decreased the overall accuracy. This is because the correlation between features and target metric (WriteThroughput) is different in the two executions than the correlation in other executions used to train the model. Thus, because the coefficient of determination is calculated by computing the distance between the predicted and measured value, while most of the predictions in Fig. 8a, b, and c are accurate, the two most inaccurate data points lead to low prediction accuracy. In conclusion, the correlation between I/O characteristics and performance is lower when the I/O performance is not the crucial factor in the application performance. However, the results demonstrated that our scheme can accurately predict I/O performance by analyzing other factors.

Figure 9 shows the prediction results of read throughput (ReadThroughput) on medium I/O-intensity applications. The combinations listed in Fig. 9a, b, and c have the coefficient of determination of 0.99, 0.99, and 0.96. Similar to read throughput prediction results for high I/O-intensity applications, the read throughput prediction results are more accurate than write throughput prediction results due to the low variance in read throughput. In addition, as shown in Fig. 9a and b, performance trends, with the



(a) KBest selector with chi-square2 and gradient boost regression

(b) KBest selector with mutual information and gradient boost regression

(c) Fdr selector with f_regression and gradient boost Regression

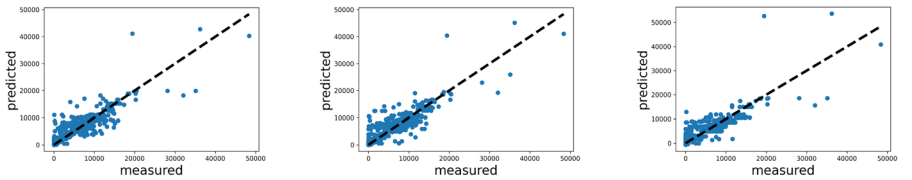
Fig. 10 Prediction results of the write throughput (WriteThroughput) on low I/O-intensity applications

groups of data points are similar (i.e., near 50 and 75 on the x-axis). This results in nearly perfect prediction accuracy. Another important factor is that features selected by the feature selection algorithms were application-specific features, such as NumOST, ReadBytesTotal, and TotalMetaReq. Thus, only a single regression algorithm was used for each combination to predict read throughput. The selected features and the accurate prediction results suggest that the selected applications share similar I/O characteristics and the performance is relatively independent of the file system status compared with other applications. Thus, these results suggest that even if the I/O-intensity applications are selected to perform the prediction if we can select the relevant features and if the correlation between the features and target metric is stable, our proposed scheme can accurately predict the I/O performance.

Low I/O-intensity applications: To evaluate our scheme in low I/O intensive applications, we used applications that have WriteBytesPct and ReadBytesPct of below 30% and predicted write and read throughput.

Figure 10 lists the prediction results of write throughput on low I/O-intensity applications. Combinations listed in Fig. 10a, b, and c have the coefficient of determination of 0.58, 0.54, and 0.49, respectively. The prediction accuracy of low I/O-intensity applications is similar to that of medium I/O-intensity applications based on the coefficient of determination. However, in the case of low I/O-intensity applications, most predictions near 0 on the x-axis are randomly scattered around the black line, and predictions beyond 100,000 are far from the black line, suggesting that the predictions are either too high or too low. Because the predictions do not show any distinct pattern, they indicate that the regression model failed to create a single model that can predict a diverse pattern. In addition, the selected features from various feature selection algorithms and scoring functions include ConsecReadPct and SeqReadPct, which are factors that do not strongly correlate to write throughput, as presented in Sect. 4. Thus, these results suggest that while the coefficient of determination is similar to that of medium I/O-intensity applications because applications have diverse I/O behavior, it can be difficult to create a model that includes diverse I/O behavior, resulting in poor prediction accuracy.

Figure 11 shows the prediction results of read throughput on low I/O-intensity applications. Combinations listed in Fig. 11a, b and c have the coefficient of determination of 0.91, 0.90, and 0.88, respectively. Compared with the prediction results of write throughput on low I/O-intensity applications, results of read throughput have very high accuracy because applications with low I/O-intensity share similar I/O characteristics in terms of read. However, similar to write throughput prediction, the predictions are

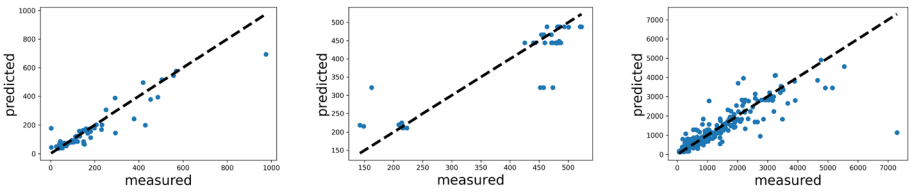


(a) KBest selector with mutual information and random forest regression

(b) KBest selector with f_regression and random forest regression

(c) KBest selector with mutual information and gradient boost regression

Fig. 11 Prediction results of the read throughput (ReadThroughput) on low I/O-intensity applications



(a) Quantum chemistry

(b) Biology1

(c) Biology2

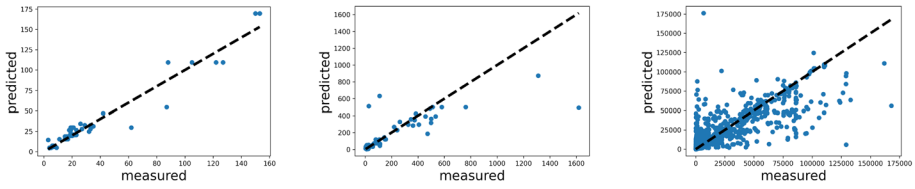
Fig. 12 Prediction results of write throughput (WriteThroughput) on targeted applications

inaccurate for applications with high read performance, suggesting that the I/O behavior of these applications is different from other applications.

I/O performance for individual applications: By evaluating high and medium I/O-intensity applications, we discovered that the prediction accuracy is related to the correlation between the application I/O behavior and the I/O performance. While categorizing applications based on I/O activity enabled us to explore how the I/O behavior of applications in each category is correlated with the performance, it also revealed the difficulty of predicting application performance accurately with different I/O behavior from other applications in the same category. We evaluated the applications by selecting target applications and predicting their performance using system application logs to validate that our proposed scheme works if we categorize the applications by application name rather than by relative I/O activity to the file system.

For target applications, we selected a quantum chemistry application and two biology applications that ran continuously in the system and have medium to high I/O-intensity. These are very well-known applications, and widely used in HPC systems. For all three applications, Kbest selector with f_regression scoring function had the highest accuracy. For the quantum chemistry application, WriteBytesTotal, ReadBytesTotal, and TotalMetaReq were selected as the most correlated features with the write throughput. For the biology applications (e.g., Biology1 and Biology2), we used WriteLess1m, TotalIOReq, and NumFile, and NumFile, TotalWriteReq, and SeqWritePct, respectively. In the case of biology applications, all correlated features were available. Thus, we did not perform additional predictions to predict the unavailable information.

Figure 12 shows the prediction results of write throughput on targeted applications. Using the RF algorithm in all phases, our proposed algorithm predicted the write throughput for the Quantum chemistry, Biology1, and Biology2 applications with 0.88. 0.73, and 0.80 coefficients of determination, respectively. Similar to high I/O-intensity applications, our prediction algorithm predicted the write throughput with



(a) High I/O-intensity Applications (b) Medium I/O-intensity Applications (c) Low I/O-intensity Applications

Fig. 13 Prediction results of runtime

high accuracy. This is because, in contrast to medium I/O-intensity applications, the I/O behavior of applications remains relatively stable between each execution which enables accurate prediction based solely on the characteristics of the target applications. Thus, the evaluation results indicate that, while the prediction results using the logs of the entire application can be inaccurate due to diverse characteristics, the prediction accuracy can be improved if we can capture a set of applications that share I/O characteristics.

Runtime prediction: In addition to write and read throughput, we performed prediction of runtime using identical data. Figure 13 shows the prediction results of runtime using high, medium, and low I/O-intensity applications. As shown in the figure, the prediction accuracy of the high, medium and low I/O-intensity applications were 0.95, 0.61, and 0.76, respectively. KBest selector with mutual information and gradient boost regression was used in both high and medium I/O-intensity applications and fpr selector with f_regression and random forest regression was used in low I/O-intensity applications. In the case of high I/O-intensity applications, features such as total written bytes, request size, and the number of metadata operations were used, suggesting that many high I/O-intensity applications are write heavy and runtime is highly correlated with the amount of written data. In the case of medium I/O-intensity applications, features such as total metadata, write, and read requests were used, suggesting that applications have a more balanced I/O pattern compared with the high I/O-intensity applications. As the applications have a more diverse pattern, the correlation between features and runtime was smaller, resulting in lower prediction accuracy. In the case of low I/O-intensity applications, while features such as sequential write percentage, average MDS OPS, and total written bytes were used, the overall correlation between the features and runtime was much lower. While the general prediction accuracy was higher as many executions follow the model, many data points are placed far from the prediction line compared with Fig. 13a and b. Thus, these runtime prediction results show that our scheme can be extended to other target metric predictions.

Experiment analysis: Compared with the general prediction accuracy of write throughput, that of read throughput is generally higher in most cases. When analyzing the evaluation results, we discovered that the number of unique applications used in both modeling and prediction can impact prediction accuracy. For example, when predicting the performance of write and read throughput of high I/O-intensity applications, while there were 39 unique applications when predicting the write throughput, there were only 29 unique applications when predicting the read throughput. Since many read

applications were benchmarks that have more stable performance, the prediction accuracy was higher when predicting read throughput than write throughput.

Similar trends were observed when predicting the write throughput of individual applications. As we manually selected the applications and build a model on an application, the prediction accuracy was very high. This is because executions of an individual application share similar I/O trends. Similar to prediction on high I/O-intensity applications, our evaluation results show that the prediction accuracy highly depends on selecting executions that share a similar correlation between features and target metric.

In the perspective of the regression algorithm, ensemble regressions such as random forest and gradient regression exhibited the most accurate prediction results compared with other regression algorithms in all evaluation scenarios. We believe that statistical approaches such as linear and polynomial regression performs worse than ensemble regressions as they create a single model for all the data points. Thus, if there are multiple groups of data points that have a different correlation between the features and the target metric, these algorithms cannot create a model that reflects a different correlation. In the case of CNN regression, we believe that our data is not suitable for CNN as it does not benefit from new features created by the convolutional layer. As our data does not have a strong correlation between each other in contrast to image or audio data, the convolutional layer does not add new information. In the case of MLP regression, we believe that there can overfit to low or high performance executions. While this can be addressed by increasing the number of hidden layers, as our prediction scheme aims to perform real-time prediction, we believed that ensemble regressions with high accuracy and low prediction overhead can outperform MLP regression with an acceptable number of hidden layers. In addition, when there is no strong correlation between features and the target metric (e.g., low I/O-intensity applications) the prediction accuracy cannot be improved even with a high number of hidden layers. In contrast, because ensemble regression divides the inputs into small subsets and performs the decision tree algorithm for each subset, it can build a model reflecting both the low-performing and high-performing executions. Thus, according to our evaluation results, we believe that using random forest and gradient regression with linear or polynomial regression which have low computational and conceptual overhead can be a good starting point when adapting our proposed scheme to different HPC environments.

In terms of feature selection algorithms, KBest feature selection algorithm was preferred in most cases. This is because, KBest selected more features compared with Fpr, Fdr, and Few feature selection algorithms which select features based on the score. Based on the prediction result, we believe that the additional features selected by KBest were essential to predict the outliers. Thus, according to our evaluation results, we believe that KBest feature selection algorithm can be a good starting point to select important features from I/O related system logs.

Related work

Understanding I/O characteristics of applications: There have been many studies that explored the characteristics of HPC applications. Lang et al. [20] analyzed hardware and software libraries and their effect on application performance. By performing experiments with different configurations, they found the correlation between configurations

and scalability from the perspective of hardware and software components in the HPC system. Teng et al. [45] proposed a method for integrating I/O logs from the HPC system and performed an analysis on the system. Their analysis revealed that comprehensive log analysis is needed to determine the root cause of performance degradation. Lockwood et al. [23] presented a year-long analysis of the HPC file system. By running an identical benchmark, they analyzed the performance while controlling the I/O behavior of the application. Their analysis showed that factors such as system upgrades and continuous execution of I/O-intensive applications could affect the application performance. Kim et al. [14] performed an analysis of the distributed file system used in the HPC environment. They discovered that most applications do not use the parallelism provided by the distributed file system and proposed an autonomous algorithm to improve I/O performance.

Our paper is in line with these studies in terms of analyzing the I/O performance in the HPC environment. These studies are focused on characterizing I/O characteristics using logs [45], and studying the impact of software/hardware configurations on the HPC application performance [14, 20, 23]. Different from previous studies we focus on analyzing I/O characteristics by integrating multiple logs and predicting I/O performance metrics using the system logs. To improve prediction accuracy, we proposed a scheme that automatically evaluates various scoring functions, feature selection algorithms, and regression algorithms, and dynamically chooses the best combination with the highest prediction accuracy. Thus, our proposed scheme predicts I/O performance in a large HPC system by utilizing multiple system logs and dynamically select the optimal combination of algorithms.

Prediction using system characteristics: There have been several studies on predicting the performance of the application to minimize interference and improve the user experience. Ernest [43] is a framework to predict the performance of large scale analytical applications by building a model based on resources. By studying hardware and application, the model predicts the application performance designed for a distributed system. Lux et al. [24] proposed a model by analyzing a benchmark with different configurations. The analysis showed that their multivariate model can accurately predict the I/O performance of the HPC system. Schmidt et al. [37] proposed a prediction scheme using an artificial neural network in an HPC system to predict the disk I/O time of each request with different size. Other works [25, 26, 49, 50] have also tried to predict various performance metrics for large clusters. Xie et al. [47, 48] proposed the regression based I/O performance prediction scheme considering I/O performance variance. They also proposed prediction based I/O middleware configuration to improve the overall I/O performance. Agarwal et al. [2] proposed a prediction-based storage system configuration adjustment scheme for recurring applications that utilize the execution history of the previous execution of the same application. Meswani et al. [28] proposed a prediction scheme that accumulates I/O calls using a base system and uses the I/O calls to build a machine learning based prediction model. In addition, there have been many researches [7, 40, 52] that proposed prediction algorithms using system characteristics in the area of aero-engine, traffic volume and cloud computing.

Table 6 shows the list of related works and the comparison with the proposed scheme. Similar to these studies, our scheme aims to predict the performance of the application

Table 6 List and comparison of related works and the proposed scheme

Paper	Target	Unit	New app prediction	Add'l Data	App
[2]	I/O configuration	App	X	X	Benchmark
[37]	I/O time	I/O req	X	O	Benchmark
[28]	I/O performance	I/O req	X	O	Benchmark
[43]	Performance	App	X	O	ML apps
Proposed	I/O performance	App	O	X	All apps

using characteristics of the system and building a machine learning model. Different from previous studies, our proposed scheme utilizes the existing system logs generated from real applications to dynamically select relevant features and build a model. This allows our proposed scheme to predict I/O performance of all applications in the system without prior benchmark results with different configurations and does not require additional information. In addition, our proposed scheme can reflect the characteristics of HPC applications that are executed in an identical environment and adapt to various system activities. By utilizing application specific I/O characteristics, the proposed scheme can accurately predict the performance of I/O intensive applications with sustainable performance. However, the approach proposed in the previous paper [48, 49] is needed to accurately predict the performance with high I/O variability.

Discussion and future works

In terms of limitation of our proposed scheme, the proposed scheme is bound by utilization of the specific logs. This is because we targeted Cori HPC system with Lustre as the file system and darshan, LMT, and SLURM logs as system logs. However, the proposed scheme can be extended to other systems that use different file systems and logging tools. Since the proposed scheme predicts the performance using information from the system logs, other systems can adapt the scheme by processing information from the logs and using the information as new features. For example, PBS pro [31] is another widely used scheduler that collects system logs. As many other system tools collect different aspects of HPC data such as storage resources and network resources, it is important to integrate the information from other sources and utilize the information for the prediction. In our proposed scheme, we dynamically select the most correlated feature using various feature selection algorithms and scoring functions. Thus, even if new system logs are introduced to the system, by parsing the log data and inserting the information into the database, our proposed scheme can dynamically integrate new information into the prediction.

For future works, we will continue to explore other system logs and features that impact the performance of HPC applications. The proposed scheme only shows high accuracy as we are utilizing I/O performance logs. For example, network-intensive or compute-intensive applications need a different approach to accurately predict the performance. With more comprehensive analysis, our study can be extended to power consumption and QoS management in large-scale systems. To do this, we will investigate other system logs and propose a scheme to better categorize the executions with

different characteristics. In addition, in terms of I/O-intensive applications, we will investigate I/O imbalance problems. In HPC systems, capacity and performance imbalance can degrade the overall I/O performance [23, 33]. In future work, we plan to utilize the performance and runtime prediction result to improve the balance between multiple storage servers in the system.

Conclusion

In this paper, we proposed an I/O performance prediction scheme for HPC environments. To do this, we first collected multiple system logs into a single integrated database and used various combinations of regression algorithms to predict the I/O performance using the database. Our analysis of logs from the example HPC system indicated that no single I/O feature can be used to accurately predict the I/O performance of applications. By selecting the most relevant features and the best regression algorithm, our scheme can predict the I/O performance with up to 99% accuracy. We believe that the presented analysis results can help users predict the I/O performance of their applications and schedule their applications efficiently, avoiding I/O interference from other applications. Also, our scheme can help system administers understand I/O behaviors in a large HPC system and efficiently allocate and manage resources in a complex system. In terms of limitations, our research only focuses on I/O performance, and including more comprehensive resources such as computation and network can improve the accuracy of the prediction. In addition, the proposed prediction scheme can be extended to manage QoS of the system and improve the overall efficiency and power consumption of complex systems.

Acknowledgements
Not applicable.

Author's information
Sunggon Kim is an assistant professor in the department of Computer Science at Seoul National University of Science and Technology (SeoulTech) since 2022. He received his B.S. degree in Computer Science from University of Wisconsin-Madison, Madison, USA, and Ph.D. degree from Seoul National University in 2015 and 2021, respectively. He was an intern at Lawrence Berkeley National Laboratory, California, USA, in 2018, 2019 and 2020. His research interests are file systems, cloud computing, distributed systems, and operating systems.
Alex Sim is currently a Senior Computing Engineer at Lawrence Berkeley National Laboratory. He authored and co-authored over 300 technical publications, and released a few software packages under open source license. His current research and development activities include data modeling, data analysis methods, learning models, distributed resource management, and high performance data systems. He is a senior member of IEEE.
Kesheng Wu is a Senior Scientist at Lawrence Berkeley National Laboratory. He works extensively on data management, data analysis, and scientific computing topics. He is the developer of a number of widely used algorithms including FastBit bitmap indexes for querying large scientific datasets, Thick-Restart Lanczos (TRLan) algorithm for solving eigenvalue problems, and IDEALEM for statistical data reduction and feature extraction.
Suren Byna received his Ph.D. degree in 2006 in Computer Science from Illinois Institute of Technology, Chicago. He was a Senior Scientist in the Scientific Data Management (SDM) Group in CRD at Lawrence Berkeley National Laboratory (LBNL). Currently, he is a professor in the Department of Computer Science and Engineering at The Ohio State University. He works on optimizing parallel I/O and on developing systems for managing scientific data. He is the PI of the ECP funded ExaIO and ExaHDF5 projects, and various projects on managing scientific data. Yongseok Son received his B.S. degree from Ajou University in 2010, and his M.S. and Ph.D. degrees from Seoul National University in 2012 and 2018, respectively. He was a postdoctoral research associate at University of Illinois at Urbana-Champaign. Currently, he is an assistant professor in Department of Computer Science and Engineering, Chung-Ang University. His research interests are operating, distributed, and database systems.

Author contributions
SK: conceptualization, methodology, software, writing. AS: conceptualization, discussion, supervision. KW: conceptualization, discussion, supervision. SB: conceptualization, discussion, supervision. YS: conceptualization, supervision, writing, corresponding author. All authors read and approved the final manuscript.

Funding
This work was supported by the National Research Foundation of Korea grant (No.2021R1C1C1010861, 2022R1A4A5034130, P0012724, RS-2022-00166541). Also, this work was supported by the Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, and also used resources of the National Energy Research Scientific Computing Center (NERSC) (Corresponding Author: Yongseok Son).

Availibility of data and materials
The datasets generated and/or analysed during the current study are not publicly available due to the Lab's policy but are available from the corresponding author on reasonable request.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 29 August 2022 Accepted: 20 April 2023
Published online: 17 May 2023

References

1. Abadi M et al. Tensorflow: a system for large-scale machine learning. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16); 2016. p. 265–83.
2. Agarwal M, Singhvi D, Malakar P, Byna S. Active learning-based automatic tuning and prediction of parallel i/o performance. In: 2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW), IEEE; 2019. p. 20–9.
3. Behzad B et al. Improving parallel I/O autotuning with performance modeling. In: Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing, Association for Computing Machinery, New York, NY, USA; 2014. p. 253–56. https://doi.org/10.1145/2600212.2600708.
4. Behzad B et al. Pattern-driven parallel I/O tuning. In: Proceedings of the 10th Parallel Data Storage Workshop, ACM, New York, NY, USA; 2015. p. 43–48. https://doi.org/10.1145/2834976.2834977.
5. Benesty J, et al. Pearson correlation coefficient. In: Davis GM, editor., et al., Noise reduction in speech processing. Heidelberg: Springer; 2009. p. 1–4.
6. Carns P et al. 24/7 characterization of petascale I/O workloads. In: 2009 IEEE International Conference on Cluster Computing and Workshops, IEEE; 2009. p. 1–10.
7. Chen Q, Sheng H, Zhang T. A novel direct performance adaptive control of aero-engine using subspace-based improved model predictive control. Aeros Sci Technol. 2022;128: 107760.
8. Chollet F. et al. Keras; 2015. https://keras.io.
9. Dudani SA. The distance-weighted k-nearest-neighbor rule. IEEE Trans Syst Man Cybern; 1976. p. 325–7.
10. Friedman JH. Greedy function approximation: a gradient boosting machine. Ann Stat. 2001;29:1189–232.
11. Friedman JH. Stochastic gradient boosting. Comput stat Data Anal. 2002;38:367–78.
12. Greenwood PE, Nikulin MS. A guide to chi-squared testing, vol. 280. Hoboken: John Wiley & Sons; 1996.
13. Khoshboresh-Masouleh M, Shah-Hosseini R. Quantum deep learning in remote sensing: achievements and challenges. Photonics Quantum. 2021;2021(11844):42–5.
14. Kim S et al. Dca-io: A dynamic i/o control scheme for parallel and distributed file systems. In: 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID); 2019. p. 351–60.
15. Kim S et al. Towards hpc i/o performance prediction through large-scale log analysis. In: Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing; 2020. p. 77–88.
16. Kira K, Rendell LA. A practical approach to feature selection. In: Sleeman D, Edwards P, editors. Machine learning proceedings. Amsterdam: Elsevier; 1992. p. 249–56.
17. Kraskov A, Stögbauer H, Grassberger P. Estimating mutual information. Phys Rev E. 2004;69: 066138.
18. Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. Adv Neural Inf Process Syst. 2012;60:1097–105.
19. Kroeger TM, Long DD. The case for efficient file access pattern modeling. In: Proceedings of the Seventh Workshop on Hot Topics in Operating Systems, IEEE; 1999. p. 14–9.
20. Lang S. et al. I/o performance challenges at leadership scale. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, IEEE; 2009. p. 1–12.
21. Liaw A, et al. Classification and regression by randomForest. R News. 2002;2:18–22.
22. Lockwood GK. et al. TOKIO on ClusterStor: connecting standard tools to enable holistic i/o performance analysis; 2018.
23. Lockwood GK, et al. A year in the life of a parallel file system. In: SC18: International Conference for High Performance Computing. Storage and Analysis. IEEE: Networking; 2018. p. 931–43.
24. Lux TC. et al. Predictive modeling of i/o characteristics in high performance computing systems. In: Proceedings of the High Performance Computing Symposium, Society for Computer Simulation International; 2018. p. 8.

25. Matsunaga A, et al. On the use of machine learning to predict the time and resources consumed by applications. In: 2010 10th IEEE/ACM International Conference on Cluster. IEEE: Cloud and Grid Computing; 2010. p. 495–504.
26. McKenna R et al. Machine learning predictions of runtime and IO traffic on high-end clusters. In: 2016 IEEE International Conference on Cluster Computing (CLUSTER), IEEE; 2016. p. 255–8.
27. McKinney W. Data structures for statistical computing in python. In: van der Walt S, Millman J, editors. Proceedings of the 9th Python in Science Conference; 2010. p. 51–6.
28. Meswani MR, Laurenzano MA, Carrington L, Snavely A. Modeling and predicting disk I/O time of HPC applications. In: 2010 DoD High Performance Computing Modernization Program Users Group Conference, IEEE; 2010. p. 478–86.
29. Min Co. SFS: random write considered harmful in solid state drives. In: FAST. 2012. p. 1–16.
30. Navot A et al. Is feature selection still necessary?. In: International Statistical and Optimization Perspectives Workshop" Subspace, Latent Structure and Feature Selection". Springer; 2005. p. 127–38.
31. Nitzberg B, et al. PBS pro: Grid computing and scheduling attributes. In: Nabrzyski J, Schopf JM, Węglarz J, editors., et al., Grid resource management. Boston: Springer; 2004. p. 183–90.
32. Pal SK, Mitra S. Multilayer perceptron, fuzzy sets, and classification. IEEE Trans Neural Netw. 1992;3:683–97.
33. Patel T. et al. Revisiting I/O behavior in large-scale storage systems: the expected and the unexpected. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis; 2019. p. 1–13.
34. Pedregosa F, et al. Scikit-learn: machine learning in python. J Mach Learn Res. 2011;12:2825–30.
35. Pfister GF. An introduction to the infiniband architecture. In: High Performance Mass Storage and Parallel I/O. 2001; ch. 42, p. 617–32.
36. Quintero D. et al. IBM Spectrum Scale (formerly GPFS). IBM Redbooks. 2017.
37. Schmidt JF, Kunkel JM. Predicting I/O performance in HPC using artificial neural networks. Supercomput Front Innov. 2016;3:19–33.
38. Schwan P. et al. Lustre: Building a file system for 1000-node clusters. In: Proceedings of the 2003 Linux symposium; 2003. p. 380–6.
39. Shan H. et al. Characterizing and predicting the I/O performance of hpc applications using a parameterized synthetic benchmark. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, IEEE Press; 2008. p. 42.
40. Shang P, Liu X, Yu C, Yan G, Xiang Q, Mi X. A new ensemble deep graph reinforcement learning network for spatio-temporal traffic volume forecasting in a freeway network. Digital Signal Process. 2022;123: 103419.
41. Snyder S, Carns P, Harms K, Latham R, Ross R. Performance evaluation of Darshan 3.0. 0 on the Cray XC30. Technical Report. Argonne National Lab.(ANL), Argonne, IL (United States); 2016.
42. Snyder S. et al. Modular HPC I/O characterization with darshan. In: 2016 5th Workshop on Extreme-Scale Programming Tools (ESPT), IEEE; 2016. p. 9–17.
43. Venkataraman S. et al. Ernest: efficient performance prediction for large-scale advanced analytics. In: 13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16); 2016. p. 363–78.
44. Verbeek JJ, Vlassis N, Kröse B. Efficient greedy learning of gaussian mixture models. Neural Comput. 2003;15:469–85.
45. Wang T. et al. Iominer: Large-scale analytics framework for gaining knowledge from I/O logs. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER), IEEE; 2018. p. 466–76.
46. Wartens CH, Garlick J. LMT-the lustre monitoring tool; 2010.
47. Xie B, Tan Z, Carns P, Chase J, Harms K, Lofstead J, Oral S, Vazhkudai SS, Wang F. Applying machine learning to understand write performance of large-scale parallel filesystems. In: 2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW), IEEE; 2019. p. 30–9.
48. Xie B, Tan Z, Carns P, Chase J, Harms K, Lofstead J, Oral S, Vazhkudai SS, Wang F. Interpreting write performance of supercomputer I/O systems with regression models. In: 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE; 2021. p. 557–66.
49. Xie B. et al. Predicting output performance of a petascale supercomputer. In: Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing; 2017. p. 181–92.
50. Xu G. et al. Simulation-based performance prediction of HPC applications: a case study of HPL. In: 2020 IEEE/ACM International Workshop on HPC User Support Tools (HUST) and Workshop on Programming and Performance Visualization Tools (ProTools), IEEE; 2020. p. 81–88.
51. Yoo AB, Jette MA, Grondona M. SLURM: Simple linux utility for resource management. In: Feitelson D, Rudolph L, Schwiegelshohn U, editors. Workshop on job scheduling strategies for parallel processing. Berlin: Springer; 2003. p. 44–60.
52. Yu J, Gao M, Li Y, Zhang Z, Ip WH, Yung KL. Workflow performance prediction based on graph structure aware deep attention neural network. J Ind Inf Integr. 2022;27: 100337.
53. Zhu Y, Chowdhury F, Fu H, Moody A, Mohror K, Sato K, Yu W. Entropy-aware I/O pipelining for large-scale deep learning on HPC systems. In: 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE; 2018. p. 145–56.

Publisher's Note