I/O Performance Characterization and Prediction through Machine Learning on HPC Systems

Lipeng Wan, Matthew Wolf, Feiyi Wang, Jong Youl Choi, George Ostrouchov, Jieyang Chen,
Norbert Podhorszki, Jeremy Logan, Kshitij Mehta, Scott Klasky and Dave Pugmire
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
Email: {wanl, wolfmd, fwang2, choij, ostrouchovg, chenj3, pnb, loganjs, mehtakv, klasky, pugmire}@ornl.gov

***Abstract*—When scientists run their applications on high-performance computing (HPC) systems, they often experience highly variable runtime I/O performance, and sometimes unexpected I/O performance degradations can dramatically slow down the applications' execution. This issue is mainly caused by I/O bandwidth contention, since the storage subsystem of HPC systems is usually shared by many concurrently running applications and the I/O performance of each application might be affected by I/O traffic from others. In order to mitigate the I/O bandwidth contention, scientific applications running on HPC systems need to schedule their I/O operations in a proactive and intelligent manner, which necessitates the capability of predicting the near-future runtime I/O performance. However, the runtime I/O performance prediction in production HPC environments is extremely challenging, as the storage subsystems are complex and the I/O operations of those running applications might have irregular patterns.**

In this paper, we formulate the I/O performance prediction on production HPC systems as a classification problem and exploit a range of machine learning techniques to address it. Our I/O prediction model is lightweight and its effectiveness is validated using real performance traces collected from two Cray supercomputer systems. Our results show that transitions between different I/O performance states can be predicted by our model with high average accuracy (75% when SVM is used). Moreover, the prediction is robust even when only limited training data is available. We also study and demonstrate how to leverage the prediction results to improve the efficiency of I/O bandwidth utilization on these two Cray supercomputer systems.

I. Introduction

As scientific applications running on HPC systems become increasingly data-intensive, one of the major challenges faced by these applications is the nondeterministic execution time caused by the highly variable I/O performance on production HPC systems. As has been observed and analyzed by several existing studies [1], [2], [3], [4], even on DoE's leadership supercomputers, it is possible for running applications to experience more than an order of magnitude variation in I/O performance, which significantly increases the uncertainty of applications' execution time. There are three main causes of I/O performance variability on production HPC systems. First, although the I/O bandwidth of HPC systems has increased steadily during the past decade, the ratio of compute capability to I/O bandwidth has continued to grow, meaning the I/O bandwidth is still limited and might not be able to satisfy all data transfer requirements, especially those from data-intensive applications. Second, due to the nature of parallel programming model, applications running on HPC systems often generate bursty I/O traffic. When the limited I/O bandwidth is shared by these applications, the potential risk of resource contention and load imbalance on storage subsystems also increases. Third, not every application developer or user has enough expertise in storage subsystems, a simple misuse or misconfiguration in their codes might lead to a system-wide I/O congestion.

As a result, developers or users often adopt conservative I/O strategies for their applications based on some empirical assumptions. These strategies are usually composed of some static policies and hard-coded into the code or job scripts. For instance, an application developer may notice that the timing on a particular I/O routine varies over an order of magnitude. In order to make sure that I/O time does not exceed 10% of the total execution time, the developer can make a pessimistic assumption on the I/O performance, and therefore let the code write data out less often than the more optimistic case. Even if the available I/O bandwidth is actually high and stable when the job is running, managing data movement based on these static policies would not be able to take full advantage of it. Since the static I/O policies are not feasible to either accurately characterize the complex storage subsystems or quickly capture the highly-variable I/O patterns in HPC environments, they are not able to help applications make intelligent I/O decisions. For example, when to read or write the data can applications achieve the best I/O performance? Where to place the data can applications make the data movement more efficient? How much data should be read or written if sacrificing some precision of the data for a better I/O performance is allowed? All these questions can be well answered only if we are able to accurately characterize and predict the runtime I/O performance on HPC systems.

I/O performance prediction is particularly challenging in HPC environments, since there might be hundreds of jobs competing for the I/O bandwidth simultaneously (with none or little performance isolation mechanism in place), and there is no way to know the I/O behaviors of these jobs in advance. To the best of our knowledge, this problem has not been well addressed by existing research efforts. Most of existing approaches were designed for system administrators rather than average users, and their models or algorithms were built upon some historical system traces that are either nonpublic or too

expensive to collect. Since the I/O performance characteristics of HPC systems change over time, the predictive model need to be updated with latest trace to achieve satisfying prediction accuracy. For instance, many existing studies, such as [5], [6], [7], build predictive models using some internal system traces. These traces are not always collected and might require administrator privileges to access, which makes continuously updating the models impossible.

In this paper, we design and develop a lightweight parallel test harness to periodically collect I/O performance and job status traces for applications running on production HPC systems. By analyzing the collected traces, we observe different I/O performance states on storage subsystems and come up with a machine learning-based approach to predict the transitions between those performance states during runtime. Specifically, we realize continuously measuring the performance of large I/O operations is not feasible as the measurement might cause too much overhead and interfere with other running jobs, thus we only periodically measure the end-to-end I/O latency by sending small write requests to storage servers of Lustre file system. However, the latency of small I/O operations does not always reflect the performance state of storage systems. To overcome this problem, we also periodically record the status of running jobs on the system by querying the job queue. We combine these two type of traces and build machine learning models to predict the time required to complete an I/O operation with certain size.

The contribution of this paper can be summarized as these aspects:

- We design and develop a lightweight parallel test harness to periodically collect I/O performance and job status traces on two Cray supercomputer systems (OLCF's Titan and NERSC's Cori).
- We present a detailed analysis of the traces we collected to understand the I/O performance characteristics of these two systems.
- We formulate the I/O performance prediction as a classification problem and exploit a range of machine learning techniques to train classification models which can be used to predict the runtime I/O performance.
- We demonstrate how to leverage the prediction results to improve the efficiency of I/O bandwidth utilization for applications running on Titan and Cori.

The remainder of this paper is organized as follows. In Section II, we present more details on the methodology for gathering our source data, while in Section III we describe and analyze the raw I/O performance and job status data. How to build, train and validate the machine learning models are introduced in Section IV. A case study that demonstrates how to use the machine learning-based prediction models to improve efficiency of I/O bandwidth utilization is presented in Section V.

II. Background

Our studies are conducted on two Cray supercomputer systems. One is OLCF's Titan supercomputer [8] and its center-wide file system Spider II, while the other is NERSC's newest supercomputer Cori [9] and its SCRATCH file system. Since these two supercomputers adopt different types of system architectures, their I/O performance characteristics are quite dissimilar from each other. By evaluating our I/O performance prediction approach on these two systems, we can have a better understanding of how generic our approach is. In order to facilitate the follow-up discussion, we provide a brief overview on both Titan and Cori.

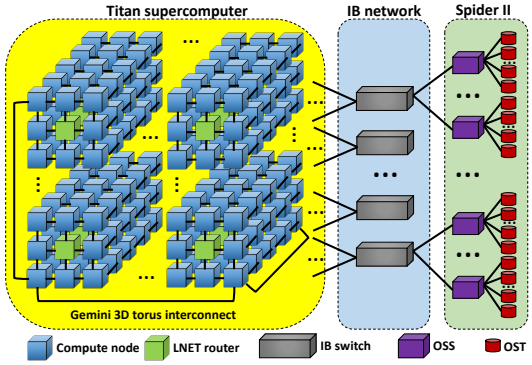


Fig. 1: Titan and its backend storage

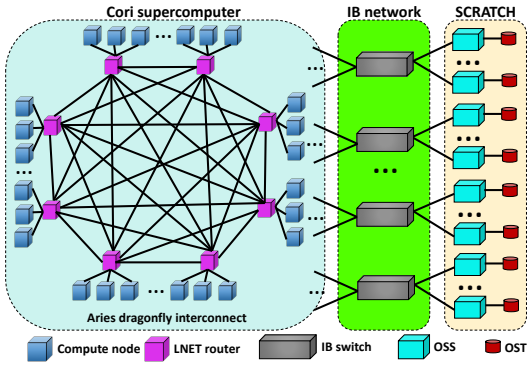


Fig. 2: Cori and its backend storage

Titan is a hybrid-architecture Cray XK7 system that features 18,688 compute nodes, and a total system memory of 710 TB. As shown in Figure 1, all XK7 nodes on Titan are part of the Gemini network in a 3D torus topology, where 440 of them are configured as Lustre I/O routers (LNET routers), which provide the connectivity to the backend storage system, which consists of 288 Lustre object storage servers (OSSs) and 2,016 Lustre object storage targets (OSTs), via InfiniBand FDR links.

Cori is a Cray XC40 system built with two different types of nodes, 2,388 Intel Xeon "Haswell" processor nodes and 9,688 Intel Xeon Phi "Knight's Landing" nodes. Cori adopts Cray's Aries dragonfly interconnect for inter-node communication. As shown in Figure 2, the compute side of Cori is also connected to its Lustre-based backend storage through LNET routers and InfiniBand network.

To summarize, production HPC systems like Titan and Cori usually have complex storage subsystems. Critical components on the I/O path, such as IB switches or OSSs, are shared by

many running jobs simultaneously. I/O contention and interference might occur on any of them at any time, which could lead to significant I/O performance variability and uncertainty. Therefore, predicting I/O performance on production HPC systems is extremely challenging.

III. Field Data Collection and Analysis

In order to build an accurate and practical I/O performance prediction model, we have developed a lightweight parallel test harness to periodically collect I/O performance and job status traces from the production HPC systems. In this section, we first introduce our test harness and data collection methodology on Titan and Cori, then provide a detailed analysis of the field data we collected.

A. Data Collection Methodology

Two principles are taken into account when we design our test harness and data collection methodology. First, although a few high-performance computing sites maintain internal I/O performance numbers, such as the throughput or IOPS of their I/O nodes, they usually do not share those numbers with application and middleware developers. Therefore, we focus on measuring and collecting I/O performance numbers at user space. Second, since the I/O performance characteristics on production HPC systems can change dramatically over time, in order to maintain the prediction accuracy, we need to collect the data and update our predictive model periodically. Our data collection is designed to be lightweight, meaning it only incurs negligible overhead so that the interference with other running applications is minimized.

In each run of our data collection routine, three types of data are measured and collected. First, we measure the end-to-end I/O latency which reflects the busyness of the storage subsystem. The default size of data transfers between the Object Storage Clients (OSCs) and OSSs on Lustre file system is usually set to 1MB. If an I/O request is larger than 1MB, Lustre will split it into multiple 1MB data chunks and send them to the OSSs through the RPC (Remote Procedure Call) protocol. In other words, the latency of I/O requests larger than 1MB can be estimated as the sum of multiple 1MB requests' latency. Therefore, we developed an MPI-based program which can periodically write 1MB data from a compute node to a specific OSS (using the *llapi* provided by Lustre file system) and measure the latency of each 1MB request. The purpose of periodically measuring the end-to-end latency is to capture the performance dynamics along each I/O path with low overhead (similar to sending probing packets to measure network delay).

Second, we snapshot and save the status of each running job by calling "qstat" or "sqs" command, depending on which job scheduler the system uses (Titan uses PBS job scheduler [10] while Slurm job scheduler [11] is used by Cori). The job status data we collected from the job scheduler includes total number of running jobs, number of compute nodes each running job occupies, amount of memory allocated to each running job, amount of wall-clock time allocated to each running job, etc. There are two purposes for collecting job status data: 1) The status of each running job might be correlated with current and future I/O performance of the storage subsystem. 2) It is easy and inexpensive for applications or I/O middleware to obtain.

Third, in order to train and validate our I/O performance prediction model, we also measure the performance of I/O requests with actual sizes as the ground-truth. We measure the performance of write requests with three different sizes (64MB, 256MB and 1GB). Figure 3 is an example of the collected data we saved in the trace file, which shows part of the job status and I/O performance traces collected in one repetition (Some of the sensitive information has been anonymized).

```
Timestamp:
  1520708288
All running jobs:
  Job Id: 3901500
    Job_Name = [redacted]_1050ps
    Job_Owner = [redacted]@titan-ext2.ccs.ornl.gov
    resources_used.mem = 8412kb
    resources_used.vmem = 168948kb
    resources_used.walltime = 03:55:35
    ...
  Job Id: 3903968
    Job_Name = [redacted]_prod
    Job_Owner = [redacted]@titan-batch5
    ...
Performance:
  OSS Id    1MB        64MB       256MB    1GB
  atlas1-0  0.00851s   0.11085s   0.5206s  1.83079s
  atlas1-1  ...
  ...
```

Fig. 3: An example of data collected in the trace file

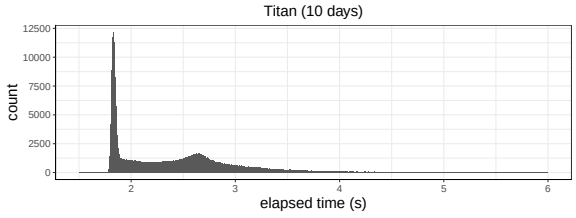
We implemented a job script for our data collection routine and submitted it to both Titan and Cori. Specifically, we reserved 288 compute nodes on Titan and 248 on Cori for our data collection jobs as there are 288 OSSs on Titan and 248 on Cori. To measure the end-to-end latency, the data collection job launched one MPI process per node and each process issued a 1MB write request to each OSS every 30 seconds. Only one of these processes is used to collect the job status data from the job scheduler every 5 minutes (the job status does not change as frequently as the end-to-end latency values). Since the scheduling policy of Titan and Cori do not allow jobs at this scale to run for long time, we have implemented a background process which monitors the job queue associated with our account and automatically resubmits the data collection job once the previous submission is completed. We ran our data collection job for 10 days on Titan and 2 days on Cori (We do not have enough hours left on our Cori account for us to run longer data collection job.).

In practice, once the I/O prediction model is trained, users do not need to launch a separate job to collect the data. Instead, the data collection code can share the compute node with their own applications and run as a background process on a single core of that computer node. Since the compute nodes on supercomputers often adopt many-core architectures, we do not expect our data collection approach will notably affect the performance of the original applications.

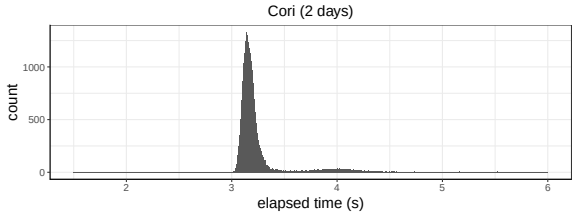
B. Analysis of the Field Data

The analysis of the data we collected is presented in this subsection.

1) I/O Performance Data: Let us have a look at the performance of writing 1GB data on Titan and Cori first. As shown in Figure 4, on both Titan and Cori, the distribution of time spent on writing 1GB data has very long tail. In fact, although most 1GB writes can be completed in 2 seconds on Titan and 3.5 seconds on Cori, a few of them might need 50 seconds or even more. This means the execution efficiency of applications running on these two systems might suffer from high I/O variability. For example, when a scientific simulation code running on Titan writes a checkpoint, there might be more than 10,000 MPI processes writing data to the storage system simultaneously. Some of them might complete the writing *much* faster than the others. However, those faster processes cannot restart the simulation until the slower ones finish writing their checkpoints, as all processes must be synchronized before starting a new simulation step.



(a) Histogram of time used to write 1GB data on Titan

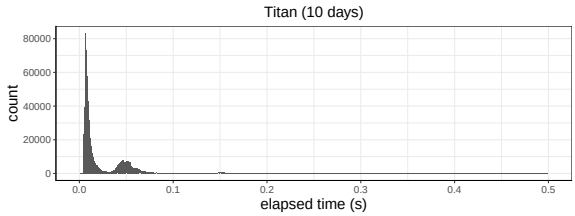


(b) Histogram of time used to write 1GB data on Cori

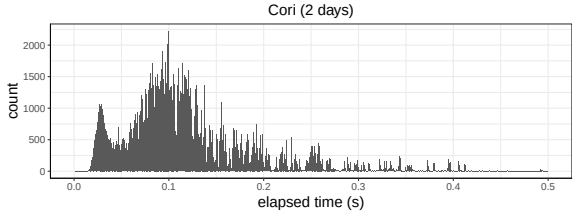
Fig. 4: Time used to write 1GB data on Titan and Cori

Another interesting observation is, on both Titan and Cori, the distribution of time spent on writing 1GB data shows some bimodal characteristics. For example, as shown in Figure 4(a), the histogram has two distinguishable peaks, one is at around 1.85 seconds and the other is at around 2.7 seconds. Similarly, in Figure 4(b), the distriubtion also shows two different modes, one is centered at around 3.2 seconds while the other is centered at around 4 seconds. This observation indicates that the degree of busyness of storage subsystems on production supercomputers can be characterized by two different performance states: idle and busy state. Besides performance of 1GB write requests, similar bimodal characteristics are also observed when writing data with other sizes (64MB and 256MB, for instance).

So why there are two different performance states? If we look at Figure 5, the distributions of end-to-end I/O latency on both Titan and Cori also show two different modes. As mentioned in previous section, Lustre splits all large I/O requests and transfers them through 1MB RPCs. The RPC traffic from different jobs are competing for the limited I/O bandwidth to the storage backend. When there is not much I/O traffic from other jobs, the storage subsystem is in idle state and the RPC traffic can be forwarded to the storage backend quickly. When the I/O bandwidth is saturated by RPC traffic, the storage subsystem is in busy state and the average latency of the RPC transfer increases significantly. Therefore, on Lustre file system, performance of writing large data has similar bimodal characteristic as that of writing 1MB data chunks.



(a) Histogram of end-to-end I/O latency on Titan



(b) Histogram of end-to-end I/O latency on Cori

Fig. 5: Latency of writing 1MB data to the storage backend on Titan and Cori

2) Job Status Data: TABLE I provides an overview of the job status data we collected from Titan and Cori. As shown in TABLE I, the average number of running jobs per day on Cori is higher than that on Titan. Moreover, it seems that most jobs running on Cori are small jobs. For example, more than 50% of the jobs running on Cori use less than 10 compute nodes, while on Titan, only 18% of the jobs use less than 10 compute nodes. Another interesting finding is that jobs running on Cori use more memory and have longer running time than those on Titan. All these differences between Titan and Cori's job status data suggest that scientific applications running on these two supercomputers might experience different I/O performance characteristics.

The I/O performance of each application often depends on other jobs running on the system since the I/O bandwidth is shared by all running jobs from all compute systems attached to the storage backend. However, in reality, finding the correlation between status of running jobs and I/O performance is difficult. In Section IV, we will demonstrate that machine learning techniques can help us bridge this gap.

		Titan (10 days)	Cori (2 days)
Avg number of jobs per day		820	1,340
Job size (node)	Min	1	1
	Median	64	8
	Max	17,920	5,468
Job memory usage (GB)	Min	0.001	1.0
	Median	3.27	360.0
	Max	3,135.9	492120.0
Job running time (hour)	Min	0.0014	0.05
	Median	1.58	1.95
	Max	28.0	48.0

TABLE I: Overview of collected job status data

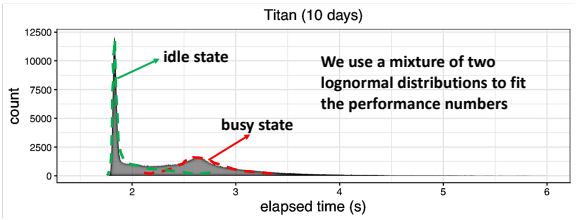


Fig. 6: A mixture of two log-normal distributions that fits performance numbers of 1GB writes

IV. I/O Performance Prediction

In this section, we demonstrate how to leverage the traces we collected and machine learning techniques to predict the transitions between different I/O performance states.

A. Problem Formulation

We formulate the I/O performance prediction on HPC system as a classification problem. As two different performance states are observed in the I/O performance data, we can use the historical data to parameterize the distribution of I/O time in each performance state. Then we train a classifier that can predict which performance state an I/O operation will encounter given the job status as well as the end-to-end I/O latency. Although we cannot obtain the exact I/O performance number from the classifier, knowing the performance state of storage subsystems and distribution of I/O time in each performance state beforehand is still useful for applications or I/O middleware to intelligently schedule their I/O operations. For example, if the classifier shows the I/O paths to some of the OSSs on Titan will be in busy state, we can write more data to those idle OSSs instead of these busy ones. More details about how we use a variety of machine learning-based approaches to train classifiers and how we validate the prediction results are presented in following subsections.

B. Data Preprocessing

Before building the classification model, we need to preprocess the data we collected and convert the raw data into response and explanatory variables which will be used to train and test the classifiers.

1) The Response Variable: The response variable is the the variable we try to classify or predict, which in our case is the performance state of I/O paths to each OSSs on Titan and Cori.

First, we use a mixture of two log-normal distributions to fit the performance numbers, which is parameterized by maximum likelihood estimation (MLE) and expectation maximization (EM) algorithm [12]. We also tried other long-tail distributions, but the log-normal distribution always fits the data best. As an example shown in Figure 6, the mixture of two log-normal distributions can capture the bimodal characteristic observed in performance numbers of 1GB writes fairly well.

Second, for each performance number we collected, we use the mixture model to compute the probability of being in each state. Since the two performance states overlap each other (short write time might belong to busy state while long write time might belong to idle state), we do not simply assign the state with higher probability as a label to the performance number, instead, we randomly generate the label for each performance number based on the calculated probability. Now the performance number becomes a binary variable which is either 0 (idle state) or 1 (busy state). This binary variable is the response variable our classification model needs to predict.

#ID	Feature
1	Total number of running jobs
2	Number of running jobs from batch queue
3	Number of running jobs from debug queue
4	Number of exta-large running jobs (nodes $\geq$ 11,250)
5	Number of large running jobs (3,750 $\leq$ nodes $<$ 11,250)
6	Number of medium running jobs (313 $\leq$ nodes $<$ 3,750)
7	Number of small running jobs (126 $\leq$ nodes $<$ 313)
8	Number of extra-small running jobs (0 $<$ nodes $<$ 126)
9	Number of new running jobs
10	Number of middle-aged running jobs
11	Number of old running jobs
12	Amount of virtual memory used by all running jobs
13	Amount of physical memory used by all running jobs
14	Current 1MB write latency
15	Average 1MB write latency in a 10-minute time window
16	Standard deviation of 1MB write latency in a 10-minute time window

TABLE II: Features of classification model built for predicting I/O performance of an OSS on Titan

2) The Explanatory Variables: In machine learning, the explanatory variable are also termed *features*, which are used to predict or explain variability in the response variable. All features we used to build the classification model for predicting I/O performance on Titan are listed in TABLE II.

In TABLE II, feature 1 to 13 are extracted from the job status data. Specifically, the definition of extra-large to extra-small job used by feature 4 to 8 are based on Titan's job priority policy, which can be found on OLCF's website [13]. For feature 9 to 11, new running jobs are those that have used less than $1/3$ of their allocated hours, middle-aged running jobs are those that have used more than $1/3$ but less than $2/3$ of their allocated hours, and old running jobs are those that have used more than $2/3$ of their allocated hours. Feature 14 to 16 are extracted from the end-to-end 1MB write latency data. Particularly, we maintained a 10-minute shifting time-window for the latency data and we calculate the average and standard deviation of end-to-end latency within that time-window. The features we selected for predicting I/O performance on Cori are similar to those listed in TABLE II.

C. Training the Classifiers

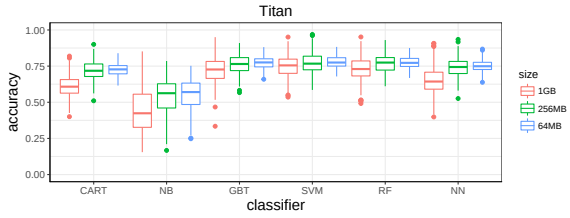
We apply six commonly used machine learning approaches to our dataset. They are classification and regression trees (CART) [14], naive bayes (NB) [15], gradient boosting (GBT) [16], support vector machines (SVM) [17], random forests (RF) [18] and neural networks (NN) [19]. All these methods are implemented based on scikit-learn library [20] in Python. For the first three methods, we use the default parameter settings provided by scikit-learn. For support vector machines, we choose the radial basis function (RBF) kernel as it gives the best *bias variance* trade-off. For random forests, we set the number of trees as 50 in our evaluation. We also tested it using up to 100 trees, but did not observe significant improvement in prediction accuracy. For neural networks, we adopt the multi-layer perceptron structure and build a network with 3 layers and 100 neurons in total. Although neural networks with complex structure and more neurons might provide better prediction accuracy, training such a model is apparently much more expensive and time-consuming, which is not very feasible in our scenario as the model needs to be updated periodically.

For each OSS, we train a classifier using the historical job status as well as the end-to-end latency measured by periodically sending 1MB data to that OSS. Specifically, for Titan, we use the data collected in the first 8 days to train a classifier for each OSS, while for Cori, we use the data of the first day as the training data. In order to avoid the overfitting/overtraining, we split the training data into 8 subsets and run 8-fold cross-validation on them to make sure the parameter setting that achieves fairly good performance on all validation splits is selected. The training usually finishes in less than 30 seconds.

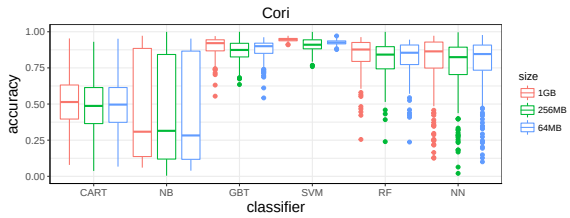
D. Prediction Results

We apply the classification algorithms to the data collected from each OSS on Titan and evaluate the prediction accuracy of these algorithms on all OSSs. As shown in Figure 11(a), each boxplot illustrates the distribution of prediction accuracy when certain classification algorithm is applied to the data collected from 288 OSSs on Titan. From this figure, we can see that SVM classifier achieves the best prediction accuracy. When SVM is used, the accuracy of I/O performance state prediction for most of the OSSs on Titan is more than 70%. Besides SVM, random forests classifier also performs fairly well. Another interesting finding is the prediction accuracy for smaller write sizes (64MB and 256MB) is more stable among different OSSs. This is because small I/O requests forward less RPC calls to the OSSs on Lustre file system, thus are less likely to be affected by I/O traffic from other running jobs.

Similarly, SVM also achieves the best prediction accuracy on data collected from Cori. As shown in Figure 7(b), when SVM is used, the accuracy of I/O performance state prediction for most of the OSSs on Cori is more than 90%. Gradient boosting classifier also performs well on Cori dataset, and the average prediction accuracy it has achieved among all OSSs is around 80%.



(a) Boxplot of prediction accuracy (Titan data is used)



(b) Boxplot of prediction accuracy (Cori data is used)

Fig. 7: Prediction accuracy of applying different classification algorithms to I/O performance data collected from all OSSs on Titan and Cori

Accuracy score is just one metric to evaluate the classification or prediction results. Another commonly used metric is the receiver operating characteristic curve (ROC curve), which is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The definition of true positive rate and false positive rate are given as follows.

$$\text{TPR} = \frac{\text{\# of true positives}}{\text{\# of true positives} + \text{\# of false negatives}} \quad (1)$$

$$\text{FPR} = \frac{\text{\# of false positives}}{\text{\# of false positives} + \text{\# of true negatives}} \quad (2)$$

Since we train a classification model for each of the OSSs on Titan and Cori, we can plot an ROC curve for each of them. Here we randomly pick three OSSs from Titan and another three from Cori, then we plot the ROC curve for each classifier

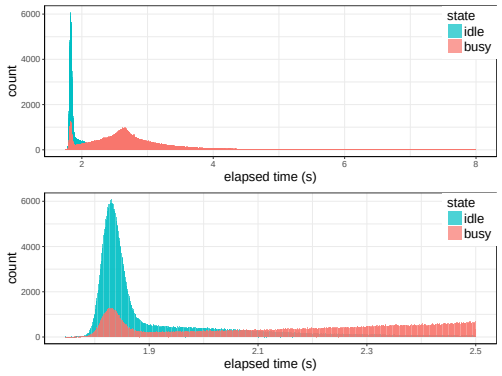


Fig. 8: Histogram of Titan's 1GB write performance numbers in the testing dataset that are predicted as being in idle and busy state. The bottom figure shows a zoom-in view where the write time is within [0, 2.5] on the x-axis. The prediction results are given by an SVM classifier.

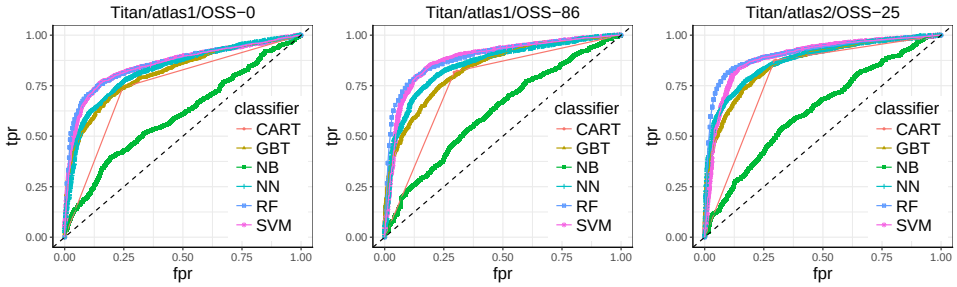


Fig. 9: ROC curves of using different classifiers to predict three Titan OSSs' I/O performance

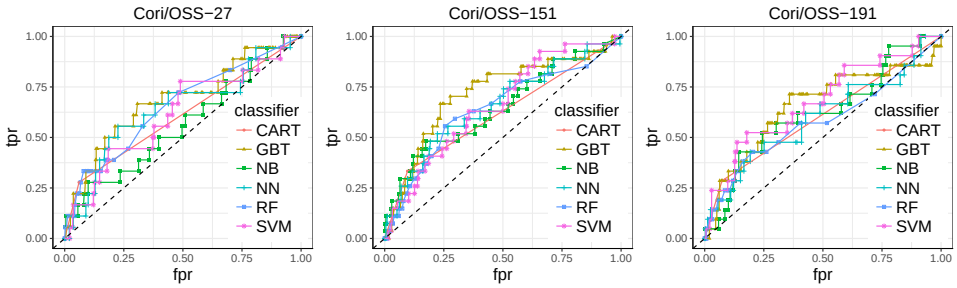


Fig. 10: ROC curves of using different classifiers to predict three Cori OSSs' I/O performance

after we apply them to predicting the 1GB write performance of these OSSs. All the ROC curves can be found in Figure 9 and 10.

After generating the ROC curve, area under the ROC curve (AUC) is often used to evaluate how good a classifier is. As shown in Figure 9, for the three OSSs on Titan, SVM and random forest classifier outperform other classifiers since the AUC of these two classifiers are greater than that of the others. On Cori, as shown in Figure 10, the difference between AUC of any two classifiers is not very significant, though the gradient boosting classifer is slightly better than the others.

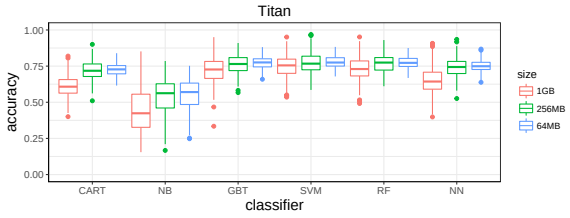
Based on the accuracy score and ROC curve, we know how accurate a classifier can be when it is used to predict the I/O performance state. Now our question is: in the testing dataset, are all the data points that have been predicted as being in the idle state have short I/O time? Or how many data points that have long I/O time are incorrectly predicted as being in the idle state? Here we separately plot the histogram of Titan's 1GB write performance numbers (from all 288 OSSs) that are predicted as being in idle and busy state in Figure 8. From the top figure we can see that almost all of the data points that have long write time are correctly predicted as being in busy state. Also, from the bottom figure we can see that most of the data points that have been predicted as being in idle state have very short write time (less than 2.1 seconds). Although, a small portion of the data points that have short write time are predicted as being in busy state, this kind of misprediction usually is not very harmful. For example, based on the prediction result that I/O performance will be in busy state, applications or I/O middleware might adopt some unambitious I/O strategies (such as skipping the current I/O step, writing less data, etc.). In that case, the I/O bandwidth might not be fully utilized, but the applications' execution efficiency will not be negatively affected.

To make it feasible for scientific applications to utilize the prediction results to dynamically schedule their I/O operations during runtime, the inference time of the trained classification models needs to be short. For all the six classification models we evaluated, the inference time can almost be ignored (less than 0.01 seconds).

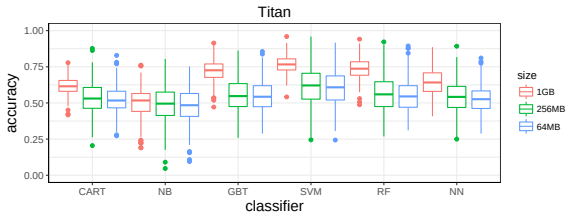
E. Robustness of Predictions

In practice, collecting a large amount of data from production HPC systems for model training might be difficult or even infeasible due to limited resources or rigid system policies. Therefore, we need to evaluate the robustness of our classification-based I/O prediction models when only limited data is available for training.

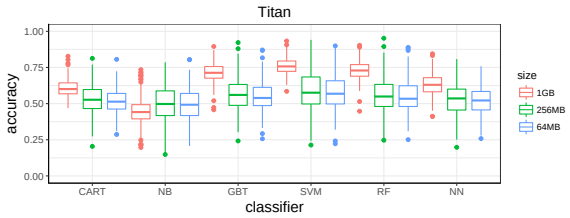
The basic idea of our evaluation is we use different percentages of the data to train our predictive model and see if the prediction accuracy changes significantly. The data collected from Titan is used for this evaluation. First, in Figure 11(a), we show the prediction accuracy our model achieved when data collected during the first 8 days is used for training and that collected during the last 2 days is used for testing. Then in Figure 11(b), only data collected during the first 6 days is used for training. Finally, in Figure 11(c), only the data of the first 4 days is used to train our model. From these three figures, we can see that when we reduce the size of the training data, the prediction accuracy of 1GB writes' performance does not decrease significantly. Particularly, for SVM classifier, even when we only use the data of the first 4 days to train it, it can still achieve an average prediction accuracy around 75% on data of the last 6 days.



(a) Boxplot of prediction accuracy (8 days for training and 2 days for testing)



(b) Boxplot of prediction accuracy (6 days for training and 4 days for testing)



(c) Boxplot of prediction accuracy (4 days training and 6 days for testing)

Fig. 11: Evaluating the robustness of our classification-based I/O prediction model

However, as we can see in 11(b) and 11(c), the prediction accuracy on performance of 64MB and 256MB writes reduce if size of the training data decreases. There are two possible causes of this accuracy degradation: 1) We adopt the default parameter settings for most of the classification models which might be sensitive to the size of the training data. 2) Some features we selected for training the classifiers, especially those collected from the job scheduler, might not have strong correlation with the performance of small writes. We believe further refinement of parameter settings and learning features will improve the robustness of our model when it is used to predict the performance of small I/O requests.

V. Case Study: Tuning Data Placement on Lustre Based on Prediction Results

In this section, we present a case study to demonstrate how to leverage the I/O performance prediction to tune the data placement on Lustre file system to improve the aggregate I/O throughput.

A. Data Placement by Proactive Performance Prediction

As we introduced in Section II, both Titan and Cori use Lustre parallel file system as their storage subsystem. In order to balance the I/O traffic among the OSSs and maximize the bandwidth usage, Lustre splits the I/O requests from applications and send them to multiple OSSs in parallel, which is also called "striping". However, in reality, such "striping" strategy does not always guarantee good I/O performance, since the parallel file system is shared by I/O traffic from many running jobs and the I/O contention and interference might happen anytime on any of those OSSs. If one of the OSSs is congested due to the I/O contention, it can lead to I/O performance degradation on a subset of MPI processes launched by those large-scale scientific applications. As a result, the aggregate I/O throughput of these applications will decrease dramatically.

Our idea to improve the aggregate I/O throughput for these large-scale applications on Lustre file system is straightforward: Based on the I/O performance prediction, we know which OSSs will be in busy state or idle state. Then we intentionally let the application shift some of its I/O traffic to those idle OSSs. Here is the model we used to decide how much data should be written to each OSS. Let us assume at any given time, n_{idle} of the OSSs are predicted as being in idle state and n_{busy} of them are predicted as being in busy state. If the total amount of data written by the application is SGB, we need to calculate how much of the data should be sent to those idle OSSs, and the rest will be sent to those busy ones. Let us assume αS data will be sent to idle OSSs, where $0 < \alpha \leq 1$. As shown in Figure 6, we have used a mixture of two lognormal distributions to fit the time on 1GB writes and estimated the parameters of these two lognormal distributions. If the mean value of these two distributions are denoted as T_{idle} and T_{busy}, then on average, writing $\alpha S/n_{idle}$GB data to an idle OSS takes $T_{idle}\alpha S/n_{idle}$ seconds. Similarly, writing $(1-\alpha)S/n_{busy}$GB to a busy OSS takes $T_{busy}(1-\alpha)S/n_{busy}$ seconds. Basically, we want the writes to idle and busy OSSs to be finished at about the same time. Therefore, we have $T_{idle}\alpha S/n_{idle} = T_{busy}(1-\alpha)S/n_{busy}$, which gives us:

$$\alpha = \frac{n_{idle}T_{busy}}{n_{idle}T_{busy} + n_{busy}T_{idle}} \tag{3}$$

B. Evaluation

We evaluate our I/O prediction-guided data placement tuning through a trace-driven simulation. In our simulation, we assume that there is an application on Titan which is writing 288GB data to Lustre at every I/O step. First, we randomly selected 100 data points (each data point includes I/O performance of all 288 OSSs measured at a specific time) from our testing dataset. Then for each data point, we apply the prediction results of performance state of each OSS to equation 3 to decide the amount of data that should be written to idle and busy OSSs. Finally, we calculate the aggregate I/O throughput using the actual write time to each OSS. We also simulate the case when no data placement tuning is used, meaning the application equally writes 1GB data to each OSS, as the evaluation baseline.

The aggregate I/O throughput of these two evaluation cases given by one run of the simulation are shown in Figure 12. As we can see, most of the time, our prediction-based data

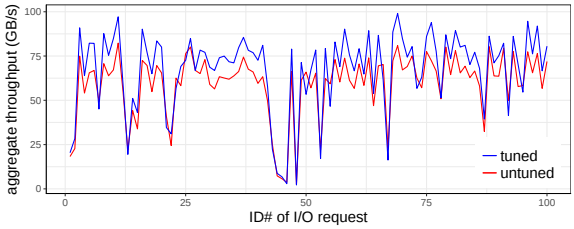


Fig. 12: Aggregate throughput with/without the prediction-based I/O tuning

placement tuning improves the aggregate I/O throughput. We also run the simulation 100 times, the average improvement of the aggregate I/O throughput by leveraging the I/O prediction-guided data placement is more than 9%.

VI. Related Work

In this section, we present a brief review of existing literature that focus on I/O performance analysis, modeling and prediction in HPC environments.

First, many existing studies have conducted a variety of I/O performance measurements and analysis on different HPC systems. Some of these studies concentrate on how hardwares or system architectures affect the I/O performance, while they do not take applications' I/O behavior into account. For example, the I/O performance of several DoE computing facilities, such as Intrepid, Edison and Titan, have been extensively measured and analyzed [21], [22], [23]. There are also some research efforts that only study the I/O performance of one or a few specific applications by analyzing the runtime traces collected when these applications were running on the HPC systems. For example, [24] presented an analysis of the application-level I/O traces collected from Intrepid, during a two-month time period, using Darshan I/O tracing tool [25]. Also, Darshan traces collected from three different supercomputing facilities during a much longer time period (6 years) were analyzed in [3]. Although these performance analysis provide some valuable insight into the I/O performance on HPC systems, most of them only focus on a specific system or application, which makes it difficult to leverage their results in other scenarios.

Second, since the I/O subsystems in HPC environments are often shared by hundreds of applications simultaneously while the I/O bandwidth is usually limited, I/O congestion and interference are highly likely to happen, which could lead to significant I/O performance variability and uncertainty. In order to have some predictability of runtime I/O performance, some other existing work try to model and characterize the I/O performance variability in production HPC systems. For example, both [1] and [26] presented some detailed analysis and discussed why I/O performance in HPC is highly variable. A grammar-based appraoch is proposed in [27], which can predict spatial and temporal I/O patterns. In [28], heavy-tailed distributions are used to fit the system response time of parallel I/O. Based on the observed properties of I/O traces collected on Titan, [4] built a hidden Markov model to characterize the end-to-end I/O performance on Titan. All these models are usually simplified based on some assumptions which might not be true in reality. For example, our approach is inspired by the study presented in [4]. However, [4] makes a strong assumption that the transition between I/O performance states on Titan follows a Markov process, which might not always be true. Our classification-based approach is more general and can be used on different HPC systems.

Third, machine learning techniques have also been applied to performance prediction and optimization in HPC environments. For example, in [29], a decision tree based model is built to predict the I/O performance on HPC systems. Similar work is done in [5], where the decision tree model is trained using long-term I/O traces collected at Lawrence Livermore National Laboratory. In [7], job scripts of 300,000 jobs from a HPC machine is used to train a deep neural network model to predict I/O performance. [6] leverages deep neural network and reinforcement learning to find the best tuning option for Lustre file system. Although these studies have shown some promising results, few of them takes the bursty nature of HPC I/O into account and can periodically update their predictive models to adapt to the runtime I/O performance variabilities, which is a norm rather than exception on production HPC systems. Therefore, our work is an important complement to existing work in this field.

VII. Conclusions and Future directions

Managing the I/O operations for large-scale scientific applications running on production HPC systems is challenging. As presented by a few existing studies as well as this paper, I/O bandwidth contention can make the runtime I/O performance highly variable which significantly increases the uncertainty of scientific applications' execution time. In order to mitigate the I/O contention and fully utilzie the available bandwidth, scientific applications or I/O middleware need to accurately characterize and predict the runtime I/O performance to schedule their I/O operations more efficiently and intelligently.

In this paper, we have explored the possibilities of leveraging machine learning techniques to predict the runtime I/O performance on production HPC systems. As demonstrated by our evaluation results, even without well crafted parameter tuning, a few commonly used machine learning models can achieve acceptable prediction accuracy. Particularly, by using SVM and random forest model, the runtime I/O performance on both Titan and Cori can be predicted with an average accuracy that is more than 70%. Moreover, we create a trace-driven simulation to demonstrate how the I/O performance prediction results can be leveraged to adaptively adjust the data placement to improve the aggregate I/O throughput on Lustre file system. The simulation results show that, on average the improvement of the aggregate I/O throughput is more than 9%, if the machine learning-based I/O performance prediction is used.

As part of our future work, we are going to study how to navigate the hyperparameter space to quickly identify the optimal parameter settings for these machine learning models to further improve the prediction accuracy. We are also going to integrate the I/O prediction models into an open source parallel

I/O framework, coupled with an approach for capturing and managing the system information that is required by model training and updating.

REFERENCES

[1] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf, "Managing Variability in the IO Performance of Petascale Storage Systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10, 2010, pp. 1–12.

[2] Q. Liu, N. Podhorszki, J. Logan, and S. Klasky, "Runtime I/O Re-Routing+ Throttling on HPC Storage," in *HotStorage*, 2013.

[3] H. Luu, M. Winslett, W. Gropp, R. Ross, P. Carns, K. Harms, M. Prabhat, S. Byna, and Y. Yao, "A Multiplatform Study of I/O Behavior on Petascale Supercomputers," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '15, 2015, pp. 33–44.

[4] L. Wan, M. Wolf, F. Wang, J. Youl Choi, G. Ostrouchov, and S. Klasky, "Analysis and Modeling of the End-to-End I/O Performance in OLCFs Titan Supercomputer," in *IEEE 19th International Conference on High Performance Computing and Communications*, ser. HPCC '17, 2017, pp. 1–9.

[5] R. McKenna, S. Herbein, A. Moody, T. Gamblin, and M. Taufer, "Machine Learning Predictions of Runtime and IO Traffic on High-end Clusters," in *2016 IEEE International Conference on Cluster Computing*, ser. CLUSTER '16, 2016, pp. 255–258.

[6] Y. Li, K. Chang, O. Bel, E. L. Miller, and D. D. E. Long, "CAPES: Unsupervised Storage Performance Tuning Using Neural Network-Based Deep Reinforcement Lear," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17, 2017.

[7] M. R. Wyatt, II, S. Herbein, T. Gamblin, A. Moody, D. H. Ahn, and M. Taufer, "PRIONN: Predicting Runtime and IO Using Neural Networks," in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP '18, 2018, pp. 46:1–46:12.

[8] "Titan: Advancing the Era of Accelerated Computing," https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan/, 2018.

[9] "Cori Supercomputer," www.nersc.gov/users/computational-systems/cori/, 2018.

[10] "Industry-Leading Workload Manager and Job Scheduler for High-Performance Computing," https://www.pbspro.org/, 2018.

[11] "Slurm Workload Manager," https://slurm.schedmd.com/, 2018.

[12] M. R. Gupta and Y. Chen, "Theory and Use of the EM Algorithm," *Foundations and Trends in Signal Processing*, vol. 4, no. 3, pp. 223–296, 2011.

[13] "OLCF POLICY GUIDE," https://www.olcf.ornl.gov/for-users/olcf-policy-guide/, 2018.

[14] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

[15] "Naive Bayes," http://scikit-learn.org/stable/modules/naive_bayes.html, 2018.

[16] "Gradient Boosting," https://en.wikipedia.org/wiki/Gradient_boosting, 2018.

[17] "Support Vector Machine," https://en.wikipedia.org/wiki/Support_vector_machine, 2018.

[18] "Random Forest," https://en.wikipedia.org/wiki/Random_forest, 2018.

[19] "Artificial Neural Networks," https://en.wikipedia.org/wiki/Artificial_neural_network, 2018.

[20] "Scikit-learn: Machine Learning in Python," http://scikit-learn.org/stable/, 2018.

[21] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock, "I/O Performance Challenges at Leadership Scale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '09, 2009, pp. 40:1–40:12.

[22] Z. Zhao, D. Petesch, D. Knaak, and T. Declerck, "I/O Performance on Cray XC30," in *Proceedings of the Cray User Group Conference*, ser. CUG '14, 2014.

[23] L. Wan, M. Wolf, F. Wang, J. Youl Choi, G. Ostrouchov, and S. Klasky, "Comprehensive Measurement and Analysis of the User-Perceived I/O Performance in a Production Leadership-Class Storage System," in *IEEE 37th International Conference on Distributed Computing Systems*, ser. ICDCS '17, 2017, pp. 1022–1031.

[24] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and Improving Computational Science Storage Access through Continuous Characterization," in *IEEE 27th Symposium on Mass Storage Systems and Technologies*, ser. MSST '11, 2011, pp. 1–14.

[25] P. H. Carns, R. Latham, R. B. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 Characterization of Petascale I/O Workloads," in *Proceedings of the First Workshop on Interfaces and Abstractions for Scientific Data Storage*, 2009.

[26] B. Xie, J. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, and N. Podhorszki, "Characterizing Output Bottlenecks in a Supercomputer," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '12, 2012, pp. 8:1–8:11.

[27] M. Dorier, S. Ibrahim, G. Antoniu, and R. Ross, "Omnisc'IO: A Grammar-based Approach to Spatial and Temporal I/O Patterns Prediction," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14, 2014, pp. 623–634.

[28] B. Dong, S. Byna, and K. Wu, "Heavy-tailed Distribution of Parallel I/O System Response Time," in *Proceedings of the 10th Parallel Data Storage Workshop*, ser. PDSW '15, 2015, pp. 37–42.

[29] J. Kunkel, M. Zimmer, and E. Betke, "Predicting Performance of Non-contiguous I/O with Machine Learning," in *30th ISC High Performance Conference*, 2015, pp. 257–273.