Improving Collective I/O Performance with Machine Learning Supported Auto-tuning

Ayşe Bağbaba
The High-Performance Computing Center Stuttgart (HLRS)
University of Stuttgart
Stuttgart, Germany
Email: bagbaba@hlrs.de

***Abstract*—Collective Input and output (I/O) is an essential approach in high performance computing (HPC) applications. The achievement of effective collective I/O is a nontrivial job due to the complex interdependencies between the layers of I/O stack. These layers provide the best possible I/O performance through a number of tunable parameters. Sadly, the correct combination of parameters depends on diverse applications and HPC platforms. When a configuration space gets larger, it becomes difficult for humans to monitor the interactions between the configuration options. Engineers has no time or experience for exploring good configuration parameters for each problem because of long benchmarking phase. In most cases, the default settings are implemented, often leading to poor I/O efficiency. I/O profiling tools can not tell the optimal default setups without too much effort to analyzing the tracing results. In this case, an auto-tuning solution for optimizing collective I/O requests and providing system administrators or engineers the statistic information is strongly required. In this paper, a study of the machine learning supported collective I/O auto-tuning including the architecture and software stack is performed. Random forest regression model is used to develop a performance predictor model that can capture parallel I/O behavior as a function of application and file system characteristics. The modeling approach can provide insights into the metrics that impact I/O performance significantly.**

***Keywords*-Collective I/O; Auto-tuning; Machine learning; MPI-IO**

I. Introduction

Input and output (I/O) performance impacts the entire system utilization and productivity. Understanding the I/O requests of user applications, parallel I/O stack and even the specification of storage systems helps the computing centers to increase their efficiency.

Collective I/O is an essential approach in high performance computing(HPC) applications. The achievement of effective collective I/O is a nontrivial job due to the complex inter-dependencies between the layers of I/O stack [1]. The complexity of today's supercomputers has significantly increased the challenges of optimizing the parallel applications' performance [2]. Different layers of the parallel I/O subsystem offers a set of tunable parameters to achieve the best possible I/O performance [1]. However, the configuration of these parameters demand a balance among diverse factors, such as the I/O application, storage hardware, data size, and number of processors. When a configuration space gets larger, it becomes difficult for humans to monitor the interactions between the configuration options. Engineers has no time or experience for exploring more than just a few good configuration parameters for each problem because of long benchmarking phase. Therefore, default configuration setup for I/O tuning is done by system administrators. In most cases, the default settings lead to poor I/O performance [1]. As the complexity of large-scale HPC systems grow, this brings further difficulties to achieve high-performance I/O through lack of global optimizations. Available I/O profiling tools can not tell the optimal system default setups by easily analyzing the tracing results by experts [3]. In this case, there is an increasing demand for new studies to address these challenges.

This paper presents a case study of a machine learning supported I/O auto-tuning for the popular collective I/O implementation of ROMIO. This study tries to tune I/O configuration parameters between layers transparently to improve the I/O performance of parallel applications and to provide statistical information to system administrators.

The collective I/O implementations are commonly used in checkpointing and analysis of computational science applications [2]. Considering its highly variable performance, collective I/O is a good target to optimize. The provision of expert knowledge from observations automatically would be useful. Analytical or black-box models could be used for predicting performance and providing statistical information of collective I/O. Using a predictive modeling approach can give insights into the file system metrics with an important impact on I/O performance [4].

The main contributions of the paper are the following. First, it states challenges of the different layers' run time factors on the optimization process. Second, it points collective I/O performance variability and work on development of an machine learning supported auto-tuning to hide the complexity of different layers from developers. Third, it evaluates development of predictive performance models to extract expert knowledge from observations automatically.

The rest of the article is structured as follows: Section II motivates this work and gives related work regarding I/O

research. Section III presents experiment results with several important performance factors in collective I/O implementations and the modeling approach. Section IV provides an overview of the proposed methodology. Section V presents implementation of auto-tuning and performance models. Section VI concludes the paper and discusses future steps.

II. Related Work

Among different optimizing potentialities, I/O request is one of the most inquired parts [3]. ROMIO [5], a widely used MPI-IO implementation, offers collective I/O optimization with the promise to improve I/O performance by merging the I/O requests of different processes in a single call.

Optimizing collective I/O has been deeply investigated. The challenge of this approach is that parameters need to be carefully evaluated and tuned. Tools such as Vampir [6] and Darshan [7] can be used for monitoring and analysis of system state and performance, however; they are helpful in the profiling, they cannot set configuration parameters [8].

Scalable I/O for extreme performance (SIOX) has an I/O tracing thread running on each compute node to inquire the appropriate I/O access patterns and to achieve a real-time parallel I/O optimization, but overhead produced by the MPI instrumentation is too high in a production environment [9]. Pattern-driven parallel I/O tuning for HDF5 applications is developed to optimize I/O performance of HDF5 applications across platforms and applications automatically [10]. A solution based on MPI-IO library could be more widely used and supports parallel HDF5. [3] presents an I/O auto-tuning framework for MPI-IO library, with an approach trying all combinations of possible configuration parameters to find the best. Using statistical methods in such a framework would improve I/O performance and save core hours. Investigating how to improve the ranking of optimal configurations and how to improve collective IO performance for small files is still an important issue in auto-tuning.

Several studies have attempted to include machine learning in the I/O analysis and optimization steps. Due to complexity of the state-of-the-art file systems, using analytical models are often inadequate and time consuming for expected predictive accuracy [11]. Upon this, several researchers have focused on empirical and machine learning approaches to model the I/O performance. Decision tree algorithm is used to develop an I/O performance model for optimization of ROMIO data sieving approach [9]. A semi-empirical solution to model the performance of MPI-IO operations is developed in [1]. Isaila et al. [2] integrated analytical and machine learning approaches to model the performance of ROMIO collectives.

III. The Problem and Setup

A. Experiments

To compare an efficient optimization to an inefficient one, Interleaved Or Random (IOR) benchmark is used to simulate the applications' I/O write requests in different problem sizes with 1200 processes on Hazel Hen (Cray XC40) with Lustre file system at HLRS. Technical details of Hazel Hen and Lustre file system are given in Table I.

Table I: Technical Details of Hazel Hen (Cray XC40)

Architecture	Cray XC40
Hardware	Intel Xeon E5-2680 v3 Cray Aries Network 7712 Compute nodes 90 Service nodes
File System	Lustre 7 MDTs 54 OSTs
Storage	Cray Sonexion 2000
Bandwidth	3.75 GB/s per OST

The MPI collective I/O simulations ran with changing configuration parameters such as data transfer size, number of processes, Lustre striping values and permission of collective I/O optimizations in ROMIO. In Figure 1, the I/O simulations were configured with MPI collective I/O write operation that access a single shared file, using same data transfer size on each process for each case, and by using MPI hints to control the Lustre striping setups. The results of each case are obtained from 10 running samples. In the first case of Figure 1, for a small data transfer size (32KB) on Lustre, the writing performance of striping over 4 Object Storage Targets (OSTs) with 1 MB stripe size is about 71 % better than the performance of striping over 16 OSTs with 1 MB stripe size. Disabling the collective buffering optimization in ROMIO, which is normally not recommended, achieved about 269 % writing performance improvement in the second case of Figure 1 for a non-small data transfer size 64 MB. Third and fourth cases of Figure 1 show that there is an optimal range for Lustre stripe size depending on data transfer size, because collective I/O performance does not keep raising together with stripe size.

The problems shown in these cases are just a tip of the iceberg. The challenge of collective I/O is that configuration parameters needs to be carefully evaluated and tuned by an expert. In long benchmarking step with different benchmarks, a lot of potential I/O optimization have been identified to improve the performance of HPC systems by avoiding unsuccessful tuning attempts.

B. Performance Factors

There are many factors involved in the collective I/O performance that can be grouped into the access pattern of application, parallel file system features, and hardware characteristics. Collective buffering is an important collective I/O optimization to limit the number of writers during a collective file operation [12].

By researching the characteristics of application, Lustre file system and the ROMIO MPI-IO library, it can be seen that the following parameters impact collective I/O performance significantly:

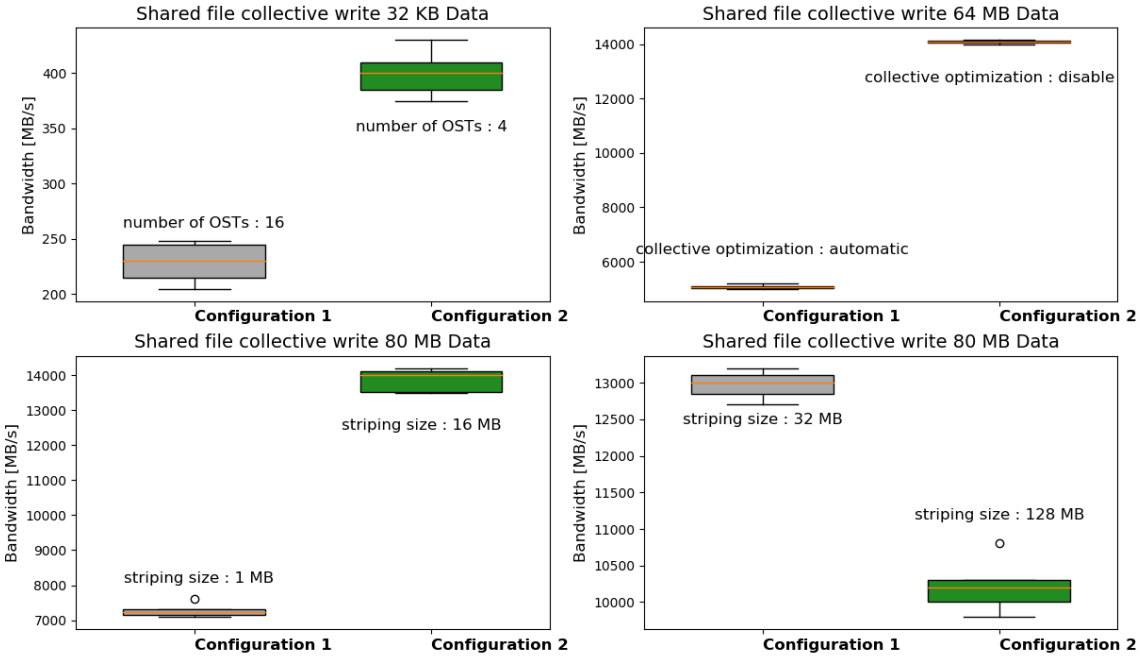

Figure 1: Collective I/O Simulation Results by Applying Different Configurations for Shared File Write.

- *number of MPI processes*: concurrency have a significant impact on the I/O performance.
- *data transfer size*: different data transfer sizes require different optimal configurations. Grouping of data transfer sizes with similar optimal configurations would be a reasonable approach.
- *MPI-IO subroutine*: tuning of different MPI-IO subroutines (collective vs. non-collective) are different.
- *romio_cb_read*: the collective buffering optimization for reading operations can be enabled or disabled.
- *romio_cb_write*:the collective buffering optimization for writing operations can be enabled or disabled.
- *striping_factor*: specifies the number of Lustre OSTs (stripe_count) to stripe new files.
- *striping_unit*: specifies the size (in bytes) of each Lustre file system OST stripe unit (stripe_size) used for new files
- *cb_buffer_size*: specifies the total buffer space that can be used for collective buffering on each target node, usually a multiple of cb_block_size. in the current default collective I/O algorithm of Cray MPI on Hazel Hen, the value of the collective buffer size, equals to the value of striping_unit.
- *cb_nodes*: specifies the number of target nodes to be used for collective buffering.

How to set the values of these configuration parameters and how to create a performance predictor based on them are the main challenges for this study.

C. Modeling

A modeling approach is considered that can model the I/O performance in terms of the application and file system characteristics. The performance model can be formally used to define the I/O performance of an application as follows:

$$\phi = f(\alpha, \zeta, \omega), \tag{1}$$

where α represents a set of observable parameters that describe application characteristics (I/O pattern, I/O operation, benchmark), ζ represents a set of observable parameters that describe file system and/or I/O characteristics (Lustre parameters), and ω represent uncontrolled non-observable parameters. In the modeling approach, our aim is to understand the relationship between ϕ and the parameters (α, ζ). For a given set of input parameter values in (α, ζ), the function f should give a prediction. If it provides distributional information (such as standard deviation) the variability in ϕ can be captured as well.

The modeling approach represents the similar cases that can be represented by using a single model. Machine learning may provide an appropriate method to analyze effects of these parameters on collective I/O behaviour.

IV. Methodology

A. I/O Auto-tuning

In this study, optimization approach consists of three basic steps:

- identifying configurations' searching scope,
- choosing the best configuration parameters,
- suggesting or tuning.

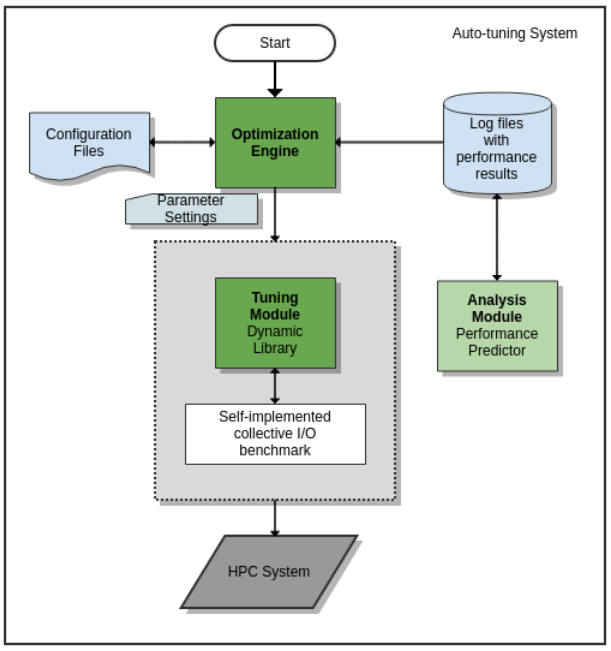

Figure 2: Overall Architecture of the Machine Learning Supported Auto-tuning Approach.

The auto-tuning system has tuning, optimization engine and analysis modules, and a monitoring module surrounds the system like a run-time tracer (Figure 2). These modules built on the MPI-IO library. As soon as user applications call MPI-IO subroutines, the tuning module is triggered to apply optimal configurations before executing the I/O operation transparently. After executing I/O operations, monitoring module records the performance results of I/O requests into log files to feed the optimization engine who extracts optimal configurations based on historical running logs for approaching a higher I/O performance in future steps. Optimization engines' task is to select proper configuration file based on number of processes and to find best configuration set in the file. Analysis module runs independently of the other modules and works like a decision support system. It supports system administrators to analyze I/O requests based on log files and helps to understand the relationship between configuration parameters and performance results by applying performance predictor models. It parses the records and generates CSV files containing all information. Different statistical approaches can be integrated into analysis module to model I/O performance in terms of performance factors.

A large amount of I/O log files are required to feed the optimizing engine to extract ideal configurations from a variety of I/O requests. To check all possible configurations for real engineering applications would need too many computer resources. Therefore, I have worked on a self-implemented I/O benchmark to simulate collective I/O. Using the I/O benchmark, modules can extract basic optimal configurations knowledge base in the training step. In time, scientists and engineers can use the auto-tuning system to optimize their applications, and so training data set could be expanded. It provides more up-to-date results.

"One process" tracing policy (rank 0 MPI process, to be responsible for the monitoring task) is used to keep overhead of monitoring module less. The file access time by each I/O process is not the same. Therefore, the longest one that indicates time of an effective I/O operation is chosen by using MPI_ALLREDUCE.

The monitoring module records all configuration parameters, the aggregated data transfer size and the processing time of the slowest process to the log file. The bandwidth (MB/s) is calculated and recorded as well by using these aggregated data transfer size and processing time. The content of a configuration file is a subset of the corresponding log file that includes only configuration parameters.

The auto-tuning approach follows the current MPI standard, runs transparently to the users and improve I/O performance automatically. These features supports to be usable for engineers and scientists with little knowledge of parallel I/O, and to be portable across multiple HPC platforms.

B. Performance Models

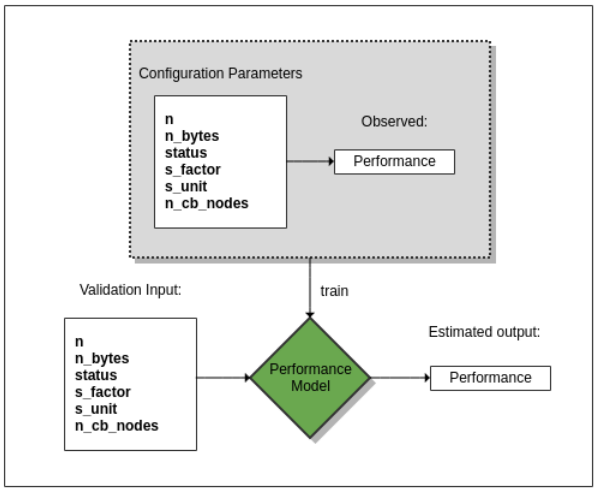

Figure 3: Performance model provides a performance estimate.

Table II: Training Set Configurations' Scope

Name	Value
n	24-1200
n_bytes	256 B - 196 MB
n_cb_nodes	1 - 16
s_factor	1 - 16
s_unit	1 MB - 32MB
status	automatic; disable; enable
IO pattern	collective

While aim is to perform the machine learning supported auto-tuning (online), in this study, the performance is modeled and machine learning algorithms are investigated offline to evaluate the performance predictions (Figure 3).

Configuration parameters may be fixed or variable. In this study, the fixed configuration parameter is the *access pattern*, "MPI_File_write_all" collective function. The variable configuration parameters are the *number of processes* (n; 24 to 1200), *number of bytes* (n_bytes; 256 B to 196 MB), *state of collective buffering optimization* (status; automatic, disable, enable), *number of collective buffering nodes* (n_cb_nodes; 1 to 16), *Lustre striping factor - number of OSTS*- (s_factor; 1 to 16) and *Lustre striping unit* (s_unit; 1 MB to 32 MB) listed in Table II). The optimization criteria is the *performance*, the bandwidth achieved under the self-implemented parallel I/O benchmark. These configuration parameters and computed performance for each configuration are stored in a CSV file. The file generates a data set to be used in training and validation of the performance model as a task of auto-tuning analysis module. The evaluation is conducted by loading the data set into the the scikit-learn Python library. Random forest regression algorithm is applied on the CSV files to learn and extract knowledge from numerous decision trees instead of producing a single decision tree.

Random forest is a supervised learning algorithm which uses ensemble learning method for classification and regression. The trees in random forests can run in parallel. There is no interaction between these trees while building the trees. It works by building a multitude of decision trees at training step and gives mean prediction (regression) of the individual trees. These decision trees are aggregated into a random forest ensemble that combines their input. Then, results are aggregated, so it can outperform any individual decision tree's output [13].

The first step is to create a performance predictor model to be expected from a given set of fixed and variable parameters listed in Table II. This performance model is trained on a number of samples so that an output value can be estimated for any given parameter set. After training the model using a subset of data set (the training set) and performance is estimated on the validation set (Figure 3).

V. Evaluation

A. Evaluation 1: Influence of Auto-tuning

Evaluations were made on Hazel Hen (Cray XC40) with an InfiniBand connected Lustre file system at High Performance Computing Center Stuttgart (HLRS). Technical details of Hazel Hen and Lustre file system are given in Table I.

As I/O benchmark software, IOR was used in evaluations to measure I/O performance. The performance factors mentioned in Section III-B have been considered in the prototype. The values of these factors were chosen in the range of values as given in Table II.

The default setup for striping configuration on Lustre was striping_factor=4 and striping_unit=1048576 on Hazel Hen. Figure 4 presents the achievement with auto-tuning by writing 32 MB data transfer size to a single shared-file collectively. When the benchmark scaled, the improvement continued to increase. The success of the approach varies between 50-130% at scale in the performance gain by finding out the optimal ones that are better than the default configurations.

In non-blocking and independent MPI-IO operations the MPI_ALLREDUCE function may be a hidden problem if the application scales out. But in the collective MPI-IO operations, this does not cause a trouble because collective functions synchronize the MPI processes implicitly. The overhead can be ignored in such cases. The overhead of the approach on 1 MPI process is measured as 0.02 seconds for MPI open and 0.2 seconds for MPI write.

B. Evaluation 2: Validation of Performance Model

To evaluate the quality of the performance model, prediction errors in MB/s for training sets under different depths of trees in the random forest are given in Table III with accuracy, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) values. The random forest regressor performs better than mean performance used as the baseline. Training set size is 2153, test set size is 539 for all tests. The Random Forest performance model is extracted based on 100 iterations. Time taken to build model is 3.19 seconds.

The observed and predicted bandwidth results are shown in Figure 5 when using depth of tree is 4 with 90.52% accuracy, in Figure 6 when using depth of tree is 10 with 99.68% accuracy. The figures show the true performance predictions (black dots) and the predicted performance (red dots). Often a predicted performance matches one of the nearby observed values; the reason is that the original data point is not contained in the training set and thus the performance model learns from nearby values and uses them as approximation.

It can be seen that RMSE score and MAE are both very good (Table III). This means that a well-fitting model have been found to predict the bandwidth value of a given

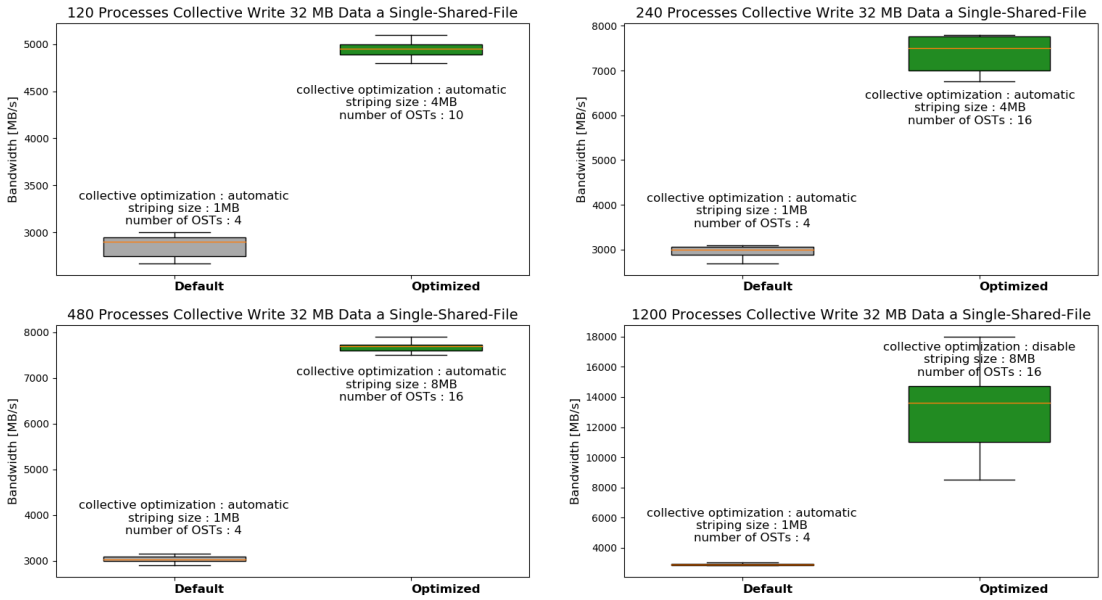

Figure 4: Default Setups on Hazel Hen vs. Auto-tuning.

Table III: Prediction errors in MB/s for training sets under different depth of each tree in the forest.

max_depth	**Prediction errors under different depths**		
	Accuracy	MAE	RMSE
3	82.16 %	495.86	963.36
4	90.52 %	287.92	576.51
5	95.15 %	147.25	325.94
7	98.87 %	46.27	180.32
10	99.68 %	24.85	167.20

configuration set and this model can provide statistical information like an expert. A decision tree with height 4 is shown in Figure 7. Using machine learning and extracting rules from such a tree can provide less time consuming and error-prone rather than the measurements by hand. The configuration set promising the best performance is chosen by the predictive model and support auto-tuning modules.

Parameter set for each configuration is stored in an attribute matrix. Therefore, the number of attributes will affect the size of the matrix, also the processing time. Attributes that can better give results are called high-level attributes and more important than the others in terms of the performance of the predictive model. Instead of using all attributes, the use of some high-level attributes not only reduces the number of transactions, but also increases performance of regression model. Feature selection algorithms can be implemented to achieve identification of effective attributes in future work. Moreover, there can be a further improvement to the metric by doing some preprocessing before fitting the data.

VI. Conclusion and Future Work

In this study, a machine learning supported collective I/O auto-tuning solution for engineering applications is presented that can be understood by engineers or scientists with little knowledge of parallel I/O without any post-processing step. The auto-tuning solution is implemented upon the MPI-IO library to be compatible with MPI based engineering applications, and be portable to different HPC platforms as well.

This study shows several challenges faced when optimizing collective I/O with collective buffering and use random forest regression to develop a predictor model in auto-tuning solution to estimate I/O performance based on the results of the previous runs. After evaluating the predictor model under various conditions, it is determined as an accurate indicator of the expected collective I/O performance. The configuration set promising the best performance is chosen by the predictive model and support auto-tuning modules. The success of the approach varies between 50-130% at scale by finding out the optimal ones that are better than the default configurations. Incorporating findings of predictive model with the auto-tuning system has the potential to further reduce the training time and size of the training set.

The parameters discussed in this paper are system dependent, but new parameters can be easily integrated to auto-

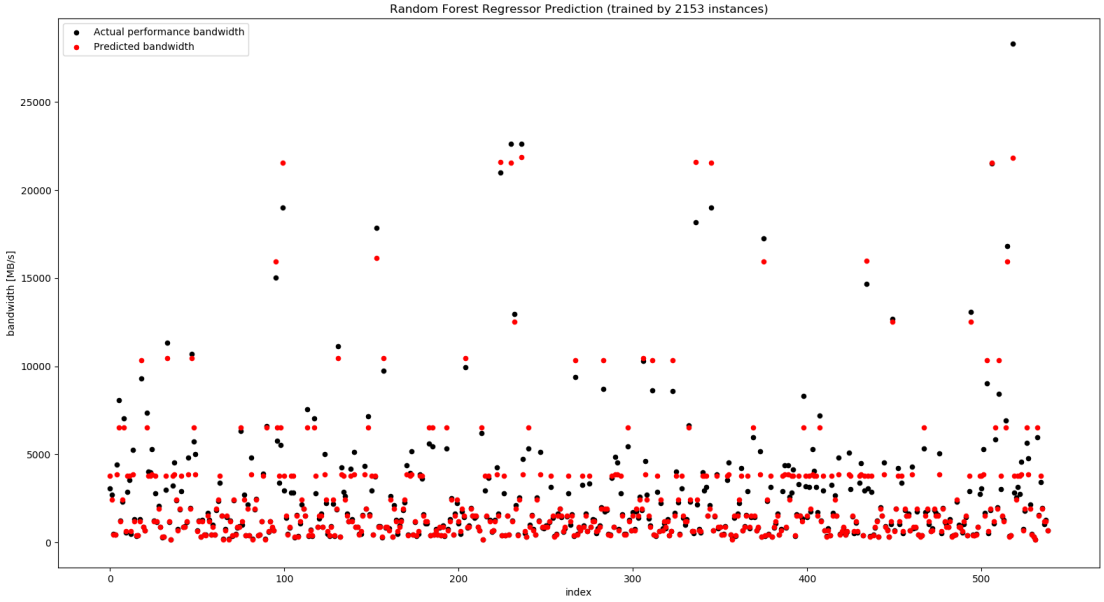

Figure 5: Random forest regression performance model with max_depth = 4, Accuracy: 90.52 %.

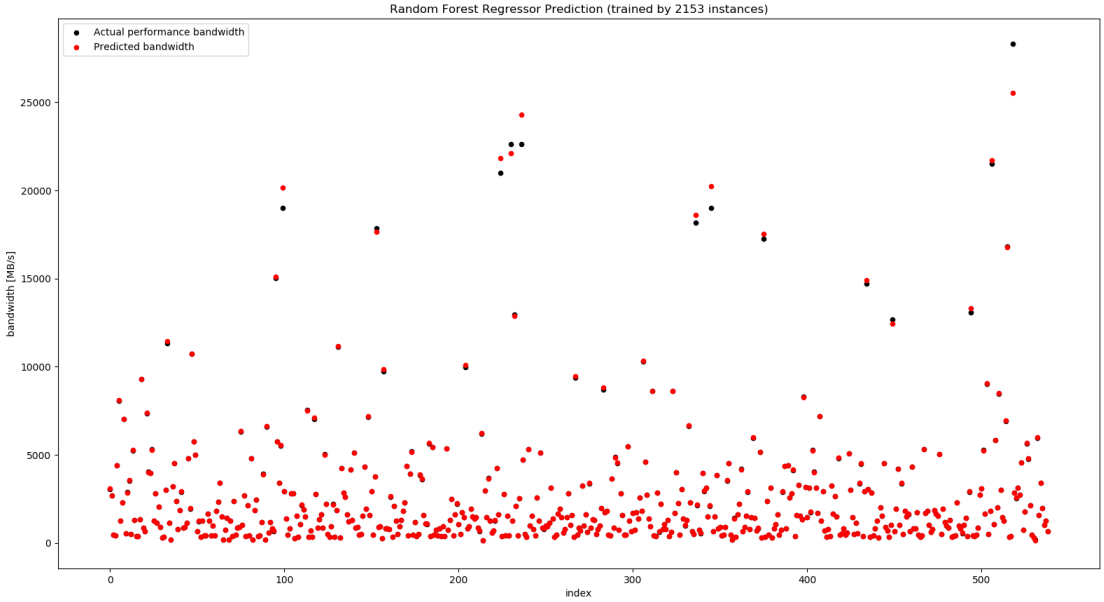

Figure 6: Random forest regression performance model with max_depth = 10, Accuracy: 99.68 %.

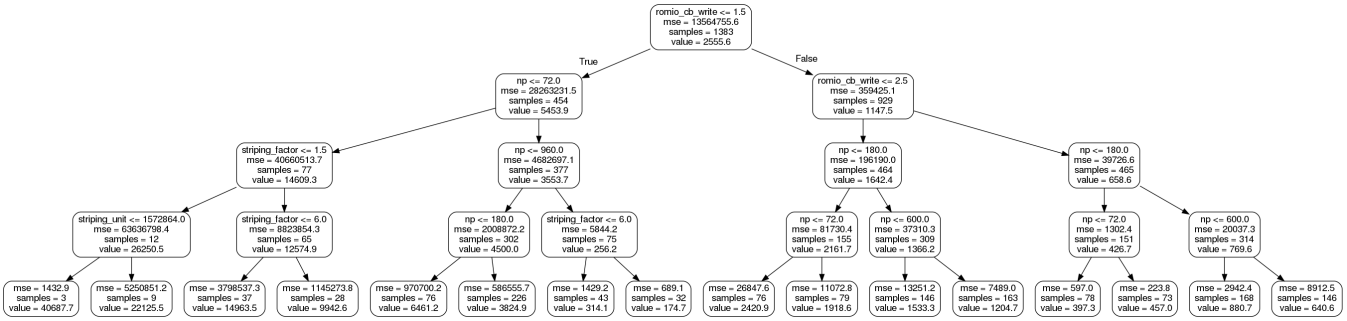

Figure 7: A subtree demonstration of random forest regression model. The dominant label is assigned to the leaf nodes.

tuning configuration files. Future efforts will further explore more accurate representations and characteristics of the configuration parameters and machine learning techniques. As future work, the auto-tuning solution will be tested on engineering applications in different professional areas to show the usability.

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 Framework Programme research and innovation programme under the Marie Sklodowska-Curie agreement No 721865. I wish to acknowledge the support of Dr. José Gracia, Christoph Niethammer and Xuan Wang.

REFERENCES

[1] B. Behzad, S. Byna, S. M. Wild, M. Prabhat, and M. Snir, "Improving parallel I/O autotuning with performance modeling", in 23rd International Symposium on High-Performance Parallel and DistributedComputing. ACM, 2014, pp. 253–256.

[2] F. Isaila, P. Balaprakash, S. M. Wild, D. Kimpe, R. Latham, R. Ross,and P. Hovland, "Collective I/O tuning using analytical and machine learning models", in International Conference on Cluster Computing. IEEE, 2015, pp. 128–137.

[3] H. Wang, (2017). A light weighted semi-automatically I/O-tuning solution for engineering applications (Doctoral dissertation). Retrieved from OPUS - Publication Server of the University of Stuttgart, http://dx.doi.org/10.18419/opus-9763.

[4] S. Madireddy, P. Balaprakash, P.H. Carns, R. Latham, R.B. Ross, S. Snyder, S.M. Wild, "Modeling I/O Performance Variability Using Conditional Variational Autoencoders", 2018 IEEE International Conference on Cluster Computing (CLUSTER), 109-113, 2018.

[5] R. Thakur, W. Gropp, E. Lusk, "Data sieving and collective I/O in ROMIO", in: FRONTIERS 1999: Proceedings of the The 7th Symposium on the Frontiers of Massively Parallel Computation, p. 182. IEEE Computer Society, Washington,DC, 1999.

[6] A. Knupfer, H.Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M.S. Muller, W.E. Nagel, "The vampir performance analysis tool-set", In: M. Resch, R. Keller, V. Himmler, B. Krammer, A. Schulz, (eds.) Tools for High Performance Computing, Proceedings of the 2nd International Workshop on Parallel Tools,pp. 139–155. Springer, Heidelberg, 2008.

[7] Argonne National Laboratory: Darshan. http://www.mcs.anl.gov/project/darshan-hpc-io-characterization-tool

[8] J. Kunkel, M. Zimmer, E. Betke, "Predicting Performance of Non-contiguous I/O with Machine Learning", in: Kunkel J., Ludwig T. (eds) High Performance Computing. ISC High Performance 2015. Lecture Notes in Computer Science, vol 9137. Springer, Cham.

[9] J. M. Kunkel, M. Zimmer, N. Hübbe, A. Aguilera, H. Mickler, X. Wang, A. Chut, T. Bönisch, J. Lüttgau, R. Michel, and J. Weging, "The SIOX Architecture — Coupling Automatic Monitoring and Optimization of Parallel I/O", in Proceedings of the 29th International Conference on Supercomputing - Volume 8488, ser. ISC 2014, Leipzig, Germany: Springer-Verlag New York, Inc., 2014, pp. 245–260, ISBN: 978-3-319-07517-4. DOI: 10.1007/978-3-319-07518-1_16. [Online]. Available: http://dx.doi.org/10.1007/978-3-319- 07518-1_16.

[10] B. Behzad, S. Byna, Prabhat, and M. Snir, "Pattern-driven Parallel I/O Tuning", in Proceedings of the 10th Parallel Data Storage Workshop, ser. PDSW '15, Austin, Texas: ACM, 2015, pp. 43–48, ISBN: 978-1-4503-4008-3. DOI: 10.1145/2834976 . 2834977. [Online]. Available: http : / / doi . acm . org / 10 . 1145/2834976.2834977.

[11] S., Madireddy, "Machine Learning Based Parallel I/O Predictive Modeling: A Case Study on Lustre File Systems", in: ISC'18: International Conference on High Performance Computing, 2018.

[12] D. Devendran, S. Byna, B. Dong, B.V. Straalen, H. Johansen, N. Keen, N.F., Samatova, "Collective I / O Optimizations for Adaptive Mesh Refinement Data Writes on Lustre File System", 2016.

[13] S. Benedict, R. S. Rejitha, P. Gschwandtner, R. Prodan and T. Fahringer, "Energy Prediction of OpenMP Applications Using Random Forest Modeling Approach," 2015 IEEE International Parallel and Distributed Processing Symposium Workshop, Hyderabad, 2015, pp. 1251-1260.