SAND2021-2465C

Interpreting Write Performance of Supercomputer I/O Systems with Regression Models

Bing Xie* Zilong Tan† Philip Carns‡ Jeff Chase§ Kevin Harms‡ Jay Lofstead¶
Sarp Oral* Sudharshan S. Vazhkudai‖ Feiyi Wang*
* Oak Ridge National Laboratory
† Carnegie Mellon University
‡ Argonne National Laboratory
§ Duke University
¶ Sandia National Laboratories

***Abstract*—This work seeks to advance the state of the art in HPC I/O performance analysis and interpretation. In particular, we demonstrate effective techniques to: (1) model output performance in the presence of I/O interference from production loads; (2) build features from write patterns and key parameters of the system architecture and configurations; (3) employ suitable machine learning algorithms to improve model accuracy. We train models with five popular regression algorithms and conduct experiments on two distinct production HPC platforms. We find that the lasso and random forest models predict output performance with high accuracy on both of the target systems. We also explore use of the models to guide adaptation in I/O middleware systems, and show potential for improvements of at least 15% from model-guided adaptation on 70% of samples, and improvements up to 10× on some samples for both of the target systems.**

***Index Terms*—high performance computing, I/O performance, machine learning**

I. Introduction

In HPC there is a need for tools to predict I/O performance accurately and interpret factors that influence performance. For supercomputer facilities, more predictable I/O performance enables more precise core-time allocations and more efficient system utilization. For domain scientists with limited budgets for compute cycles charged on application runtimes [1], accurate prediction offers the potential to reduce I/O-induced idle time in their runs (§II-A1, §IV-D).

It is a challenge to predict write performance accurately on production supercomputers. One major obstacle is *performance variability* due to resource contention. Supercomputer I/O systems are shared by applications from different science domains and with varying I/O outputs. The shared systems therefore can deliver write bandwidths with large variations. To illustrate this point, consider our results in Figure 1, showing CDFs of performance variations across identical runs at different times. Cetus [2], at the Argonne Leadership Computing Facility (ALCF), is relatively stable. Titan [3] and Summit [4] at the Oak Ridge Leadership Computing Facility (OLCF) have progressively worse variability. Moreover, this high variability exists widely on other supercomputer I/O systems [5], [6].

‖ The author conducted this research when he was with Oak Ridge National Laboratory.

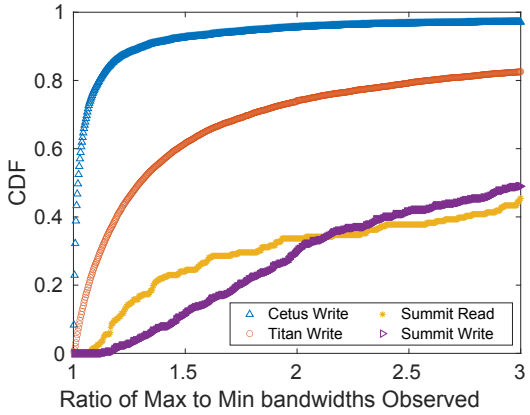


Fig. 1: CDFs of I/O performance variations on Cetus, Titan and Summit. The x-axis represents the measures ($\frac{max}{min}$) of the I/O bandwidths, each point represents the ratio of the max to the min bandwidths of the identical IOR executions (§III-D).

The other major obstacle is the limited user-level visibility into supercomputer I/O systems, which we refer to as the *black-box issue*. In most cases, end users and I/O libraries treat these systems as black boxes with minimal system-level insight. There are user-level tools providing limited information on application I/O behaviors such as Darshan (§II-A2). Unfortunately, these tools leave much of the system-side behaviors unexplored and insufficient for performance prediction. Thus, for end users it is hard to understand the write behaviors and the associated underlying root causes.

Despite the forementioned two challenges, the target applications, machines and filesystems show the characteristics that allow us to build learning models. Specifically, we find that a large group of scientific applications generate writes with regular and predictable patterns; the mean performance is effective to address their write behaviors (§II-A). On the system side, although end-user applications by themselves do not have sufficient I/O performance data collected, each I/O write cycle can be considered as a multi-stage path (see Figure 2) from the I/O issued by compute processes on some compute nodes to the data committed to the persistent storage devices. For each such stage, I/O performance data can be

derived from write patterns coupled with system design and configuration (§III).

These findings motivate us to take a regression approach: 1) To address the high variability, we model the mean I/O write performance and introduce a convergence-guaranteed sampling method to generate training data (§III-D); 2) To address the black-box issue, we consider the target systems as multi-stage write paths, and treat the aggregate load, load skew and resources in use on individual stages as I/O performance-related parameters. We build model features on performance-related parameters, and derive them from write patterns and system design and configuration (§III-A). 3) To analyze I/O interference in production environment, we address the interference as part of the model features in the target systems. 4) We tried 5 popular learning models (linear, lasso, ridge, decision tree and random forest) and find that the chosen lasso and random forest models are highly accurate.

We apply this approach on two production supercomputers, Cetus and Titan, deployed with GPFS and Lustre filesystems, respectively. In particular, we conclude that:

1. Our approach identifies highly accurate models. The chosen lasso models can attain $\leq 30\%$ relative error for up to 98.31% (Cetus) and 100% (Titan) of the samples on the target systems.

2. Our approach locates the most relevant features. Based on our analysis, the write behaviors of GPFS systems are dominated by the metadata load and load skew within supercomputers; the write behaviors of Lustre systems are heavily determined by the aggregate load, load skew and resources in use within supercomputers.

3. Our models are useful to improve write performance. We explore the potential of using the chosen lasso models to guide write adaptations by I/O middleware used in supercomputers. The results show that there is significant potential to achieve substantial improvements in write performance on the target systems automatically.

II. Background

A. Properties of Target Scientific Codes

1) Write Behaviors: On HPC platforms, a large group of applications are numerical simulations and analyses of physical phenomena. These applications are submitted as *jobs* (runs), execute iterative computations and generate output data periodically.

For typical applications, a run solves a fixed-space problem such as a magnetically confined torus (XGC [7]), a multi-dimensional mesh (Astrophysics) or a 3D grid (S3D [8]). The run has a job description file specifying the numbers of compute nodes, cores, and problem settings. The problem space is partitioned into a group of equal-sized subspaces; each process runs on a different core[1] for a different subspace; all processes execute the same numerical *solver* over a sequence of iterations and synchronize after each iteration.

A run produces output data according to one or more write patterns. Each pattern has a number of synchronous output bursts originating from a set of compute nodes/cores. It repeats on a fixed interval (*write frequency*) with each burst recording the states (numerical values) of the same variables. When writing bursts, the entire execution stalls until the last byte of the data is acknowledged by the filesystem I/O servers. Specifically, each core produces the same-size bursts across iteration-intervals[2] and the load is balanced among the engaged cores. To summarize, these applications exhibit three major properties:

Write patterns are generally fixed and predictable. The number of bursts (determined by the number of I/O write issuing cores), burst size (determined by the variables/spaces recorded per burst), and write frequency, are all preconfigured as problem settings.

Job execution time is predictable. The total execution time is the sum of the times spent on computation and I/O writes. The computation time can be estimated based on the problem and its resource settings [10]. With an accurate prediction of write times, a run's execution time can be predicted.

Write cost is tunable. Users may want to control write cost. For example, they may want to limit the checkpointing cost to 10% of job execution times. With the time estimates on computation and writes, users can control the checkpointing cost by choosing its write frequency appropriately.

Scientific codes also produce data using different mechanisms such as write-sharing, where processes write-share data to a single file [6], or dynamic writes, such as AMR codes where write load may be imbalanced among processes; this imbalance may vary across operations.

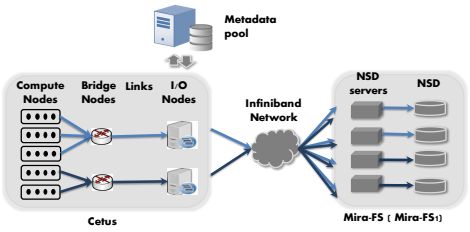
2) Write Patterns in Production Load: We analyze Darshan [11] logs collected from the machines at ALCF from January 2017 to August 2018. Darshan characterizes the I/O behaviors of scientific codes at the application level. It monitors job executions and collects I/O statistics. In particular, it records the metadata of I/O operations starting from when a job calls *MPI_Init* and stopping when *MPI_Finalize* is executed. Over the course of a year, Darshan has monitored/recorded the application I/O profiles consuming $\sim$30% of compute-core hours on ALCF systems.

The collected Darshan data comprises of 514,643 entries. Each entry summarizes the I/O behaviors of a job, such as the number of participating processes and burst-size histograms. Specifically, Darshan reports the histograms of conventional burst-size ranges. For example, for a process in an entry, Darshan records "CP_SIZE_WRITE_10M_100M 17"meaning that for the burst-size ranges 10M—100M the process writes 17 times. Darshan's histogram summary is useful to understand the repetitions of write patterns.

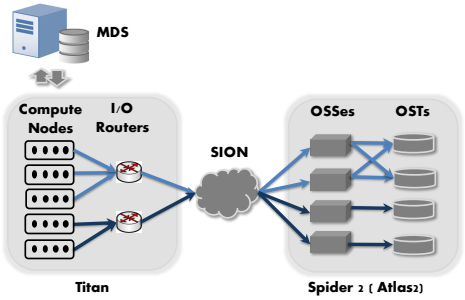
In summary, the recorded ALCF jobs execute on the scales of 1—1,048,576 processes; spend 0.01—23.925 compute-core

[1]Blue Gene/Q machines [9] utilize 4-way hyperthreading and support 4 concurrent processes per core; but many users still choose to run each process on a different core because of the memory limitation (e.g., 16GB memory per node on Cetus).

[2]For some codes (e.g., XGC), data imbalance and variation may occur across processes and operations, but in most cases it is ignorable [10].[2]



(a) Cetus and Mira-FS1



(b) Titan and Atlas2

Fig. 2: Architectures of the target I/O systems. Cetus and Titan take different strategies for relaying filesystem operations. For each system, one mapping policy is employed to connect compute nodes to bridge nodes (Cetus) and to I/O routers (Titan) (discussed in §II-B1 and §II-B2), respectively.

hours for execution; produce Byte—Gigabyte bursts. Across burst-size ranges, the write repetition per burst-size range is 3, 9, 66 times for quantile 0.3, 0.5 and 0.7, respectively.

Observation ①. The analysis suggests that scientific writes span over wide ranges for both scaling in terms of the number of processes and the burst size. It motivates us to build datasets for both representativeness and randomness across write scales, burst sizes, and repetitions (§III-D).

B. Supercomputers and Their Filesystems

1) GPFS on Cetus and Mira-FS1: Cetus is an IBM Blue Gene/Q machine, hosted at ALCF and connected to Mira-FS: a GPFS filesystem serving Cetus and many other supercomputers at ALCF. Figure 2a presents Cetus and Mira-FS.

On Cetus, 4,096 compute nodes are connected by a 5-D torus interconnect with 16 CPUs per node. Moreover, 32 *I/O forwarding nodes*, called I/O nodes, are evenly distributed through the interconnect, each forwarding metadata and data requests to Mira-FS for a group of 128 compute nodes. Cetus routes I/O traffic statically: it points each compute node group to a dedicated I/O node by sharing 2 designated bridge nodes, each bridge node connecting to its I/O node by a single link.

Mira-FS has two partitions: Mira-FS0 and Mira-FS1, each a single namespace on a metadata pool and a data pool. We experiment on Mira-FS1 with 1 and 336 **NSD**s (**N**etwork **S**torage **D**isks) for metadata and data services. For data service, 48 NSD servers manage 336 NSDs.

GPFS striping policy. To absorb bursts in parallel, GPFS stripes burst data across NSDs in its data pool, as shown in Figure 3a. For each burst, GPFS partitions the data into a sequence of equal-size blocks and distributes the block sequence across an NSD sequence in a round-robin way. The block size (*GPFS block size*) is configured at filesystem creation time. The NSD sequence starts from a random NSD chosen independently for each burst and may range over the entire data pool. Thus users do not control either of these parameters. Mira-FS1 configures GPFS block size as 8MB.

GPFS subblock policy. GPFS manages filesystem fragmentation with *subblocks*. Specifically, GPFS divides each block into 32 equal-size subblocks. When the last block of a file is $<$ block size, GPFS partitions the block data into a group of subblocks and merges/migrates the subblocks to the local and/or remote NSDs to form full blocks. This policy works at *file_close* when the block size is determined.

2) Lustre on Titan and Atlas2: Titan is a Cray XK7 machine at OLCF and connected to a Lustre deployment Spider 2. Figure 2b shows the system in detail.

On Titan, 18,688 compute nodes are connected by a 3-D torus interconnect; each node has a 16-core CPU and a GPU, and runs a Lustre software stack serving as a Metadata Client (**MDC**) and an Object Storage Client (**OSC**). Titan is connected to Spider 2 by a Scalable I/O Network (SION).

Spider 2 has two partitions (Atlas1 and Atlas2), each a Metadata Server (**MDS**) and 1,008 Object Storage Targets (**OST**s). We focus on Atlas2, in which 144 Object Storage Servers (**OSS**es) manage the I/O traffic for 1,008 OSTs (144×7) in a round-robin way. Titan nodes access Spider 2 via 172 I/O routers; the routers are evenly distributed through the torus and route I/O traffic statically [12], [13]: a compute node is connected to a fixed group of "closest" I/O routers.

In contrast to GPFS, data striping in Lustre is user-controlled, as shown in Figure 3b.

Lustre striping policy. A burst is partitioned into a sequence of equal-sized blocks, distributed across a sequence of OSTs in a round-robin way. The block size, OST-sequence length, and OST start index are three configurable parameters, called *stripe size*, *stripe count* and *starting OST*. Atlas2 takes 1MB (stripe size), 4 (stripe count) and a randomly chosen starting OST as its default configuration.

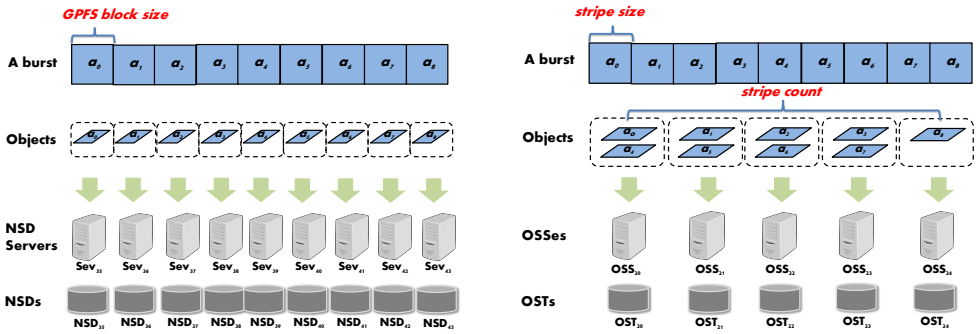
Across two representative filesystems, we find that:

Observation ②. From the view of I/O writes, supercomputer I/O systems are multi-stage write paths, including the stages from compute node to storage target (shown in Figure 2).

Observation ③. For a write pattern on a write path, aggregate load on each stage can be estimated/predicted according to the burst size and the numbers of compute nodes/cores in use.

Observation ④. For a write pattern of a job on a supercomputer, the locations of the compute nodes are known at job allocation. The resources used (e.g., the number of I/O routers) and load distribution among the resources for both compute nodes and network interconnect are known based on the job location and the supercomputer's network configuration.

Observation ⑤. For a write pattern of a job on a filesystem, the resources used and load distribution on the servers/targets



(a) GPFS striping. For concurrent bursts of a write operation, each burst is striped independently (§II-B1).

(b) Lustre striping For concurrent bursts of a write operation, users can configure *stripe size*, *stripe count*, and *starting OST* (§II-B2).

Fig. 3: GPFS and Lustre Striping Policies

for metadata and data services can be estimated based on the write pattern, striping policy and server-target mapping.

III. Modeling Write Performance of Supercomputer I/O Systems

A. Parameters Influencing Write Performance

We consider the target systems as multi-stage write paths. For each such path, we build features on the *performance-related parameters* that potentially affect the time cost on individual and correlated I/O stages, and interference (noise).

For a stage on a write path, we consider three performance-related parameters: *aggregate load*, *load skew*, and *resources in use*. For a metadata server stage, aggregate load represents the number of overall metadata operations of a write pattern. For the remaining stages, aggregate load represents the total Bytes of data generated by the pattern. We define load skew of a stage based on its straggler. Specifically, load skew is the maximum load (maximum number of metadata operations or maximum Bytes of data) on a single component of a stage. For example, for I/O router stage on Lustre deployments, load skew is the maximum Bytes of data processed by a single router. Moreover, resources in use is the number of components used in a stage. For example, at compute node stage, resources in use is the number of the compute nodes issuing write bursts. As our follow-on analysis shows that load skew is an important factor to consider for prediction accuracy and performance improvement.

Resources on some stages are not partitionable. Cetus/Mira-FS1 has one such stage: the Infiniband Network (the network and links are always used concurrently); Titan/Atlas2 has two such stages: the metadata server and the SION network. For these stages, we consider only one parameter: aggregate load.

With no loss of generality, we consider a write pattern that produces $m \times n$ bursts (size K Bytes) from m nodes with n cores per node. Our modeling approach can also be used to predict the performance of more flexible/dynamic write patterns when the write load and the compute nodes/cores in use are known before issuing writes. In particular, the load imbalance among compute nodes can be addressed as load skew at the compute-node stage.

For the pattern on a stage of a system, the performance-related parameters can be collected or estimated. We demonstrate the process on locating the parameters below and summarize the relative notations in Table I.

Besides collecting parameters from m, n, K, for Cetus/Mira-FS1, we estimate the number of subblocks per burst (n_{sub}) based on the write pattern and GPFS subblock policy (§II-B1). We collect the numbers of bridge nodes in use (n_b), links in use (n_l), and I/O nodes in use (n_{io}) based on the locations of the m nodes and the network configurations in Cetus (**Observation** ④). Similarly, we estimate load skew on bridge nodes, links and I/O nodes by collecting the information on s_b, s_l, s_{io}, which represent the sizes of the largest node groups in the m nodes that connect to the same bridge node, the same link, and the same I/O node, respectively.

Moreover, on Mira-FS1, for each of the $m \times n$ bursts we estimate the number of NSDs in use (n_d) based on the write pattern, GPFS block size and GPFS striping policy (§II-B1), and estimate the number of NSD servers in use (n_s) based on n_d and the mapping from NSD servers to NSDs (**Observation** ⑤). Moreover, according to GPFS striping policy each burst chooses starting NSDs randomly and independently. Theoretically, for the $m \times n$ bursts the numbers of NSDs in use (n_{nsd}) and NSD servers in use (n_{nsds}) are both random. Statistically, these numbers are bound to m, n, n_d, n_s: i) The more bursts ($m \times n$) a write pattern produces, the larger n_{nsd}, n_{nsds} are likely in use. ii) The more NSDs/NSD servers each burst uses (n_d, n_s), the larger n_{nsd}, n_{nsds} are likely in use. Accordingly, we estimate n_{nsds}, n_{nsd} based on m, n, n_s, n_d.

For Titan/Atlas2, we collect the number of I/O routers in use (n_r) and estimate load skew on routers based on the largest node group (s_r) in the m nodes that connect to the same router. n_r and s_r are known from the node-locations and the node to I/O router mapping in Titan (**Observation** ④).

On Atlas2, we estimate the number of OSTs in use (n_{ost}) by write pattern and the configurations on Lustre striping (discussed in §II-B2). We estimate the number of OSSes in use

TABLE I: Notations used for the performance-related parameters. Section III-A presents the definitions in detail.

Target System	Collectable Parameters	Predictable Parameters
Cetus/Mira-FS1	m, n, K, $n_{sub}, n_b, n_l, n_{io}, s_b, s_l, s_{io}$	$n_d, n_s, n_{nsd}, n_{nsds}$
Titan/Atlas2	m, n, K, n_r, s_r	$n_{ost}, n_{oss}, s_{ost}, s_{oss}$

(n_{oss}) based on n_{ost} and the mapping from OSSes to OSTs in Atlas2. We estimate the load skew on OSTs (s_{ost}) and OSSes (s_{oss}) according to the striping configurations and OSS-OST mapping (**Observation** ⑤).

B. Features

For a performance-related parameter on a stage of a write path, we derive two features for positive and inverse correlations. In particular, we take only positive feature for subblock-related parameters on Cetus/Mira-FS1: when a burst has no subblock (e.g., 8MB burst), the positive feature value is 0.

Moreover, we build features for interference by following the observations on Titan's I/O system [10]. Specifically, in a production environment I/O interference is positively correlated to the number of compute nodes (m) and inversely correlated to the aggregate burst size ($\frac{1}{m\times n\times K}$). Thus, for the target systems, we use three features to address interference, including m, $\frac{1}{m\times n\times K}$, and $\frac{m}{m\times n\times K}$.

1) Features for GPFS filesystems: Table II reports the 8 stages of the write path of a typical GPFS deployment, which is also given in Figure 2a.

Metadata stage. On Cetus/Mira-FS1, we consider the features for both metadata services and subblock operations. In particular, the aggregate load of metadata comes from two parts: the numbers of *file_open* and *file_close* ($m \times n$) and the number of subblocks for the $m \times n$ bursts ($m \times n \times n_{sub}$).

Stages within supercomputers (Compute Node, Bridge Node, Link, I/O node). For these stages, aggregate load is the aggregate data size of the write pattern ($m \times n \times K$), and the resources in use (m, n, n_b, n_l, n_{io}) can be collected directly (§III-A). The load skew on the stages of compute node, bridge node, link and I/O node can be estimated by $n\times K$, $s_b\times n\times K$, $s_l \times n \times K$, $s_{io} \times n \times K$, respectively.

Infiniband Network stage. Aggregate load is $m \times n \times K$.

Stages within storage systems (NSD server, NSD). Aggregate load is $m \times n \times K$. The resources in use for NSD servers and NSDs (n_{nsds}, n_{nsd}) are determined by $m \times n \times n_s$, $m \times n \times n_d$, respectively (§III-A). Moreover, we consider there is no noticeable load skew on these two stages and accordingly build no feature on it. In practice, load skew may occur on NSDs and NSD servers; and from the application viewpoint, the skew is unpredictable. In this study, we find and report in Section IV that the write performance of Cetus/Mira-FS1 can be estimated accurately, suggesting GPFS striping policy performs effectively in balancing write load across the entire data pool. We conclude that, for large-scale writes on GPFS filesystems, the load skew on these two stages is negligible.

Besides the features for individual stages, we also develop cross-stage features for correlated stages. Compared to CPU resources on supercomputers, I/O bandwidth is a shared and a finite resource; I/O bottlenecks may occur on one or more stages concurrently [6], [10]. To address the concurrent bottlenecks from multiple stages, we consider adjacent stages on the write paths as correlated stages, and build cross-stage features to address concurrent load skew (potential bottlenecks) on them. For example, for Compute Node and Bridge Node stages, we build a cross-stage feature as $(n\times K)\times(s_b\times n\times K)$.

In summary, we build overall 41 features for a GPFS write path, including 34 and 4 features for individual and correlated stages, and 3 for interference, respectively.

2) Features for Lustre Filesystems: Tables III reports the 6 stages of a typical Lustre filesystem, also given in Figure 2b.

Metadata stage. Aggregate load is metadata operations for *file_open* and *file_close* ($m \times n$).

Stages within supercomputers (Compute Node, I/O Router). For these two stages,aggregate load is the aggregate data size ($m \times n \times K$) and the resources in use for compute node (m) and I/O routers (n_r) can be collected directly (§III-A). The load skew on compute node and I/O routers can be estimated by $n \times K$ and $s_r \times n \times K$ respectively.

SION stage. Aggregate load is $m \times n \times K$.

Stages within storage systems (OSS, OST). Aggregate load is the aggregate data size ($m \times n \times K$). The resources in use for OSS (n_{oss}) and OST (n_{ost}) and the load skew for these two stages (s_{oss}, s_{ost}) can be predicted (§III-A).

Similar to Cetus/Mira-FS1, we build cross-stage features for adjacent stages on the Lustre write path. In summary, a Lustre write path has 30 features, including 24 and 3 features for individual and correlated stages, and 3 features for interference, respectively.

C. A Cross-Platform Modeling Method

1) Building regression models: We model the mean end-to-end write time (t) as a function (f) of a feature set $X = \{x_1, x_2, ..., x_{n-1}\}$, where f is a regression technique and each x_i is a feature.

$$t = f(x_1, x_2, ..., x_{n-1}) \tag{1}$$

To improve model interpretability and enhance prediction accuracy, we build models with popular regression techniques from three groups: (1) Linear models, (2) linear models with feature selection (lasso and ridge), (3) nonlinear models with feature selection (decision tree and random forest). Another group of nonlinear models is also popular in performance prediction. This group includes SVR (Support Vector Regression) and Gaussian process. They maintain all features and transform the feature forms with kernels to enhance prediction accuracy. In this work, we train SVR and Gaussian models with two widely used kernels (RBF and polynomial), and receive low prediction accuracy for both Cetus/Mira-FS1 and Titan/Atlas2. We conclude that these techniques fail to provide

TABLE II: Features of individual stages of GPFS deployments

Performance-Related Parameter	Metadata Cost		Data-absorption Cost					
Stage	Metadata	Subblock	Compute Node	Bridge Node	Link	Network	NSD server	NSD
Aggregate Load	$m \times n$, $\frac{1}{m \times n}$	$m \times n \times n_{sub}$	$m \times n \times K$, $\frac{1}{m \times n \times K}$					
Load Skew	$s_{io} \times n$, $\frac{1}{s_{io} \times n}$	$s_{io} \times n \times n_{sub}$	$n \times K$, $\frac{1}{n \times K}$, K, $\frac{1}{K}$	$s_b \times n \times K$, $\frac{1}{s_b \times n \times K}$	$s_l \times n \times K$, $\frac{1}{s_l \times n \times K}$			
Used Resources	n_{io}, $\frac{1}{n_{io}}$		m, $\frac{1}{m}$, n, $\frac{1}{n}$	n_b, $\frac{1}{n_b}$	n_l, $\frac{1}{n_l}$		n_s, $\frac{1}{n_s}$, n_{nsds}, $\frac{1}{n_{nsds}}$	n_d, $\frac{1}{n_d}$, n_{nsd}, $\frac{1}{n_{nsd}}$

TABLE III: Features of individual stages on Lustre deployments

Performance-Related Parameter	Metadata Cost	Data-absorption Cost				
Stage	Metadata	Compute Node	I/O Node	SION	OSS	OST
Aggregate Load	$m \times n$, $\frac{1}{m \times n}$	$m \times n \times K$, $\frac{1}{m \times n \times K}$				
Load Skew	n, $\frac{1}{n}$	$n \times K$, $\frac{1}{n \times K}$, K, $\frac{1}{K}$	$s_r \times n \times K$, $\frac{1}{s_r \times n \times K}$		s_{oss}, $\frac{1}{s_{oss}}$	s_{ost}, $\frac{1}{s_{ost}}$
Used Resources	m, $\frac{1}{m}$	m, $\frac{1}{m}$, n, $\frac{1}{n}$	n_r, $\frac{1}{n_r}$		n_{oss}, $\frac{1}{n_{oss}}$	n_{ost}, $\frac{1}{n_{ost}}$

accurate predictions for our target systems, or at least they require tuning.

2) Selecting the best models: For each method, we search for the best model from a regression model space, trained across subsets of the training set and the values of model parameters. We select the trained models that deliver the lowest MSEs (Mean Square Errors) on the *validation set*. In this study, we train models on benchmark data from runs with 1-128 nodes, which are relatively cheap. We choose 20% of the samples from each size range in 1-128 nodes at random for the validation set, and use the remaining 80% of samples for training. In §IV we evaluate how well the selected models predict performance on a *test set* of runs with 200-2000 nodes, which are not used for training or model selection.

D. A Convergence-Guaranteed Sampling Method

To train and evaluate the models, we benchmarked output performance on Cetus and Titan with multiple experiment samples under a range of parameters and conditions. We choose IOR as a burst generator. IOR is widely used in the HPC community to measure new deployments of supercomputer I/O systems, to capture the I/O characteristics of scientific applications in filesystem studies [6], [10], [14]–[16], and to compare the performance of storage systems and I/O tuning techniques, e.g., for IO-500. This work uses IOR to generate synthetic writes with various patterns and burst sizes on various numbers of compute nodes and cores, and measures their delivered performance. We summarize the method as five steps capturing the key aspects of the method.

Step 1. To obtain good coverage across write scales under a range of parameters, each *experiment* generates data for a fixed write scale (m) by following one or more *templates*. A template is a job script structured as multiple levels of for loops, each loop varying a parameter. For the experiments on GPFS deployments, each template varies the number of cores per node (n) and burst size (K); for Lustre deployments, each template varies n, K, W (stripe count).

Step 2. Each template attains good coverage on burst sizes (K) by strategically choosing burst-size ranges and randomly producing bursts in each chosen range. We consider bursts in a wide range: 1MB—10GB (§II-A2). To guarantee balanced burst-size coverage, we break it into 10 size ranges (Column 3 in Table IV and Table V), and generate a random burst size for each range.

Step 3. Each template attains good coverage on the number of cores per node (n) and stripe count (W). Similar to the process of generating burst sizes, for Lustre deployments like Titan we generate the number of cores per node (n) and stripe count (W) randomly. Specifically, for a template for Titan, we generate 4 or 8 random numbers in the range of 1—16 (the maximum number of CPUs in a Titan compute node), each number used as the number of cores per node; for stripe count (shown as Table V Column 4), we consider the range 1—64 from observed production use, break it into 5 stripe-count-ranges, and choose a random value within each stripe-count-range. For GPFS deployments like Cetus Mira-FS1, the systems limit users to take n from $\{1, 2, 4, 8, 16\}$ (16 is the maximum number of cores in a Cetus compute node), so we vary n within that range.

Step 4. Each template attains samples on different compute-node locations and background I/O interference by executing many *jobs* from the template at different times. To avoid internal interference, we execute one job at a time. Following the template, a job performs several *rounds* of IOR executions, each round executing IOR with the varied parameter values specified in the multi-level for-loops. Thus, the jobs of a template yield benchmark data for the same parameters and patterns, but at a sampling of times and conditions.

Step 5. A *sample* is the mean write time of identical IOR executions. We consider two IOR executions to be identical if they have the same parameters and patterns. A sample may be generated from different jobs of the same template. To assure that the data is stable, we estimate sample convergence with the central limit theorem, which is widely used on data with an unknown mean. [3] In particular, for the sample of r identical IOR executions with write times:$\{t_0, t_1, ..., t_{r-1}\}$, we consider the sample is converged with a confidence level $(1-\alpha)$ and error estimator (ζ). $z_{\alpha/2}$ is the value from the standard normal distribution for the selected confidence level $1-\alpha$; σ and t are the standard deviation and mean of the r times. If we presume the expected mean is $\bar{t}$, Formula 2 guarantees that $\|\frac{\bar{t}-t}{t}\| \leq \zeta$ with the confidence level $1-\alpha$.

$$\| \frac{z_{\alpha/2} \times \left(\frac{\sigma}{\sqrt{n-1}} \right)}{t} \| \leq \zeta \tag{2}$$

[3] In this study, the mean write time of identical IOR executions is unknown beforehand.

TABLE IV: Write patterns on Cetus/Mira-FS1. The first row generates data for both training and testing. The second row generates larger bursts for training purpose only. The third row repeats the write patterns of production codes (§IV-A).

Scale (m)	Cores per Node (n)	Burst Size (K)
1, 2, 4, 8, 16, 32, 64, 128, 200, 256, 400, 512, 800, 1000, 2000	1, 2, 4, 8, 16	1MB—5MB 6MB—25MB 25MB—100MB 101MB—250MB 251MB—500MB 501MB—1024MB 1025MB—2560MB
1, 2, 4, 8, 16, 32, 64, 128	1, 2, 4, 8, 16	2561MB—5120MB 5121MB—7680MB 7681MB—10240MB
1000, 2000	1, 2, 4, 8, 16	4MB, 23MB, 59MB, 69MB, 121MB, 376MB, 750MB, 1024MB, 1280MB

TABLE V: Write patterns on Titan/Atlas2. We generate these patterns following the same rule presented in Table IV.

Scale (m)	Cores per Node (n)	Burst Size (K)	*stripe_width* (W)
1, 2, 4, 8, 16, 32, 64, 128, 200, 256, 400, 512, 800	8 *from* 16	1MB—5MB 6MB—25MB 25MB—100MB 101MB—250MB 251MB—500MB 501MB—1024MB 1025MB—2560MB	1—4 5—8 9—16 17—32 33—64
1, 2, 4, 8, 16, 32, 64, 128	4 *from* 16	2561MB—5120MB 5121MB—7680MB 7681MB—10240MB	1—4 5—8 9—16 17—32 33—64
1000, 2000	1, 4	4MB, 23MB, 59MB, 69MB, 121MB, 376MB, 750MB, 1024MB, 1280MB	4, 5—64

IV. Experimental Evaluations

We use IOR to generate data for model training and testing and optimize the write performance through model-guided I/O adaptations.

A. Experiment Data

We train models on small-scale writes ($\leq$128 compute nodes). Then, we test the trained models on medium (200—800 nodes) and large-scale (1000, 2000 nodes) writes. For the small and medium-scale writes, we generate the data with extensive write patterns by following the benchmarking method (III-D). On the large-scale writes, we generate the testing data by repeating the write patterns of real applications, including XGC, GTC [17], S3D, PlasmaPhysics, Turbulence1, Turbulence2, and AstroPhysics reported in [18]. We determine the write scale of training data as 1 — 128 compute nodes in consideration of the ratios of compute node to I/O node (128:1 on Cetus) and compute node to I/O router (110:1 on Titan). We produce the training and testing data by following the templates in Tables IV and V.

We focus on the writes $\geq$5 seconds, since in production runs smaller writes are usually hidden by the client-side page cache and make little impact on application performance. We produced 3,899 and 4,004 converged samples (defined in §III-D) for training on Cetus/Mira-FS1 and Titan/Atlas2, respectively. For the training set on Cetus/Mira-FS1 a write scale has 394 — 646 samples; for the set on Titan/Atlas2 a write scale has 427 — 569 samples. We evaluate our approach on 4 test sets of a target system, with 3 sets for converged samples and 1 for unconverged samples. The 3 converged sets are grouped on the write scales of the samples: small set (on 200, 256 nodes), medium set (on 400, 512 nodes), large set (on 800, 1000, 2000 nodes). For small/medium/large sets, each set has 278, 174, 133 samples on Cetus/Mira-FS1, 237, 226, 273 samples on Titan/Atlas2. The unconverged sets are generated on the write scales on 200—2000 nodes, including 169 and 180 samples for Cetus and Titan respectively.

B. Chosen Models and Features

For each regression technique, we train models across 255 training sets, each a combination of datasets built on the write scales in 1 — 128 nodes. By following the modeling method (§III-C), we identify 10 models each for a regression technique on a target system, including two linear models ($linear_{best\_cetus}$, $linear_{best\_titan}$), two lasso models ($lasso_{best\_cetus}$, $lasso_{best\_titan}$), two ridge models ($ridge_{best\_cetus}$, $ridge_{best\_titan}$), two decision tree models ($tree_{best\_cetus}$, $tree_{best\_titan}$), and two random forest models ($forest_{best\_cetus}$, $forest_{best\_titan}$).

We also train models with the training data on 1 — 128 nodes as the baseline cases. In summary, we address 10 additional models, each named as a base model for a regression method on a target system (e.g., $linear_{base\_cetus}$).

C. Model Evaluation

1) Evaluating the modeling method: We apply the modeling method (§III-C) on five representative regression models and compare the prediction accuracy to the results of the corresponding baseline models (defined in §IV-B). Figure 4 presents the results. For all regression techniques on the testing sets of both target systems, the chosen models report higher prediction accuracy. Specifically, for a learning technique on Cetus/Mira-FS1, the chosen model show 1.34× — 52.6× better prediction accuracy in MSE (mean square error). For a technique on Titan/Atlas2, the chosen models show 1.21× — 1.62× better prediction accuracy in MSE. Moreover, the results also suggest that the chosen lasso models deliver the best prediction accuracy on both of the target systems and for both converged and unconverged samples.

2) Model accuracy: We use *relative true error* (ϵ) as the error estimator. Consider the mean time of the ith sample is t_i, its prediction result is t'_i,

$$\epsilon_i = \frac{t'_i - t_i}{t_i} \tag{3}$$

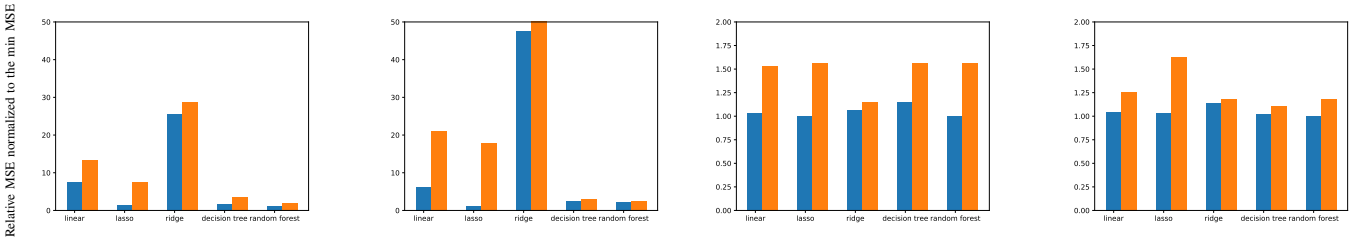
$\epsilon_i > 0$ suggests that t_i is over estimated; $\epsilon_i < 0$ suggests that t_i is under estimated; $\|\epsilon_i\|$ quantifies prediction accuracy: smaller $\|\epsilon_i\|$ indicates higher accuracy. We focus on: $\|\epsilon\|$=0.2 and $= 0.3$. We choose these two thresholds in considerations of two factors: 1) These two thresholds are widely used as the conventional numbers in accuracy measurements [19]; 2) for HPC applications, the time cost on I/O writes are expected to be ∼10% of the total runtimes (see §II-A1). Plus 0.2—0.3 prediction error, we expect the guaranteed I/O cost as of 7% — 13% of the total runtimes, which is generally acceptable for production runs [10], [13], [20].

TABLE VI: The chosen lasso models on Cetus/Mira-FS1 and Titan/Atlas2. The models are defined in §IV-B. Each row reports the parameters of a model, including training set, value of the shrinkage parameter (λ) and selected features. For each lasso model, we report the intercept, the selected features and their corresponding coefficients.

Model Name	Training Set	λ	Intercept	Selected Features									
$lasso_{best\_cetus}$	{32 — 128}	0.01	0.902	0.0864	5.812^{-4}	4.301^{-5}	2.646^{-4}	0.0022	2.535^{-4}	5.167^{-4}	3.238^{-6}	5.958^{-13}	2.97^{-10}
				n	$s_l \times n \times K$	$s_b \times n \times K$	$m \times n$	$n \times K$	n_{nsds}	$s_{io} \times n \times K$	n_{nsd}	$(s_l \times n \times K) \times (s_b \times n \times K)$	$(s_b \times n \times K) \times n_{nsds}$
$lasso_{best\_titan}$	{16 — 128}	0.01	1.7826	3.485^{-4}	-0.010	2.91^{-4}		8.963^{-5}		1.463^{-6}	2.116^{-4}	9.315^{-11}	1.925^{-10}
				K	n_r	$s_r \times n \times K$		s_{ost}		$m \times n \times K$	$n \times K$	$(n \times K) \times (s_r \times n \times K)$	$(s_r \times n \times K) \times n_{oss}$

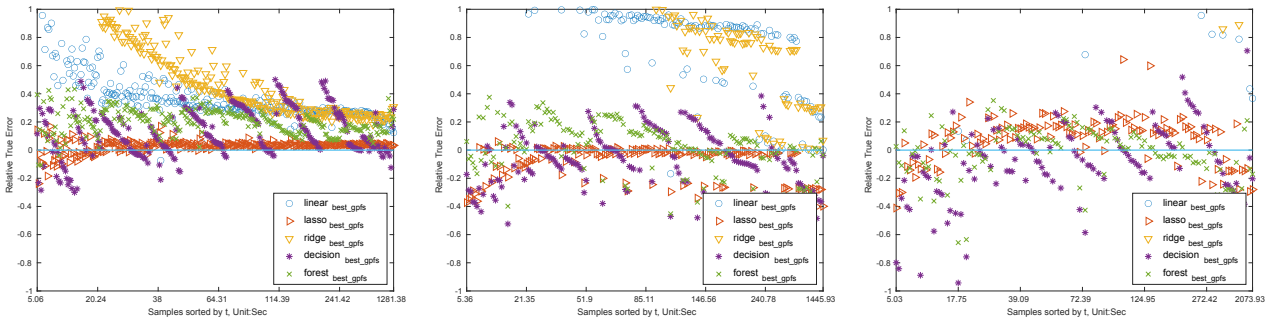
TABLE VII: Prediction accuracy of the chosen lasso models. This table presents the accuracy of the chosen lasso models (defined in §IV-B and shown in Table VI) on the four test sets for the two target systems.

Target System	Model	small set		medium set		large set		unconverged samples	
		$\|\|\epsilon\|\| \leq 0.2$	$\|\|\epsilon\|\| \leq 0.3$	$\|\|\epsilon\|\| \leq 0.2$	$\|\|\epsilon\|\| \leq 0.3$	$\|\|\epsilon\|\| \leq 0.2$	$\|\|\epsilon\|\| \leq 0.3$	$\|\|\epsilon\|\| \leq 0.2$	$\|\|\epsilon\|\| \leq 0.3$
Cetus/Mira-FS1	$lasso_{best\_cetus}$	99.64%	100%	74.14%	90.8%	76.69%	93.98%	44.97%	63.91%
Titan/Atlas2	$lasso_{best\_titan}$	96.2%	98.31%	93.36%	94.69%	82.42%	84.25%	12.78%	20.56%



(a) converged on Mira **(b) unconverged on Mira** **(c) converged on Titan** **(d) unconverged on Titan**

Fig. 4: MSEs on Cetus/Mira-FS1 and Titan/Atlas2. In each subfigure, we report the results of five regression techniques. For a technique in a subfigure, we report the results of two models: the best model chosen by our approach (left) and the baseline model (right); for a model in a subfigure, we normalize its MSE (mean square error) to the minimum MSE among the models on the same testing set.



(a) results on small set **(b) results on the medium set** **(c) results on the large set**

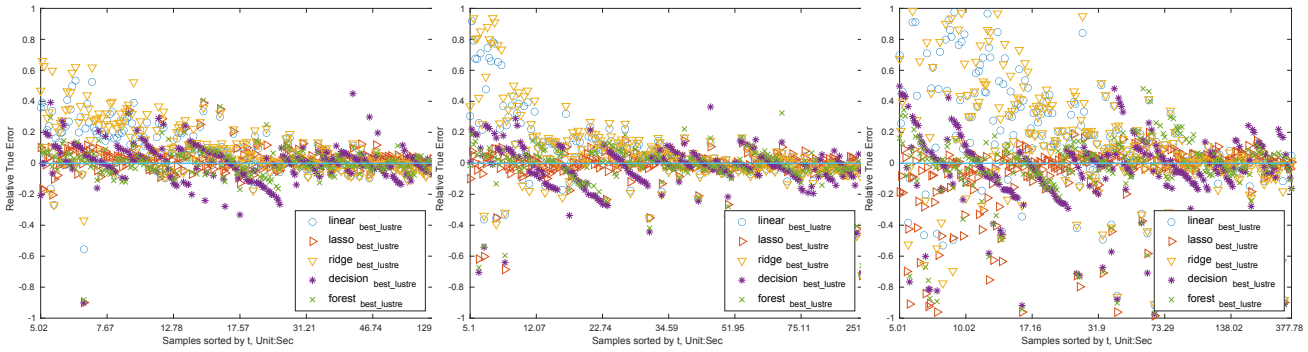
Fig. 5: Model accuracy on the converged test sets of Cetus/Mira-FS1. Each subfigure plots the relative true errors (ϵ) of five models on a test set; the errors are sorted along the x-axis based on t. The test sets, ϵ and t are defined in §IV-C.

We focus on the performance of the best models identified by our learning approach. Figures 5 and 6 plot the error accuracy of the models on the three converged test sets for the two target systems. It is clear that the best lasso models deliver the best overall prediction accuracy for both of the target systems. We report the chosen lasso models in Table VI and summarize the prediction accuracy in Table VII. In particular, for the lasso models on Cetus/Mira-FS1 and Titan/Atlas2, 74.14% — 99.64% of the samples on the small/medium/large sets report $\|\epsilon\| \leq$0.2, 84.25% — 100% of the samples on the sets report $\|\epsilon\| \leq$0.3. In conclusion, the chosen lasso models are highly accurate on the target systems.

Moreover, our approach identifies the effective features in the target environments. As listed in Tables VI, lasso can interpret the write behaviors of the target systems effectively. In particular, we find that: (1) The write behaviors of Cetus/Mira-FS1 are dominated by the metadata load, load skew in the supercomputer, and the resources used in the filesystem; (2) The write behaviors of Titan/Atlas2 are heavily determined by the aggregate load, load skew, and resources used within supercomputers.

D. Model-Guided I/O Middleware

Finally, we consider the potential value of using the performance models to configure I/O middleware systems for best



(a) results on the small set **(b) results on the medium set** **(c) results on the large set**

Fig. 6: Model accuracy on the converged test sets of Titan/Atlas2. Each subfigure plots the relative true errors (ϵ) of five models on a test set; the errors are sorted along the x-axis based on t. The test sets, ϵ and t are defined in §IV-C.

performance. We focus on the chosen lasso models for their high prediction accuracy.

Many supercomputer applications use I/O middleware systems like ADIOS and ROMIO because they offer flexible I/O APIs and opportunities to improve I/O performance transparently. At launch time, the run is assigned to a group of compute nodes and issues write requests from its nodes/cores periodically. An I/O middleware system provides *write adaptation*: it selects a subset of the engaged nodes/cores as *aggregators*, directs the output data from compute nodes to selected aggregators, and writes the output to the external storage systems from the aggregators. Today, write adaptations are configured by various heuristics.

In this study, we use the chosen lasso models to select an aggregator configuration based on predicted performance of the candidates. We determine the number of aggregators, the burst-size per aggregator, and the aggregator locations. As is discussed in Section IV-C2, for both of Cetus/Mira-FS1 and Titan/Atlas2 the load skew within supercomputers affects the write performance heavily. Thus, we strategically choose the aggregator locations to use the links and I/O nodes (for Mira) or the I/O routers (for Titan) in a balanced way. Each run in the benchmark data gives the node locations for the runs, allowing us to compute load skew on the I/O routers generated by the run. The model-guided search selects from among candidate aggregator configurations with a balanced load. On Lustre, the search also considers the striping parameters of the candidates.

We compared predicted performance for the best candidate adaptations for samples in the test set (samples with 200-2000 compute nodes) on both targets. We estimate the potential gain from I/O adaptation by $t'' + e$, where t'' is the predicted value by the chosen lasso model with the adapted feature values, $e = t' - t$ is the error of the predicted value by the model with the original feature values (t') and the observed write time (t). Specifically, we presume that, for a write pattern on a target system, the prediction error does not change. And under this condition we measure the expected performance improvement by $\frac{t''+e}{t}$. Note that this measure does not consider any overheads to move data to the aggregators, which we expect to reduce the benefit modestly.

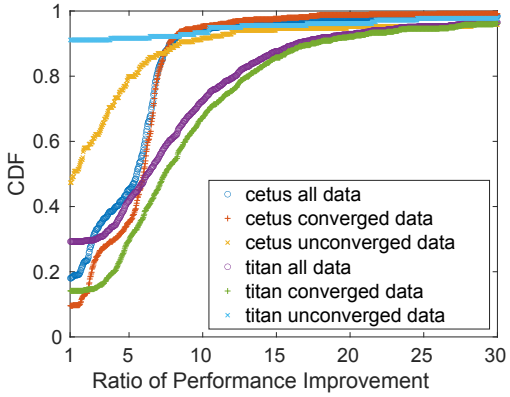


Fig. 7: Predicted performance improvement using model-guided I/O adaptation to configure aggregators for the I/O patterns of samples in the test set.

Figure 7 reports the results. It shows that, for Cetus/Mira-FS1, 82.4% of the overall samples report $\geq$1.1$\times$ performance improvement; for Titan/Atlas2, 71.6% of the overall samples show $\geq$1.15$\times$ performance improvement. We conclude that, under the guidance of our models, I/O adaptation within supercomputers can attain performance improvement. We leave the verification on real applications as our future work.

V. Related Work

A group of studies [21]–[23] adopt learning models to predict the I/O behaviors of HPC applications. In particular, Kunkel et.al [21] build decision trees to predict applications' non-contiguous I/O behaviors and further determine the parameter combinations for data sieving in ROMIO [24]. Omnisc'IO [22], inspired by Sequitur algorithm, introduces a context-free grammar to learn I/O patterns of HPC applications. Mckenna et.al [23] model application runtimes and I/O patterns by building training set on job logs and system monitoring tools. Compared to the works in this group, our study presumes that applications' I/O patterns/behaviors are known and given, and builds learning models on the

features derived from both I/O patterns and filesystem's design and configuration. Another group of studies [25], [26] use learning models to predict the I/O performance variability of supercomputers. Wan et.al [25] builds a hidden Markov model for a Lustre filesystem to learn the variability of I/O latencies from a single client to a single OST. Madireddy [26] et.al uses Gaussian process to predict the I/O performance variability of a Lustre-based supercomputer and derive system-specific features from Lustre monitoring tools. Different from the studies in this group, we address the variability issue in two ways: 1) modeling the mean performance of various I/O patterns, 2) deriving features to address I/O interference from competing loads. Xie et.al [14] also analyzes the I/O performance of large-scale parallel filesystems with machine learning techniques. Differently, they predict the I/O performance with one regression model but we introduce a systematic modeling method on five popular regression methods and apply the modeling results on performance improvement.

VI. Conclusions

In this paper, we build regression models to predict write performance for GPFS- and Lustre-based filesystems under production load. We take the mean write time as the model target; generate features to address the load, load skew and resources in use on separate and correlated stages of the target multi-stage write paths; introduce a convergence-guaranteed sampling method to produce a training set space with a good variety and low benchmarking cost; and, finally propose an approach to search for the best model from a rich model space for each target write path.

Acknowledgments

This research used resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at the Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725. This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References

[1] W. Shin, C. Brumgard, B. Xie, S. Vazhkudai, D. Ghoshal, S. Oral, and L. Ramakrishnan, "Data Jockey: Automatic data management for HPC multi-tiered storage systems," in *IPDPS'19*, 2019.

[2] Argonne Leadership Computing Facility, "Cetus," https://www.alcf.anl.gov/user-guides/computational-systems.

[3] Oak Ridge Leadership Computing Facility, "Titan Cray XK7," https://www.olcf.ornl.gov/computing-resources/titan-cray-xk7/.

[4] Oak Ridge Leadership Computing Facility (OLCF), "Summit," https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/.

[5] G. Lockwood, W. Yoo, S. Byna, N. J. Wright, S. Snyder, K. Harms, Z. Nault, and P. Carns, "UMAMI: a recipe for generating meaningful metrics through holistic i/o performance analysis," in *PDSW'17*, 2017.

[6] B. Xie, J. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, and N. Podhorszki, "Characterizing output bottlenecks in a supercomputer," in *SC'12*, 2012.

[7] G. Bateman, S.-H. Ku, J. Cummings, C.-S. Chang, and A. Kritz, "Xgc documentation," http://w3.physics.lehigh.edu/xgc/, 2016.

[8] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo, "Terascale direct numerical simulations of turbulent combustion using S3D," *Computational Science and Discovery*, 2009.

[9] M. Gilge *et al.*, *IBM system blue gene solution: blue gene/Q application development*, 2014.

[10] B. Xie, Y. Huang, J. Chase, J. Y. Choi, S. Klasky, J. Lofstead, and S. Oral, "Predicting output performance of a petascale supercomputer," in *HPDC'17*, 2017.

[11] Argonne National Laboratory, "Darshan: HPC I/O Characterization Tool," http://www.mcs.anl.gov/research/projects/darshan.

[12] Y. Kim, R. Gunasekaran, G. M. Shipman, D. A. Dillow, Z. Zhang, and B. W. Settlemyer, "Workload characterization of a leadership class storage cluster," in *PDSW'10*, 2010.

[13] B. Xie, J. Chase, D. Dillow, S. Klasky, J. Lofstead, S. Oral, and N. Podhorszki, "Output performance study on a production petascale filesystem," in *HPC-IODC'17*, 2017.

[14] B. Xie, Z. Tan, P. Carns, J. Chase, K. Harms, J. Lofstead, S. Oral, S. Vazhkudai, and F. Wang, "Applying machine learning to understand write performance of large-scale parallel filesystems," in *PDSW'19*, 2019.

[15] B. Xie, S. Oral, C. Zimmer, J. Y. Choi, D. Dillow, S. Klasky, J. Lofstead, N. Podhorszki, and J. Chase, "Characterizing output bottlenecks of a production supercomputer: Analysis and implications," *ACM TOS'20*, 2020.

[16] B. Xie, "Output performance of petascale file systems," Ph.D. dissertation, Duke University, 2017.

[17] Z. Lin, T. S. Hahm, W. Lee, W. Tang, and R. B. White, "Turbulent transport reduction by zonal flows: Massively parallel simulations," *Science*, 1998.

[18] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *MSST'12*, 2012.

[19] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, 2001.

[20] C. S. Chang, S. Klasky, J. Cummings, R. Samtaney, A. Shoshani, L. Sugiyama, D. Keyes, S. Ku, G. Park, S. Parker *et al.*, "Toward a first-principles integrated simulation of tokamak edge plasmas," *Journal of Physics: Conference Series*, 2008.

[21] J. Kunkel, M. Zimmer, and E. Betke, "Predicting performance of non-contiguous I/O with machine learning," in *ISC'15*, 2015.

[22] M. Dorier, S. Ibrahim, G. Antoniu, and R. Ross, "Omnisc'IO: a grammar-based approach to spatial and temporal I/O patterns prediction," in *SC'14*, 2014.

[23] R. McKenna, S. Herbein, A. Moody, T. Gamblin, and M. Taufer, "Machine learning predictions of runtime and IO traffic on high-end clusters," in *CLUSTER'16*, 2016.

[24] R. Thakur, W. Gropp, and E. Lusk, "Data sieving and collective i/o in ROMIO," in *Frontiers' 99*, 1999, pp. 182–189.

[25] L. Wan, M. Wolf, F. Wang, J. Y. Choi, G. Ostrouchov, and S. Klasky, "Analysis and modeling of the end-to-end I/O performance on OLCF's titan supercomputer," in *HPCC/SmartCity/DSS*, 2017.

[26] S. Madireddy, P. Balaprakash, P. Carns, R. Latham, R. Ross, S. Snyder, and S. M. Wild, "Machine learning based parallel I/O predictive modeling: A case study on lustre file systems," in *ISC'18*, 2018.