Towards HPC I/O Performance Prediction Through Large-scale Log Analysis

Sunggon Kim
Seoul National University
Seoul, Republic of Korea
skim@dcslab.snu.ac.kr

Alex Sim, Kesheng Wu, Suren Byna
Lawrence Berkeley National Laboratory
Berkeley, USA
[asim,kwu,sbyna]@lbl.gov

Yongseok Son
Chung-Ang University
Seoul, Republic of Korea
sysganda@cau.ac.kr

Hyeonsang Eom
Seoul National University
Seoul, Republic of Korea
hseom@cse.snu.ac.kr

ABSTRACT

Large-scale high performance computing (HPC) systems typically consist of many thousands of CPUs and storage units, while used by hundreds to thousands of users at the same time. Applications from these large numbers of users have diverse characteristics, such as varying compute, communication, memory, and I/O intensiveness. A good understanding of the performance characteristics of each user application is important for job scheduling and resource provisioning. Among these performance characteristics, the I/O performance is difficult to predict because the I/O system software is complex, the I/O system is shared among all users, and the I/O operations also heavily rely on networking systems. To improve the prediction of the I/O performance on HPC systems, we propose to integrate information from a number of different system logs and develop a regression-based approach that dynamically selects the most relevant features from the most recent log entries, and automatically select the best regression algorithm for the prediction task. Evaluation results show that our proposed scheme can predict the I/O performance with up to 84% prediction accuracy in the case of the I/O-intensive applications using the logs from CORI supercomputer at NERSC.

KEYWORDS

High Performance Computing; Distributed File System; I/O Performance Prediction; Log Analysis

ACM Reference Format:
Sunggon Kim, Alex Sim, Kesheng Wu, Suren Byna, Yongseok Son, and Hyeonsang Eom. 2020. 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 (HPDC '20), June 23–26, 2020, Stockholm, Sweden.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3369583.3392678

1 INTRODUCTION

Due to a large amount of data produced by both traditional HPC applications and recent machine learning and big data applications, the I/O performance on HPC systems has a major impact on overall application 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 would also allow the systems to better allocate CPU, I/O and networking resources, and enable computer centers to better provision the resources when they purchase the 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 hard to predict.

The I/O performance on HPC systems mainly depends on I/O libraries and underlying parallel file systems, such as Lustre [29] and IBM's SpectrumScale (previously known as GPFS) [26]. 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 the consistency of the file system. There are many ways that an application might experience a poor I/O performance, 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 performance of applications in large HPC environments and orchestrate them in the perspective of efficiency and priority.

HPC systems are continuously monitored by a number of different tools. For example, Slurm workload manager [37] records the progress of each job, Lustre Monitoring Tool (LMT) [35] logs the file system activities, and system monitoring tools such as Darshan [31] monitor the I/O activities of each application. 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 for understanding the application behavior.

To understand the I/O performance using HPC system logs, there have been many studies. Lockwood et al. [15] ran I/O-intensive scientific benchmarks and studied the logs to find various factors that impact the performance. Using application I/O and scheduler logs, TOKIO [16] provides a comprehensive graphical display that

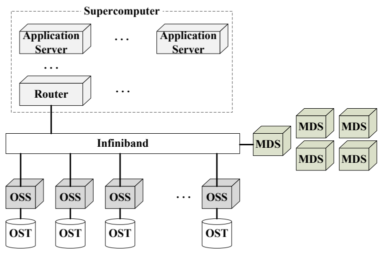


Figure 1: Overall architecture of CORI supercomputer.

helps users to understand the I/O behavior of the application and its impact on the overall system. There are various efforts to predict the I/O performance. For instance, I/O performance models [2, 3] 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), find key features that impact the I/O performance, and predict the I/O performance using the features.

Overall, we analyze system and I/O logs from a large system and propose a performance prediction scheme to predict the I/O performance of HPC applications. Our analysis results showed that the I/O performance of applications is affected by not only the application behavior but also the file system behavior. Also, the correlation between features and the I/O performance changes according to the I/O intensity of applications. Though the analysis, our paper presents deep understanding on I/O systems in a large HPC environments and help both system administers and users to understand the system and coordinate multiple I/O intensive applications to avoid performance degradation. With the findings from analysis, we proposed I/O performance prediction scheme that selects the features that impact the I/O performance from the logs and uses combinations of various regression algorithms to predict the I/O performance. Our evaluation using logs from the example system showed that combinations of regression algorithms can predict the performance with the accuracy of up to 84%.

Our contributions are as follows:

- We propose the I/O performance prediction scheme for HPC environments (Section 3.3 and 3.4) using system logs (Section 3.1 and 3.2) to help users understand the performance in complex systems.
- We analyze various system logs and the correlation between application behavior (Section 4.1 and 4.3), file system behavior (Section 4.2) and the relation to overall I/O performance (Section 4).
- We evaluate our prediction scheme using logs from a large scale HPC environment and applications with different characteristics (Section 5).

The rest of this paper is organized as follows: Section 2 describes the background. Section 3 presents our proposed performance prediction scheme. Section 4 describes the analysis results in CORI supercomputer at NERSC. Section 5 presents the evaluation results. Section 6 presents related works. Section 7 concludes the paper.

2 HPC ENVIRONMENT AND TOOLS

2.1 HPC environment and parallel file system

In Figure 1, we show the architecture of CORI supercomputer at NERSC with compute nodes, network interconnects (e.g., Infiniband), and a storage system. The storage subsystem we show is based on Lustre parallel file system [29]. 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 and every metadata operation is handled by MDS. Lustre's Object Storage Servers (OSS) are responsible for storing and retrieving user data. It is equipped with 248 OSSes and each OSS is connected to 1 Object storage target (OST) which is a set of HDDs that are grouped by RAID. The computation and storage systems are connected via Infiniband [25] providing a fast storage area network between the computation server and storage server.

2.2 System analysis tools in HPC

In this study, we use resource scheduler logs, I/O logs, and file system logs. Slurm workload manager [37] is a workload manager to allocate compute nodes and processes on HPC systems. It stores a complete history of jobs, username, number of processes and other information as logs. Slurm can provide a history of application execution using a JobID.

Darshan I/O characterization tool [6] is an I/O characterization tool that stores I/O information of a job. It stores I/O characteristics such as the number of bytes written, a histogram of request size, 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 the single application, Lustre monitoring tool (LMT) [35] is designed to monitor the file system activities. It collects information such as MDS CPU usage, MDS operations per second, OSS write throughput and more. For example, LMT is used to determine the file system performance after a major update to the system by analyzing abnormal MDS or OSS usage. This information can give insight into the abnormal behavior of the file system.

3 I/O PERFORMANCE PREDICTION METHOD

To predict the I/O performance of HPC applications, it is crucial to understand the existing logs in the system and process the information. In this section, we first describe storing information from various system logs into a database and then describe the prediction method.

3.1 Integrated database for system logs

Slurm, Darshan, and LMT logs have different fields and stored in different locations. For using them, we first store them in an integrated SQLite database to collect, store, and access the information during the prediction phase. In Figure 2, we show the overall procedure of building an integrated database. We build the database using entries of Darshan logs, which create an I/O log for each program execution. We use the JobID as a unique key in the database. We

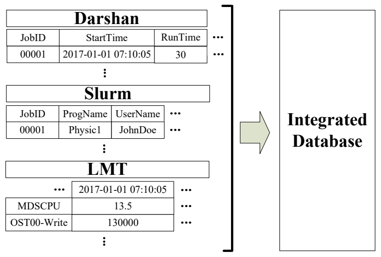


Figure 2: Overall procedure of building integrated database.

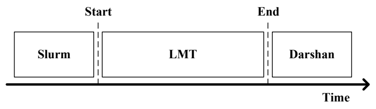


Figure 3: Availability of features in perspective of application run time.

first process Darshan logs by using the Darshan-parser [6] tool that parses a Darshan compressed format and the output is written to a text file format. We then read the output file and extract I/O related information such as StartTime, RunTime, and TotalBytesWritten. To extract the information from a Slurm log, we use the JobID from the Darshan log to search the entry in the Slurm database. As Slurm stores the history of jobs using MySQL, we perform a simple select operation with JobID and store the fetched information to the integrated database. In contrast to Darshan and Slurm, LMT continuously collects the information using the 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 [16], we extract the file system usage from the StartTime and end time (StartTime + RunTime) of the application. By continuously updating the integrated database with information collected from these three logs, our proposed scheme can effectively collect and access the I/O behavior of applications and the file system.

3.2 Selecting the features

In Figure 3, we show the availability of information when running an application. At the start of execution when the prediction has to be made, there is only limited information available. For example, although LMT collects the file system information continuously in 5-second intervals, information collected before the application execution can be irrelevant and the information reflecting the application behavior is not available until the end of execution. 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 run time. 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 has to specify the resources such as the number of processes and the number of nodes. In addition to Slurm, users need to specify the stripe size and stripe count of output directory on Lustre before the application execution by using a Lustre command. Darshan logs from the previous executions of the same application can be used for analysis. Darshan collects the I/O information of all applications that executed in the same environment. This information is important because many HPC applications, especially the I/O-intensive applications, have a 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 utilize that information for the prediction. Execution history with identical application name represents that the application is executed multiple times with similar I/O characteristics. For example, out of 3,543,538 application executions we have studied, there are only 2,039 distinct application names, suggesting that only a small set of applications are executed multiple times in the example system. This allows our scheme to utilize the information such as access pattern and request size from the previous executions to predict the performance of a target application,

In addition to the availability of information, it is also important to determine which information is important when predicting the I/O performance of an application. While many features are available, using many irrelevant features can reduce the accuracy of prediction and increase the computational overhead based on both our experiments and previous studies [10, 22]. To determine which information to use, we calculate the correlation factor between file system activity and information that is available. According to our analysis, the relevant information is different in every condition. For example, the correlation between stripe count and write throughput is far greater in a set of I/O-intensive applications than a set of other applications. This is because I/O-intensive applications can exploit high available throughput provided by the high number of storage devices. However, this is not the case for computation-intensive applications as the I/O requirement of those applications is already minimal and using more storage devices does not improve the I/O performance. Thus, to determine which information to use, we calculate the correlation factor at the start of prediction rather than using a set of predefined features.

3.3 Tools for prediction

Regression algorithms are typically used to estimate an unknown value from known values. The unknown value can be either the unavailable information such as file system activity or the target metric such as I/O We have selected various regression algorithms that have distinct characteristics. For statistical regressions, we use 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 more traditional machine learning-based regressions, we use k-Nearest Neighbor (kNN) [8] and Random Forest (RF) [14] algorithms. Finally, we used Multi-layered Perceptron (MLP) [23] and Convolutional Neural Network (CNN) [11] regression algorithms for multi-neuron based machine learning regressions. We use multiple of these algorithms because it is hard to select a single algorithm due to the complexity of data records. 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 the I/O performance using

PROCEDURE 1 Overall algorithm of proposed prediction scheme.

```
/* Predefined variables */
RegressionAlgorithms = [Linear, Polynomial, kNN, RF, MLP, CNN]
DB = integrated database
AvailableFeatures, UnavailableFeatures = []
ApplicationFeatures = [Features from Darshan logs]
FileSystemFeatures = [Features from LMT logs]

Function SelectFeatures (TargetFeature)
    /* Feature selection phase */
    HistoryRecords, SelectedFeatures = []
    If UnavailableFeatures.find(TargetFeature) == false
        HistoryRecords = SELECT * FROM DB
    Else
        HistoryRecords = SELECT AvailableFeatures FROM DB
    SelectedFeatures = Correlation(TargetFeature, HistoryRecords)
    /* Sort and Select the correlated features */
    return SelectedFeatures

Function DataPreperation (SelectedFeatures, TargetFeature)
    /* Data for building a model */
    X_train = SELECT (SelectedFeatures) FROM DB
    Y_train = SELECT (Target) FROM DB
    /* Data for the prediction */
    TargetRecords = SelectedFeatures of the target application
    For FeatureValue in TargetRecords
        If FeatureValue != Available and FeatureValue.Feature == ApplicationFeatures
            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 = MissingFeatureValue[0] //Value from the most recent execution
        If FeatureValue != Available and FeatureValue.Feature == FileSystemFeatures
            TempFeatureList = SelectFeatures(FeatureValue.Feature)
            X_train, Y_train, X_test = DataPreperation(TempFeatureList, FeatureValue.Feature)
            MissingFeatureValue = Prediction(X_train, Y_train, X_test)
            TargetRecords.replace(FeatureValue, MissingFeatureValue)
    X_test = TargetRecords
    retrun X_train, Y_train, X_test

Function Prediction (X_train, Y_train, X_test)
    /* Prediction phase */
    Prediction, Coefficient = 0
    For Algorithm in 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

/* Main procedure */
Target = [StartTime, ProgName, UserName, GroupID] //Information of the target application
TargetFeature = Read/Write Throughput
If Select Target.ProgName From DB == [] //No previous record of the target application
    AvailableFeatures = [Features from Slurm logs]
    UnavailableFeatures = [Features from LMT and Darshan logs]
Else
    AvailableFeatures = [Features from Slurm and Darshan logs]
    UnavailableFeatures = [Features from LMT logs]
TargetFeatureList = SelectFeatures (Target.Feature)
X_train, Y_train, X_test = DataPreperation (TargetFeatureList, Target.Feature)
PredictionResult = Prediction (X_train, Y_train, X_test)
```

various combinations of regression algorithms that exhibit different characteristics.

3.4 I/O prediction algorithm

To predict the I/O performance from system logs, we first select the important features, prepare the data, and perform prediction using the data. In Procedure 1, we show the simplified algorithm of the proposed method. Note that in this algorithm, we represent the name of the *selected feature* (e.g., SeqWritePct) as an example feature and the actual value of the feature (e.g., 99%). as a *feature value.*

As shown in the procedure, our scheme has three functions (SelectFeatures, DataPreperation, and Prediction) as well as the main procedure. At the start of the main procedure, we first initialize the values (e.g., TargetProgName) from the target application and feature. We then 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 at the start of an execution. As mentioned before, if any previous records exist (i.e., recurring application), we use the feature values of Slurm logs and Darshan logs for the most recent record to make a prediction. However, if there are no previous executions, we cannot use Darshan log of the previous records, so we set the available features to only features from the corresponding Slurm logs.

In the feature selection function, our proposed scheme selects the features that are correlated to the target feature (TargetFeature) that needs to be predicted. To do this, we select the list of features that can be used to predict the target feature. The target feature can be either the read/write throughput or an unavailable feature that has a strong correlation with the read/write throughput based on the prediction model but the value is not available until the end of the execution. In the case of the read/write throughput, we select all features that exist in our integrated database as the possible features that can have a correlation with the target feature. However, in the case of the unavailable feature, we select the features that are already available and can be used to predict the unavailable feature. With the records selected from the features (HistoryRecords), we determine the correlation between the target feature and other features using the Pearson correlation coefficient [5], which is a widely used algorithm to quantify the correlation between features. We report the selected features that have a strong correlation with the target feature.

In the data preparation function, 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 between the selected features and the target feature. To do this, we select all the records of 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 a model to find the value of the target feature. However, the information of selected features related to the application (from Darshan) and file system (from LMT) behavior is not available until the end of execution. To overcome this issue, for the unavailable features related to the application behavior, we use the values from the most recent execution in the case of recurring application[1]. We first select the records that share the application name, user name, and group ID. If there are no records that share all the information, we relax the select condition to find the records that share any of the three conditions. Finally, we select the record from the most recent execution since the application behavior can change from the application library and system update. For the unavailable features related to the file system behavior, we use the predicted file system values by performing the prediction using the available information to predict the file system behavior. By recursively calling the SelectFeatures, DataPreperation, and Prediction function, our scheme selects the features that are available and related to the unavailable file system feature, uses the values from current (from Slurm) and the most

[1]In the case of a new application that has no previous records, only the available information (i.e., Slurm logs) are selected from the feature selection function. Thus, there is no unavailable information related to the application behavior.

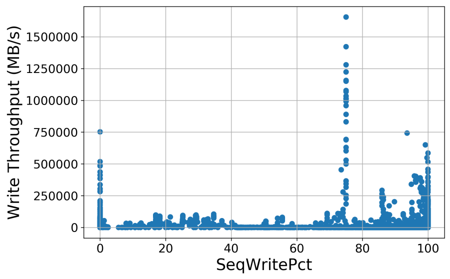


Figure 4: Correlation between SeqWritePct and WriteThroughput.

recent execution (from Darshan), and makes the prediction of the unavailable feature. Thus, by using the application features from the recent execution and performing the prediction of file system features, our scheme can prepare the data for building a model (X_train, Y_train) and the prediction (X_test)

With the prepared data, we perform 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 of selected features (X_train) and 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 of the selected feature (X_test) related to the target application. Finally, by comparing the coefficient of determination, we select the regression algorithm with the highest coefficient of determination and return the prediction result of that regression algorithm.

For the implementation, we used python to build our proposed prediction algorithm. For the database operations, we used *sqlite3 DB-API* provided in Python. For the Pearson correlation factor, we used the algorithm provided by Pandas dataframes [20]. In the case of the regression algorithms, we used *scikit-learn* library [24] with the exception of CNN using *TensorFlow* [1, 7].

4 ANALYSIS OF I/O LOGS

In this section, we present our analysis results on the I/O behavior of HPC applications and the correlation between their I/O characteristics and performance. We used application logs acquired from CORI system. There are 3,543,538 logs from October 2017 to January 2018. Note that this is not the number of distinct applications but the number of jobs dispatched to the system. For example, if an application is executed for 10 occasions, it creates 10 different logs.

4.1 Application I/O characteristics

While the I/O performance of an application is affected by many factors, there is no doubt that the I/O behavior of application itself is one of the key factors in both small and large scale system [12, 21, 30]. To find the effect of the I/O behavior on the performance, we conducted an analysis using the information acquired by Slurm and Darshan. In Table 1, we show 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.

Table 1: List of information extracted from Slurm and Darshan logs.

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

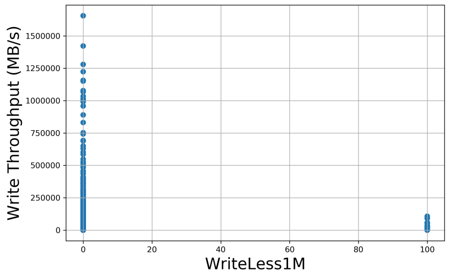


Figure 5: Correlation between WriteLess1M and WriteThroughput (Each color represents a group of applications that share similar characteristic)s.

In Figure 4, we show the correlation between the access pattern (SeqWritePct) and write throughput. The sequential write percentage is distributed fairly evenly from 0 to 100%. However, the write throughput has no correlation with sequential write throughput since the write throughput of higher sequential write percentage is not significantly higher than that of lower sequential write percentage. This suggests that although there might be a correlation between the access pattern and write throughput, it is not deterministic.

We illustrate the correlation between the request size (WriteLess1M) and write throughput in Figure 5. WriteLess1M refers to the percentage of write requests that have a request size of less than 1 megabyte. Thus, if the WriteLess1M is higher, the application issues more write requests with a small request size. The distribution of WriteLess1M is skewed towards 0 and 100 percent. This suggests that HPC applications can be categorized into small and large I/O applications. Similar to SeqWritePct, the correlation between request size and write throughput is not deterministic. Note that some data points at 0 percent have extremely high write throughput because many applications have very short write time, yielding very high write throughput. However, the sustained write throughput should be lower.

As shown in both figures, our analysis results show that the correlation between each I/O behavior metric and write throughput is inconclusive. As we analyzed the correlation between write

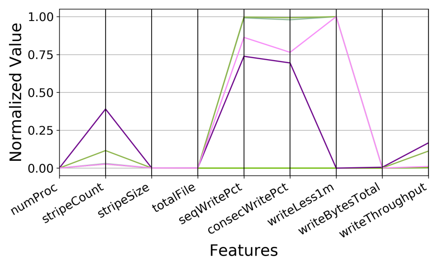


Figure 6: Correlation between I/O behavior metrics and WriteThroughput.

throughput and metrics shown in table 1, the results can be categorized into two categories similar to two figures. The first category is similar to SeqWritePct where the distribution of data points is fairly uniform. In other cases, they are similar to WriteLess1M where the distribution is skewed towards very low and high ends. In both cases, the correlation between independent metric and write throughput was inconclusive.

To find the correlation between the I/O behavior and write throughput more comprehensively, we analyzed the correlation using all the metrics presented in table 1. By clustering the history of logs in clusters, we created clusters with a distinct I/O behavior and check the correlation between a set of I/O behavior and write throughput. We used the Gaussian mixture model [33] from the Scikit-learn python library [24] to build clusters. 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 6 shows the analysis result using the clustering algorithm. As shown in the figure, there are 5 clusters represented by each line and two clusters represented as purple and light green line have high write throughput compared to others. The common features of two clusters are that they have a relatively high number of files, sequential/consecutive write percentage, and total bytes written. In contrast, the request size which is represented by writeLess1M is different. However, the analysis result is inconclusive because other clusters share similar trends in a few metrics. If a metric or combination of metrics has a strong correlation with write throughput, a cluster that shares the trend must have similar write throughput. However, other clusters that share similar metrics have low write throughput suggesting that in addition to the application behavior, the effect of other applications and file system activity must be considered to accurately predict the I/O performance in HPC environments.

4.2 File system activities

The analysis results from the previous subsection suggested that to understand the I/O performance, analysis of the file system activities is needed 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 show very unusual performance [15]. Thus, we focused on the LMT which collects the lustre file system information in a 5-second interval.

Table 2 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 collect system activity during the application run time. To analyze the system activity in both normal and abnormal cases, we use the average value (e.g., mdsCPUmean) and the 95 percentile of the highest value (e.g., mdsCPU95) during the run time. This is because even if the system is stable in most of application run time, a single peak can degrade the overall performance of the application.

Table 2: List of information extracted from LMT logs.

Name	Description
mdsCPU[Mean/95]	mean and 95 percentile CPU usage of a MDS server during the application runtime
mdsOPS[Mean/95]	mean and 95 percentile operations per second of a MDS server during the application runtime
ossCPU[Mean/95]	mean and 95 percentile CPU usage of OSS servers during the application runtime
ossWrite[Mean/95n]	mean and 95 percentile write throughput of OSS servers during the application runtime
ossRead[Mean/95]	mean and 95 percentile read throughput of OSS servers during the application runtime

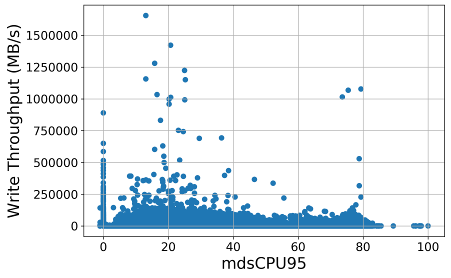


Figure 7: Correlation between mdsCPU95 and write throughput.

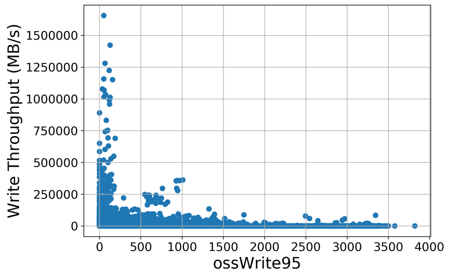


Figure 8: Correlation between ossWrite95 and write throughput.

Figure 7 shows the correlation between MDS CPU activity and write throughput. As shown in the figure, while the MDS CPU usage is uniformly distributed, there is no correlation between write throughput. It is important to notice that there are only a handful of cases where MDS CPU usage is over 80 percent. This result suggests that the file system is rarely experiencing the bottleneck caused by MDS and the file system has sufficient computing power to handle the metadata request from multiple users.

Figure 8 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 1,000. These linear data points suggest that in certain types of applications, as the write activity of the file system increases, the

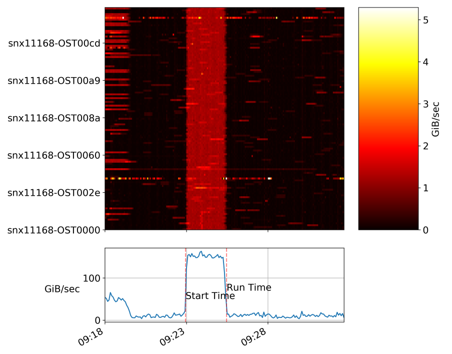


Figure 9: I/O heatmap of the file system when IOR benchmark is running.

write performance of the application increases as well. Thus, while not all the HPC applications follow identical correlation with write throughput, in certain applications, the general activity of the file system can have a strong correlation with write throughput.

4.3 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 figure 8, we observed that there can be a correlation between two metrics in a certain group of applications. To do this, we analyzed applications with various I/O characteristics and how the file system activity such as mdsOPS and ossWrite was affected.

Figure 9 shows the write activity heatmap of the file system when an instance of IOR benchmark [4] is running. In the example system, benchmarks such as IOR are scheduled to run every day to observe the impact of the file system upgrade. Thus, by studying regularly scheduled applications, we can observe the impact of the file system on the performance of applications. The graph was generated using the pytokio tool which is TOKIO tool based on python [16]. Each snx11168-OST of the y-axis represents an OST, and the x-axis represents the time. The graph includes the file system activity of 5 minutes before the application start time and 5 minutes after the application run time which are labeled as Start Time and Run Time, respectively. Note that each line parallel to the x-axis represents the busyness of an OST, with red representing a busy OST.

As shown in the figure, the write activity of the file system increases during the run time of the IOR benchmark. From our analysis, there are 16 applications that were running during the run time of the IOR benchmark. However, while multiple applications are running in the system, we are certain that file system activity is increased due to the IOR benchmark because of two observations. First, the write activity increases as soon as the benchmark was executed and decreases as soon as the benchmark was terminated. Second, the number of bytes written during the high activity period is similar to the total number of bytes written by the benchmark, and the other 16 applications have very a small number of bytes written. 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,

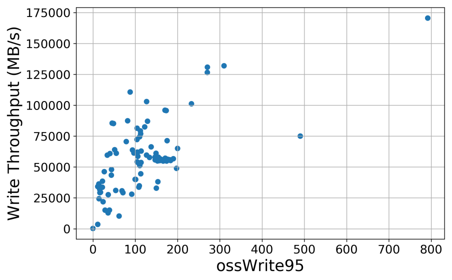


Figure 10: Correlation between ossWrite95 and write throughput on high I/O intensity applications.

the correlation between file system activity and application write throughput is more significant when the write activity of certain application dominates the file system activity.

From the analysis, we found that I/O-intensive applications can have a dominant effect on the file system. However, finding applications that induce heavy I/O activity is difficult since Darshan does not record the relative I/O usage. To do this, we extract the list of OSTs that are used by an application. Although the Lustre usage is not available for all logs, applications that are compiled with the Lustre information flag have the Lustre usage information such as which OSTs are used by the application. With the list of used OSTs, we propose a new metric called WriteBytesPct.

$$WriteBytesPct = \frac{WriteBytes_{UsedOSTs}}{WriteBytes_{AllOSTs}} \tag{1}$$

Equation 1 shows how we calculate WriteBytesPct. 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. Thus, we utilize this metric to select applications that have high I/O intensity in terms of the effect on the file system.

With the new metric (WriteBytesPct), we revisit figure 8 and select high I/O intensity applications. To find the applications that have a high impact on the file system, we selected the high I/O intensity applications that have WriteBytesPct of 80 percent or higher which suggests that the application accounts for more than 80% of file system activity. Figure 10 shows the correlation between OSS write activity and application write throughput on high I/O intensity applications. As shown in the figure, the performance of applications has a very strong correlation with file system activity. This is because when an application dominates the I/O activity of the file system, the performance of the file system is the performance of the application itself.

Based on our analysis, there are two major observations on the I/O characteristics of HPC applications. First, no single I/O characteristic is dominant enough to observe a clear correlation between write throughput. This demonstrates that HPC systems have a very complex I/O performance model. Second, there is a strong correlation between the 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 certain scenarios where I/O performance is dominant in overall application performance, the I/O characteristics can be used to accurately predict the I/O performance of HPC applications.

5 EVALUATION

5.1 Preparation of the data

For evaluation, we used identical logs as mentioned in Section 4. To validate our proposed scheme and compare the predicted performance with the actual performance, we divided our logs into two groups. For the training phase to build the prediction model, we used the data from October 1st to November 30th of 2017. For the test phase to validate our prediction, we used the data from December 1st of 2017 and January 31st of 2018. Thus, our scenario was that while we have a set of logs from training phases, we are trying to predict the application performance of test data and validate the prediction accuracy.

Another important consideration is how to normalize our data since features collected have different units. For example, percentage values such as SeqWritePct are between 0 and 100 while byte values such as WriteBytesTotal can be any positive integers. To standardize the values, we used a standard scaler to balance the values with the different units. Note that although we standardized the values, we did not change the unit of write throughput when reporting prediction result to easily estimate our prediction accuracy and application performance.

5.2 High I/O intensity applications

We first performed the evaluation using the application with high I/O intensity. During the feature selection phase, we selected the top three features that have a strong correlation with write throughput during the evaluation. By using the correlation algorithm, ossWrite95, WriteBytesTotal, and SeqWritePct had the most correlation with write throughput. While WriteBytesTotal and SeqWritePct are available from Darshan logs of previously executed applications, ossWrite95 is not available until the application finishes running. Thus, to predict the unavailable feature, we performed another correlation algorithm to find the top three most correlated features with ossWrite95. Our evaluation gave NumFile, WriteBytesTotal, and StripeCount. Thus, we used NumFile, WriteBytesTotal, and StripeCount to predict the ossWrite95, and used the predicted ossWrite95, WriteBytesTotal, and SeqWritePct to predict the write throughput.

Figure 11 shows the prediction results of ossWrite95 using NumFile, WriteBytesTotal and StripeCount. While the x-axis represents the measured ossWrite95 from LMT log, the y-axis represents the predicted ossWrite95. Thus, the prediction is more accurate if the data points are placed near the black dotted line. For evaluation, we used the six regression algorithms mentioned in Section 3. As shown in the figure, random forest (RF) regression had the most accurate prediction while linear, polynomial and convolutional neural network (CNN) regression showed a skewed prediction result. This is because the ossWrite95 does not have a data pattern that can be accurately estimated using simple linear and polynomial equations.

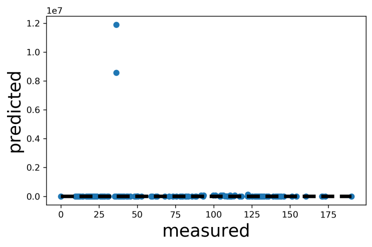
With the predicted ossWrite95, we performed the prediction of write throughput. Table 3 shows the list of regression algorithms

Table 3: List of regression algorithms and their coefficients of determination on high I/O intensity applications.

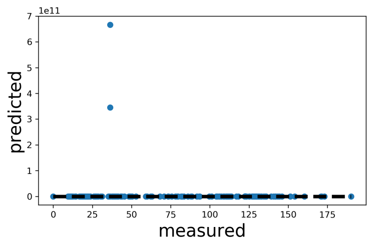
Regression Algorithm	Coefficient of Determination
Linear -> RF	0.84
Polynomial -> RF	0.83
kNN -> RF	0.81
MLP -> RF	0.81
RF -> kNN	0.79
⋮	⋮
kNN -> MLP	0.40
RF -> MLP	0.39
Polynomial -> Linear	0.35
kNN -> Linear	0.34
MLP -> Linear	0.33

used in prediction and their coefficients of determination when predicting write throughput. The coefficient of determination represents the variance in predicted values from measured value [27]. If the value is closer to 1, the prediction is identical to the measured value and the number decreases if the distance between the prediction and the measured increases. Note that the first algorithm before the arrow represents the algorithm used to predict ossWrite95 and algorithm after the arrow represents the algorithm used to predict write throughput. As shown in the table, the prediction is the most accurate when linear regression is used to predict ossWrite95 and RF regression is used to predict write throughput. This is interesting because as shown in Figure 11, linear regression had two data points that have very low prediction accuracy. However, since a combination of linear and RF regression has the lowest conceptual overhead, using the combination is beneficial not only from the perspective of the accuracy but also the computational overhead. Also, multi-layer perceptron (MLP) and CNN regression which are considered as more modern algorithms did not perform particularly well when predicting both ossWrite95 and write throughput. Finally, the accuracy was not great when an identical algorithm was used twice which suggests that a different approach is needed when predicting ossWrite95 and write throughput. Thus, in the case of this evaluation scenario, no algorithm is superior to another in every cases and combinations of regression algorithm must be evaluated for different sets of application data to accurately predict the I/O performance.

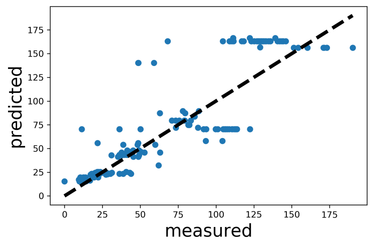
Figure 12 shows graphs of prediction results using different combinations of regression algorithms. The top three figures show the combinations with high accuracy and the bottom three figures show those with low accuracy. As shown in Table 3, combinations listed in Figure 12a, Figure 12b and Figure 12c have coefficients of determination of 0.84, 0.83 and 0.81, respectively, while combinations listed in Figure 12d, Figure 12d and Figure 12f have coefficients of determination of 0.35, 0.34 and 0.33, respectively. Figure 12a, Figure 12b and Figure 12c have data points close to the black dotted line which represents 100 percent accuracy. This suggests that in the case of high I/O intensity applications, the write throughput can be predicted accurately which is in line with our observation from the analysis. However, the inaccurate data points are shared among predictions that suggest that either prediction of ossWrite95 or other inputs used to predict write throughput does not follow the model accurately. In case of combinations with low accuracy,



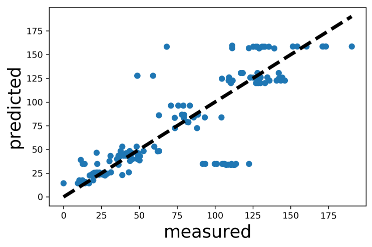
(a) Linear Regression



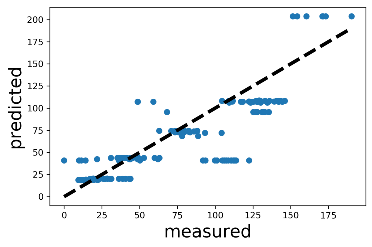
(b) Polynomial Regression



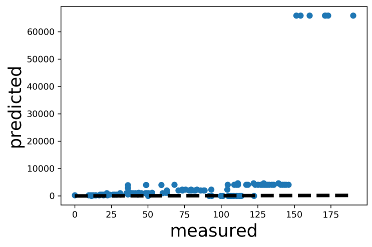
(c) k-nearest Neighbor Regression



(d) Random Forest Regression

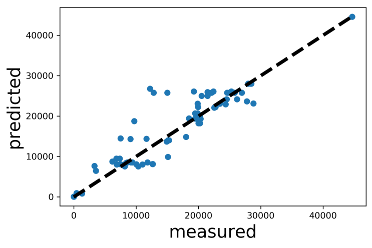


(e) Multi-layer Preceptron Regression

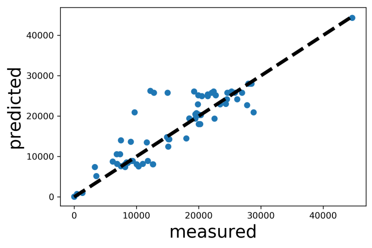


(f) Convolutional Neural Network Regression

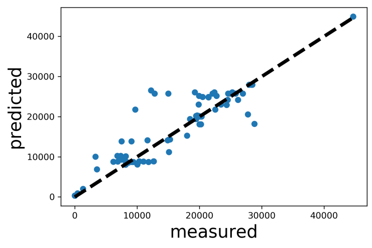
Figure 11: Prediction results of global file system activity (ossWrite95) on high I/O intensity applications.



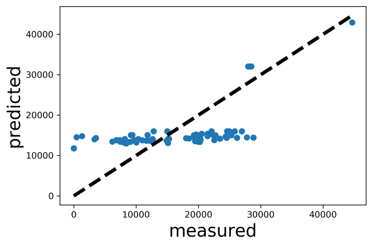
(a) Linear Regression and Random Forest Regression



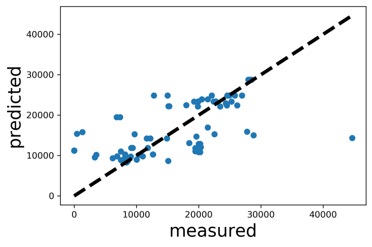
(b) Polynomial Regression and Random Forest Regression



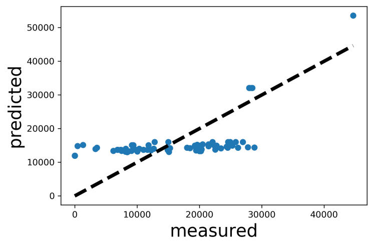
(c) k-nearest Neighbor Regression and Random Forest Regression



(d) Polynomial Regression and Linear Regression



(e) k-nearest Neighbor Regression and Linear Regression



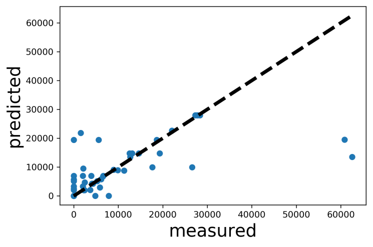
(f) Multi-layer Preceptron Regression and Linear Regression

Figure 12: Prediction results of Write throughput (WriteThroughput) on high I/O intensity applications.

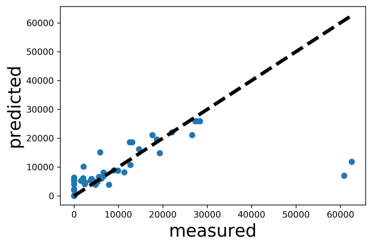
while combinations listed in Figure 12d and Figure 12f are similar and the prediction results are inaccurate, the combination listed in Figure 12e seems more accurate. This suggests that while different combinations can have a similar coefficient of determination, some might have fairly accurate predictions with few vastly inaccurate predictions.

5.3 Medium I/O intensity applications

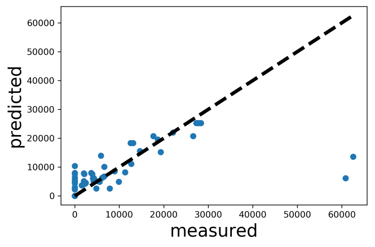
To evaluate our scheme in a different scenario, we evaluated our scheme by predicting the I/O performance of applications that have less I/O activity. This is important because our analysis already confirmed that features of high I/O intensity applications have a strong correlation with write throughput. However, it is also important to verify our scheme in a different scenario where the correlation



(a) Polynomial Regression and k-nearest neighbor Regression



(b) Polynomial Regression and Random Forest Regression



(c) Multi-layer Perceptron Regression and Random Forest Regression

Figure 13: Prediction results of Write throughput (WriteThroughput) on medium I/O intensity applications.

Table 4: Top five regression algorithms and their coefficients of determination on medium I/O intensity applications.

Regression Algorithm	Coefficient of Determination
Polynomial -> kNN	0.48
Polynomial -> RF	0.45
MLP -> RF	0.44
RF -> RF	0.44
Linear -> RF	0.41

between the features and write throughput is less significant. To do this, we used applications that have WriteBytesPct between 30% and 80%. This suggests that OSTs used by applications were generating 30% to 80% of file system activity.

For evaluation, we used identical logs and divided training and test data equally. Through the correlation analysis, the three most correlated features with write throughput were WriteBytesTotal, SeqWritePct, and ossWrite95. While the features remain identical from high I/O intensity applications, the ordering changed, especially the order of ossWrite95, suggesting that the importance of file system activity decreased. From the correlation analysis on ossWrite95, we have selected WriteBytesTotal, WriteLess1M, and StripeCount. Thus, similar to the evaluation of high I/O intensity applications, we first predicted the ossWrite95 and used that value to predict write throughput.

Table 4 shows the top five regression algorithms which had the highest coefficient of determination when predicting write throughput. As shown in the table, the overall accuracy decreased compared with the evaluation with high I/O intensity applications. This is because the features used to predict both ossWrite95 and write throughput had less correlation than that of high I/O intensity applications. While the high I/O intensity applications have a similar I/O behavior and effect on the file system, medium I/O intensity applications have diverse characteristics. Thus, while the regression algorithms are identical, the overall accuracy decreased significantly.

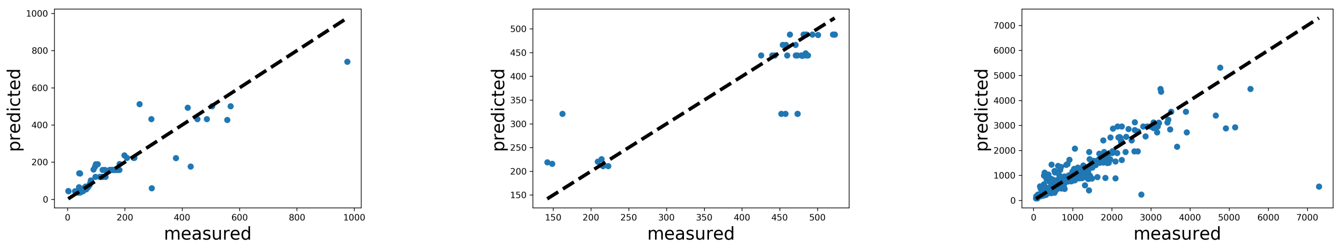
Figure 13 shows the prediction graph of the top three combinations of algorithms that have the highest coefficient of determination as listed in Table 4. As shown in the figure, while most of the predictions were accurate, the prediction result of two data points at 60,000 of x-axis decreased the overall accuracy. It is also noteworthy that although Figure 13a seems to show more inaccurate results than that of Figure 13b and Figure 13c, the coefficient of determination is higher. This is because the coefficient of determination is calculated by computing the distance between the predicted and measured value. Thus, since the combination of Figure 13a leads to a more accurate prediction of the two most inaccurate data points compared with the combinations of Figure 13b and Figure 13c, the overall distance value is smaller even though the overall prediction accuracy is lower. In conclusion, the correlation between I/O characteristics and the performance is lower when the I/O performance is not the key factor in application performance. However, the results showed that by analyzing other factors, our scheme can accurately predict the I/O performance.

5.4 I/O performance of individual application

Through the evaluation of high and medium I/O intensity applications, we discovered that the prediction accuracy is related with 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 are correlated with the performance, it also revealed that it can be difficult to accurately predict the performance of an application that has different I/O behavior from the other applications in the same category. To validate that our proposed scheme works if we categorize the applications by application name rather than relative I/O activity to file system, we performed evaluation by selecting target applications and predict the performance of each targeted application using system logs of the application.

For the target applications, we have selected a quantum chemistry application and two biology applications that ran continuously in the system and have medium to high I/O intensity. For the quantum chemistry application, the write throughput was correlated with WriteBytesTotal, TotalMetaReq and mdsCPUMean. Since mdsCPUMean is not available at the start of execution, we used TotalMetaReq, ReadBytesTotal, and StripeCount to predict mdsCPUMean. For the biology applications (e.g., biology1 and biology2), we used WriteLess1m, TotalIOReq, & NumFile, and NumFile, WriteBytesTotal, & SeqWritePct, respectively. In the case of biology applications, all the correlated features were available. Thus, we did not perform additional prediction to predict the unavailable information.

Figure 14 shows the prediction result of write throughput on targeted applications. Using the RF algorithm in all phases, our proposed algorithm predicted the write throughput of quantum chemistry, biology1, and biology2 applications with 0.83. 0.73, and 0.77 coefficients of determination, respectively. Similar to high I/O



(a) Quantum chemistry

(b) Biology1

(c) Biology2

Figure 14: Prediction results of write throughput (WriteThroughput) on targeted applications.

intensity applications, our prediction algorithm predicted the write throughput with 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 solely based on the characteristics of the target applications. Thus, the evaluation result shows that while the prediction result 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.

In the perspective of the regression algorithm, in the case of our evaluation, RF regression showed the most accurate prediction result compared with other regression algorithms in all the evaluation scenarios. We believe that while linear and polynomial regression is simple, the pattern between the low performing executions and the high performing execution can be different and these algorithms cannot build a model reflecting the dynamic changes. Similarly, MLP and CNN regressions tend to overfit to either low or high performing executions. In contrast, since RF 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 and high performing executions. Thus, according to our evaluation results, we think that using RF regression with linear or polynomial regression which has low computational and conceptual overhead can be a good starting point when adapting our proposed scheme to different HPC environments.

6 RELATED WORK

6.1 Understanding I/O characteristics of HPC applications

There have been many studies that explore the characteristics of HPC applications. Lang et al. [13] 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 in the perspective of hardware and software components in the HPC system. Teng et al. [34] proposed a method for integrating I/O logs from the HPC system and performed analysis on the system. Their analysis showed that comprehensive log analysis is needed to find the root cause of performance degradation. Lockwood et al. [15] 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 upgrade and continuous execution of I/O-intensive applications can affect the application performance. Kim et al. [9] performed an analysis of the distributed file system used in the HPC environment. They discovered that most of the applications are not utilizing 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 I/O performance of the application in the HPC environment. In contrast, we focus on both I/O behavior of application and file system and propose a scheme that predicts the performance of the application using the history of logs.

6.2 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 [32] is a framework to predict the performance of large scale analytical applications by building a model based on resources. By studying hardware and application, it predicts the performance of application that is designed for the distributed system. Lux et al. [17] 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. [28] proposed a prediction scheme using artificial neural network in HPC system. Other works [18, 19, 36] also tried to predict various performance metrics for large clusters. Similar to these studies, our scheme aims to predict the performance of the application using characteristics of the HPC environment. However, we focus on analyzing the system using various logs and make the prediction based on the history of logs collected in the same environment. This enables our proposed scheme to make an adjustment for various system activities that can cause sudden changes in the model and make a prediction based on I/O behavior of complex HPC applications.

7 CONCLUSION

In this paper, we propose a regression based I/O performance prediction scheme for HPC environments. To do this, we collect the multiple system logs into a single integrated database and use various combinations of regression algorithms to predict the I/O performance using the database. Our analysis of logs from the example HPC system showed that no single I/O feature can be used to accurately predict the I/O performance of applications. By selecting the most relevant features, the most recent history of application, and the best regression algorithm, our scheme can predict the I/O performance of applications with up to 84% accuracy according to our evaluation results. We believe that the presented analysis results can help users to predict the I/O performance of their applications and schedule their applications efficiently, avoiding I/O interference by other applications. Also, our scheme can help system administers to understand I/O behaviors in a large HPC system and efficiently allocate and manage resources in a complex system. We share our code and sample data set in following link: https://bitbucket.org/berkeleylab/piop

8 ACKNOWLEDGMENTS

We want to thank anonymous reviewers and our shepherd Gerald Lofstead from Sandia National Lab for helpful reviews that improved the quality of this paper. This work was supported by by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2016M3C4A7952587, No.2018R1C1B5085640), and the BK21 Plus for Pioneers in Innovative Computing (Dept. of Computer Science and Engineering, SNU) funded by National Research Foundation of Korea (NRF) (No. 21A20151113068). 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).

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] Babak Behzad, Surendra Byna, Prabhat, and Marc Snir. 2015. Pattern-driven Parallel I/O Tuning. In *Proceedings of the 10th Parallel Data Storage Workshop (PDSW '15)*. ACM, New York, NY, USA, 43–48. https://doi.org/10.1145/2834976.2834977

[3] Babak Behzad, Surendra Byna, Stefan M. Wild, Mr. Prabhat, and Marc Snir. 2014. Improving Parallel I/O Autotuning with Performance Modeling. In *Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '14)*. Association for Computing Machinery, New York, NY, USA, 253–256. https://doi.org/10.1145/2600212.2600708

[4] IOR Benchmark. 2020. https://asc. llnl. gov/sequoia/benchmarks/IOR summary v1. 0. pdf. *Accessed January* 5 (2020).

[5] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. In *Noise reduction in speech processing*. Springer, 1–4.

[6] Philip Carns, Robert Latham, Robert Ross, Kamil Iskra, Samuel Lang, and Katherine Riley. 2009. 24/7 characterization of petascale I/O workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 1–10.

[7] François Chollet et al. 2015. Keras. https://keras.io.

[8] Sahibsingh A Dudani. 1976. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics* 4 (1976), 325–327.

[9] Sunggon Kim, Alex Sim, Kesheng Wu, Suren Byna, Teng Wang, Yongseok Son, and Hyeonsang Eom. 2019. 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)*. 351–360.

[10] Kenji Kira and Larry A Rendell. 1992. A practical approach to feature selection. In *Machine Learning Proceedings 1992*. Elsevier, 249–256.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[12] Tom M Kroeger and Darrell DE Long. 1999. The case for efficient file access pattern modeling. In *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*. IEEE, 14–19.

[13] Samuel Lang, Philip Carns, Robert Latham, Robert Ross, Kevin Harms, and William Allcock. 2009. I/O performance challenges at leadership scale. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. IEEE, 1–12.

[14] Andy Liaw, Matthew Wiener, et al. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.

[15] Glenn K Lockwood, Shane Snyder, Teng Wang, Suren Byna, Philip Carns, and Nicholas J Wright. 2018. A year in the life of a parallel file system. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE Press, 74.

[16] Glenn K Lockwood, Nicholas J Wright, Shane Snyder, Philip Carns, George Brown, and Kevin Harms. 2018. TOKIO on ClusterStor: connecting standard tools to enable holistic I/O performance analysis. (2018).

[17] Thomas CH Lux, Layne T Watson, Tyler H Chang, Jon Bernard, Bo Li, Li Xu, Godmar Back, Ali R Butt, Kirk W Cameron, and Yili Hong. 2018. Predictive modeling of I/O characteristics in high performance computing systems. In *Proceedings of the High Performance Computing Symposium*. Society for Computer Simulation International, 8.

[18] Andréa Matsunaga and José AB Fortes. 2010. On the use of machine learning to predict the time and resources consumed by applications. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE, 495–504.

[19] Ryan McKenna, Stephen Herbein, Adam Moody, Todd Gamblin, and Michela Taufer. 2016. Machine learning predictions of runtime and IO traffic on high-end clusters. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 255–258.

[20] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman (Eds.). 51 – 56.

[21] Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. 2012. SFS: random write considered harmful in solid state drives.. In *FAST*, Vol. 12. 1–16.

[22] Amir Navot, Ran Gilad-Bachrach, Yiftah Navot, and Naftali Tishby. 2005. Is feature selection still necessary?. In *International Statistical and Optimization Perspectives Workshop" Subspace, Latent Structure and Feature Selection"*. Springer, 127–138.

[23] Sankar K Pal and Sushmita Mitra. 1992. Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on neural networks* 3, 5 (1992), 683–697.

[24] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[25] Gregory F Pfister. 2001. An introduction to the infiniband architecture. *High Performance Mass Storage and Parallel I/O* 42 (2001), 617–632.

[26] Dino Quintero, Luis Bolinches, Puneet Chaudhary, Willard Davis, Steve Duersch, Carlos Henrique Fachim, Andrei Socoliuc, Olaf Weiser, et al. 2017. *IBM Spectrum Scale (formerly GPFS)*. IBM Redbooks.

[27] Calyampudi Radhakrishna Rao, Calyampudi Radhakrishna Rao, Mathematischer Statistiker, Calyampudi Radhakrishna Rao, and Calyampudi Radhakrishna Rao. 1973. *Linear statistical inference and its applications*. Vol. 2. Wiley New York.

[28] Jan F Schmidt and Julian M Kunkel. 2016. Predicting I/O performance in HPC using artificial neural networks. *Supercomputing Frontiers and Innovations* 3, 3 (2016), 19–33.

[29] Philip Schwan et al. 2003. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux symposium*, Vol. 2003. 380–386.

[30] Hongzhang Shan, Katie Antypas, and John Shalf. 2008. 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, 42.

[31] Shane Snyder, Philip Carns, Kevin Harms, Robert Ross, Glenn K Lockwood, and Nicholas J Wright. 2016. Modular hpc i/o characterization with darshan. In *2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*. IEEE, 9–17.

[32] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. 2016. Ernest: efficient performance prediction for large-scale advanced analytics. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*. 363–378.

[33] Jakob J Verbeek, Nikos Vlassis, and Ben Kröse. 2003. Efficient greedy learning of Gaussian mixture models. *Neural computation* 15, 2 (2003), 469–485.

[34] Teng Wang, Shane Snyder, Glenn Lockwood, Philip Carns, Nicholas Wright, and Suren Byna. 2018. IOMiner: Large-Scale Analytics Framework for Gaining Knowledge from I/O Logs. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 466–476.

[35] CM Herb Wartens and Jim Garlick. 2010. LMT-The Lustre Monitoring Tool.

[36] Bing Xie, Yezhou Huang, Jeffrey S Chase, Jong Youl Choi, Scott Klasky, Jay Lofstead, and Sarp Oral. 2017. Predicting output performance of a petascale supercomputer. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. 181–192.

[37] Andy B Yoo, Morris A Jette, and Mark Grondona. 2003. Slurm: Simple linux utility for resource management. In *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 44–60.