Model Driven Auto Tuning Parallel IO using Active Learning

Divyansh Singhvi
divyanshsinghvi@gmail.com

Megha Agarwal
agarwalmegha1308@gmail.com

I. Abstract

Parallel IO has been used in wide array of scientific applications. However current stack of parallel IO contains various tunable parameters. While these parameters increases the speed of IO many-fold, the application developer usually resides to default values. Therefore, we propose an auto-tuning algorithm for the parameters inspired from automated hyper parameter tuning in Machine learning models. We have used Bayesian Optimization that finds the value of parameters that minimizes an objective function. Our model builds a surrogate function (probability model) based on past evaluation results of the objective.[1] It requires fewer efforts with results suggesting this approach can achieve better performance on the test set while requiring fewer iterations. The time taken is also less compared to random search or Genetic Algorithms as it makes an informed search .It requires no manual effort beyond the initial set-up. We have automated tuning of Lustre parameters and MPI-IO hints.
To validate our auto-tuning system, we applied it to two I/O benchmarks (S3D-IO and BT-IO) that reproduce the I/O activity of the scientific applications (S3D and BT). The auto tuning model improved over the default runtimes in all cases.

II. Introduction

Parallel IO is an important component of High Performance computing. In most of the HPC applications, IO affects the considerable performance of codes. The parallel file system and middleware layers provide a collection of parameters for improvement of performance. Unfortunately, obtaining the best set of values for these parameters is difficult due to the complex interaction between hardware, Application and various intermediate software layers or IO stack. Tuning all these parameters require an in-depth understanding of IO stack. But as an application developer, one would aim at optimizing algorithms of code and not IO parameters to improve efficiency. They should obtain reasonable performance across all systems. While there have been considerable efforts for application specific tuning on particular systems but ideally there should be a single algorithm for all applications independent of the platform to give the best set of parameters that enhance performance.
For better use of HPC and saving laborious efforts by Developers of HPC, we aim to design a system that gives best set of IO parameters with minimal human intervention and execution time.
In this paper, we have discussed the approach developed by us to achieve the required goal. Our work is different from existing methods in the field and gives a new scope for improvement. We are able to implement a tuning system that hides the complexity of IO interactions at various layers from the developer and is independent of the application.
The rest of this paper is organized as follows. Related work has been discussed in Section III. An overview of our algorithm is presented in Section IV, and the technical details and computation methodology are explained in Section V. Section VI containes overview of 3 models that we have proposed. An experimental evaluation of the system along with results is presented in Section VIII and VII, followed by our concluding remarks in Section IX.

III. Related Work

Auto Tuning is required in various fields of computer science. While training machine learning models the hyperparameters, it may not be always possible to tune them without an expert. Various auto tuning methods are used such as gradient descent, Bayesian Optimization, or evolutionary algorithms to conduct a guided search for the best hyperparameters. [1] Our work is inspired by similar approaches where instead of tuning hyperparameters of ml model we tune MPI-IO hints and Lustre parameters.

Genetic Algorithm are set of search algorithms used to address problem of finding best values from given parameter set. There are many works oriented to this approach. [2] GA chooses a set of values for parameters and assign them a fitness level via a fitness function. Higher fitness represents better solution. The whole sample space constitutes the population. At every stage members of population undergoes reproduction, crossover and mutation. The probability of selection of particular value depends on its fitness. Mutation ensures that solution does not converge on local minima. There have been many developments in this field lately.
B.Behzad et al. used a heuristic-based search with a Genetic Algorithm in order to tune I/O performance. The heuristic-based search has a long run time and could not be applied on a different configuration than the trained configuration. Howison et al. studied manually tuning HDF5 applications on Lustre file systems [3]. To manually tune IO parameters is a cumbersome process and requires experienced parallel developers while the application developer usally leave the application on default parameters. McLay et al. study tuning parallel I/O on a specific

system and suggest that maximizing stripe count has a significant impact on performance [4] There have been several efforts in predicting parallel I/O performance, where I/O experts use the system and application knowledge to develop predictive models. We have used ml models in Model 2 and 3 to predict performance. Lee and Katz [5] developed analytical models of disk arrays to approximate their utilization, response time, and throughput. Song et al [6] proposed an analytical model to predict the cost of read operations for accessing data. Data organization in different layouts on the file system is used to predict cost. Herbein et al. [7] use a statistical model, called surrogate-based modeling to predict the performance of the I/O operations of HPC applications. Barker et al. [8] used analytical performance models for two applications to predict their performance for new storage system deployment. But due to complex layout of filesystem developing analytical models is often time consuming and insufficient to obtain expected predictive accuracy. So several researchers started using ML models for modelling IO performance. Kunkel et al. [9] used decision trees to build an I/O performance model and used it optimize ROMIO data sieving. Behzad et al. [10] developed a semi-empirical approach to model the performance of MPI-IO operations. Isaila et al. [11] combined analytical and machine learning approaches for modeling the performance of ROMIO collectives. Xie et al. [12] developed microbenchmarks to characterize storage system write performance, identified the most important input parameters, and developed machine learning-based models. Similar to our work, the modeling is used to reduce the time to search for the optimal parameters. Here we have used a new approach that uses active learning to predict best set of performance and ML modelling in case of Model 2 and Model 3 to reduce the run time for obtaining best set of parameters.

IV. Model

Our aim is to find values of parameters that gives best performance in terms of IO bandwidth for a given IO size of the particular application and underlying file system. Mathematically we can represent our problem as

$$x* = \arg\max_{x \in \mathcal{X}} f(x)$$

where f(x) represents the objective function to minimize which in our case denotes the run-time of application.(we can incorporate the IO bandwidth by taking its inverse and multiplying with IO size), x* denotes the set of values for parameters which minimizes function and and x can take any value in space of parameters denoted by $\mathcal{X}$.

The key problems auto tuning models face are :

- finding best set of parameters in optimal time
- feeding those parameters without modifications in code

We have tackled second problem by inserting the parameter values obtained from first either through environment variables or executing their corresponding commands before running the code. For finding the best set of parameters one can think of tuning values manually. But as run-time of applications is generally large this approach is not feasible. A naive strategy to get best set of parameters is to run application on specific system across whole parameter space with all possible values and obtain the best set. But the task is tedious and knowing that search space is continuous the computation overhead will be large. We can reduce the search space analytically and improve efficiency by doing random search but again random search is unreliable and may not result in best set or close to best set every time. Also in some cases parameters selected from random search results in performance degradation. So we choose the method of Bayesian Optimization.

A. Bayesian Optimization

Bayesian Optimization chooses set of parameters as informed decision based on performance in previous runs whereas random search chooses values independent of results in previous trial runs. The concept is: *limit expensive evaluations of the objective function by choosing the next input values based on those that have done well in the past.* [1] When the number of parameters to be tuned increases, random search performs miserably. Bayesian Optimization keeps track of previous evaluations by computing

$$P(score|parameters)$$

In the literature, this model is named as "surrogate" for objective function and is denoted by P(y/x). The Bayesian method optimizes this surrogate rather than objective function. Fig 1 denotes the steps taken by Bayesian Optimization.

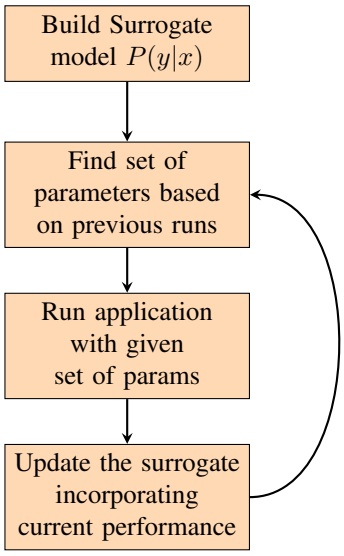


Fig. 1: Steps Taken By Bayesian Optimization

At each step decision is taken by the model in informed manner. The time spent in taking this decision for choosing values of parameters for next run is insignificant compared to computational resource invested in running application on multiple cores. In the next section we will see implementation details.

V. Implementation details

Sequential Model Bayesian Optimization Algorithms are formalization of Bayesian Optimization. As the name suggests sequential denotes running trials one after another each time finding parameters by applying bayesian reasoning and updating surrogate. There are five aspects to optimization as discussed in [1]. We have used *hyperopt* library to implement the bayesian optimization model. There are other python libraries also available for same.

A. Domain

Domain of values of parameters to be tuned. Domain in our case is represented by Lustre parameters and MPI-IO hints.

```
space = {
    'romio_ds_read'
    'romio_ds_write'
    'romio_cb_read'
    'romio_cb_read'
    'romio_cb_write'
    'cb_buffer_size'
    'setstripe-size'
    'setstripe-count'
}
```

Search space of each of these is stated as range in case of continuous space and set of values in case of discrete space. On the basis of intuition one can associate the probability function with search space associating higher probability to region where optimal parameters are most likely to be available.

B. Objective function

It takes in values of parameters as input and outputs the score or f(x) that we want to minimize. The objective function runs the application code and hence is the most expensive part. The aim of entire model is to minimize the calls to this function and try to converge in minimum possible iterations.

C. "Surrogate" model of objective function

As discussed above surrogate model is probabilistic model built using previous evaluations. There are various functions for modelling surrogate like Gaussian Processes and Random Forest. We have used Tree-structured Parzen Estimator (TPE) for our purpose. TPE constructs the model by using bayes rule. Instead of directly using $p(y|x)$ it uses

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

, where $p(x|y)$ is probability of set of values of parameters given score.

D. Selection function

It denotes the criteria by which next set of values of parameters are chosen from the surrogate function. We have used Expected Improvement criteria.

$$EI_{y*}(x) = \int_{-\inf}^{y*} (y^* - y)p(y|x)dx$$

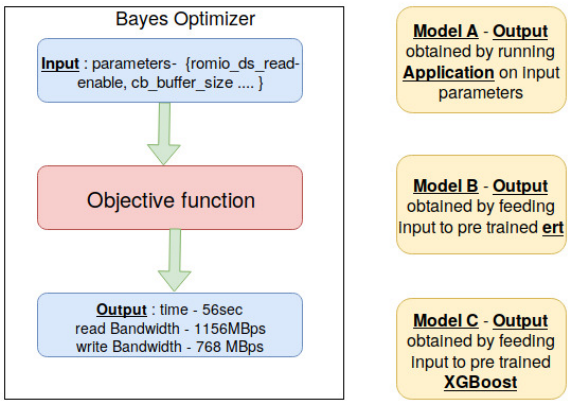


Fig. 2

where y* is a threshold value of the objective function, y is the actual value of the objective function using parameter set x. The objective of model is to maximize Expected Improvement wrt x.

E. History

Each time the algorithm finds a new set of candidate values of parameters, it evaluates them with the actual objective function and records the result in a pair (score, value of parameters). These records form the history and are used in subsequent runs.

VI. Three Approaches

As auto-tuning is time consuming task we have implemented three models with trade-off in time vs efficiency. The differences in three models is in objective function. Objective function takes as input the set of parameters and returns the run-time and IO bandwidth as output. The details of each model and how it is automated is discussed in this section.

A. Model A

This model runs the whole application in objective function and returns the run-time and IO bandwidthon the fly which are further used by Selection function. This model provides good enough results in 20 Iterations. More number of iterations ensures better parameters. The time taken by this model is dependent on number of runs of Application code. In this model we find optimal parameters for given input values (example : size of grid in S3D-IO) named as a *input configuration*.

B. Model B/C

In many scenarios, developers are using Application on their system from long time and have experimented with different parameter values for various input configurations. These runs are usually logged and data collected from logs can be used for auto tuning. Also instead of fine tuning each configuration one may require a model which gives parameters - not optimal but better than default performance - in constant time. As we noticed computing objective function is bottleneck of above

pipeline. So we propose a model that utilizes this data to significantly reduces the run-time of tuner generating good set of parameters for given input configuration.

The flow of model is as follows: In *Model B* extremely randomized tree(ert) is trained on logged data as suggested in [13]. Now in the objective function, instead of running whole application code we load this pre trained model and use it to predict output values i.e. IO bandwidth(in our case we predicted only write bandwidth but we think that this could be generalize to predict IO bandwidth by duplicating the model where the first model will predict the write bandwidth only while the second will predict read bandwidth only. And then in the objective function these two models could be called to predict the IO Bandwidth. for given input configuration. Thus the run-time of objective function is reduced to constant time and hence training time of Auto tuner is improved significantly.

But there is a limitation in ert. If the difference in values of two consecutive iteration is not large enough, ert predicts same runtime. To overcome this we used regression based model-XGBoost instead of ert. This model is named as *Model C*

VII. Experimental Evaluation

A. Application I/O Kernels

1) S3D-IO: This software benchmarks the performance of PnetCDF method implementing the I/O kernel of S3D combustion simulation code [14]. [15]. It writes and reads two four-dimensional variables and two three-dimensional variables. While the three dimensions are user input the fourth dimension is constant for the two variables. In our tests we ran it for several configurations with each dimension ranging from 100 to 400 with processors ranging from 2-16 nodes with each node running 8 processes.

2) BT-IO: This software benchmarks the performance of PnetCDF and MPI-IO methods for the I/O pattern used by the NASA's NAS Parallel Benchmarks (NPB) suite (http://www.nas.nasa.gov/publications/npb.html). The evaluation method is strong scaling. BTIO presents a block-tridiagonal partitioning pattern on a three-dimensional array across a square number of MPI processes [16].

VIII. Results

A. Model1

First we ran our model 1 on the two benchmarks for various configurations. We obtained the optimum read bandwidth and write bandwith by running 20 iterations of the model. Fig 4a and Fig 4b corresponds to the comparision of read and write bandwidth of various configurations when run using default parameters and the optimum parameters obtained from the model. The model was pretty fast but the time to run the benchmark over the predicted parameter values on the fly was a time consuming step. Hence the overall time the model take to predict one optimal set of parameters was equal to time it took to run the model 20 times. The readers should note that the ranges in the plots are different. The default experiment values corresponds to the default parameter setting that a HPC developer would run on if he make no changes to the parallel IO tunable paramaters.

Fig 3a shows how the density of the sample space varies before running the model and after running the model. We could see that our hypothesis in all cases of cb_buffer_size was uniform distribution between a range while the bayesian optimization model finds the better size and tuned the density to predict it. We could also see that our model was able to find that in the benchmark of S3D-IO, romio_cb_read and romio_cb_write were not changed by hint and hence not affecting the result. While figures Fig 3a and Fig 3c clearly show that nature. While in the Fig 3b there was a huge bias given to the hint which we think could reduce in more runs of the benchmark instead of 20 as even the loss graph is more flat in comparison to other plots. We also see that stripe count which is given a prior of loguniform probability distribution flattens somewhere and rises somewhere which shows that even if our initial hypothesis of the sample space is wrong the model will eventually find the optimal parameters.

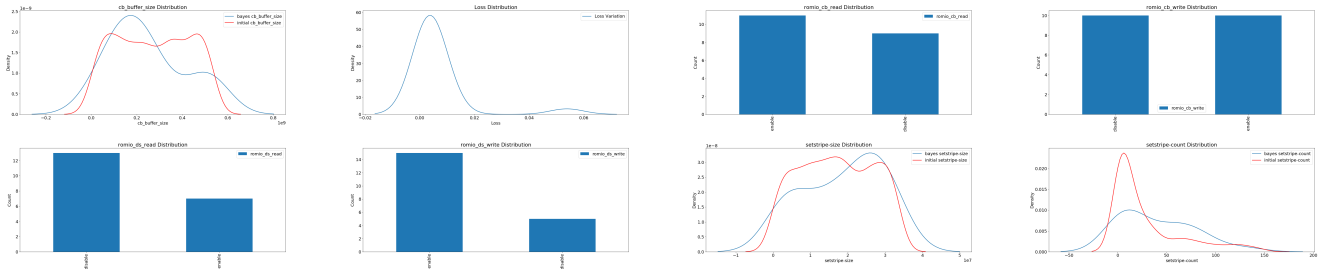
Fig 4 denotes comparison of the default read and write bandwidths vs model 1 optimized read and write bandwidths. We see that as the cores increased the improvement is significant in both the benchmarks. In S3D-IO in all cases we see a significant improvement. For BT-IO the results for the configurations 100-100-100 were variable since the data was too less that the time difference between two iterations for different bandwidths was not too much and in return giving sometimes bad result. This may also happen due to noise in system. While for all other cases we see significant improvement in overall bandwidths.

B. Model2

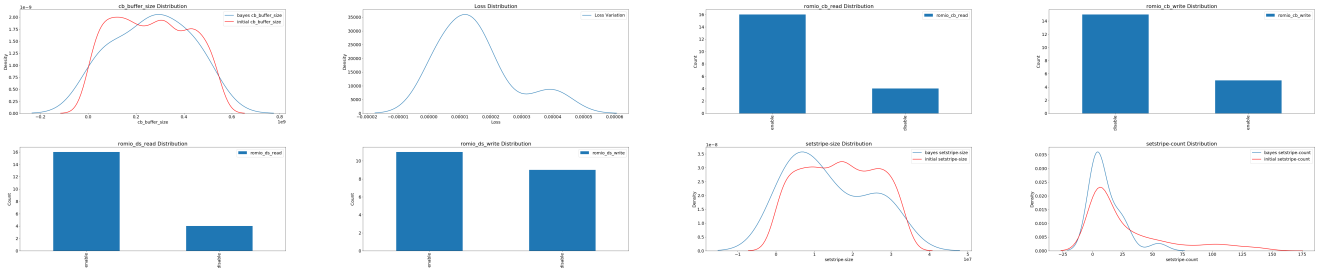
We had 1136 data points for S3D-IO benchmark which include runs of various configurations over various MPI and lusture parameters and each datapoint contains the IO Bandwidth, the tuning parameters and the configuration it was run on. Similarly for BTIO we had 414 data points containing the similar values. These were logs generated while writing our first model. To evaluate the model we take the coefficient of determination (R2) between the observed values and the predicted values. This metric assesses the goodness of the model fit. The maximum value of R2 can be 1, which in turn indicates that the model predictions perfectly fit the data.

We ran the extremely randomized tree model over the data logs of previous runs for the two benchmarks to predict the write Bandwidth. The amount of time for such small dataset was less than a minute and to dump the model on disk storage may take few minutes depending on storage disk speed. We see the results of the benchmark in Table I. We find that it requires a huge amount of data to give a good enough R^2 score which was not suitable for our case. Also it was only able to predict integers and hence we had to multiply the bandwidth with 100 to avoid floating point errors and to get a true accuracy. Hence we modified the machine learning model in Model 3 replacing ETR with XGBoost for regression.

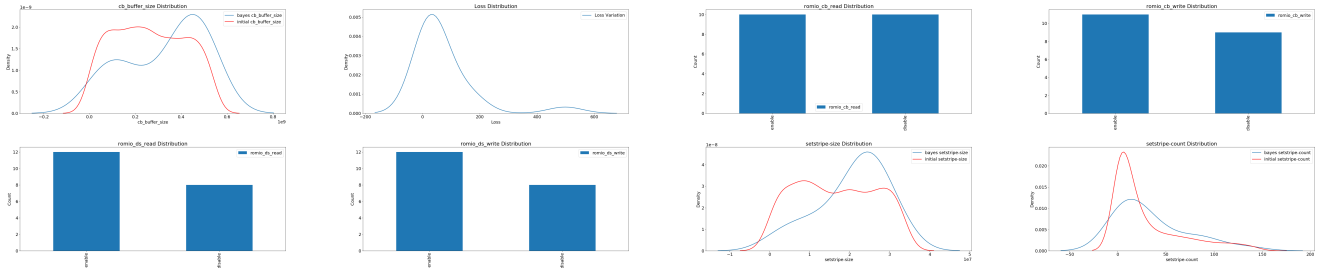
Fig. 3: Visualization of tunable parameters - how the tunable parameter density distribution varies from initial(red) to final density plot(blue) after using bayesian optimization for 20 runs. The histogram represents the probability changed from 0.5 of both possible outcomes to more preference to one with better parameters. Following figures represent : (1,1) : cb_buffer_size, (1,2) : loss, (1,3) : romio_cb_read, (1,4) : romio_cb_write, (2,1) : romio_ds_read, (2,2) : romio_ds_write, (2,3) : lusture stripe size, (2,4) : lusture stripe count



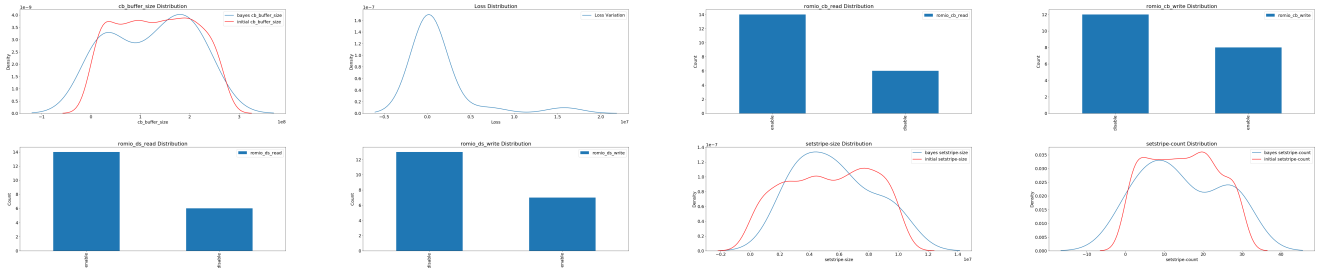
(a) The figure corresponds runs on configuration of S3D-IO ($nx\_g = 200, ny\_g = 400, nz\_g = 400, npx = 4, npy = 4, npz = 8, ppn = 8$)



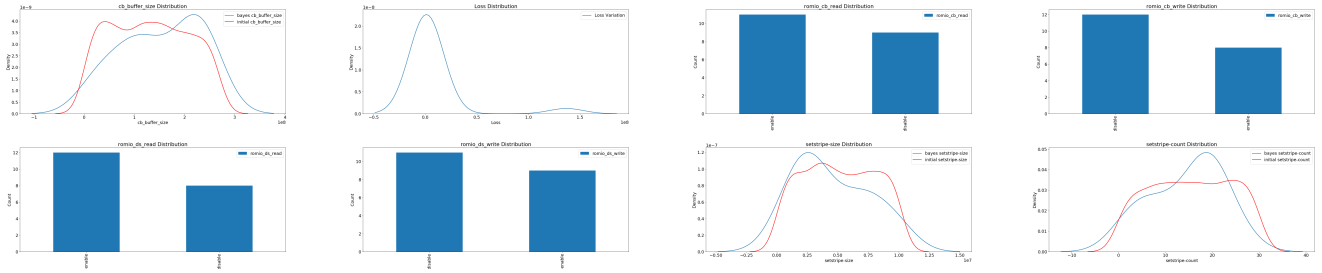
(b) The figure corresponds runs on configuration of S3D-IO ($nx\_g = 100, ny\_g = 100, nz\_g = 200, npx = 4, npy = 4, npz = 4, ppn = 8$)



(c) The figure corresponds runs on configuration of S3D-IO ($nx\_g = 100, ny\_g = 200, nz\_g = 200, npx = 2, npy = 2, npz = 4, ppn = 8$)

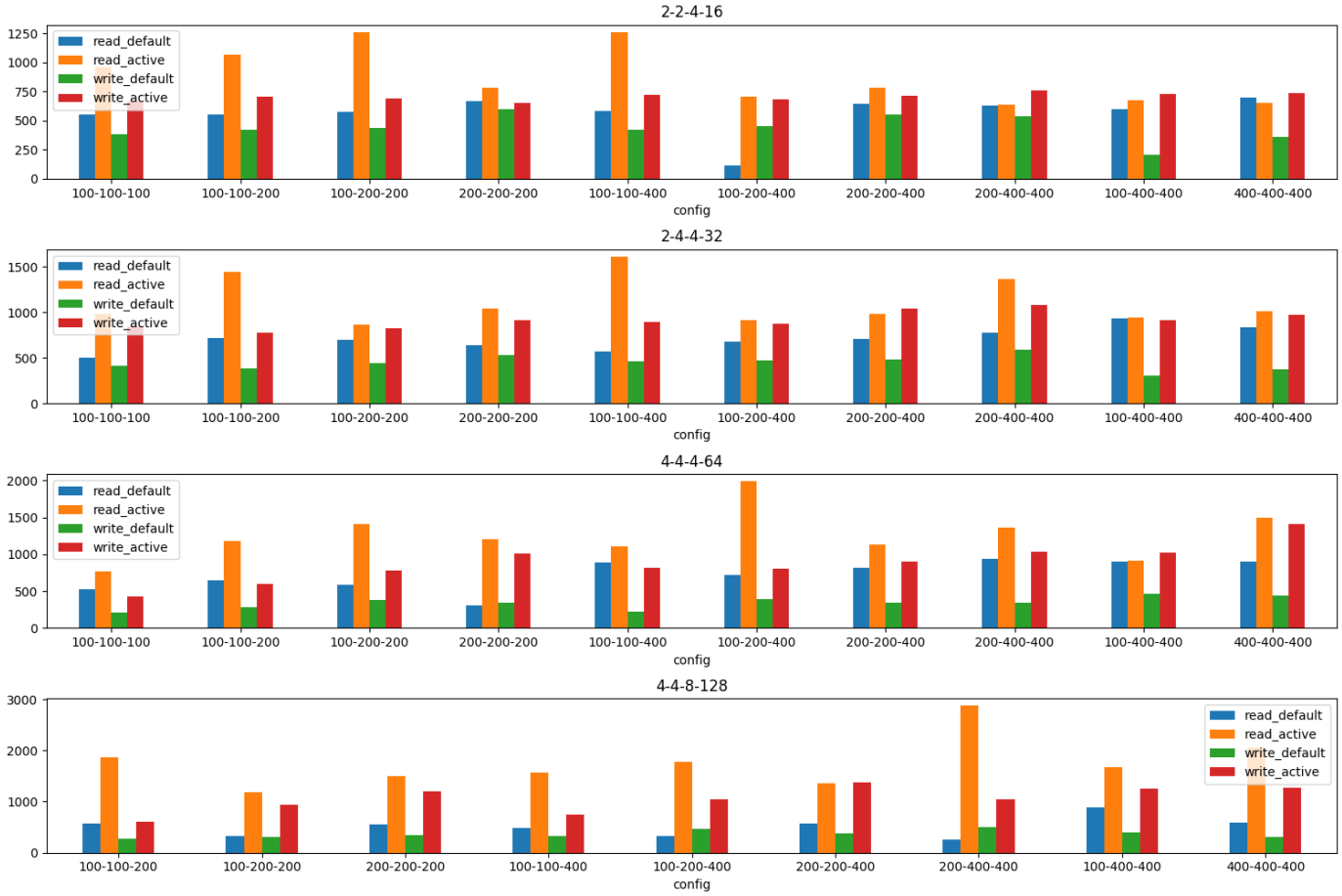


(d) The figure corresponds runs on configuration of BT-IO ($nx\_g = 400, ny\_g = 400, nz\_g = 400, nodes = 8, ppn = 8$)

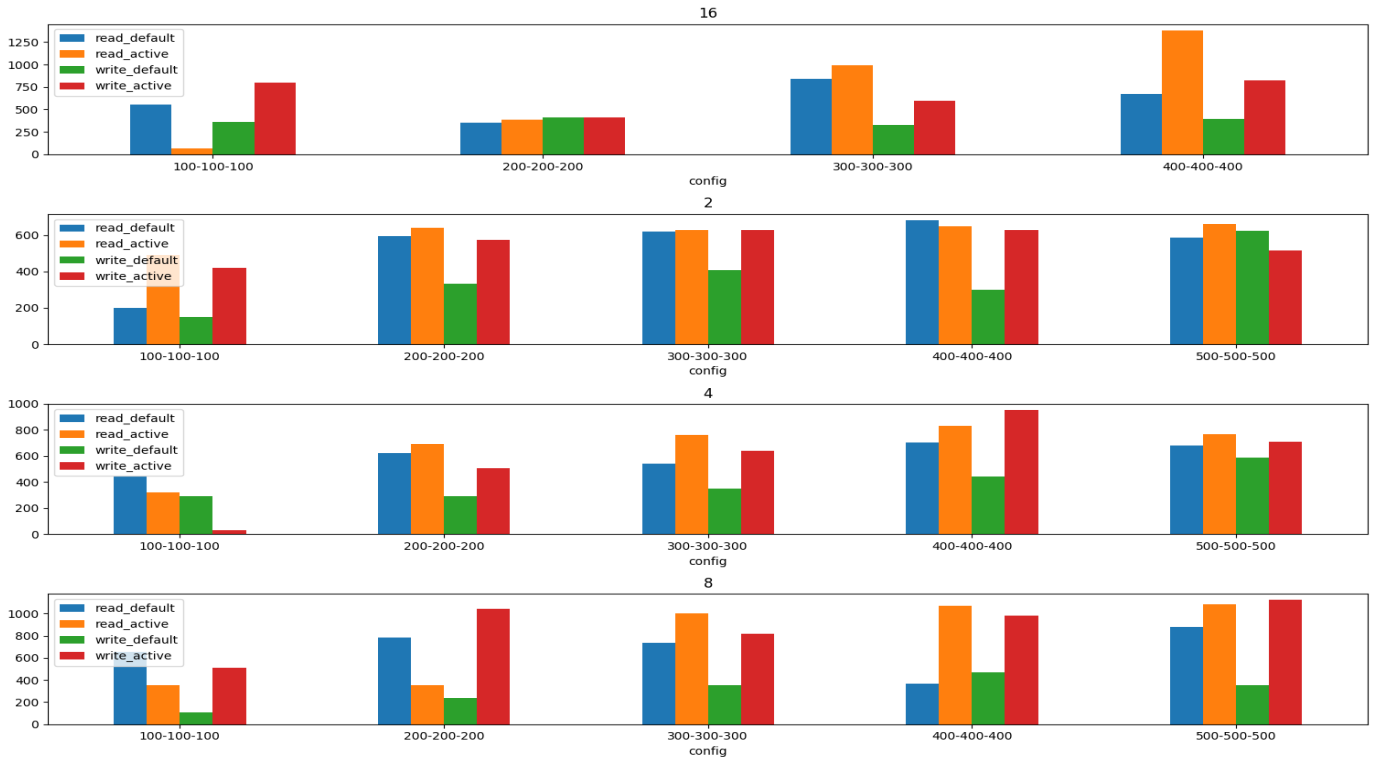


(e) The figure corresponds runs on configuration of BT-IO ($nx\_g = 500, ny\_g = 500, nz\_g = 500, nodes = 8, ppn = 8$)

Fig. 4: Following figures represents comparison of read and write bandwidth of default with the one predicted by Model 1. The IO bandwidth scales are different in each plot.



(a) The graphs represents default vs optimum bandwidth from Model 1 on S3D-IO benchmark for various configurations. Plots title a-b-c-d represents $npx = a, npy = b, npz = c$ and d represents total number of cores. Xtics for each plot of format x-y-z represents $nx\_g = x; ny\_g = y; nz\_g = z$



(b) The graphs represents default vs optimum bandwidth from Model 1 on BT-IO benchmark for various configurations. Plots title a represents nodes; For nodes = 4 and 16 - ppn=4 and nodes = 2 and 8 - ppn =8.Xtics for each plot of format x-y-z represents $grid\_points(1) = x, grid\_points(2) = y, grid\_points(3) = z$

Train/Test	HyperParamterers of etr	R^2 Score
70/30	max_depth=100, n_estimators = 30	0.763
70/30	max_depth=100, n_estimators = 300	0.80
80/20	max_depth=1000, n_estimators = 300	0.80
80/20	max_depth=100, n_estimators = 300	0.73
80/20	max_depth=100, n_estimators = 30	0.753

TABLE I: ERT results trained on S3D-IO

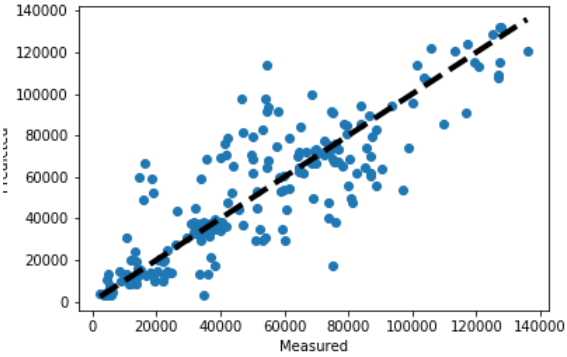


Fig. 5: ERT predicted value vs measuerd value of write bandwidth *100. The ideal graph should be dashed black line and the actual values are denoted by blue dots. The deviations give us good idea of R2 score being lesser in it.

C. Model3

We ran the XGBoost [17] linear regressor model over the data logs of previous runs for the two benchmarks to predict the write Bandwidth. The model took around 1 minute to train. We found the results to be extremely good for even a split of 20/80 train/test and 30/70 train/test with R2 score going to 0.85 for S3D-IO. The R2 score for BT-IO was a bit less which is because of it's lesser number of data points for training.

Train/Test	R^2 Score
5/95	0.627
10/90	0.812
15/85	0.813
20/80	0.846
30/70	0.858
50/50	0.87

TABLE II: XGBoost results trained on S3D-IO

Train/Test	R^2 Score
10/90	0.683
20/80	0.725
30/70	0.767
50/50	0.80
60/40	0.82
70/30	0.827

TABLE III: XGBoost results trained on BT-IO

Since Model 3(XGBoost) performed much better than Model 2(ERT) and required very less data for predictions. We used XGBoost predictions in the objective function to calculate loss and then predict the optimum parameters and ran it for 100 iterations trained on 30/70 split. The results after running over optimized vs default parameters are shown in Fig8. We see that S3D-IO performed very good with this model over various configurations tested for write bandwidth. We ran the benchmark over varied configurations which were not in out trained or test data set and were completely new for the model. While for BTIO also some configurations performed good. Since our original dataset contained configurations like 100-100-100, 200-200-200 to 500-500-500 we never trained it over a model containing multiple of 50 but not 100 and hence our prediction falter heavily over 350-350-350 which is justifiable. Hence, the model was a good performant for various configuration and also reduced the training time and finding the optimal configuration under 2 minutes.

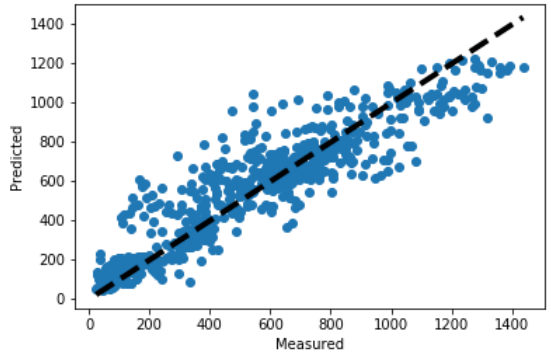


Fig. 6: XGBBoost predicted value vs measuerd value of write bandwidth for S3D-IO for 30/70 split of train/test. The ideal graph should be dashed black line and the actual values are denoted by blue dots. The deviations give us intuition of good R2 score.

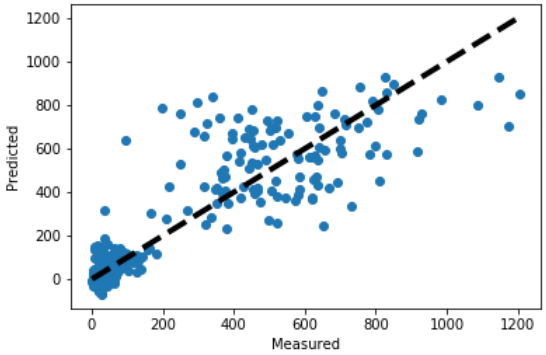
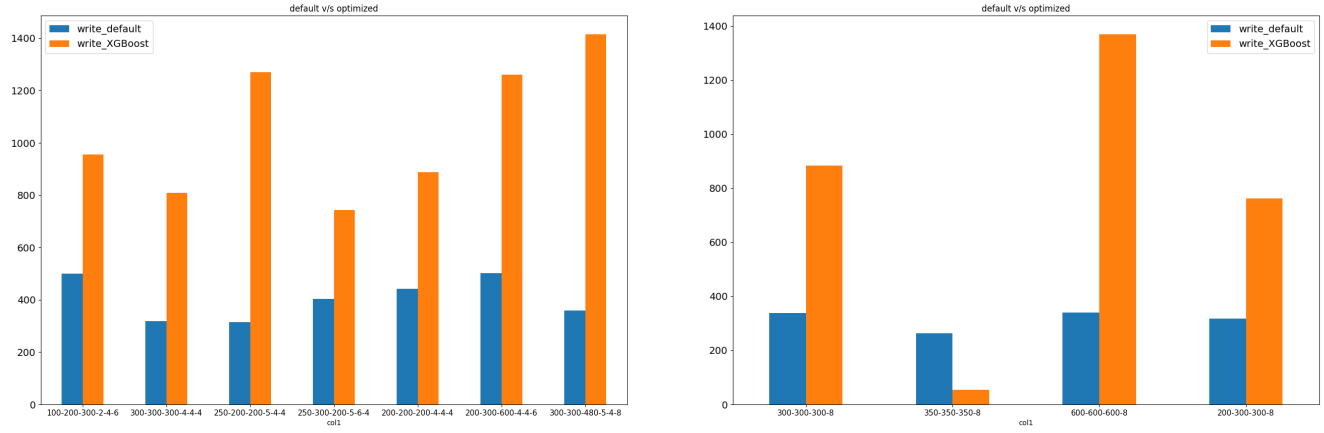


Fig. 7: XGBBoost predicted value vs measuerd value of write bandwidth for BT-IO for 30/70 split of train/test. The ideal graph should be dashed black line and the actual values are denoted by blue dots. The deviations give us intuition of average R2 score. Most possible reason is lesser data and that too being highly localized to bandwidth within 200MB/s



(a) S3D-IO

(b) BTIO

Fig. 8: Following figures represents comparison of write bandwidth of default with the one optimized by Model 3. Blue Bars are default write Bandwidth while the red bars are Model 3's optimized write bandwidth

IX. Conclusion

We developed an autotuning framework with a unique approach that is extremely fast with very good results. We solved the major hurdle of auto tuning being a time consuming process when machine learning models are used while the anlytical models though are fast usually don't give good results. By careful analysis of bottleneck, we developed a novel autotuning framework that is independent of the IO code and could tune various lusture and MPI parameters to give optimum parameters.

X. Future Work

Currently on running model 3, we output the best performant by only running the whole procedure only once. To increase the robustness we may run the benchmark three times and run the model over those three configuration to output the best result parameters. Various other machine learning models or combination of models could be used to give a better accuracy than the current scenario. Also, currently Model 2 and Model 3 predict over write bandwidth only, we think that the same procedure could be followed to predict read bandwidth by training same model with different output(read bandwidth) and their combination could be used to give an overall optimum IO bandwidth as in Model 1.

References

[1] Automated hyperparameters tuning. [Online]. Available: https://www.kaggle.com/willkoehrsen/automated-model-tuning

[2] B. Behzad, H. V. T. Luu, J. Huchette, S. Byna, R. Aydt, Q. Koziol, M. Snir *et al.*, "Taming parallel i/o complexity with auto-tuning," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 68.

[3] M. Howison, "Tuning hdf5 for lustre file systems," 2010.

[4] R. McLay, D. James, S. Liu, J. Cazes, and W. Barth, "A user-friendly approach for tuning parallel file operations," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 229–236.

[5] E. K. Lee and R. H. Katz, "An analytic performance model of disk arrays," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 21, no. 1. ACM, 1993, pp. 98–109.

[6] H. Song, Y. Yin, Y. Chen, and X.-H. Sun, "Cost-intelligent application-specific data layout optimization for parallel file systems," *Cluster computing*, vol. 16, no. 2, pp. 285–298, 2013.

[7] 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," in *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2014, pp. 568–575.

[8] K. J. Barker, K. Davis, and D. J. Kerbyson, "Performance modeling in action: Performance prediction of a cray xt4 system during upgrade," in *2009 IEEE International Symposium on Parallel Distributed Processing*, May 2009, pp. 1–8.

[9] J. Kunkel, M. Zimmer, and E. Betke, "Predicting performance of non-contiguous i/o with machine learning," in *International Conference on High Performance Computing*. Springer, 2015, pp. 257–273.

[10] B. Behzad, S. Byna, S. M. Wild, M. Prabhat, and M. Snir, "Improving parallel i/o autotuning with performance modeling," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*. ACM, 2014, pp. 253–256.

[11] 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 *2015 IEEE International Conference on Cluster Computing*. IEEE, 2015, pp. 128–137.

[12] B. Xie, Y. Huang, J. S. Chase, J. Y. Choi, S. Klasky, J. Lofstead, and S. Oral, "Predicting output performance of a petascale supercomputer," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2017, pp. 181–192.

[13] P. Malakar, P. Balaprakash, V. Vishwanath, V. Morozov, and K. Kumaran, "Benchmarking machine learning methods for performance modeling of scientific applications," in *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 2018, pp. 33–44.

[14] J. H. Chen, A. Choudhary, B. De Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W.-K. Liao, K.-L. Ma, J. Mellor-Crummey, N. Podhorszki *et al.*, "Terascale direct numerical simulations of turbulent combustion using s3d," *Computational Science & Discovery*, vol. 2, no. 1, p. 015001, 2009.

[15] W.-k. Liao and A. Choudhary, "Dynamically adapting file domain partitioning methods for collective i/o based on underlying parallel file system locking protocols," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 3.

[16] W.-k. Liao, "Design and evaluation of mpi file domain partitioning methods under extent-based file locking protocol," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 2, pp. 260–272, 2011.
[17] Xgboost. [Online]. Available: https://xgboost.readthedocs.io/en/latest/