Applying Machine Learning to Understand Write Performance of Large-scale Parallel Filesystems

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*—In high-performance computing (HPC), I/O performance prediction offers the potential to improve the efficiency of scientific computing. In particular, accurate prediction can make runtime estimates more precise, guide users toward optimal checkpoint strategies, and better inform facility provisioning and scheduling policies. HPC I/O performance is notoriously difficult to predict and model, however, in large part because of inherent variability and a lack of transparency in the behaviors of constituent storage system components. In this work we seek to advance the state of the art in HPC I/O performance prediction by (1) modeling the mean performance to address high variability, (2) deriving model features from write patterns, system architecture and system configurations, and (3) employing Lasso regression model to improve model accuracy. We demonstrate the efficacy of our approach by applying it to a crucial subset of common HPC I/O motifs, namely, file-per-process checkpoint write workloads. We conduct experiments on two distinct production HPC platforms—Titan at the Oak Ridge Leadership Computing Facility and Cetus at the Argonne Leadership Computing Facility—to train and evaluate our models. We find that we can attain $\leq$ 30% relative error for 92.79% and 99.64% of the samples in our test set on these platforms, respectively.**

***Index Terms*—Large-scale parallel filesystem, write performance, production supercomputer, machine learning**

I. Introduction

Motivation. HPC productivity could be greatly enhanced by more accurate I/O performance prediction. At supercomputing facilities, scientists are allocated limited compute cycles and charged based on their application execution times. They are therefore motivated to limit or reduce the time consumption on I/O. For the facilities, more predictable I/O performance will enable more precise compute-time allocations and further enable additional jobs on the same platform, increasing the system utilization. For example, for many scientists the ideal time spent on writing state snapshots is $\sim$10% of the total application execution times. In practice, however, the cost may span a wide range (e.g., 7%–20% for XGC code [39]) because of lack of guidance on I/O configuration. With accurate write performance prediction, scientists can manage the cost by configuring the write frequencies appropriately (§II-A).

Challenge. For production supercomputers, predicting I/O performance is challenging. One major obstacle is the high performance variability. Supercomputers and their filesystems

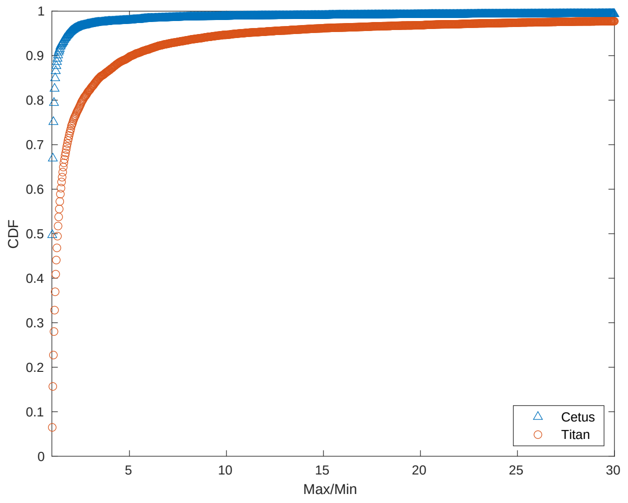


Fig. 1: CDFs of write performance variations on Cetus and Titan. The x-axis represents the relative measures ($\frac{max}{min}$) of the write bandwidths of the samples[1] used in experiments (§IV).

are shared by all running applications. These applications can be from different science domains with varying I/O outputs. The shared systems therefore exhibit significant performance variability to individual users. As an example, we present in Figure 1 performance variations with synthetic I/O benchmarks on the production supercomputers Cetus [1] at the Argonne Leadership Computing Facility (ALCF) and Titan [24] at the Oak Ridge Leadership Computing Facility (OLCF). Of the samples[1], 10.41% on Cetus and 49.71% on Titan report the peak write bandwidths as >1.3× their minimum bandwidths; in the worst cases, the max/min bandwidths differ by over three orders of magnitude for both machines. This high variability also exists widely on other supercomputing platforms and filesystems [17], [30], [36], [37], [40].

The other major obstacle is the limited user-level visibility into the I/O subsystems, which we refer to as the *black-box issue*. For most cases, end users and I/O middleware libraries

[1]For each sample point, we generate writes repeatedly with the same set of settings, including a specific write pattern, specific I/O configurations, and system settings (see §IV-A).

treat these subsystems as black boxes with minimal system-level insight. At supercomputing facilities, often the computing platforms and filesystems are monitored separately and operated by separate teams, with systemwide logs available only to their administrators [34]. Therefore, end users find it hard to understand I/O behavior and associated underlying root causes.

Our Approach. We address these two obstacles with a regression approach:

1) To address the high performance variability, we model the mean performance. 2) To address the black-box issue, we consider the target systems as multistage storage 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 the parameters and derive them from I/O patterns and system design and configuration. 3) To improve prediction accuracy, we build models with Lasso.

This approach is an extension of our earlier study on supercomputer Titan and its Lustre-based filesystem Spider [39]. Compared to the Titan study, this work discusses the write performance for both Lustre- and GPFS-based filesystems and builds predictive models with Lasso over linear regression for higher model accuracy.

We investigate the potential of our approach by applying it to a common I/O motif: the file-per-process data checkpoint. In this scenario, the engaged processes write data simultaneously, and the data from each process is stored in a separate file. This is a key initial step toward a more generalized model for I/O performance. We empirically evaluate our approach on two production supercomputers, Cetus at the ALCF and Titan at the OLCF, deployed with GPFS and Lustre filesystem software, respectively. We draw two major conclusions:

1. Our approach works effectively. For the best models on the GPFS and Lustre systems, up to 92.79% and 99.64% of the samples of a test set attain $\leq$30% relative errors, respectively.

2. Our approach identifies the most relevant features for write performance prediction. Our analysis shows that the write behaviors of the GPFS system are dominated by the metadata load and load skew within the supercomputer and the resources used in its filesystem; the write behaviors of the Lustre system are heavily determined by the aggregate load, load skew, and resources in use within the supercomputer and the load skew within the filesystem.

For end users and system administrators working on the systems similar to Cetus and Titan, our approach and the selected best models can be used to manage the I/O configurations at the application level or to configure burst parameters (e.g., tuning I/O configurations on Lustre filesystems) dynamically at system level during job allocations and based on the historical statistics of their I/O behaviors. Our initial success in modeling file-per-process write workloads indicates that this is a promising method for more generalized HPC I/O performance modeling. We intend to explore its application to other I/O access patterns in future work.

II. Background

A. Scientific Writes

At supercomputing facilities, a large number of codes are numerical simulations. Such like XGC [2] and S3D [6] codes, these applications are submitted as *jobs* (runs) to execute iterative computations and produce data periodically.

A job/run has a job description file, specifying the numbers of cores, nodes, and problem settings. Its problem space is partitioned into a group of equal-sized subspaces; each process runs on a different core[2] for a different subspace; all processes execute the same numerical methods (*solver*) over a sequence of iterations and sync with each other at the end of each iteration.

A run produces data that can be represented by one or more write patterns. Each pattern has a number of synchronous output bursts originating from a set of allocated compute nodes/cores; it repeats on a fixed time interval (write frequency) with each burst recording the states (numerical values) of the same variables. For each operation in a pattern, each core produces the same-size bursts across iteration intervals,[3] and the load is balanced among the engaged cores. To summarize, these applications exhibit three major properties:

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

Secondly, 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 setting [39]. In addition, with an accurate prediction of write times, a job's execution time can be predicted.

Thirdly, 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.

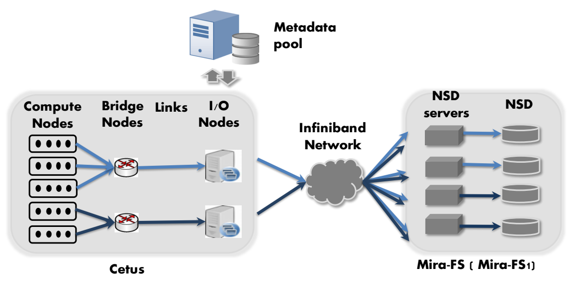
We limit our attention to the applications: for each write pattern, a fixed set of cores produce equal-size bursts (independent files) repeatedly. For these applications, the mean performance of output patterns is effective to address the write behaviors of applications [39].

B. Supercomputers and Their Filesystems

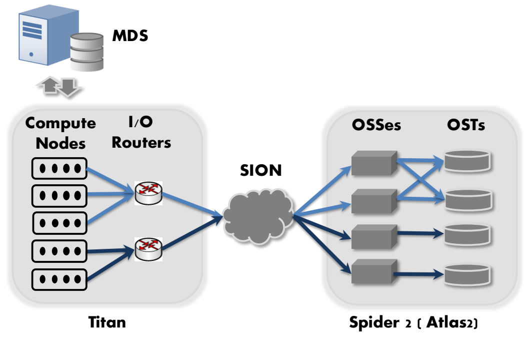
This section discusses two target supercomputer I/O subsystems: Cetus/Mira-FS1 and Titan/Atlas2, deployed with GPFS and Lustre filesystem software respectively.

[2]Blue Gene/Q machines [11] utilize 4-way hyperthreading and support 4 concurrent processes per core; but most users still choose to run one process on a different core because of the memory limitation (e.g., 16 GB memory per compute node on Cetus).

[3]For some codes (e.g., XGC), data imbalance and variation may occur across processes and operations, but in most cases it is ignorable [39].[3]



(a) Cetus and Mira-FS1



(b) Titan and Atlas2

Fig. 2: Architectures of the target I/O subsystems. 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.

1) GPFS on Cetus and Mira-FS1: Cetus is an IBM Blue Gene/Q machine, hosted at the ALCF and connected to Mira-FS: a center-wide GPFS filesystem providing I/O services to Cetus and other supercomputers at the ALCF. Figure 2a presents Cetus and Mira-FS.

On Cetus, 4,096 compute nodes are connected by a 5-D torus interconnect with 16 CPU cores per node. In addition to compute nodes, 32 dedicated I/O forwarding nodes are used to forward filesystem operations from the compute nodes to the Mira-FS filesystem. These I/O forwarding nodes are evenly distributed through the torus, but not connected to compute nodes directly. In particular, they are connected to subsets of compute nodes via *bridge nodes*. In each subset, 128 compute nodes shares two designated bridge nodes and one I/O forwarding node.

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 Network Storage Disks (NSDs) for metadata and data services, respectively. In the data pool, 48 NSD servers manage 336 (48×7) NSDs in a round-robin way.

To absorb bursts in parallel, GPFS stripes burst data across NSDs in its data pool, as shown in Figure 3a.

GPFS striping policy. For a burst, GPFS partitions the data into a sequence of blocks and distributes the block sequence across an NSD sequence in a round-robin way. The block size (*GPFS block size*) and the NSD sequence in use are not controlled by the users; the GPFS block size is determined at the creation time of a GPFS filesystem; the NSD sequence starts from a randomly chosen NSD and may span over the entire data pool. In Mira-FS1, GPFS block size is configured as 8 MB.

GPFS subblock policy. GPFS manages filesystem fragmentation with *subblocks*. It divides each block into 32 equal-size subblocks. When a file or the last block of a file is less than the block size, GPFS partitions the file/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 file/block size is determined.

2) Lustre on Titan and Atlas2: Titan is a Cray XK7 machine at the OLCF connected to a Lustre filesystem Spider 2, shown in Figure 2b.

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 is a center-wide Lustre file system with two partitions (Atlas1 and Atlas2), each having a metadata server (MDS) and 1,008 object storage targets (OSTs). An MDS stores the metadata of files and objects on RAID devices, called metadata targets (MDTs). We focus on Atlas2 in which 144 object storage servers (OSSes) manage the I/O traffic for 1,008 OSTs (144×7) in a round-robin way.

Unlike Blue Gene machines that utilize bridge nodes and I/O forwarding nodes to route I/O operations, compute nodes in Titan access Spider 2 via 172 I/O routers. The routers are evenly distributed through the torus and route I/O traffic statically [12], [38]: a compute node is connected to a fixed group of "closest" I/O routers. Figure 2 illustrates the distinction between the architectures of Cetus and Titan I/O subsystems.

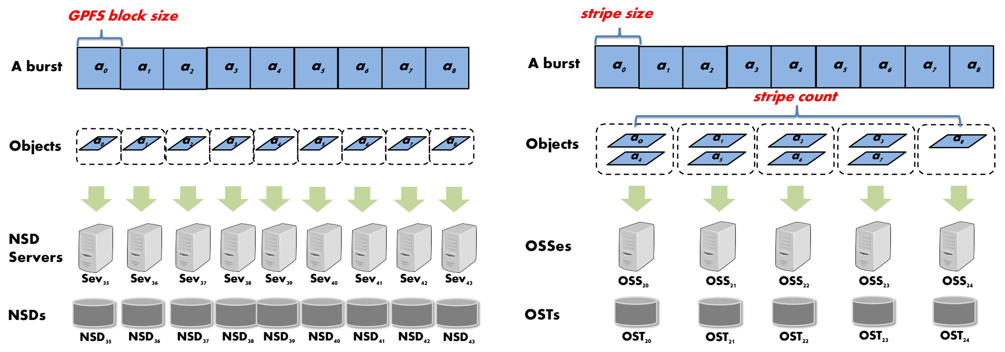
Different from GPFS, data striping in Lustre is user controlled, as shown in Figure 3b.

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*. As its default configuration, Atlas2 takes 1 MB as its stripe size, 4 as its stripe count, and a randomly chosen starting OST.

3) Observations: Across two representative filesystems, we find the following:

First, from the view of I/O writes, supercomputers and their filesystems are multistage write paths, including the stages from compute node to storage target (Figure 2).

Secondly, for a write pattern on a write path, the aggregate load on each stage can be predicted based on the numbers of nodes/cores in use and the burst size.



(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

Thirdly, for a write pattern of a job on a supercomputer, the compute-node locations are known at job allocation; the resources in use (e.g., the number of routers) and load distribution among the resources for both compute nodes and network interconnect are known based on a supercomputer's network configuration.

Finally, for a write pattern of a job on a filesystem, the resources in use and load distribution on storage servers/targets can be estimated based on the write pattern, striping policy and server-target mapping.

III. Modeling Write Performance of Large-scale Parallel Filesystems

A. Features

1) Overview: In this study, we build predictive models to address the mean write time (t) as a function of features (x_1, x_2, ..., x_n), as shown in Formula 1.

$$t = f(x_1, x_2, ..., x_n) \tag{1}$$

Consider a write pattern (discussed in §II-A) that produces $m \times n$ concurrent K-bytes bursts/files from n nodes with m cores per node. For this pattern, each model feature is an I/O write characteristic that affects the accuracy of machine learning models for write performance prediction.

Since the target systems are multistage write paths (Figure 2), a write time on a path is the sum of times spent on each of its stages. We build model features on the *performance-related parameters* that potentially affect the time cost on each stage. In this study, we take three such parameters: *aggregate load*, *load skew*, and *resources in use*. For each performance-related parameter, we derive two features for positive and inverse correlations[4].

Specifically, for metadata stage on a path, aggregate load means the number of overall metadata operations generated by a write pattern. For the remaining stages, aggregate load means the total bytes of data generated by the pattern. Moreover, we define load skew on a stage based on the straggler. Specifically, we measure the maximum load (maximum number of metadata operations or maximum bytes of data) of a single component on a stage and use it for load skew. For example, for I/O router stage, load skew is the maximum bytes of data processed by a single router. Finally, for a stage we consider the number of components used as resources in use. For example, at compute node stage, resources in use means the number of compute nodes issuing output bursts. As our follow-on analysis shows, load skew is an important factor to consider for prediction accuracy.

2) Features for GPFS Filesystems: Figure 2a presents Cetus/Mira-FS1 as an example of GPFS deployments. There are eight stages on the write path. Table I summarizes the features for each of the eight stages. We build these features based on our observations (II-B3) and the write operations discussed in III-A1.

Metadata stage. Aggregate load of metadata service ($m \times n$) and subblock service ($m \times n \times p_{sub}$) can be estimated by the write pattern and GPFS subblock policy. Here, p_{sub} represents the number of subblocks per burst. Resources in use are the number of I/O nodes (n_{io}). Load skews of metadata ($s_{io} \times m$) and subblock ($s_{io} \times m \times p_{sub}$) can be estimated by the number of cores per node (m), the locations of the n nodes, and the compute node to I/O node mapping. s_{io} represents the size of the largest node group in the n engaged nodes that connect to the same I/O node.

Stages within supercomputers (compute node, bridge node, link, I/O node). Aggregate load ($m \times n \times K$) can be estimated by the write pattern. The numbers of compute nodes in use (n), bridge nodes in use (n_b), links in use (n_l), and I/O nodes in use (n_{io}), load skew on compute nodes ($m \times K$), bridge nodes ($s_b \times m \times K$), links ($s_l \times m \times K$), and I/O nodes ($s_{io} \times m \times K$) can be estimated by the locations of the n compute nodes and the network mapping configurations from compute node to

[4]We take only positive features for subblock-related parameters on Cetus/Mira-FS1: when a burst has no subblock (e.g., 8 MB burst), the positive feature value is 0.

bridge node, to link, and to I/O node, respectively. Here, s_b and s_l represent the sizes of the two largest node groups in the n nodes that connect to the same bridge node and the same link respectively.

Infiniband network stage. Aggregate load is $m \times n \times K$. Since Infiniband network serves I/O writes as a whole, at this stage we does not consider load skew and resources in use.

Stages within GPFS filesystems (NSD server, NSD). For each of the concurrent $m \times n$ bursts, the number of NSDs (n_d) used can be estimated by the burst size (K), GPFS block size, and GPFS striping policy. The number of NSD servers (n_s) used can be estimated by n_d and the mapping from NSD servers to NSDs. Moreover, according to the striping policy, the $m \times n$ bursts choose starting NSDs randomly and independently. Theoretically, for these bursts, the numbers of NSDs (n_{nsd}) and NSD servers (n_{nsds}) used are both random. Statistically, the numbers are bound to $m \times n$, n_d, n_s: (1) the more bursts ($m \times n$) a write produces, the larger n_{nsd}, n_{nsds} it is likely to use; and (2) the more NSDs/NSD servers a burst uses (n_d, n_s), the larger n_{nsd}, n_{nsds} the $m \times n$ bursts are likely to use. Thus, we estimate $n_{nsds} = m \times n \times n_s$, $n_{nsd} = m \times n \times n_d$.

For the stages of the NSD server and NSD, we develop features only on aggregate load and resources in use and does not consider load skew on these two stages. 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. This suggests that the GPFS striping policy performs effectively in balancing the 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 develop cross-stage features for correlated stages. Compared with CPU resources on supercomputers, I/O bandwidth is a shared and finite resource; I/O bottlenecks may occur on one or more stages concurrently [37], [39]. To address the concurrent bottlenecks from multiple stages, we consider adjacent stages on the write paths as correlated stages, and we 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 $(m \times K) \times (s_b \times m \times K)$.

Moreover, I/O interference is inevitable under production load. We build features for it by following the observations on Titan [39]. Specifically, the interference is positively correlated to the number of compute nodes in use (n) and inversely correlated to the aggregate burst size ($\frac{1}{m \times n \times K}$). Thus, for the target environment, we build three features for interference: n, $\frac{1}{m \times n \times K}$, and $\frac{n}{m \times n \times K}$.

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

3) Features for Lustre Filesystems: Figure 2b presents Titan/Atlas2 as an example of Lustre filesystems. There are six stages on the write path. Table II reports the features for each of the six stages. Similar to GPFS deployments, we build features based on observations (§II-B3) and the write patterns discussed in §III-A1.

Metadata stage. The aggregate load is $m \times n$. Since metadata server provides services as a whole, at this stage we does not consider load skew and resources in use.

Stages within supercomputers (compute node, I/O router). The aggregate load is $m \times n \times K$. The number of I/O routers used (n_r) and the load skew on compute nodes ($m \times K$) and I/O routers ($s_r \times m \times K$) can be estimated by the locations of the n compute nodes and the mapping from compute node to I/O router. Here s_r represents the largest node group in the n nodes that connect to the same I/O router.

SION stage. Aggregate load is $m \times n \times K$. Similar to the Infiniband network stage on Cetus/Mira-FS1 (§III-A2), at this stage we only build features on aggregate load.

Stages within Lustre filesystems (OSS, OST). The aggregate load ($m \times n \times K$) can be derived from the write pattern. The number of OSTs used (n_{ost}) can be estimated by the write pattern and the configurations on the Lustre striping policy. The number of OSSes (n_{oss}) used can be estimated by n_{ost} and the mapping from OSSes to OSTs. The load skew on OSTs (s_{ost}) and OSSes (s_{oss}) can be estimated by the striping configurations and OSS-OST mapping.

Similar to §III-A2, we build cross-stage features for adjacent stages on the Lustre write path, and for interference as: n, $\frac{1}{m \times n \times K}$, and $\frac{n}{m \times n \times K}$.

In summary, a Lustre write path has 30 features: 24 for individual stages, 3 for correlated stages, and 3 for interference.

B. Cross-Platform Modeling Methodology

1) Feature Selection: Feature selection is widely used to improve feature or model interpretability and enhance model accuracy with the right subset of features. Selection models can be divided into three major groups: filter methods, wrapper methods, and embedded methods [26]. In particular, embedded methods were developed recently to overcome the issues of collinearity and overfitting of the other two methods.

In this study, we use Lasso from embedded methods to both select features and train models. Lasso is built on linear relationships. It solves a least square problem by shrinking the absolute sum of the coefficients for features, with *shrinkage parameter* λ.

Compared to our earlier prediction work on Titan [39], this study chooses Lasso over linear regression for its better interpretation and higher model accuracy (discussed in §IV). Such like decision tree and Random forest, other learning techniques can also be used for performance modeling and feature selection. We leave the further exploration on other techniques as our future work.

2) Building Regression Models: In this study, we model write time (y) as a function (f) of dataset X. Specifically,

Performance-Related Parameter	Metadata Cost		Data-Absorption Cost						
Stage	Metadata	Subblock	Compute Node	Bridge Node	Link	I/O Node	Network	NSD server	NSD
Aggregate Load	$m \times n$, $\frac{1}{m \times n}$	$m \times n \times p_{sub}$	$m \times n \times K$, $\frac{1}{m \times n \times K}$						
Load Skew	$s_{io} \times m$, $\frac{1}{s_{io} \times m}$	$s_{io} \times m \times p_{sub}$	$m \times K$, $\frac{1}{m \times K}$, K, $\frac{1}{K}$	$s_b \times m \times K$, $\frac{1}{s_b \times m \times K}$	$s_l \times m \times K$, $\frac{1}{s_l \times m \times K}$	$s_{io} \times m \times K$, $\frac{1}{s_{io} \times m \times K}$			
Resources in Use	n_{io}, $\frac{1}{n_{io}}$		n, $\frac{1}{n}$, m, $\frac{1}{m}$	n_b, $\frac{1}{n_b}$	n_l, $\frac{1}{n_l}$	n_{io}, $\frac{1}{n_{io}}$		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 I: Features of individual stages of GPFS deployments. For each stage, we derive three performance-related parameters. For a parameter, we build two features to address positive and inverse correlations.

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	m, $\frac{1}{m}$	$m \times K$, $\frac{1}{m \times K}$, K, $\frac{1}{K}$	$s_r \times m \times K$, $\frac{1}{s_r \times m \times K}$		s_{oss}, $\frac{1}{s_{oss}}$	s_{ost}, $\frac{1}{s_{ost}}$
Resources in Use	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}, $\frac{1}{n_{ost}}$

TABLE II: Features of individual stages on Lustre deployments. For each stage, we derive three performance-related parameters. For a parameter, we build two features to address positive and inverse correlations.

Scale (n)	Cores per Node (m)	Burst Size (K)
1, 2, 4, 8, 16, 32, 64, 128, 200, 256, 400, 512	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

TABLE III: Generating 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.

Scale (n)	Cores per Node (m)	Burst Size (K)	*stripe_count* ((W))
1, 2, 4, 8, 16, 32, 64, 128, 200, 256, 400, 512	8 *from* 16	1MB—5MB 6MB—25MB 25MB—100MB 101MB—250MB 251MB—500MB 501MB—1024MB 1 025MB—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

TABLE IV: Generating write patterns on Titan/Atlas2. The first row generates data for both training and testing; The second row generates larger bursts for training purpose only.

we use Lasso() from the scikit-learn toolkit [29] to train the models.

3) Searching for the best model: We search for the best model from a regression model space, trained across training sets and the values of model parameters. Specifically, Lasso has one model parameter (shrinkage parameter λ);

We evaluate the trained models by their mean square error (MSE) on the validation set. In this study, we train models on small-scale writes and test the trained models on large-scale writes; we use the randomly chosen samples of test data as the validation set; the set size is 20% of the test set.

For each two trained models in Lasso, we consider one model is better if its MSE is smaller. Following the same rule, we search for the best models with minimum MSEs.

IV. Experimental Evaluations

This section presents the evaluation of our machine learning approach on Cetus/Mira-FS1 and Titan/Atlas2. We use IOR as a load generator to produce data for model training, validation, and testing. We conduct experiments on Cetus in Aug. 2017 — Feb. 2018 and on Titan in Dec. 2016 — Feb. 2018, respectively.

A. Experiment Data

We train models with small-scale writes ($\leq$128 compute nodes[5]) and test them on medium-scale writes (200–512 nodes). Tables III and IV present the write patterns generated on GPFS/Mira-FS1 and Titan/Atlas2, respectively.

We generate samples from these patterns by following the *semi-random sampling method* proposed in [39]. Moreover, to address performance variability, in this study we further categorize the collected samples into *converged* and *unconverged* samples.

In particular, for a write scale (n) on Cetus/Mira-FS1, we generate samples for different write patterns by varying the number of cores per node (m) and burst size (K). For a write scale on Titan/Atlas2, we produce samples for different patterns by varying m, K and W (stripe count). Thus, a write pattern is a combination of (m, n, K) on Cetus/Mira-FS1 and a combination of (m, n, K, W) on Titan/Atlas2, respectively. For each pattern on Cetus, we use m cores from all of the possible choices; for a pattern on Titan, we choose m cores randomly in the range of 1 — 16 cores. For both of the systems, we design the burst-size ranges based on our observations from the machines hosted by the ALCF. For a pattern, we choose K-bytes from a burst-size range randomly. Moreover, for the write patterns on Titan/Atlas2, we take the Lustre configuration as 1MB (stripe size)[6], W, and the starting OSTs as an OST sequence [39].

For each write pattern, we repeat the identical IOR executions and collect the write time of each execution from the minimum of *file_open()* to the maximum of *file_close()* among the coordinated write bursts. A sample is the mean write time of a pattern. We estimate the sample convergence by the central limit theorem [28] for its effectiveness on the dataset with an unknown mean.[7] In the central limit theorem, we take a confidence level =0.9 (Titan) and =0.98 (Cetus) and

[5]We choose the number 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) (§II-B).

[6]Stripe size does not affect write performance when setting in 1MB—32MB, or leads to performance degradation for the values >32MB [37].

[7]In this study, the mean time of a sample is unknown beforehand.

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}
				m	$s_l \times m \times K$	$s_b \times m \times K$	$m \times n$	$m \times K$	n_{nsds}	$s_{io} \times m \times K$	n_{nsd}	$(s_l \times m \times K) \times (s_b \times m \times K)$	$(s_b \times m \times K) \times n_{nsds}$
$Lasso_{best\_titan}$	{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	$s_r \times m \times K$		s_{ost}		$m \times n \times K$	$m \times K$	$(m \times K) \times (s_r \times m \times K)$	$(s_r \times m \times K) \times n_{oss}$

TABLE V: Best Lasso models. The models are defined in §IV-B. Each row reports the information of a model, including the training set, value of the shrinkage parameter (λ), intercept and selected features.

error estimator =0.1 for both of the systems by following the conventional use. To reduce benchmarking cost, we terminate a pattern's IOR execution if the pattern reaches a number of repetitions but its sample is not converged. Specifically, we take 9 as the termination threshold based on the statistics of Darshan logs [4] at the ALCF and in consideration of limited I/O impact on production load.

We focus on writes >5 seconds, since in production runs smaller writes are usually hidden by the client-side kernel buffer of page cache and have little impact on application performance. For Cetus/Mira-FS1 and Titan/Atlas2, we produced 4,715 and 3,465 converged samples for training, and 874 and 668 samples for testing (including both converged and unconverged samples), respectively. For Cetus/Mira-FS1, the sample size of a write scale ranges from 553 to 660 samples for training and from 73 to 371 samples for testing; for Titan/Atlas2, the sample size of a write scale ranges from 346 to 545 samples for training and from 142 to 281 samples for testing.

B. Chosen Models and Features

For each target system, we train models across 255 training sets, each set a combination of datasets built on different write scales on 1–128 compute nodes and generated by the write patterns in Tables III and IV. We vary the shrinkage parameter in Lasso from 0 to 10 and identify two best models: $Lasso_{best\_cetus}$, $Lasso_{best\_titan}$, representing the best models located by the learning approach on Cetus and Titan, respectively. We list the details of the best models in Table V.

To demonstrate the effectiveness of our approach and the performance of Lasso, we also address three additional models for each target system, including the Lasso models trained on the write scales of 1—128 compute nodes ($Lasso_{base\_cetus}$, $Lasso_{base\_titan}$), the linear models trained on the scales of 1—128 compute nodes ($Linear_{base\_cetus}$, $Linear_{base\_titan}$), and the best linear models identified by our learning approach ($Linear_{best\_cetus}$, $Linear_{best\_titan}$).

C. Model Evaluation

We use *relative true error* (ϵ) as the error estimator. Consider that the mean time of the ith sample is t_i and its prediction result is t'_i.

$$\epsilon_i = \frac{t'_i - t_i}{t_i} \tag{2}$$

Then $\epsilon_i > 0$ suggests that t_i is overestimated; $\epsilon_i < 0$ suggests that t_i is underestimated; and $\|\epsilon_i\|$ quantifies prediction accuracy: smaller $\|\epsilon_i\|$ indicates higher accuracy. We focus on $\|\epsilon\|$=20% and =30% in consideration of two factors: (1) these two thresholds are widely used as the conventional numbers in accuracy measurements in statistics [10] and (2) for HPC applications, the time cost on I/O writes is expected to be ~10% of the total runtimes (discussed in Section II-A). With a 20%–30% prediction error, we expect the guaranteed I/O cost to be 7%–13% of the total runtimes, which is generally acceptable for production runs [5], [36], [38], [39].

For each target system, we evaluate the models each on three test sets, with two sets for converged samples and one for unconverged samples (discussed in §IV-A). The two converged sets are grouped on the write scales of the samples: small set (on 200, 256 nodes), medium set (on 400, 512 nodes). For small and medium sets, each set has 483, 399 samples on Cetus/Mira-FS1 and 423, 414 samples on Titan/Atlas2. The unconverged sets are generated on the write scales on 200–512 nodes, including 98 and 47 samples for Cetus and Titan, respectively. Figures 4 and 5 plot the error accuracy of the models on the two converged test sets; Table VI presents the accuracy summary for all sets.

In summary, we find our approach is valid and draw two major conclusions.

1. The approach locates the best models. Compared to the linear models, the Lasso models are more accurate, and the best Lasso models attain the highest accuracy on both of the target systems. In particular, as presented in Table VI and for a set of the small/medium sets, 85.56%–98.56% of the samples report $\|\epsilon\|$ ≤30% for $Lasso_{best\_cetus}$; 95.58%–99.65% of the samples report $\|\epsilon\|$ ≤30% for $Lasso_{best\_titan}$.

2. The approach identifies the effective features. As shown in Table V, we find that, (a) the write behaviors of Cetus/Mira-FS1 are dominated by the metadata load and the load skews within Cetus, and the resources used in Mira-FS1; and (b) the write behaviors of Titan/Atlas2 are heavily determined by the aggregate load, the load skews, and resources used within Titan, and the load skew within Atlas2.

D. Inaccurate Estimates

For $Lasso_{best\_cetus}$ and $Lasso_{best\_titan}$, 11.1% and 6.29% of the overall samples on Cetus/Mira-FS1 and Titan/Atlas2 report $\|\epsilon\|$ >30%, respectively. Moreover, we observe that 90.72% of these inaccurate samples for Cetus and 100% for Titan are underestimated (ϵ <-30%).

We focus on the underestimated samples and find that for Cetus 53.1% of the samples are generated by the write patterns from 16 cores per node; for Cetus and Titan, 90.8% and 71.43% of the samples are generated by 400 and 512 compute nodes, respectively. In order to reduce these inaccurate samples, adaptive I/O libraries such as ADIOS can be used to

Target System	Model	small set		medium set		unconverged samples	
		$\|\epsilon\| \leq 20\%$	$\|\epsilon\| \leq 30\%$	$\|\epsilon\| \leq 20\%$	$\|\epsilon\| \leq 30\%$	$\|\epsilon\| \leq 20\%$	$\|\epsilon\| \leq 30\%$
Cetus/Mira-FS1	$Lasso_{best\_cetus}$	97.12%	98.56%	65.56%	85.56%	41.84%	60.2%
Titan/Atlas2	$Lasso_{best\_titan}$	98.61%	99.65%	96.10%	98.2%	19.15%	25.53%

TABLE VI: Summary of model accuracy

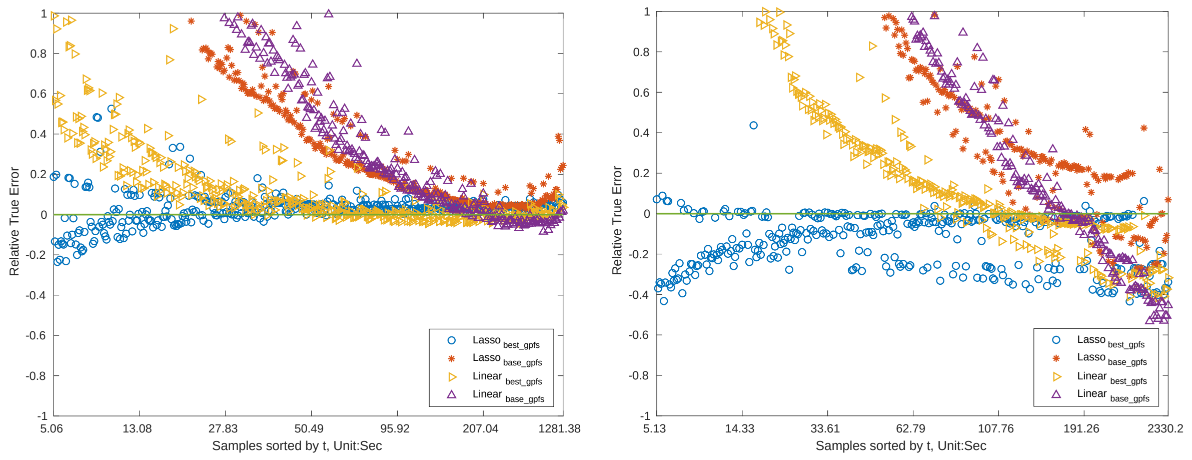


Fig. 4: Evaluation of the Lasso models on Cetus/Mira-FS1. From left to right, the 2 subfigures present the accuracy of the two Lasso models (Table V) on small and medium test sets on Cetus/Mira-FS1, respectively; a subfigure plots the relative true errors (ϵ) of the models on a test set; the errors are sorted along the x-axis based on t. The test sets ϵ and t are defined in §IV-C.

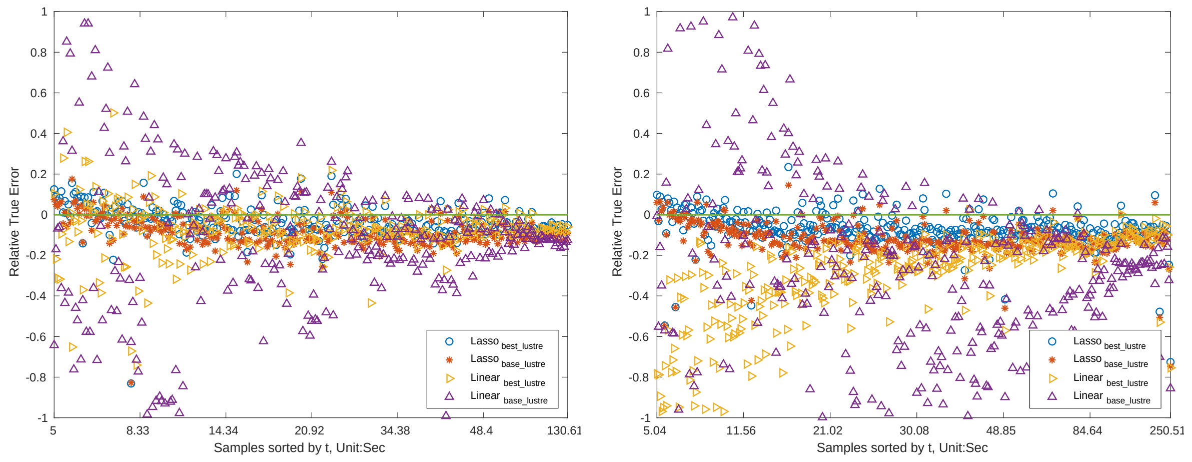


Fig. 5: Evaluation of the Lasso models on Titan/Atlas2. From left to right, the 2 subfigures present the accuracy of the two Lasso models (Table V) on small/medium/large test sets on Titan/Atlas2, respectively; a subfigure plots the relative true errors (ϵ) of the models on a test set; the errors are sorted along the x-axis based on t. The test sets ϵ and t are defined in §IV-C.

adapt write bursts to a smaller number of cores of a node for Cetus or to a smaller write scale for both Cetus and Titan. We leave this study for our future work.

V. Related Work

1. I/O performance studies. The Darshan team from Argonne National Laboratory [4], [21] investigated the I/O behavior of supercomputer platforms by analyzing logs and traces collected by continuous monitoring software installed on compute nodes. Xie et al. [37] and Uselton et al. [33] take similar approaches by adopting a statistical method to analyze the I/O behavior of large-scale parallel filesystems; they report on the straggler effects on the high degree of parallelism of large-scale I/O operations. The I/O performance of Lustre-based file systems has also been studied by several researchers. Liu et al. [16] probe the application behaviors by analyzing the server-side I/O logs. Logan and Dickens [20] address the signatures of I/O performance obstacles on scientific codes with MPI-IO. Yu et al. [41] characterize the I/O behaviors of Titan's predecessor Jaguar for both individual components and the entire system. Kim et al. [12] study the combined application behavior on the server side by monitoring the performance of storage servers.

2. Techniques on I/O performance improvement. To improve the performance of Lustre-based file systems, Shipman et al. devise a class of techniques to support the following: a multithreading pool in OSC [31], asynchronous journaling[25], and a fine-grained, load-balancing and network routing algorithm[9]. Ren et al. focus on optimizing the I/O performance by reducing the cost of metadata operations[27], [18]. Researchers have also developed middleware systems (e.g., ADIOS) [7], [13], [19] at the application level to improve the I/O performance by adapting the I/O patterns and configurations. Their work is complementary to ours in that our prediction results can be directly used as the input to their systems to guide I/O adaptation and configuration.

3. Machine learning studies on HPC I/O. A group of studies [3], [8], [14], [15], [23] adopt learning models to predict the I/O behaviors of HPC applications. In particular, Behzad et al. [3] and Kumar et al. [14] are among the first to use machine learning techniques to tune configurable I/O parameters at application level. Different from these works, we build machine learning models to predict I/O performance with the features derived from I/O access patterns, system design and architecture. Kunkel et al. [15] build decision trees to predict applications' noncontiguous I/O behaviors and further determine the parameter combinations for data sieving in ROMIO [32]. Omnisc'IO [8], 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 with 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 use learning models to predict the I/O performance variability of supercomputers. For example, Wan et al. [35] build a hidden Markov model for a Lustre filesystem to learn the variability of I/O latencies from a single client to a single OST; and Madireddy [22] et al. use a 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 and (2) deriving features to address I/O interference from competing loads.

VI. Conclusions

In this paper, we have presented 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 multistage write paths; introduce a convergence-guaranteed sampling method to produce a training set space with a good variety and low benchmarking cost; and describe an approach to search for the best model from a rich model space for each target write path. We measure our approach on two production supercomputers, Titan and Cetus; the results suggest that the approach can locate good models with high prediction accuracy for large-scale writes on both machines.

In future work we will investigate how these strategies can be applied to a wider variety of HPC I/O workloads, and we will continue to evaluate their accuracy on additional platforms.

VII. Acknowledgments

We are thankful to the anonymous reviewers and our shepherd Julian Kunkel for their invaluable input, and to many staff at ALCF and NCCS for answering numerous questions on supercomputers Cetus and Titan.

This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This work 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 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. This work used resources of Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References

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

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

[3] B. Behzad, H. V. T. Luu, J. Huchette, S. Byna, R. Aydt, Q. Koziol, M. Snir, et al. Taming parallel I/O complexity with auto-tuning. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13)*, 2013, pp. 68–79.

[4] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley. 24/7 characterization of petascale I/O workloads. In *Proceedings of 2009 IEEE International Conference on Cluster Computing (Cluster'09)*, 2009, pp. 1–10.

[5] 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 125*, 1 (2008), 012042.

[6] 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 2*, 1 (Jan. 2009), 015001.

[7] A. Choudhary, W. Liao, K. Gao, A. Nisar, R. Ross, R. Thakur, and R. Latham. Scalable I/O and analytics. *Journal of Physics: Conference Series 180*, 1 (2009), 012048.

[8] 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 (SC'14)*, 2014, pp. 623–634.

[9] M. Ezell, S. Oral, F. Wang, D. Tiwari, D. Maxwell, D. Leverman, and J. Hill. I/O router placement and fine-grained routing on Titan to support Spider II. In *Proceedings of the Cray User Group Conference (CUG'14)*, 2014, pp. 1–6.

[10] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*. Springer series in statistics New York, 2001.

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

[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 *Proceedings of the 5th Petascale Data Storage Workshop (PDSW'10)*, 2010, pp. 1–5.

[13] S. Kumar, J. Edwards, P.-T. Bremer, A. Knoll, C. Christensen, V. Vishwanath, P. Carns, J. A. Schmidt, and V. Pascucci. Efficient I/O and storage of adaptive-resolution data. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*, 2014, pp. 413–423.

[14] S. Kumar, A. Saha, V. Vishwanath, P. Carns, J. Schmidt, G. Scorzelli, H. Kolla, R. Grout, R. Latham, R. Ross, et al. Characterization and modeling of PIDX parallel I/O for performance optimization. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13)*, 2013, pp. 67–78.

[15] J. Kunkel, M. Zimmer, and E. Betke. Predicting performance of non-contiguous I/O with machine learning. In *Proceedings of the International Conference on High Performance Computing (ISC'15)*, 2015, pp. 257–273.

[16] Y. Liu, R. Gunasekaran, X. Ma, and S. S. Vazhkudai. Automatic identification of application I/O signatures from noisy server-side traces. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST'14)*, 2014, pp. 213–228.

[17] 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 *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW'17)*, 2017, ACM, pp. 55–60.

[18] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Adaptable, metadata-rich I/O methods for portable high performance I/O. In *Proceedings of the 23rd IEEE International Parallel & Distributed Processing Symposium (IPDPS'09)*, 2009, pp. 1–10.

[19] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf. Managing variability in the I/O performance of petascale storage systems. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, 2010, pp. 1–12.

[20] J. Logan, and P. Dickens. Towards an understanding of the performance of MPI-IO in Lustre file systems. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'08)*, 2008, pp. 330–335.

[21] 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 (HPDC'15)*, 2015, pp. 33–44.

[22] 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 *Proceedings of the International Conference on High Performance Computing*, 2018, pp. 184–204.

[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 *Proceedings of the 2016 IEEE International Conference on Cluster Computing (CLUSTER'16)*, 2016, pp. 255–258.

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

[25] S. Oral, F. Wang, D. Dillow, G. Shipman, R. Miller, and O. Drokin. Efficient object storage journaling in a distributed parallel file system. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*, 2010, pp. 143–154.

[26] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence 27*, 8 (2005), 1226–1238.

[27] K. Ren, Q. Zheng, S. Patil, and G. Gibson. Indexfs: scaling file system metadata performance with stateless caching and bulk insertion. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*, 2014, pp. 237–248.

[28] M. Rosenblatt. A central limit theorem and a strong mixing condition. *Proceedings of the National Academy of Sciences 42*, 1 (1956), 43–47.

[29] scikit learn. scikit-learn:machine learning in Python. http://scikit-learn.org/.

[30] 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 *Rroceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS'19)*, 2019, pp. 511–522.

[31] G. Shipman, D. Dillow, D. Fuller, R. Gunasekaran, J. Hill, Y. Kim, S. Oral, D. Reitz, J. Simmons, and F. Wang. A next-generation parallel file system environment for the OLCF. In *Proceedings of the Cray User Group Conference (CUG'12)*, 2012, pp. 1–12.

[32] R. Thakur, W. Gropp, and E. Lusk. Data sieving and collective I/O in ROMIO. In *Frontiers of Massively Parallel Computation, 1999. Frontiers' 99. The Seventh Symposium on the*, 1999, IEEE, pp. 182–189.

[33] A. Uselton, M. Howison, N. J. Wright, D. Skinner, N. Keen, J. Shalf, K. L. Karavanic, and L. Oliker. Parallel I/O performance: from events to ensembles. In *Proceedings of the 24th IEEE International Parallel & Distributed Processing Symposium (IPDPS'10)*, 2010, pp. 1–11.

[34] S. S. Vazhkudai, R. Miller, D. Tiwari, C. Zimmer, F. Wang, S. Oral, R. Gunasekaran, and D. Steinert. GUIDE: a scalable information directory service to collect, federate, and analyze logs for operational insights into a leadership HPC facility. In *Proceedings of the 2017 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'17)*, 2017, pp. 45–54.

[35] 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 *Proceedings of the 19th IEEE International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2017, pp. 1–9.

[36] B. Xie. *Output Performance of Petascale File Systems*. PhD thesis, Duke University, 2017.

[37] 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 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'12)*, 2012, pp. 1–11.

[38] 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 I/O in the Data Center Workshop (HPC-IODC'17)*, 2017, pp. 187–200.

[39] B. Xie, Y. Huang, J. Chase, J. Y. Choi, S. Klasky, J. Lofstead, and S. Oral. Predicting output performance of a petascale supercomputer. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing (HPDC'17)*, 2017, pp. 181–192.

[40] B. Xie, S. Oral, C. Zimmer, J. Y. Choi, D. Dillow, S. Klasky, J. Lofstead, N. Podhorszki, and J. S. Chase. Characterizing output bottlenecks of a production supercomputer: Analysis and implications. *ACM Transactions on Storage (TOS'19) 15*, 4 (2019), 1–39.

[41] W. Yu, J. Vetter, and S. Oral. Performance characterization and optimization of parallel I/O on the Cray XT. In *Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08)*, 2008, pp. 1–11.