Dynamic Model-driven Parallel I/O Performance Tuning

Babak Behzad*, Surendra Byna[†], Stefan M. Wild[‡], Prabhat[†], Marc Snir*[‡]

* *University of Illinois at Urbana-Champaign*
Urbana, IL 61801
[†] *Lawrence Berkeley National Laboratory*
Berkeley, CA 94720
[‡] *Argonne National Laboratory*
Argonne, IL 60439

***Abstract*—**
Parallel I/O performance depends highly on the interactions among multiple layers of the parallel I/O stack. The most common layers include high-level I/O libraries, MPI-IO middleware, and parallel file system. Each of these layers offers various tunable parameters to control intermediary data transfer points and the final data layout. Due to the interdependencies and the number of combinations of parameters, finding a good set of parameter values for a specific application's I/O pattern is challenging. Recent efforts, such as autotuning with genetic algorithms (GAs) and analytical models, have several limitations. For instance, analytical models fail to capture the dynamic nature of shared supercomputing systems and are application-specific. GA-based tuning requires running many time-consuming experiments for each input size.

In this paper, we present a strategy to generate automatically an empirical model for a given application pattern. Using a set of real measurements from running an I/O kernel as training set, we generate a nonlinear regression model. We use this model to predict the top-20 tunable parameter values that give efficient I/O performance and rerun the I/O kernel to select the best set of parameter under the current conditions as tunable parameters for future runs of the same I/O kernel. Using this approach, we demonstrate 6X - 94X speedup over default I/O time for different I/O kernels running on multiple HPC systems. We also evaluate performance by identifying interdependencies among different sets of tunable parameters.

***Keywords*-Parallel I/O, Parallel I/O Tuning, Performance Optimization, Performance Modeling**

I. Introduction

Large-scale simulations are increasingly used to study complex scientific phenomena across many science domains. For instance, studying plasma behavior in solar weather, understanding particle interactions in accelerator physics, computational fluid dynamics, atmospheric modeling, and combustion all require high-resolution, large-scale simulations. Due to advances in computing capabilities, many large-scale simulations produce massive amounts of data that need to be stored on file systems for further analysis. It is typical for the simulations using hundreds of thousands of CPU cores to generate tens to hundreds of Terabytes (TB).

Since simulations often wait idly while writing the data, it is imperative that parallel I/O performance be efficient. However, achieving such performance for high-performance computing (HPC) applications is nontrivial because of complex interdependencies among layers of the parallel I/O software and hardware stacks. The most common parallel I/O stack contains high-level I/O libraries such as HDF5 and PNetCDF, parallel I/O middleware such as MPI-IO, and parallel file systems such as Lustre and GPFS. The data transferred among these layers must align properly to achieve efficient performance. Each of these layers offers application-level tunable parameters to select how data is moved across intermediary nodes and how it is stored on disks. For instance, for MPI-IO, applications can specify the number of intermediary nodes (aggregators) and the distribution of data to those nodes. Parallel file systems, especially Lustre, provide parameters to select the number of storage targets to write the data to and the amount of contiguous data chunks to be written to a storage target.

Selecting a set of tunable parameters for the entire stack that would result in efficient performance is challenging. Because of the large number of discrete options for various parameters, the search space is enormous and running an application with all possible combinations of configurations is impractical. We have recently explored an autotuning approach using genetic algorithms (GA) to traverse the search space systematically [3]. The GA approach initializes this traversal with random sets of parameters and produces new generations of parameter sets by applying mutation and crossover operations. The GA eventually determines parameter values that give near-optimal I/O performance. While GA approach reduces the number of configurations significantly, it is still time consuming as the number of experiments required to converge could be prohibitively large. Another limitation is that parameter values are specific to each application and its input size.

To overcome the limitations of the GA-based traversal, in this paper, we present a statistical approach for automatic generation of an empirical performance prediction model that is used for pruning the search space significantly.

There have been a few research efforts targeting prediction of parallel I/O performance accurately (see, e.g. [18], [15], [17], [13], [22], [19], [9]). Many of these models are relatively complex and their applicability is limited to specific systems or I/O workloads. We use a statistical approach to train an empirical base model that can be used to prune the search space. The base model then can be tuned further by running on the reduced sample space in order to capture the dynamic runtime conditions of a system. The advantages of our proposed method include fast reduction of the search space compared to a GA approach and consideration of dynamic conditions of a parallel I/O subsystem.

The contributions of this paper are as follows:

- We propose a statistical approach for generating empirical prediction models for parallel I/O performance.
- We demonstrate the use of our models for selecting tunable parameters that achieve efficient I/O performance on multiple platforms, for multiple I/O kernels extracted from real scientific simulations running at different problem sizes and scales.
- We evaluate interdependencies among various parallel I/O tunable parameters.

The remainder of the paper is organized as follows. In Section II, we present the background on the parallel I/O subsystem. In Section III, we discuss the experimental setup and characteristics of I/O kernels of applications. We propose our approach of generating empirical performance models and their dynamic tuning in Section IV. We demonstrate the use of our model generation for selecting tunable parameters and evaluate interdependencies of the parameters in Section IV-A1. In Section VI, we discuss our work in the context of existing research efforts and compare this work with the other state-of-the-art I/O autotuning approaches. Section VII concludes the paper with a brief discussion of future work.

II. Parallel I/O

A parallel I/O subsystem typically consists of various layers of middleware libraries and hardware. The most common parallel I/O stack in current HPC machines has high-level I/O libraries and file formats (e.g., HDF5, NetCDF, and ADIOS), I/O middleware (e.g., MPI-IO and POSIX), parallel file systems (e.g., Lustre, GPFS, and PVFS), and storage and I/O hardware. When parallel applications perform I/O operations, the data moves from individual processors to the storage hardware through the multiple layers of the stack.

To achieve good I/O performance, each of the layers offers optimization strategies. For instance, MPI-IO provides two modes of writing data to disks: independent I/O and collective I/O [21]. With independent I/O, each MPI process writes the data to storage independent of other processes of the application. In collective I/O mode, the data is collected at a few aggregator processes and the aggregators write the data to storage. The collective I/O mode is preferable when the number of MPI processes is large because too many requests to the file system degrade I/O performance. Throughout this paper, we focus on the write operations that originate from large simulations using collective I/O.

A typical implementation of a collective I/O write operation includes two phases: the data collection phase at aggregators and the I/O phase [7]. Each MPI process first analyzes its request to the file and calculates the *start offset* and *end offset*. These two variables identify the segment of the file accessed by the processor. After calculating these variables, each process sends their values to all the other processes. The aggregators then compute the partitions, called *file domains*, of the file they are responsible for writing. In ROMIO [20], the basis for many MPI-IO implementations, the aggregators split the range of the file being updated equally in a block-cyclic distribution.

Note that, we define *write time* as a higher-level library write operation, consisting of all the communication and I/O time needed for this operation.

III. Experimental Setup

A. HPC Platforms

We conducted all the experiments presented in this paper on three platforms, named Edison, Hopper, and Stampede, located at two supercomputing centers.

1) Edison: Edison is a supercomputer at the National Energy Research Scientific Computing Center (NERSC). It is a Cray XC30 system comprising $5,576$ twenty-four core nodes with 64GB of memory per node. It has Cray Aries with Dragonfly topology and three Lustre file systems with aggregate bandwidth of 168 GB/s. For the experiments conducted in this study, we used the scratch2 file system in these experiments with a maximum of 96 OSTs and 48 GB/s peak I/O bandwidth.

2) Hopper: Hopper is another supercomputing system located at NERSC. It is a Cray XE6 system containing $6,384$ twenty-four core nodes with 32GB of memory per node. It employs the Gemini interconnect with a 3D torus topology. We used a Lustre file system with 156 OSTs and a peak bandwidth of about 35GB/s for storing data.

3) Stampede: Stampede is a Dell PowerEdge C8220 cluster at the Texas Advanced Computing Center. It has $6,400$ sixteen core nodes with 32GB of memory per node. It uses Mellanox FDR InfiniBand technology with a two-level fat-tree topology. Stampede's Lustre file system with 160 OSTs (in the testing experiments for consistent comparisons we use 156 OSTs as the maximum stripe count for Stampede as well) has shown a peak of 159 GB/s I/O bandwidth.

B. I/O kernels

We used three I/O kernels in this study: VPIC-IO, VORPAL-IO, and GCRM-IO. These kernels are extracted from a particle physics simulation (VPIC [5], [6]), a computational plasma framework (VORPAL [14]), and a global atmospheric circulation model framework (GCRM [16]). The kernels perform I/O operations representative of real problem configurations. We now summarize the I/O kernels:

1) VPIC-IO: VPIC-IO uses the H5Part library [4] to initiate and write data pertaining to particles. The code is run in a weak-scaling mode, where each MPI process writes eight million particles. Each particle has eight (six floating point and two integer) variables. All processes issue one write call per variable (i.e., eight write calls) in order to write the data into a single shared HDF5 file.

2) VORPAL-IO: VORPAL-IO leverages the H5Block library [4], which uses the HDF5 library to handle block structured data. VORPAL-IO partitions a 3D grid of points into a 3D grid of processes. Each process writes a sub-block of points in its partition. For example, in a 128-process run with a block of size $300 \times 100 \times 60$ and a decomposition of $(8, 4, 4)$, the size of the total block is going to be $2400 \times 400 \times 240$. This kernel is also configured to run in a weak-scaling mode.

3) GCRM-IO: GCRM-IO also leverages the H5Block library [4], but has a less complex pattern than VORPAL-IO. GCRM-IO partitions a semi-structured geodesic mesh between processes. Each process writes a sub-block of the mesh in its partition. The grid and sub-domain resolution are controlled by the user. Unlike the other two, this kernel is configured to run in a strong-scaling mode, where the total data size does not increase with the number of processes.

IV. Dynamic Model-driven I/O Tuning

In Figure 1, we show a high-level workflow of our proposed dynamic model-driven I/O tuning process. We extract the I/O kernel of an application using tracing tools such as I/O Tracer [2] or Skel [10] and run the kernel with a preselected training set of tunable parameters. We define the training set based on the parameters for different levels of the I/O stack and for multiple problem sizes. Using the measured I/O performance of the kernel, we develop an empirical performance model (described in detail in Section IV-A).

We use the developed performance model to predict I/O performance for an exhaustive set of all combinations of tuning parameters. We then select the best performing tuning parameter sets by sorting the predicted performance for further exploration. The number of best parameter sets for exploring the current conditions of a HPC system is a configurable option. Based on the measured I/O performance of the top *k* parameter sets, we select the set that has the best I/O performance as the tuned I/O configuration for the I/O kernel for a given scale. One can fine tune the I/O model further by evaluating the performance results of the top *k* configurations iteratively, which is optional. In this paper, we select the best performing I/O parameter set from running the top $k = 20$ configurations. In the following subsection, we explain model development and configuration selection process with more details.

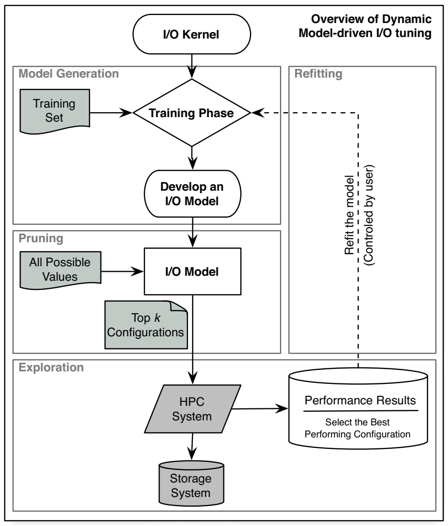


Figure 1: Overview of our proposed dynamic model-driven I/O tuning process.

A. Development of Empirical Performance Models

We now examine nonlinear regression models in the context of modeling I/O write times for a given application. As discussed previously, the main I/O parameters on a Lustre file system are Lustre stripe settings (e.g., stripe count and stripe size) and MPI-IO collective buffering settings (e.g., number of collective buffering nodes and collective buffering size). In order to identify potential challenges and illustrate our approach, we first vary only the stripe settings. In subsequent tests, we also vary the collective buffering settings and consider multiple file sizes.

1) Model for a single node writes: We begin by examining the problem of building a single model for write times as Lustre settings are modified. In order to isolate Lustre settings, we developed a micro-benchmark that uses POSIX I/O from a single node to write a single file on the Lustre file system on Hopper. We fixed the file size to about 20 GB (20 * 1024 = 20480 MB). Since we have a single node using POSIX I/O, the number of I/O aggregators is also fixed. Table I shows the different stripe settings that comprised the set of training configurations in this first set of experiments.

Parameter	Tested Values	# of Values
c, stripe count	1,2,4,8,16,32,64,96,128,156	10
s, stripe size (MB)	1,2,4,8,16,32,64,96,128	9

Table I: Training configurations (90 in total) tested as part of the single-node experiment.

One of our goals in this initial analysis was to inspect write time variability in simple settings (in this case, using a single node). Therefore, we evaluated all the 90 training configurations (from Table I) in four different experiments (each taking place on different days of a week) in order to increase our chances of encountering different levels of interference from the I/O activity of other jobs running on a shared system, such as Hopper.

Figure 2 shows the 360 write times recorded as part of these four experiments. Here, the 90 training configurations are sorted by the minimum write time across the four experiments. Variability within a particular configuration is illustrated by a vertical line connecting the four write times for that experiment. It can be seen that, even in this single-node setting, interference/noise, possibly from other jobs sharing the Lustre file system can have a significant impact on the I/O performance.

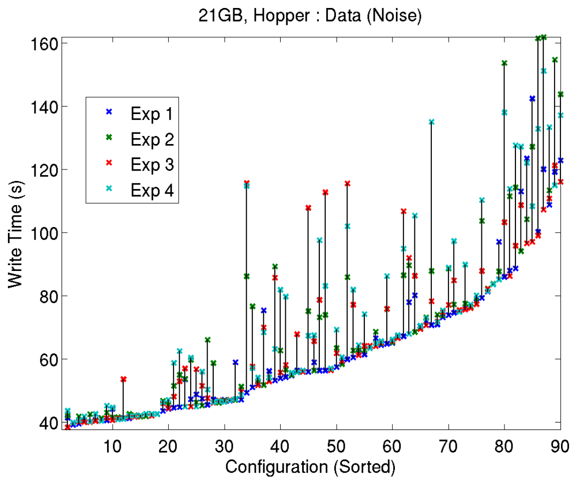


Figure 2: Performance variability and effect of interference on a single-node writing to a file.

This variability can significantly complicate the modeling process since it necessitates a more careful definition of the modeling objectives prior to performing experiments. For example, if one wishes to model "average" I/O performance, the experimental setup would need to sufficiently sample across different system states/sources of the variability. Furthermore, since variability differs among different configurations, modeling it over the entire tunable parameter space would be a challenging task.

In our context, we are interested in identifying sets of high-performing configurations (that are not already in the training set) for subsequent evaluation. While prediction of I/O performance may not be accurate in an absolute sense, obtaining high-performing configuration has been our goal. In Figure 2, we observe that the highest-performing configurations tend to be less sensitive to noise; reordering the configurations based on the mean or median of the four experiments has little effect on the constituents of the highest-performing quartile. Consequently, we have decided to use the minimum time of each of the experiments in building our models dynamically.

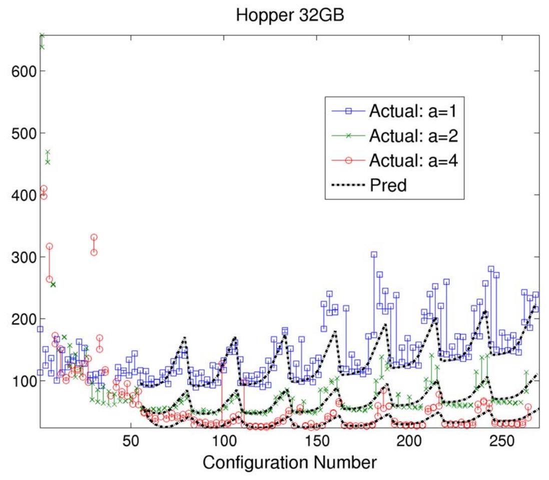


Figure 3: Raw data and nonlinear model of Equation (2) for VPIC-IO write times as the number of aggregators is varied.

To form nonlinear regression based prediction model, we use all possible low polynomials (-1, 0, and 1) of all given parameters. Given the two Lustre file system parameters, c (the stripe count) and s (the stripe size), there are 9 possible terms in the basis set of our nonlinear regression model. Using a *forward-selection* approach [1], we select $\{1, c, s, \frac{1}{c}, \frac{1}{s}, \frac{s}{c}\}$ as basis functions. Using these nonlinear basis functions and all the 90 data points (from Table I), we obtain the following model for predicting the data write time, where β_i are constants.

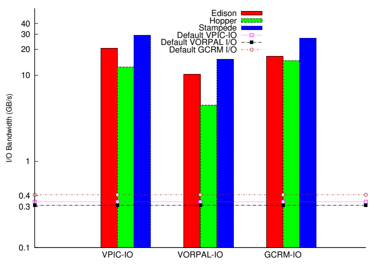
$$m(c, s) \quad = \beta_1 + \beta_2 c + \beta_3 s + \beta_4 \tfrac{1}{c} + \beta_5 \tfrac{1}{s} + \beta_6 \tfrac{s}{c}. \qquad (1)$$

2) Write time models for multiple nodes: Having observed that nonlinear regression models can predict the trend of I/O performance when one node is writing to one file, we show how such a model can be used when writing to a shared file from multiple nodes.

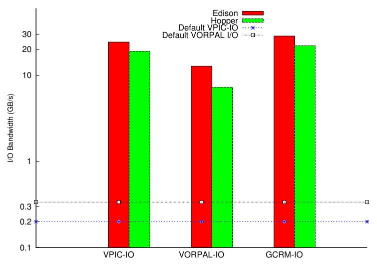
To this end, we use the VPIC-IO benchmark with 128 cores and a file size of 32 GB.

For training, we consider the same 90 combinations of the stripe size, s, and stripe count, c, shown in Table I, but we enrich the configuration space to include aggregators, a. In particular, we consider one, two, and four collective buffering nodes, for a total of 270 (c, s, a) configurations. We performed two different runs of the 270 configurations, with the training data again taken as the minimum write time over these two runs.

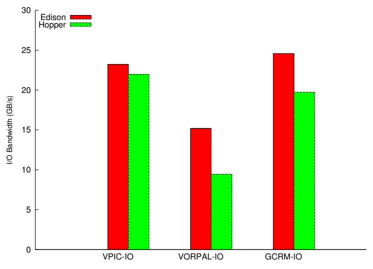
The data, shown in Figure 3, reveal that, for small stripe count values ($c \leq 2$), the write behavior is difficult to predict with the simple models considered here. Consequently, we



(a) 4K cores



(b) 8K cores



(c) 16K cores

Figure 4: Summary of the best I/O performance obtained in the top-20 configurations for each I/O benchmark running on (a) 4K cores, (b) 8K cores, and (c) 16K cores. Note that (a) and (b) are log-scale plots.

formed our model on the basis of the remaining 216 configurations. As illustrated in Figure 3, the five-term model is as follows:

$$m(c, s, a) = \beta_1 + \beta_2 \frac{c}{a} + \beta_3 \frac{s}{a} + \beta_4 \frac{1}{a} + \beta_5 \frac{a}{cs}. \quad (2)$$

We observed this model reproducing the training data well. The accuracy is assessed by having acceptable statistical measures (e.g. R-squared of 0.95 for VPIC on Stampede). Furthermore, even though the models we consider in this paper do not directly account for the variability, we observe that our realized predictions tend to yield more accurate predictions for those configurations where little variability is seen as can be seen in Figure 2.

Thus far, we have only considered a single file size when building nonlinear regression models. This modeling approach reflects the typical workflow in automatic empirical performance tuning, where one wishes to determine parameter values for actionable decisions. One of the main benefits of our models is that their simple, parameterized, algebraic form allows us to very quickly solve optimization problems involving them.

B. Training the Performance Models

We now consider models for multiple different file sizes. The four independent variables, i.e., $\mathbf{x} = (c, s, a, f)$, form a total of 81 possible terms in the basis set.

We conducted experiments for all the three I/O kernels mentioned in Section III-B and different file sizes on all the three platforms, i.e., Hopper, Edison, and Stampede. The training set size on 512 cores was 336 configurations, on 1028 cores it was 180 and on 2048 cores it was 96. The size of the training set is decreased as the core counts and file sizes increase due to the increase in the required resources.

The selection of training set can be automatic with simple heuristics of limits on the allowable value ranges in order to cover the parameter space well. For example, the maximum number of aggregators are limited by the number of MPI processes of the application. Additionally, commands such as "*lfs osts*" obtains the number of OSTs available on a Lustre file system, which can be stored in a configuration file. Once the limits are known, to establish a training set one can use all discrete integer values as possible tunable parameter values. Another strategy is to use powers-of-two or halves-of-max-allowable values. An expert can set these values more judiciously. Since the training is done infrequently, this can be decided based on the training set exploration time budget.

Following the forward-selection approach on the entire training data set, as we defined in [1], we obtain one model for each application on each platform. Due to lack of space, we only provide the model for VPIC-IO on Edison:

$$m(\mathbf{x}) = \beta_1 + \beta_2 \frac{1}{s} + \beta_3 \frac{1}{a} + \beta_4 \frac{c}{s} + \beta_5 \frac{f}{c} + \beta_6 \frac{f}{s} + \beta_7 \frac{cf}{a}, \quad (3)$$

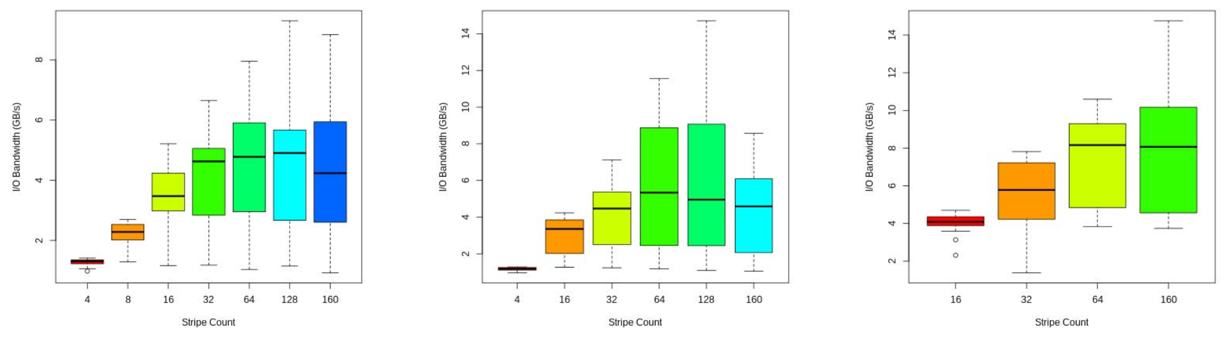
with a fit to the data yielding

$$\hat{\beta} = [10.59,\ 68.99,\ 59.83,\ -1.23,\ 2.26,\ 0.18,\ 0.01]\,.$$

The terms in (3) are interpretable from the parallel I/O point of view. For instance, write time would have an inverse relationship with number of aggregators and stripe count, because as we increase those the I/O performance tend to increase; It should have a linear relationship with file size as increasing the file size causes an increase in the write time. We describe the detailed validation of this model to section V-B. Additionally, in the next section we will analyze in detail this model's ability to perform space reduction and optimization for a variety of I/O tuning tasks.

C. Refitting the Performance Models

After training the model for the search space pruning step, the process of choosing the top k configurations only involves evaluating the model, a task whose computational expense is negligible (relative to evaluation of a configuration) for our simple choice of models. Therefore, using such an approach will only require an evaluation of a few configurations on the platform, decreasing the optimization time significantly. In our experiments, the top twenty configurations always resulted in high I/O bandwidth. As opposed to our existing GA-based approach, our approach does not



(a) 512 cores

(b) 1K cores

(c) 2K cores

Figure 5: Effect of Lustre's stripe count at three different scales of VPIC-IO on Stampede with (a) 512 cores, (b) 1K cores, and (c) 2K cores

spend excessive time in evaluating configurations that have low performance.

In case our approach is not able to achieve I/O rates competitive with those in the training set, it is possible to simply refit the model using the new results gathered at the exploration step. We note that the choice for the number of top performing configurations is a variable parameter that one can choose.

V. Experimental Results

In this section, we first present the I/O performance results for the three I/O kernels at different scales on the three platforms. We compare the achieved I/O bandwidth and the overall improvement compared to the default I/O settings. We then analyze the interdependencies of the I/O parameters by taking a closer look at these results.

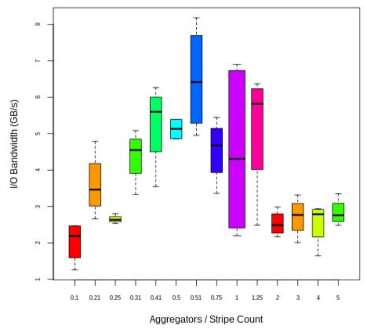
A. *Overall Performance Improvement*

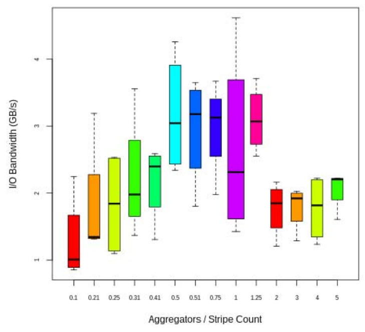
The best I/O bandwidth results we have obtained for each of the applications on different platforms are summarized below. For each experiment, this is the best performing configuration among the top-20 configurations predicted by the model. Figure 4 shows the I/O bandwidth grouped by the number of cores from 4K to 16K. For all these experiments we used the training phase experiments without a refitting phase. As can be observed, the I/O bandwidths of the kernels are in the range of 5-30 GB/s, which is efficient performance for writing to one shared file on these platforms at their respective scales. We also show the default I/O performance of the applications for their respective concurrencies at 4K and 8K on the Hopper platform. Compared to the default performance, which is in the range of 0.3-0.4 GB/s, our tuned configurations perform 6X-94X better. As the number of Lustre OSTs on Edison and on Stampede are similar to that of Hopper, we expect the default performance and our speedup to be at the same level. Note that for the Stampede platform, we have scaled our runs only up to 4K cores due to queue policies in running large-scale tests.

Table II summarizes the achieved I/O bandwidths for the three I/O kernels running at different concurrencies on the three platforms. The table also shows the size of the data written to the file system. The time to traverse the search space after training less than three hours. In most cases, exploring the top twenty configurations took less than one hour, resulting in significant improvements to overall parallel I/O performance.

# cores	I/O Kernel	File Size (GB)	Edison (GB/s)	Hopper (GB/s)	Stampede (GB/s)	Hopper Default (GB/s)
512	VPIC	128	8.19	3.00	9.30	0.39
	VORPAL	140.625	3.24	2.67	7.76	0.44
	GCRM	166.4	9.78	5.27	11.62	-
1K	VPIC	256	14.24	5.09	14.71	0.32
	VORPAL	281.25	9.91	2.34	9.10	0.41
	GCRM	166.4	14.63	6.70	13.28	-
2K	VPIC	512	19.72	8.18	14.75	0.40
	VORPAL	562.5	17.81	4.63	12.67	0.36
	GCRM	665.6	23.96	6.82	21.05	0.24
4K	VPIC	1024	20.57	12.57	29.20	0.34
	VORPAL	1197	10.26	4.50	15.35	0.31
	GCRM	2600	16.64	10.59	26.99	0.41
8K	VPIC	2048	24.32	18.93	-	0.20
	VORPAL	2250	12.77	7.26	-	0.33
	GCRM	10400	28.60	22.09	-	-
16K	VPIC	512	23.21	21.96	-	-
	VORPAL	4394	15.20	9.45	-	-
	GCRM	10400	24.58	19.73	-	-

Table II: Highest bandwidth achieved for the three applications by selecting the best-performing configuration suggested by our proposed framework.


(a) VPIC-IO


(b) VORPAL-IO

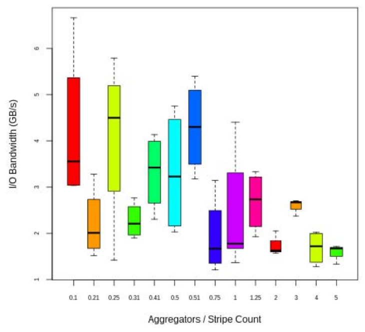

(c) GCRM-IO

Figure 6: Ratio of MPI-IO's aggregators and Lustre's stripe count on three different applications on 2K cores of Hopper

B. Analysis of Interdependencies Among I/O Layers

In this subsection, we analyze some of the interdependencies of the parallel I/O tunable parameters by looking at the results of the experiments we conducted. All the data gathered for all the applications on the platforms is not possible to put in this paper due to space restrictions; therefore we try to have each analysis on different applications and on different platforms. We first analyze the impact of individual tuning parameters (stripe count, number of aggregators, and stripe size) on performance, and then discuss the combined impact of stripe count and aggregators, stripe size and aggregators.

In order to analyze the effect of Lustre's stripe count parameter on I/O performance of an application we look at the three different scales for which we ran VPIC-IO on Stampede. Figure 5 shows the box plots for these experiments, where each of the plots contains all the training set configurations for corresponding concurrency (i.e., 512 cores, 1024 cores, and 2048 cores). We can observe that as the stripe count increases the I/O performance improves (especially at higher concurrencies of) VPIC-IO application since the amount of data to be written is large. This behavior is exactly reflected in the model since it tries to use all available OSTs for VPIC-IO.

Figure 7 shows the variation of the number of aggregators on VPIC-IO's training set on Stampede. Similar to Lustre's stripe count, increasing the number of aggregators helps in improving the I/O performance for VPIC-IO. Therefore, the model has taken this into account and tries to maximize the number of aggregators for the larger-scale testing sets.

Figure 8 shows the same box plots for the stripe size of VPIC-IO on Stampede. As illustrated the plot, Lustre's stripe size does not have the same behavior as the stripe count and each of the values chosen in the training set for the stripe size has shown both good and bad I/O performance, depending on the values of other I/O parameters. As we show next, the model's behavior for this value is interesting.

Now that the variations and the performance of individual parameters are observed, we analyze the top twenty

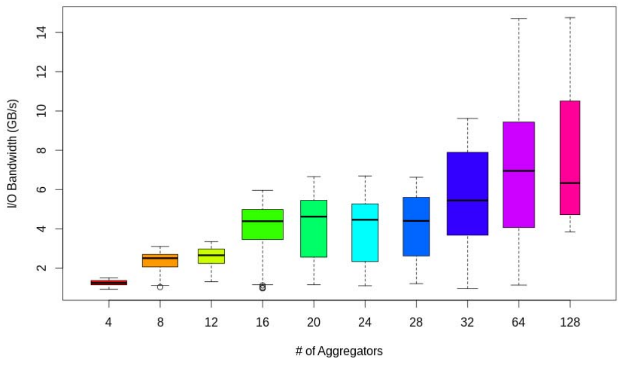


Figure 7: Effect of MPI-IO aggregators using the training set of VPIC-IO on Stampede.

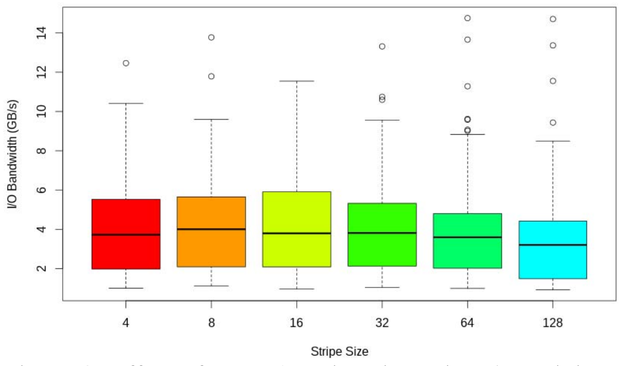


Figure 8: Effect of Lustre's stripe size using the training set of VPIC-IO on Stampede..

configurations predicted by the model for a larger-scale VPIC-IO experiment on Stampede. Table III contains the configurations proposed by the model as the top performing configurations for VPIC-IO using 4K cores of Stampede, which leads to an output file of size 1024 GB. As noted before, the number of aggregators is chosen to be the maximum of 1024 and the stripe counts are varying from 156 (maximum in the testing set) to 64. Since there is no strong correlation in stripe size and I/O performance in the training sets, all the stripe sizes in the testing sets are chosen by the model to be tested. Looking at this table, we can observe that for experiment (exp_id) 6, the highest I/O bandwidth of roughly 30 GB/s is achieved with a stripe size of 64MB.

exp_id	c	s	a	f (GB)	time (s)	bandwidth (GB/s)
0	156	1	1024	1024	58.87	17.39
1	156	2	1024	1024	49.84	20.54
2	156	4	1024	1024	47.06	21.75
3	156	8	1024	1024	42.11	24.31
4	156	16	1024	1024	38.99	26.25
5	156	32	1024	1024	40.28	25.41
6	**156**	**64**	**1024**	**1024**	**35.06**	**29.20**
7	156	128	1024	1024	44.96	22.77
8	128	1	1024	1024	61.33	16.69
9	128	2	1024	1024	65.87	15.54
10	128	4	1024	1024	58.94	17.37
11	128	8	1024	1024	54.72	18.71
12	128	16	1024	1024	68.53	14.94
13	128	32	1024	1024	61.76	16.57
14	128	64	1024	1024	49.47	20.69
15	128	128	1024	1024	57.31	17.86
16	64	1	1024	1024	104.13	9.83
17	64	2	1024	1024	95.14	10.76
18	64	4	1024	1024	129.01	7.93
19	64	8	1024	1024	78.20	13.09

Table III: The top-20 configs predicted by the model and their I/O bandwidth for VPIC-IO on 4K cores of Stampede (c: stripe count; s: stripe size; a: aggregators; f: file size).

Another interesting behavior we found in the results of the training sets experiments is a relationship between the number of aggregators and the stripe count. We analyze this relationship using the "ratio of the number of aggregators to the stripe count". This relationship makes sense from the parallel I/O perspective as the number of aggregators each OST handles has an impact on concurrency of Lustre and the communication between an aggregator and an OST.

Figure 6 shows the impact of the ratio of aggregators to the stripe count for various I/O kernels running on Hopper at a concurrency of 2K. We can observe that for both VPIC-IO and VORPAL-IO, the impact of the ratio is similar, while GCRM-IO shows a different behavior. It is not surprising to see that the higher ends of the spectrum is not performing well for all kernels as they are related to those experiments with lower stripe count. There is a peak in the middle of this plot, where we can obtain the best I/O performance for VPIC-IO and VORPAL-IO. This is where both stripe count and aggregators are large enough to get the most parallelism, but not too large so that the overhead causes the performance to drop. This is different for GCRM because the stripe count should be large but the number of aggregators should not be that large.

Analyzing the top twenty results predicted by our model once we ran them on the platforms provides insight as well. Here we show some of the insights that we think are important for the scientific community to achieve efficient parallel I/O performance.

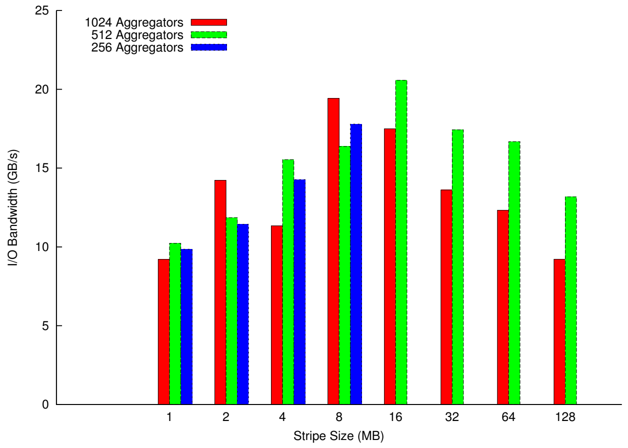


Figure 9: Effect of Lustre's stripe size on performance of the top-20 VPIC-IO configurations on 4K cores of Edison. Stripe count is fixed at 96 (maximum # of OSTs on Edison).

The first insight we gained is the role of Lustre's stripe size. Figure 9 compares the performance of the top twenty configurations obtained by the model for VPIC-IO on 4K cores of Edison. The stripe count for all these configurations are fixed to 96 thus it is easy to compare the impact of stripe size in one plot. The three bars in different colors show the numbers for three different sets of aggregators chosen by the model and the X-axis shows different values for stripe size in MB. We can observe the difference between poor performing and best performing configurations is almost two-fold. This behavior is similar to what we observed in Table III, which is the same application at the same concurrency on Stampede; However, on Edison, the best I/O performance was gained when stripe size is equal to 16, while on Stampede that is 64. This shows that depending on the platform, the values of these parameters are different and underscores that the selection of stripe size has an impact on I/O performance contrary to a recent study [12] that downplays this impact.

Another insight we gained from the results is that unlike Lustre's stripe count, where increasing the number of OSTs gives better performance, the number of aggregators exhibits a sweet spot depending on the I/O pattern of an application. Figure 10 demonstrates this impact for the VORPAL application on 16K cores of Edison. Fourteen configurations out of the top twenty proposed by our model for this experiment have stripe count equal to 96 and therefore we can compare the effect of aggregators for each stripe size value. Based on the plot, one can conclude that having too many aggregators does not provide good performance (most likely because of high overhead). On the other hand, having too few aggregators is suboptimal because nodes are not able to saturate the I/O bandwidth. We can conclude from

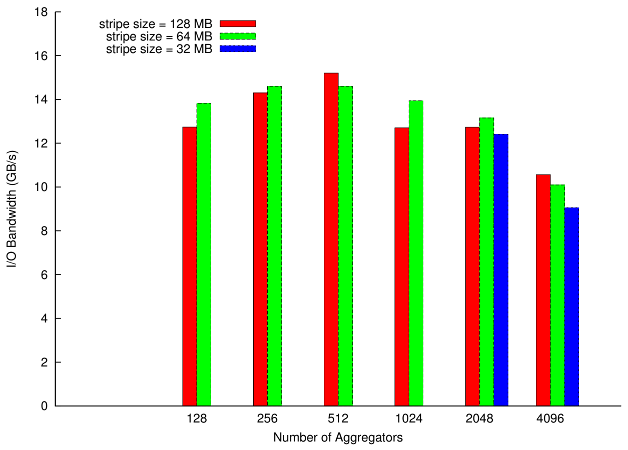


Figure 10: Effect of aggregators on performance of 14 configurations of the top 20-experiments on 16K cores of Edison. Stripe count is fixed at 96 (maximum # of OSTs on Edison)

the plot that the value of stripe size has a role in choosing the number of aggregators as well proving the existence of the interdependency among various I/O parameters.

In summary, we list the following findings from our analysis of I/O performance:

- Irrespective of the platform, as we increase the size of a file, increasing Lustre's stripe count, causes more parallelism and therefore results in an improvement in the I/O bandwidth.
- Lustre's stripe size is an important factor in tuning I/O performance. It can have a dramatic impact on I/O performance and its values depend on the I/O operations and the other I/O parameters (e.g., aggregators) and the HPC platform.
- The number of MPI-IO aggregators should be specified carefully and not blindly minimized or maximized. This parameter also depends on the I/O operations, other I/O parameters, the platform, and the amount of communication happening in the application (i.e., the I/O pattern of the application).

VI. Related Work

Tuning the I/O subsystem has unique challenges. Although the computation kernels run for a few milliseconds, evaluation of I/O functions can take minutes. Due to the complexity and interdependency among multiple layers of the I/O system, searching for tuned parameters is a cumbersome process. There have been several efforts in predicting I/O performance of applications and parallel systems and in tuning I/O performance. In the following, we briefly discuss the most related work to our exploration.

Our recent work used a heuristic-based search with a GA in order to tune I/O performance [3]. However, this heuristic search process has a prohibitive run time and limited applicability at a different concurrency from the trained problem size. Our performance modeling approach filters the number of combinations to a small number and then searches within the smaller space. Table IV shows a comparison of these approaches. With default configuration without any I/O tuning, each application will take more than 3 hours. With Genetic Algorithms, for each application and scale, a cost of more than 10 hours is paid for tuning. With current approach, the cost of training is paid once and then for each application applying the model takes less than an hour with fast application run time. Howison et al. studied manually tuning HDF5 applications on Lustre file systems [8]. Using analytical models, our work automates the tuning process. McLay et al. study tuning parallel I/O on a specific system and suggest that maximizing stripe count has a significant impact on performance [12]. As mentioned earlier, our study explores the interdependencies of various parameters and shows the impact of multiple parameters on the overall performance.

Method	Training Phase	Applying the Model	Per App. & Scale Tuning	App. Runtime (VPIC-8192 on Hopper)
GA	N/A	N/A	$>$ 10 hours	118 seconds
Model Fitting	$>$ 10 hours (can reuse)	$<$ 1 minute (automatic)	$<$ 1 hour	100 seconds
Default Config.	none	none	none	$>$ 3 hours

Table IV: A comparison of GA, modeling and default configuration.

There have been several efforts in predicting parallel I/O performance. Shan et al. [17] use the IOR benchmark to match the I/O patterns of an application and predict I/O performance. Meswani et al. [13] use a similar strategy by running the I/O operations of an application on a reference system and calibrate the performance of the reference system with a target system. Smirni et al. [18] use a queuing network model to predict the performance of RAID-3 disks. Song et al. [19] propose an analytical model to predict the cost of read operations for accessing data organized in different layouts on the file system. While many of these efforts seek to predict I/O performance accurately, our work uses the models to identify fruitful parameter values and then iterates in the executing and refitting stages by searching among this smaller set of parameter values. Using this approach, we have shown that our technique is fast and effective in achieving good I/O performance. Herbein et al. [11] use a statistical model, called surrogate-based modeling to predict the performance of the I/O operations of HPC applications. Similar to our work, the modeling is used to reduce the time to search for the optimal parameters. Although the modeling is general, the parameters used in [11] are more specific to ADIOS library while the parameters we used in this paper is general and can be applied to any parallel I/O application using Lustre.

VII. Conclusion and Future Work

Parallel I/O is an integral part of modern HPC, however it remains challenging to obtain maximum performance from I/O subsystems. This is mainly due to interdependencies among multiple layers of the parallel I/O stack. The values for the parameters at each layer of this stack are critical to the I/O performance and they vary across applications, platforms, and the concurrency of the application.

In this paper, we present a model-driven tuning framework which exploits nonlinear regression models to find the top performing values for these parameters in order to decrease the amount of I/O time in HPC applications. We show that our approach achieves significant portion of the available I/O performance of various HPC platforms for a range of applications.

Another main contribution of this paper is shedding light on the complex interdependencies of different parallel I/O tunable parameters and how they vary with different experiments. We show that Lustre's stripe count and stripe size along with MPI-IO aggregators are all critical factors for I/O performance. Our dynamic, model-driven approach makes the search process for selecting these parameters easier.

As future work, we plan to tie the models we developed in this paper to the I/O patterns of applications. Additionally, we will explore isolating interference in storage systems, mostly due to various jobs sharing the same resources. We plan on studying the network and storage activity and their impact on I/O performance.

Acknowledgment

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract numbers DE-AC02-05CH11231 and DE-AC02-06CH11357. This research used resources of the National Energy Research Scientific Computing Center and Texas Advanced Computing Center

References

[1] B. Behzad, S. Byna, S. M. Wild, M. Prabhat, and M. Snir. Improving Parallel I/O Autotuning with Performance Modeling. HPDC '14, 2014.

[2] B. Behzad, H.-V. Dang, F. Hariri, W. Zhang, and M. Snir. Automatic Generation of I/O Kernels for HPC Applications. PDSW '14, pages 31–36, Piscataway, NJ, 2014.

[3] B. Behzad, L. Huong Vu Thanh, J. Huchette, S. Byna, Prabhat, R. Aydt, Q. Koziol, and M. Snir. Taming Parallel I/O Complexity with Auto-Tuning. SC '13, 2013.

[4] E. W. Bethel, J. M. Shalf, C. Siegerist, K. Stockinger, A. Adelmann, A. Gsell, B. Oswald, and T. Schietinger. Progress on H5Part: A portable high performance parallel data interface for electromagnetics simulations. PAC 07, 2007.

[5] K. J. Bowers, B. J. Albright, L. Yin, B. Bergen, and T. J. T. Kwan. Ultrahigh performance three-dimensional electromagnetic relativistic kinetic plasma simulation. *Physics of Plasmas*, 15(5):7, 2008.

[6] S. Byna, A. Uselton, Prabhat, D. Knaak, , and Y. He. Trillion particles, 120,000 cores, and 350 TBs: Lessons learned from a hero I/O run on Hopper. 2013.

[7] J. M. del Rosario, R. Bordawekar, and A. Choudhary. Improved parallel I/O via a two-phase run-time access strategy. *SIGARCH Comput. Archit. News*, 21(5):31–38, 1993.

[8] M. Howison, Q. Koziol, D. Knaak, J. Mainzer, and J. Shalf. Tuning HDF5 for Lustre File Systems. IASDS '10, Sept. 2010.

[9] S. Kumar, A. Saha, V. Vishwanath, P. Carns, J. A. Schmidt, G. Scorzelli, H. Kolla, R. Grout, R. Latham, R. Ross, M. E. Papkafa, J. Chen, and V. Pascucci. Characterization and modeling of PIDX parallel I/O for performance optimization. SC '13, pages 67:1–67:12, 2013.

[10] J. Logan, S. Klasky, J. Lofstead, H. Abbasi, S. Ethier, R. Grout, S.-H. Ku, Q. Liu, X. Ma, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. Skel: Generative Software for Producing Skeletal I/O Applications. ESCIENCEW '11, pages 191–198, Washington, DC, 2011. IEEE Computer Society.

[11] M. Matheny, S. Herbein, N. Podhorszki, S. Klasky, and M. Taufer. Using Surrogate-based Modeling to Predict Optimal I/O Parameters of Applications at the Extreme Scale. 2014.

[12] R. McLay, D. James, S. Liu, J. Cazes, and W. Barth. A user-friendly approach for tuning parallel file operations. SC '14, pages 229–236, Piscataway, NJ, 2014.

[13] M. Meswani, M. Laurenzano, L. Carrington, and A. Snavely. Modeling and predicting disk I/O time of HPC applications. HPCMP-UGC '10, pages 478–486, 2010.

[14] C. Nieter and J. R. Cary. VORPAL: A Versatile Plasma Simulation Code. *Computational Physics*, 196:448–472, 2004.

[15] J. Oly and D. A. Reed. Markov model prediction of I/O requests for scientific applications. ICS '02, pages 147–155, 2002.

[16] D. A. Randal and A. Arakawa. Design and Testing of a Global Cloud-Resolving Model. Report, 2009.

[17] H. Shan, J. Shalf, and K. Antypas. Characterizing and Predicting the I/O Performance of HPC Applications using a Parameterized Synthetic Benchmark. SC' 08. ACM/IEEE, Austin, TX, 2008.

[18] E. Smirni, C. L. Elford, D. A. Reed, and A. A. Chien. Performance modeling of a parallel I/O system: An application driven approach. PPSC. SIAM, 1997.

[19] H. Song, Y. Yin, Y. Chen, and X.-H. Sun. Cost-intelligent application-specific data layout optimization for parallel file systems. *Cluster Computing*, 16(2):285–298, June 2013.

[20] R. Thakur, W. Gropp, and E. Lusk. *Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation*. Mathematics and Computer Science Division, Argonne National Laboratory, Oct. 1997.

[21] R. Thakur, W. Gropp, and E. Lusk. Data sieving and collective I/O in ROMIO. FRONTIERS '99, page 182, Washington, DC, 1999. IEEE.

[22] H. You, Q. Liu, Z. Li, and S. Moore. The design of an auto-tuning I/O framework on Cray XT5 system. CUG '11, Fairbanks, Alaska, May 2011.