Prediction and Interpretability of HPC I/O Resources Usage with Machine Learning

Alexis Bandet, Francieli Boito, Guillaume Pallez

Prediction and Interpretability of HPC I/O Resources Usage with Machine Learning

Alexis Bandet, Francieli Boito
Univ. Bordeaux, CNRS, Bordeaux INP, Inria, LaBRI, UMR 5800, F-33400
Talence, France
{alexis.bandet, francieli.zanon-boito}@inria.fr

Guillaume Pallez
Inria
Rennes, France
guillaume.pallez@inria.fr

***Abstract*—I/O management tools proposed for high-performance computing (HPC) environments usually rely on accurate I/O bandwidth estimations to make their decisions. However, finding the correct I/O subsystem configuration that provides this maximal I/O bandwidth is particularly hard. In this work, we focus on finding a good estimate of the number of I/O resources (e.g., OSTs and I/O nodes) that provides the maximal bandwidth while minimizing the system occupation and taking into account the natural I/O variability. We use machine learning techniques to do so, focusing on intrinsic application features and system configurations. We show I/O resource usage is predictable and further study the impact of different features. We also validate our models with four I/O kernels from real applications. Finally, we show that our model, when used for resource allocation, can improve application performance.**

***Index Terms*—High-Performance Computing, Parallel I/O, Parallel File System, performance prediction, resource allocation**

I. Introduction

High-performance computing (HPC) facilities often use remote shared Parallel File Systems (PFS), such as Lustre or BeeGFS to store persistent data. PFS cuts files into chunks, called stripes, and distributes them across several Object Storage Targets (*OSTs*). Each OST can be accessed by multiple applications at the same time, which leads to contention [25], [26].

The fact that I/O performance depends on what others are doing leads to higher performance variability and less predictability in job duration [11], [17], which complicates resource management. Many techniques have been proposed to reduce contention, including scheduling accesses to the PFS [8], burst buffers [1], and I/O-aware batch scheduling [13].

The problem of avoiding contention is made more complicated by the fact that applications' I/O performance (i) may behave differently with different numbers of I/O resources; and (ii) may behave differently depending on the run (even in isolation). This is illustrated in Fig. 1, where we can see the performance of 12 different access patterns as a function of the number of OSTs being used (also called the *stripe count*). While some applications may benefit from using more OSTs, others may not be sensitive to this parameter, or even be harmed by extra resources. The same phenomenon was observed in the literature for other I/O resources such as I/O nodes [15], [3].

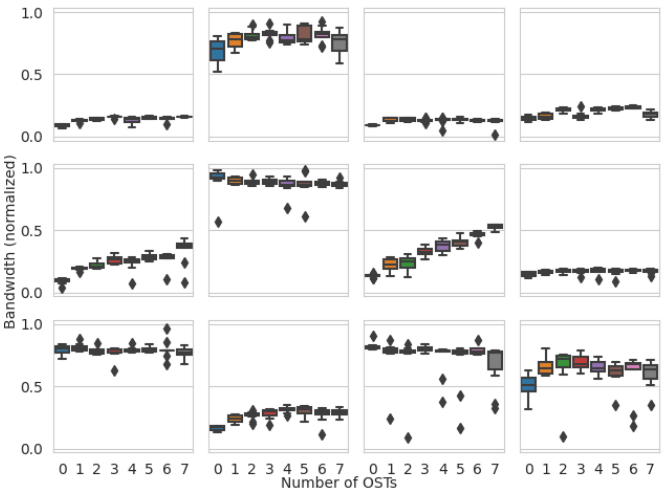


Fig. 1: Performance of 12 access patterns as a function of the number of used OSTs, normalized by the system's peak I/O performance ($\times$12 GB/s). Details about these experiments are presented in Section II-A.

In addition, it has been shown by Bandet et al. [2] that it may not be in the system best interest to select the number of OST that maximizes the I/O bandwidth of an application, disregarding the impact on other applications. Indeed, while allocating too few would prevent some applications from reaching their best possible performance, allocating too many would induce more contention by making applications share more resources unnecessarily.

In this work, our goal is to provide a way to predict the behavior of these I/O patterns based on limited information on the I/O accesses, and hence to provide a way for resource manager to opt for the number of I/O resources to allocate to each applications. Our solution accounts for the intrinsic limits of I/O variability.

Recently, Bandet et al. [2] have demonstrated that scheduling techniques could be developed for dealing with concurrent applications sharing distributed I/O resources (OSTs, I/O nodes), even if the bandwidth information was inaccurate. Nonetheless, so far the only way of knowing the application's performance behavior as a function of the number of I/O

resources is to run the application multiple times with different numbers of resources. That can be prohibitive in time, and the extra profiling cost could offset the improvements brought by better allocation algorithms. Instead, in this paper, we propose to use an offline-trained model to predict applications' I/O performance behavior. Such a model can then be used by an online scheduler to categorize applications and take decisions accordingly.

Our main contributions can be summarized as follows.

- We provide a large dataset of write performance measurements with various I/O configurations on BeeGFS as a function of the stripe count.
- We provide a theoretical framework for the definition of *peak bandwidth prediction* when there is variability between repetitions of an experiment.
- We provide a solution to predict I/O performance behavior as a function of the number of I/O resources based on application characteristics. We also compare different machine learning techniques to do so.
- We analyze the importance of different input features, identifying the ones that impact I/O performance behavior the most.
- We validate our model with four I/O kernels from real applications, and also show our model can be effectively used for OST allocation through a use case.

The rest of the paper is organized as follows. Section II presents our methodology, the data set used for this research and the prediction models. That is followed by their evaluation in Section III and the OST allocation use case in Section IV. We discuss the state of the art in Section V and final remarks in Section VI.

II. A model for predicting I/O performance behavior

We start by presenting our methodology for creating a data set using a flexible benchmarking tool launched across multiple configurations, and a preliminary analysis of this data, in Section II-A. Then, we define what is the correct number of I/O resources we would like to predict in Section II-B. Lastly, in Section II-C we present the methodology for training and predicting the best configuration for each application with machine learning techniques across multiple data scenarios and the questions we aim to answer in Section III.

A. Data collection and preliminary analysis

The dataset we used for this research contains performance measured with different numbers of OSTs for 528 access patterns (or *configurations*), which were created with the IOR benchmarking tool. It is publicly available at [4].

The 528 configurations were created by varying several I/O variables as described in Table I. We call a configuration a combination of all variables (except Stripe Count). There are 540 possible configurations, however 12 are not valid because the amount of data per process is smaller than the request size, leaving 528 configurations. Each configuration is evaluated ten times with each number of OSTs (1, 2, 3, ... 8). The whole set of tests was executed, one at a time, in a random order over a period of six months.

TABLE I: *Variables* present in the dataset, there are 10 runs for each combination of variables (i.e. configuration).

Variable name	Description	Values
nodes	Number of compute nodes	$\{1, 4, 8, 16\}$
procs	Total number of processes	$\{1, 8, 16\}$ per node
filestrategy	File Strategy	{Shared-file, file-per-process}
reqsize	Request size	{32KiB, 8MiB, 16MiB}
totaldata	Amount of transferred data	{256MiB, 1GiB, 4GiB, 16GiB, 64GiB}
spatiality	Spatiality of accesses	{Contiguous, 1-D strided (only for shared-file)}
ost_number	Stripe count	$\{1, 2, 3, \ldots, 8\}$

The experiments were conducted on the PlaFRIM experimental platform. The values given to the different parameters above were chosen to be lower or equal to values that consistently allow for reaching peak PFS performance in this system. That is the range where the stripe count has an impact, as reaching peak performance means benefiting from using all OSTs.

We used the Bora cluster, composed of 2x18-core Intel CascadeLake with 192 GB of RAM memory. All nodes are connected through a OmniPath 100Gbit/s network that is also used for BeeGFS storage. BeeGFS is deployed over two hosts, used for both data and metadata. Each host executes one OSS with four OSTs and one metadata server. Each OST uses 12 Toshiba AL15SEB18EQY HDDs, each with 1.8 TB of capacity and running at 10,000 RPM. These HDD are organized in RAID-6. We used the system's default stripe size of 512 KiB, and different stripe counts were used by making IOR write to folders that were previously created and configured with each count. The I/O infrastructure of PlaFRIM has been studied in [6].

Preliminary data analysis: The goal of this work is to study whether a tool can predict based on application features the number of OSTs that an application should use. We start by showing that there is no trivial way to obtain this information, hence the need of using machine learning-based solutions.

First, in Fig. 2 we plot the cumulative over each configuration of the smallest number of OSTs whose median behavior over 10 runs gives the peak bandwidth (see Definition 1 for an exact definition). This corresponds to what we want to predict.

As can be seen from this figure, there is no clear stripe count that is optimal independently of the configuration. On the contrary, the choices are evenly distributed among all numbers of OSTs.

The second question that we ask is whether there is a correlation between various variables (from Table I) and the I/O performance. Such a strong correlation could allow to infer behavior.

We show in Fig. 3 the Pearson correlation between all vari-

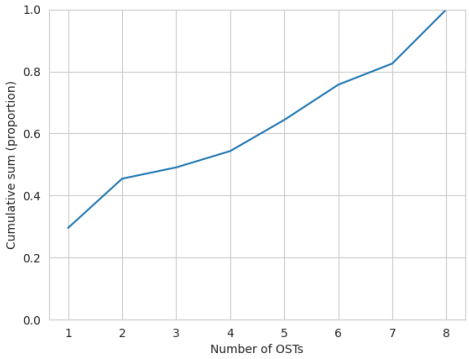


Fig. 2: Cumulative function of the number of OST whose median behavior provides peak bandwidth over each configuration.

Fig. 3: Pearson correlation coefficients between bandwidth and the different application variables, separated by class of behavior.

ables[1] and the I/O bandwidth. We found no strong correlation, meaning that these variable do not determine performance *by themselves*.

From the observations in this section, we conclude that predicting I/O performance behavior according to the number of OSTs is not at all straightforward, as there is no single variable that allows for predicting it, nor a single number of OSTs that resolves the problem.

B. Evaluation of a peak performance predictor

We now discuss the limits of a prediction for the specific context of I/O bandwidth where two identical runs do not produce the same values.

Theoretically, given application A and its $n \to b(n)$ function that gives bandwidth according to n the stripe count, the goal of a predictor would be to predict n_{perf}, the number of I/O resources that maximizes application bandwidth:

$$n_{perf} = \underset{n}{\operatorname{argmax}}\, b(n) \tag{1}$$

Property (I/O variability)**.** *I/O performance presents high variability [22], [20], even in isolation. Hence Equation* (1) *is not well defined since it may be dependent on the run.*

In practice, this means that even if n_0 is the theoretical right stripe count, there could be cases where using n_0 OSTs provides lower performance than if one had used n_1 OSTs. This underlines the difficulty of validating the *correct* prediction.

[1]Because the Pearson correlation does not yield any significant results for categorical variables, for *file strategy* and *spatiality* we arbitrarily transformed the binary variables into -1 and 1.

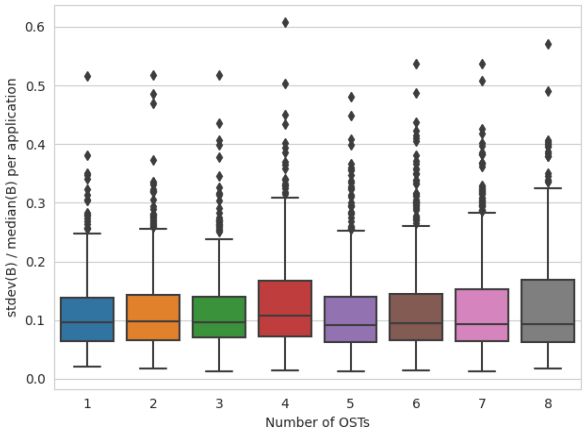


Fig. 4: I/O bandwidth standard deviation divided by median for each scenario, as a function of n.

The first consequence of this is to encourage to use the median value of the bandwidth instead of the maximum value. Let $b^*(n)$ be the median value over hence we define n^*_{perf}, the stripe count whose median performance is the highest among other stripe counts:

$$n^*_{perf} = \underset{n}{\operatorname{argmax}}\, b^*(n) \tag{2}$$

This is what we now target to predict.

The second property of performance with different numbers of OSTs is what we can observe in Fig. 1: many values for stripe count may obtain similar performance. This raises the question of what do we call a *valid* prediction?

This question is a hard question and relies on the objective behind the usage of the predictor. Indeed, even without variability, if by using fewer resources, we lose 0.1% of performance, which is the best prediction, the one with fewer resources or the one with the best performance? How about if we lose 10% performance, but we get to use half the number of resources?

Here we do not resolve this question in general as it is a Pareto optimization problem that only the user can configure. We still decide on a solution for this work, but this is highly reconfigurable, and one could use our techniques with other parameters.

To decide this question, we plot in Fig. 4 the bandwidth variability observed in our dataset, as a function of the number of used OSTs. The first important observation is that variability does not seem dependent on the number of I/O resources, and its median value is around 10%.

Based on this, we select that a prediction is valid if its median performance is within 10% of the median performance of that of n^*_{perf}:

$$n^{\psi}_{perf} \text{ is a valid prediction } \iff b^*(n^{\psi}_{perf}) \geq 0.9 \cdot b^*(n^*_{perf}) \tag{3}$$

Indeed, this means that 50% of the measurements for n^{ψ}_{perf} are larger a value that can generally be expected from n^{*}_{perf} given the variability observed in Fig. 4.

The final step of our decision process is which of these values do we want to predict. At this point, given that we have decided that all these values are *almost* equivalent (they are all valid predictions), and because we want to minimize contention (hence resource occupation), we decide that the *best prediction* is the prediction that minimizes the resource usage. With this we have fully defined what our predictor aims to predict:

Definition 1 (Stripe count predictor/best prediction)**.** *The best prediction we can provide is defined as:*

$$\operatorname{argmin}\left\{n^{\psi}_{perf}\right\} \tag{4}$$

C. Prediction models and training scenarios

We create multiple models to predict the *best* number of I/O resources an application should use, as defined in the previous section.

1) Models under consideration: For that, we compare four machine learning methods: Random Forest (RF), Extra Trees (ET) [12], Gradient Boosting (XGB), and K-Nearest Neighbors (KNN). The first three models are tree-based classifiers, while the last one is a cluster classifier. Majors differences between RF and XGB are the training process to obtain decision trees and the resilience to overfitting. XGB integrates security to avoid overfitting at the cost of training time. However, our dataset is relatively small, and training time is not an issue. Performance of ET and RF are expected to be about the same, but ET is less robust to outliers but more resilient against variance. We used their implementation from the Scikit Learn Python library. All code used needed to train our models and reproduce our results is available at *[the link will be available once the paper is published]*

2) Training vs Validation: These models are compared in multiple *scenarios* that were designed to answer our questions regarding the features. Note that for all scenarios, the training data is selected by picking uniformly at random 1/5 of all the extended *admissible* configurations (i.e. including ost_number, which is *not* a feature). To give an example, when we use the complete data set for training, we pick a fifth of the $528 \times 8 = 4224$ extended configurations.

The validation is done by picking uniformly at random 1/10 of the 528 configurations. Each experiment is repeated 10 times.

Q1: Is the best number of OSTs predictable from the considered application variables? To answer that, we use all configurations in our data set. We call this scenario AllData.

Q2: Can we predict it even when the data set used for training is not as diverse as the real applications are? For this question, we consider the case where some elements had not been considered for training, i.e. we prune the initial data set to select the training data:

1) **Configuration pruning**: we remove from the training data set some configurations. We selected the two variables that had the most important correlation with the bandwidth behavior (see Fig. 3):
   - NoSharedFile consists in removing all shared-file configurations (variable "filestrategy").
   - NoContiguous consists in removing all configurations with contiguous accesses (variable "spatiality").
2) **Behavior**: we remove from the training data set some configurations based on their I/O performance:
   - NoNeutral consists in removing all configurations where the highest median bandwidth is smaller than 1.3 times the lowest median bandwidth for that configuration. In other words, we remove all applications that are not very sensitive to stripe count.
   - AllNeutral consists in doing the opposite: removing all configurations where the highest median bandwidth is larger than 1.3 times the lowest median bandwidth for that configuration (i.e. all applications that are very sensitive to stripe count).

Q3: Are there features that contain redundant information? Fig. 5 shows different projections of a principal component analysis (PCA) on our data, looking for redundant features. We can see *procs* and *nodes* are always close. Indeed, it is plausible that those two variables contribute with very similar information. If we think about how to obtain this information from applications at execution time, it can be easier to obtain the number of compute nodes than the number of processes doing I/O. To verify if they are redundant, we create models without the "procs" feature and without both "procs" and "nodes". If they are redundant, the former should have results close to AllData. If this information is useful, the latter (without both) should have low accuracy.

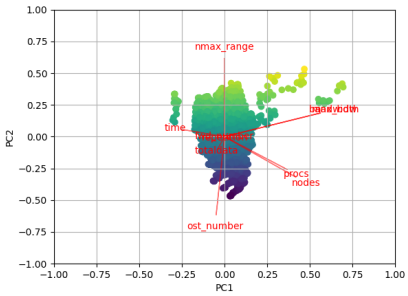
Q4: What features are the most important for prediction (and which ones are not)? To answer that, we train our models with different sets of features, excluding one at a time, to verify their impact on the obtained accuracy.

III. Evaluation

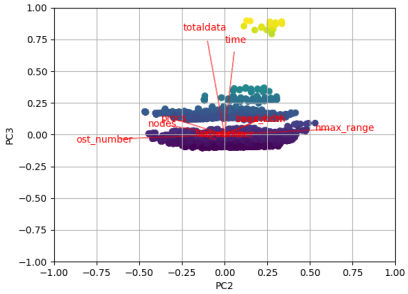
In this section, we present the results we obtained from experiments designed to answer the questions listed in Section II-C. We first present the metrics used to evaluate our predictions in Section III-A, then the results for the cases where we have different training data sets in Section III-B, and the results with different features in Section III-C. Finally, in Section III-D we further validate our model by presenting results obtained with real application I/O kernels.

A. Metrics

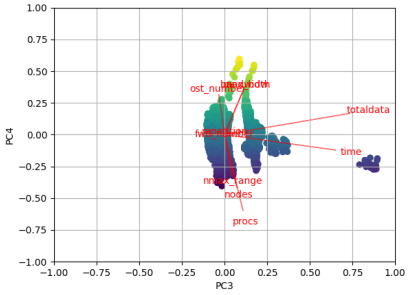
We are interested in the *accuracy* of our models in each scenario. An accuracy of 0.95, for example, means that a model gives the right prediction 95% of the times. If y is the true value (the best prediction, as defined in Definition 1) and $\hat{y}$ is the predicted value, then:



(a) First and second components



(b) Second and third components



(c) Third and forth components

Fig. 5: Different projections of the PCA analysis

$$accuracy(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N} \mathbb{1}_{(\hat{y}_i = y_i)}$$

Nonetheless, just knowing some answers are wrong does not give the whole information, namely how badly the wrong predictions could impact the system and applications performance. Hence we define additional metrics below.

$$PredictErr = \frac{1}{N}\sum_{i=0}^{N} |y_i - \hat{y}_i|$$

$PredictErr$ gives the mean deviation between the best values and what was predicted, in number of I/O resources. Ideally, it should be as close to 0 as possible. We further decompose $PredictErr$ into the cases where the number of OSTs is underestimated ($LowPredictErr$) and overestimated ($HighPredictError$).

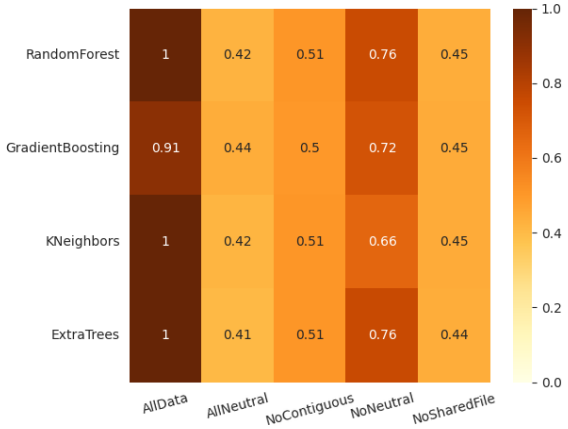


Fig. 6: Accuracy obtained with different machine learning models trained on the different scenarios

$$LowPredictErr = \frac{1}{N}\sum_{i=0}^{N} max(y_i - \hat{y}_i, 0)$$

$$HighPredictError = \frac{1}{N}\sum_{i=0}^{N} max(\hat{y}_i - y_i, 0)|$$

The $HighPredictError$ metric quantifies how much a model potentially makes an allocation algorithm waste system resources. On the other hand, $LowPredictErr$ corresponds to making the allocation algorithm give applications less resources than it would need to reach its peak performance. We have:

$$PredictErr = LowPredictErr + HighPredictError$$

In both cases, under or overestimating the number of OSTs, the predicted value could still be a good prediction, as defined by Equation (3), and hence have only a small impact on application performance. Therefore, we also want to measure the difference in performance that would be caused to applications by using one of our models for I/O resource allocation.

$$BdwLost = \frac{1}{N}\sum_{i=0}^{N} max(\frac{b(y_i) - b(\hat{y}_i)}{b(y_i)}, 0)$$

We use the max function in the formula to only consider the cases where the prediction $\hat{y}$ has a negative impact on the bandwidth.

B. Results with different training data sets

Figure 6 shows accuracy for all combinations of machine learning techniques and scenarios. As detailed in the previous section, these results are complemented by the $PredictErr$ metric shown in Fig. 7, stacked with $LowPredictErr$ and $HighPredictError$, and $BdwLost$ in Fig. 8. In the following, we analyze these results aiming to answer the questions listed in Section II-C.

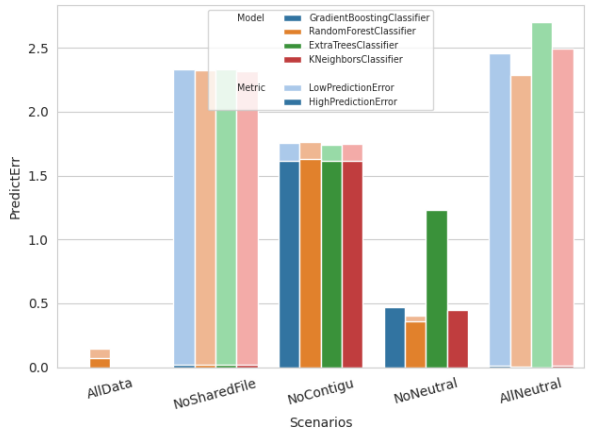


Fig. 7: Prediction error decomposed into Low (lighter color, shown on top) and High (darker color, shown on the bottom) $PredictErr$ for all ML models and scenarios

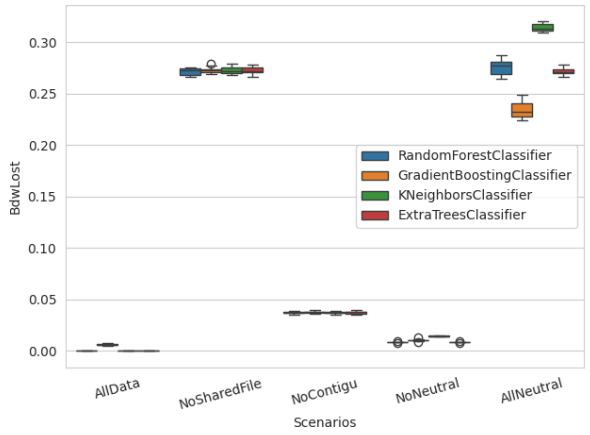


Fig. 8: $BdwLost$ for all ML models and scenarios

1) Results for the AllData scenario: As seen in Figure 6, when we train on a random sample of 20% of all data (AllData), all models perform well. XGB and KNN have results that are worse than the ones obtained by Random Forest and Extra Trees. Fig. 9 further details these results by separating them by the expected prediction. We can see XGB and KNN have lower accuracy especially when they should predict numbers of OSTs that are less common in the data, such as 3 and 7 (in contrast to good accuracy for 1). However, we can also see in Fig. 7 that they do not have a tendency of always underestimating or overestimating this value.

Q1: Is the best number of OSTs predictable from the considered application variables?

Yes, the results show it is possible to train a model that receives these variables as features and predicts the best number of OSTs to use. Among the tested models, Random Forest and Extra Trees seem to be the most suitable.

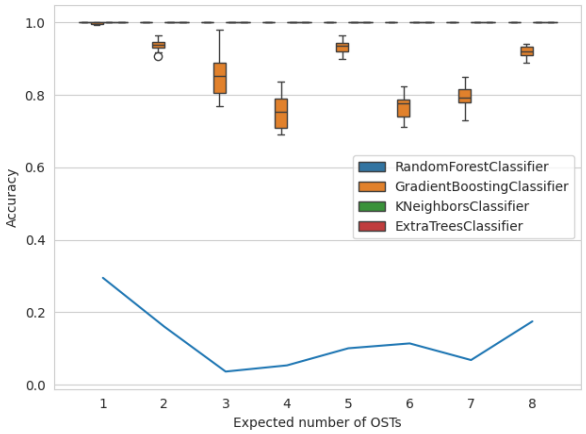


Fig. 9: Accuracy for all models for the AllData scenario, separated by the value that should be predicted as the best number of I/O resources. The boxplots group the 10 accuracy results. The blue line shows the proportion of the data that corresponds to each number of OSTs (a different view of the same data from Figure 2).

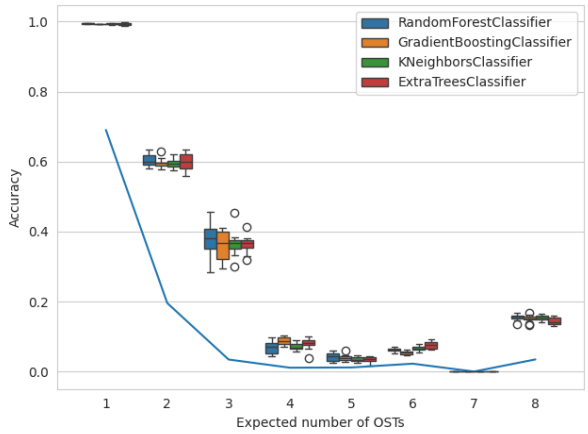
2) Less diverse training sets: We now look into the scenarios where we removed some applications from the training set to purposely make them less diverse, as discussed in Section II-C.

a) NoSharedFile and NoContiguous (shown in Fig. 10): We can see that, for NoSharedFile (Fig. 10a), roughly 70% of this training set are applications for which 1 is the best prediction. That causes all models to have a good accuracy for this value, and then quickly drop for other values. We can confirm in Fig. 7 that they consistently predict a lower number of OSTs than required.

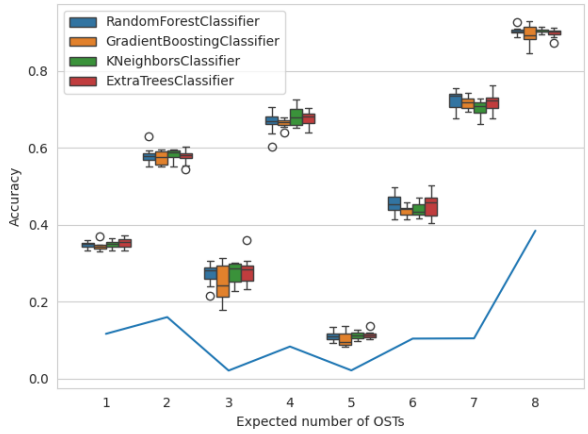
We have a similar conclusion with NoContiguous dataset, shown in Fig. 10b. In this case, most of the training data set corresponds to applications that have 8 as the best prediction, and hence this model tends to overestimate the number of OSTs, as seen in Fig. 7. Its impact on performance, seen in Fig. 8, is lower than the impact from NoSharedFile because the latter tends to underestimate the number of resources, and that is worse in this data set.

b) NoNeutral and AllNeutral (shown in Fig. 11): We can see AllNeutral results look like the NoSharedFile ones, and that NoNeutral results look like the NoContiguous ones, in the sense that the training data sets have either too many or not enough cases where the best prediction is a low number of OSTs. That causes these models to consistently under (AllNeutral) or over (NoNeutral) estimate the number of resources (Fig. 7), and for this data set underestimating it has a heavier impact on performance (Fig. 8).

It is important to notice that, even if the accuracy (Figure 6) and performance (Fig. 8) results seem good for NoNeutral, we conclude none of the four scenarios discussed here yield good predictors. That is the case because they have particularly bad results for some of the applications (the ones that were under

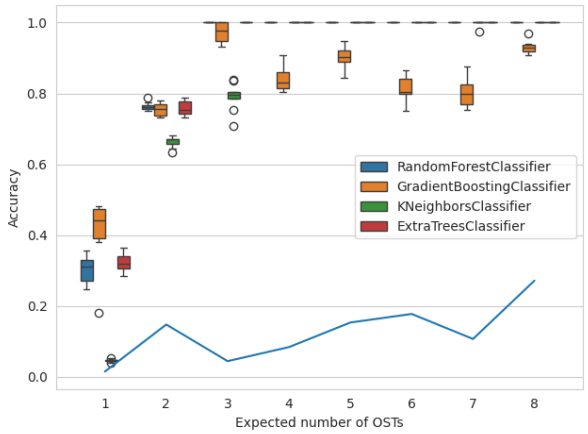


(a) NoSharedFile

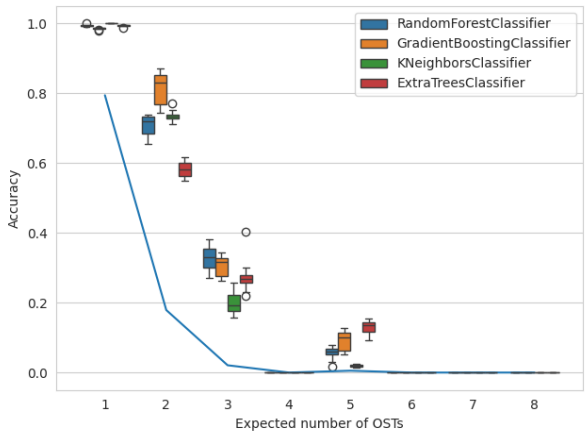


(b) NoContiguous

Fig. 10: Accuracy (boxplots) for the NoSharedFile and NoContiguous scenarios, separated by the expected number of OSTs. The blue line shows the proportion of each expected prediction in the training data set (file-per-process applications only for NoSharedFile, and contiguous accesses only for NoContiguous).



(a) NoNeutral (all classes are present for training except Neutral)



(b) AllNeutral (only the Neutral class is present for training)

Fig. 11: Accuracy (boxplots) for two scenarios of limited classes of behaviors, separated by the expected number of OSTs. The blue line shows the proportion of each expected prediction in the training data sets.

represented in the training data set).

Q2: Can we predict it even when the data set used for training is not as diverse as the real applications are?

We conclude it is very important for all possible predictions to be present in the training set in order to create a model that yields good results. The models should be trained on a diverse application set.

C. Results with different features

We want to know if every variable is important for prediction, and if there are features that are more important than others. Fig. 12 shows the accuracy for each prediction model when not using some features. Those model are still trained on a sample of all data, but just ignoring a variable.

a) Information redundancy: The intuition from Fig. 5 is that "nodes" and "procs" could contribute with the same knowledge for the prediction model. If that was the case, it would be ineffective to provide both features. Although the accuracy is approximately 80% without "nodes", it decreases to approximately 55% by further removing "procs". This implies that both these features are important for prediction. Moreover, Fig. 12 does not show any features that can be removed without a loss in accuracy when compared to the scenario with all data as presented in Fig. 9.

Q3: Are there features that contain redundant information?

While we initially thought we could have redundant features, that appears to be untrue. All of the features contribute with necessary information to predict the number of resources that will give the best performance.

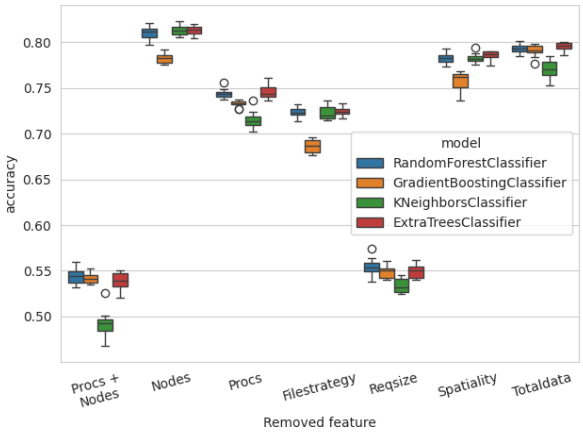


Fig. 12: Accuracy obtained by models when ignoring different features

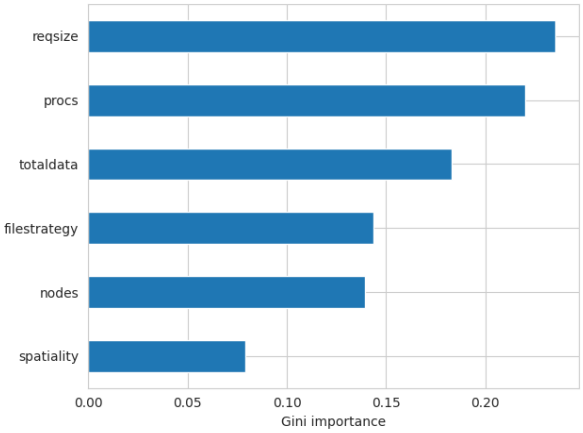


Fig. 13: Random Forest features importance (MDI)

b) Feature importance: While Fig. 12 shows the impact of each feature on accuracy, Fig. 13 proposes a more in-depth look by showing the Mean Decrease in Impurity (MDI) from the Random Forest model. MDI counts the number of nodes where a feature is used in decision trees and weight it by the probability to encounter that node during prediction. The highest this value, the more important the feature is for prediction. The feature of request size emerges as three times more significant than the feature of spatiality for prediction. Therefore, this clearly demonstrates that certain features are of greater significance in achieving accurate prediction. Despite the non-linear correlation between MDI and accuracy, the conclusion drawn from both figures is remarkably similar.

Q4: What features are the most important for prediction (and which ones are not)?

We have come to the conclusion that all the features presented are important for enhancing prediction accuracy. We found that request size and scale (the number of processes and of nodes used by the application to perform I/O) are the most important features. We can still achieve an accuracy close to 80% if we remove spatiality and the total amount of data, which is an interesting result as these features are clearly more difficult to obtain from applications at execution time than the others.

D. Results with real applications

After evaluating our models and answering our questions in the previous sections, we now further validate our approach by testing it with I/O kernels from real applications:

- HACC-IO, from the Hardware Accelerated Cosmology Code (HACC) application [10]. We executed the write-only version with 28256364 particles.
- MSLIO, from the MHM multiscale simulation library [5]. We executed it using ccross 7 and 128 subelements.
- BT-IO, part of the NAS parallel benchmarks [16]. We executed two classes, C and D, which in this context count as different applications. We used the MPI-IO full version.

For each of them, we manually obtained the features for our model by studying the code and/or a Darshan trace obtained from a separate run. These features are presented in Table II. Automatically obtaining the features will be the subject of future work.

Results are presented in Table III. We can see our models are able to predict the best number of OSTs to two of the four considered cases (MSLIO and HACC-IO). For the other two, the performance obtained with the predicted number is very close to what would be obtained by the expected prediction (a difference of less than 5%).

Experiment with real applications

We conclude our models can generalize to other applications than the ones used in the training sets, and could be used to decide how many OSTs they should use.

IV. Use Case: Scheduling of OSTs

Lastly, we evaluate the use of two of our models, Random Forest and KNN (from the AllData scenario), for OST allocation (i.e. deciding how many OSTs each application should use). For that, we recreated an experiment from Bandet et al. [2]. In this experiment, 20 applications are randomly generated and follow different behaviors from the data set described in Section II-A. There are 20 OSTs, which are scheduled in two steps: first the allocation algorithm decides how many each application uses, then a placement algorithm is used to decide which OSTs are used by each application. Then the execution is simulated, taking into consideration the contention generated

TABLE II: Features (described in Table I) of the four considered applications

application	nodes	procs	filestrategy	reqsize	totaldata	spatiality
HACC-IO	16	64	file-per-process	116 MB	67 GB	contiguous
MSLIO	1	16	file-per-process	13 bytes	1.7 GB	contiguous
BT-IO C	4	16	shared-file	14 MB	6.8 GB	contiguous
BT-IO D	8	64	shared-file	16 MB	133 GB	contiguous

TABLE III: For four real application I/O kernels, executed 10 times with different numbers of OSTs, the expected best prediction and the prediction obtained from our Random Forest model, together with the median bandwidth obtained by those stripe counts.

	Expected prediction		Prediction by RF		Prediction by KNN		Prediction by ET		Prediction by XGB	
	number	bandwidth (MB/s)	number	bandwidth (MB/s)	number	bandwidth (MB/s)	number	bandwidth (MB/s)	number	bandwidth (MB/s)
HACC-IO	1	9798	1		8	9118	1		1	
MSLIO	1	1493	1		1		1		1	
BT-IO C	3	1532	8	1605	8		8		8	
BT-IO D	2	774	8	778	8		8		8	

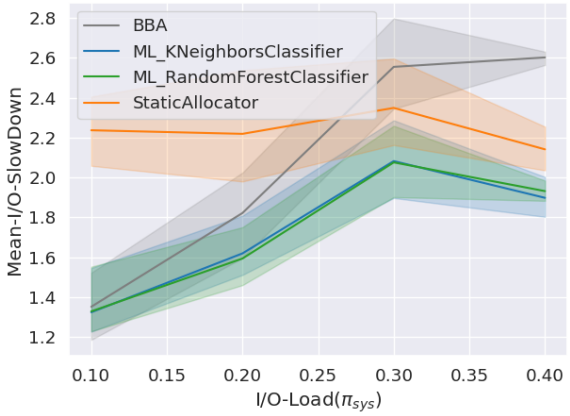


Fig. 14: Mean I/O slowdown (the lower, the better) obtained when our machine learning models are used for OST allocation

in the shared resources. The described process is repeated multiple times, using different sets of random applications. From each execution, we compute the I/O slowdown, which represents how much the I/O performance of each application is slowed down because of sharing the system with the 19 others (compared to running in isolation). This slowdown is a result of both the allocation and the placement, as both the used number of OSTs and the sharing of resources impact performance.

We use a simple greedy placement algorithm (that tries to assign the same number of applications to each OST) and compare our model to:

- BestBandwidth Allocator (BBA), a solution that gives n^*_{perf} (as defined in Eq. (2)) to every application. This is a theoretical solution that assumes we have a perfect knowledge of applications;
- Static that gives a number of OSTs that is proportional to the number of compute nodes (a solution when there is no knowledge on the impact of the stripe count on performance).

These are two heuristics proposed by [2] (BBA being the best heuristic among those proposed on these scenarios). The evaluation is performed when the I/O occupation (I/O load) of the system is below 40%: when the I/O load is high, Bandet et al. [2] showed that precise knowledge of application behavior was less important, and that the Static heuristic was sufficient.

The evaluation shows that the solution that uses the output of our predictor outperforms the other solutions. This demonstrates that:

1) the accuracy of our prediction model is sufficient to be used for resource allocation;
2) the choices that we made to define what a "valid prediction" is (Section II-B), converging to Definition 1, were good choices for this goal.

V. Related work

I/O performance prediction has been a hot-topic recently. Generally most of the work focus on predicting the expected bandwidth. Typu et al. [18] used neural network to predict I/O bandwidth over seismic data on lustre file system for very specific format, even if they extrapolated their results to more general MPI IO conclusion. They provided around 80% accuracy for their data format. Features used are number of node and process, strips configuration and chunk size.

Xu et al. [24] focus on predicting I/O variability, using surrogate models. They definied three categories for seven variables, such as Hardware, OS and Application. From Application side they have IO Operation mode, Number of thread, file size by record size. Hardware is just the CPU frequency. OS is IO and VM schedulers. The benchmark used is IOZone. Interpolation and extrapolation test. Interpolation is whenever they are testing data that have the same parameters than training data.

Isakov et al. [14] used several years of Darshan log to create a tool that could find unusual I/O behavior to detect bottleneck. The approach is based on many clustering and decision trees machine learning techniques. Their goal is to analyze the impact of each metrics role in I/O bottleneck and to spot applications with bottleneck based on these variables.

Wong et al. [21] propose an approach that creates a fingerprint of an application, that tries to describe the application based on its behavior. They predict the execution time of certain application phases with a high accuracy. However, this approach does not provide any information about application nor system parameters, and remains a black box model.

The work that we provide here is different from those, since instead of trying to predict an (inaccurate) bandwidth value, we focus on predicting a configuration that maximizes this bandwidth, in order to inform resource management strategies.

Other work have studied the impact of OST placement and stripe count on the I/O performance of application. Boito et al. [6] studied the impact of target placement on different servers and concluded on the importance of balancing OST over the different OSS. Their work was limited to one access pattern. A previous study by Chowdhury et al. [9] with the same file system, but using other applications, concluded the number of OSTs had little or no impact. On the other hand, Xu et al. [23] concluded that for collective reads, using more OSTs actually degraded performance. In our work, we construct a more general data set that shows that using OST is a more complicated problem that highly depends on I/O variables.

Finally, several work [3], [2] have shown how one can use the information about OST prediction into the design of I/O scheduling techniques, and have shown that an accurate prediction can be extremely beneficial to I/O performance.

VI. Conclusion and future work

In this paper, we proposed an approach to predict the best number of I/O resources, such as OSTs, to assign to applications according to their characteristics. Related work has demonstrated the value of such information for resource allocation algorithms that improve application performance and system usage. However, as we showed, predicting this value is not trivial. Indeed, the only way of obtaining it so far was to test the application with multiple numbers of OSTs multiple times (to account for variability), which is prohibitive in cost and limits the usage of such resource allocation techniques.

For this study, we created a data set of 528 different access patterns (created with the IOR benchmarking tool) and measurements of performance for them with different numbers of OSTs. Our approach trains a model using a machine learning technique to predict the best number of OSTs while accounting for intrinsic variability: the *best* is a number that yields performance that is close enough to the highest one, but using as few resources as possible to prevent contention caused by sharing resources with other applications. Training such a model obviously has a cost, but it would only have to be done once per system. Indeed, it is common for newly deployed I/O systems to undergo extensive testing to confirm they meet requirements and to configure parameters such as default stripe size and count. This testing phase could include the experiments required to train the model. In this context, it is important to notice the good results obtained for the AllData scenario were generated by training only on 20% of the data set.

While only a part of the data set is required to train a model that is able to generate the desired predictions with great accuracy, we observed it is very important for this subset to be representative of the different observed behaviors. When we removed whole classes of applications from the training set, that resulted in predictors that would consistently under or overestimate the best number of OSTs. Moreover, we tested four different machine learning models — Random Forest, Extra Trees, Gradient Boosting (XGB), and K-Nearest Neighbors (KNN) — and found that Random Forest and Extra Trees were the best ones regarding accuracy, prediction error, and impact on performance.

We also investigated the importance of the different features on the models results. Some studies have shown that not all features are equally important by measuring the accuracy when removing one variable at the time. These experiments led to a drop in accuracy of at least 20%, showing that every feature is important for prediction. Furthermore, we demonstrate that there is no feature redundancy in our dataset, meaning every feature has a piece of knowledge to give to the prediction model.

Furthermore, our models were validated with four I/O kernels from real applications and provided good predictions for them (either the actual best predictions or predictions that would lead to performance that is less than 5% different than that). Finally, we showed our model provides useful information for resource allocation.

As future work, we plan to integrate our solution in a monitoring system and make it communicate with a parallel file system, allowing for a resource allocation algorithm to be used in the latter to better allocate OSTs. For example, a recent work by Walton et al. [19] gives Darshan [7] ability to connect to an application that is already running, giving real-time insight. This tool could be exploited to get features in real time to provided to our pre-trained model.

Acknowledgements

As part of the “France 2030” initiative, this work has benefited from a State grant managed by the French national research agency (*Agence Nationale de la Recherche*) attributed to the Exa-DoST project, and bearing the reference ANR-22-EXNU-0004. It was also supported by the Adaptive multitier intelligent data manager for Exascale (ADMIRE) project, funded by the European Union’s Horizon 2020 JTI-EuroHPC Research and Innovation Programme (grant 956748). Experiments were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and *Conseil Régional d’Aquitaine* (see https://www.plafrim.fr).

References

[1] G. Aupy, O. Beaumont, and L. Eyraud-Dubois. Sizing and partitioning strategies for burst-buffers to reduce io contention. In *2019 IEEE international parallel and distributed processing symposium (IPDPS)*, pages 631–640. IEEE, 2019.

[2] A. Bandet, F. Boito, and G. Pallez. Scheduling distributed I/O resources in HPC systems. In *30th International European Conference on Parallel and Distributed Computing 26 - 30 August 2024 Madrid, Spain 30th International European Conference on Parallel and Distributed Computing*, Madrid, Spain, Aug. 2024.

[3] J. L. Bez, A. Miranda, R. Nou, F. Z. Boito, T. Cortes, and P. Navaux. Arbitration policies for on-demand user-level i/o forwarding on HPC platforms. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 577–586. ISSN: 1530-2075.

[4] F. Boito. Write performance with different numbers of OSTs for BeeGFS in PlaFRIM, Jan. 2024.

[5] F. Boito, A. T. A. Gomes, L. Peyrondet, and L. Teylo. I/O performance of multiscale finite element simulations on HPC environments. In *WAMCA 2022 - 13th Workshop on Applications for Multi-Core Architectures*, Bordeaux, France, Nov. 2022.

[6] F. Boito, G. Pallez, and L. Teylo. The role of storage target allocation in applications' i/o performance with BeeGFS. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 267–277. ISSN: 2168-9253.

[7] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley. 24/7 characterization of petascale i/o workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10. IEEE.

[8] J. Carretero, E. Jeannot, G. Pallez, D. E. Singh, and N. Vidal. Mapping and scheduling hpc applications for optimizing i/o. In *Proceedings of the 34th ACM International Conference on Supercomputing*, pages 1–12, 2020.

[9] F. Chowdhury, Y. Zhu, T. Heer, S. Paredes, A. Moody, R. Goldstone, K. Mohror, and W. Yu. I/o characterization and performance evaluation of BeeGFS for deep learning. In *Proceedings of the 48th International Conference on Parallel Processing*, ICPP '19, pages 1–10. Association for Computing Machinery.

[10] CORAL. Coral benchmark codes, 2014.

[11] E. Costa, T. Patel, B. Schwaller, J. M. Brandt, and D. Tiwari. Systematically inferring i/o performance variability by examining repetitive job behavior. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, pages 1–15. Association for Computing Machinery.

[12] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63:3–42, 2006.

[13] S. Herbein, D. H. Ahn, D. Lipari, T. R. Scogland, M. Stearman, M. Grondona, J. Garlick, B. Springmeyer, and M. Taufer. Scalable i/o-aware job scheduling for burst buffer enabled hpc clusters. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, pages 69–80, 2016.

[14] M. Isakov, E. d. Rosario, S. Madireddy, P. Balaprakash, P. Carns, R. B. Ross, and M. A. Kinsy. HPC i/o throughput bottleneck analysis with explainable local models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13.

[15] J. Luca Bez, F. Z. Boito, A. Miranda, R. Nou, T. Cortes, and P. O. A. Navaux. Towards On-Demand I/O Forwarding in HPC Platforms. In *PDSW 2020 : 5th IEEE/ACM International Parallel Data Systems Workshop*, Atlanta, Georgia / Virtual, United States, Nov. 2020.

[16] NAS. Nas parallel benchmarks, 2006.

[17] S. Oral, J. Simmons, J. Hill, D. Leverman, F. Wang, M. Ezell, R. Miller, D. Fuller, R. Gunasekaran, Y. Kim, S. Gupta, D. T. S. S. Vazhkudai, J. H. Rogers, D. Dillow, G. M. Shipman, and A. S. Bland. Best practices and lessons learned from deploying and operating large-scale data-centric parallel file systems. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 217–228. ISSN: 2167-4337.

[18] A. J. S. Tipu, P. O. Conbhui, and E. Howley. Applying neural networks to predict HPC-i/o bandwidth over seismic data on lustre file system for ExSeisDat. 25(4):2661–2682.

[19] S. Walton, O. Aaziz, A. L. V. Solórzano, and B. Schwaller. LDMS darshan connector: For run time diagnosis of HPC application i/o performance. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 626–634. ISSN: 2168-9253.

[20] T. Wang, S. Byna, G. K. Lockwood, S. Snyder, P. Carns, S. Kim, and N. J. Wright. A zoom-in analysis of i/o logs to detect root causes of i/o performance bottlenecks. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 102–111, 2019.

[21] A. Wong, D. Rexachs, and E. Luque. Parallel application signature for performance analysis and prediction. 26(7):2009–2019. Conference Name: IEEE Transactions on Parallel and Distributed Systems.

[22] B. Xie, Z. Tan, P. Carns, J. Chase, K. Harms, J. Lofstead, S. Oral, S. S. Vazhkudai, and F. Wang. Applying machine learning to understand write performance of large-scale parallel filesystems. In *2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*, pages 30–39, 2019.

[23] C. Xu, S. Byna, V. Venkatesan, R. Sisneros, O. Kulkarni, M. Chaarawi, and K. Chadalavada. Lioprof: exposing lustre file system behavior for i/o middleware. In *2016 Cray User Group Meeting*, 2016.

[24] L. Xu, T. Lux, T. Chang, B. Li, Y. Hong, L. Watson, A. Butt, D. Yao, and K. Cameron. Prediction of high-performance computing input/output variability and its application to optimization for system configurations.

[25] B. Yang, W. Xue, T. Zhang, S. Liu, X. Ma, X. Wang, and W. Liu. End-to-end i/o monitoring on leading supercomputers. 19(1):1–35.

[26] O. Yildiz, M. Dorier, S. Ibrahim, R. Ross, and G. Antoniu. On the root causes of cross-application i/o interference in HPC storage systems. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 750–759. IEEE.