Model Driven Auto Tuning Parallel IO using Active Learning

Prof Preeti Malakar

Submitted By- Divyansh Singhvi
Megha Agarwal

Problem Statement

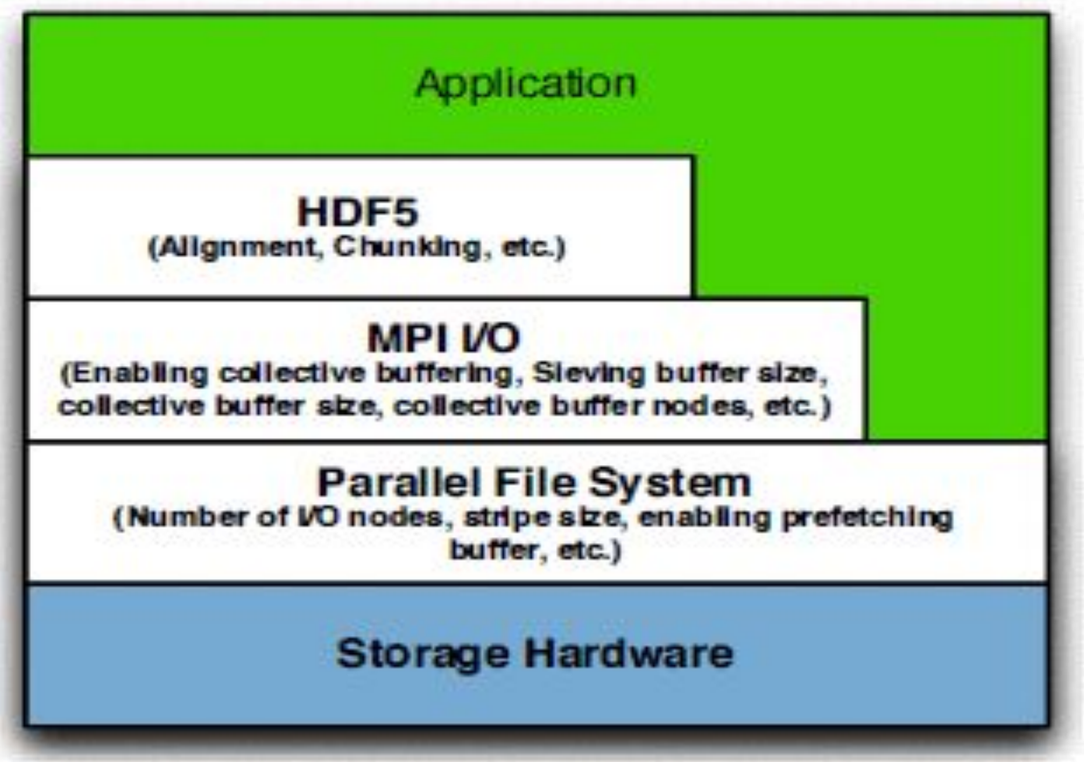
Automating Tuning of parameters for optimizing Parallel IO

Parameters : Lustre (file system), MPI-IO(MPI-IO middleware)

Introduction

- Parallel I/O performance depends on interaction of multiple layers of parallel I/O Stack(high-level I/O libraries, MPI-IO middleware, and file system)
- Each layer has several tunable parameters
- Since the layers are interdependent finding best parameters for a given stack is challenging for a specific application’s I/O pattern
- A normal HPC application developer(expert in their scientific domain) resorts to default parameters resulting in poor performance.

Introduction

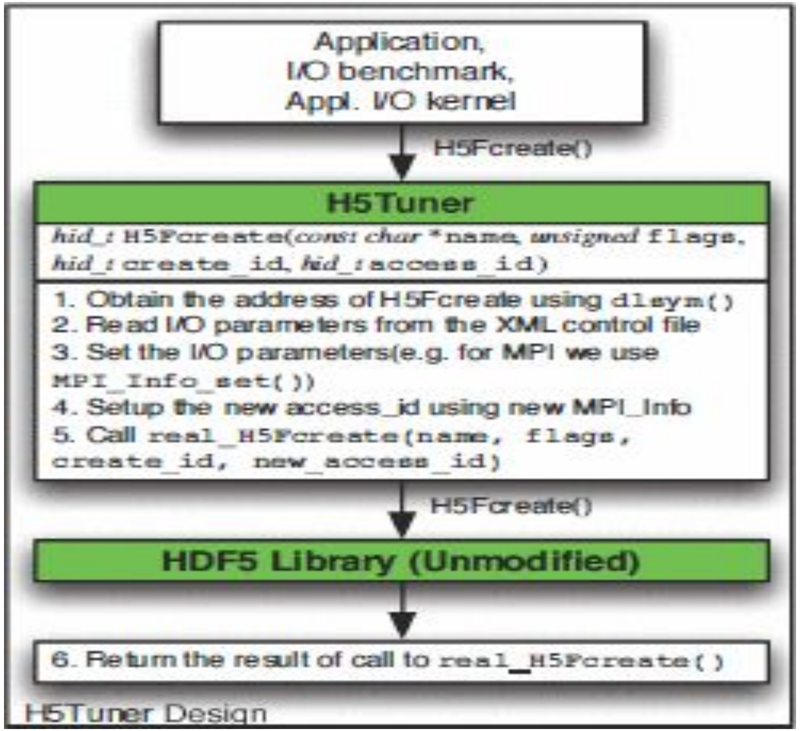


Src - **Taming parallel I/O complexity with auto-tuning**

Parallel Performance dependencies

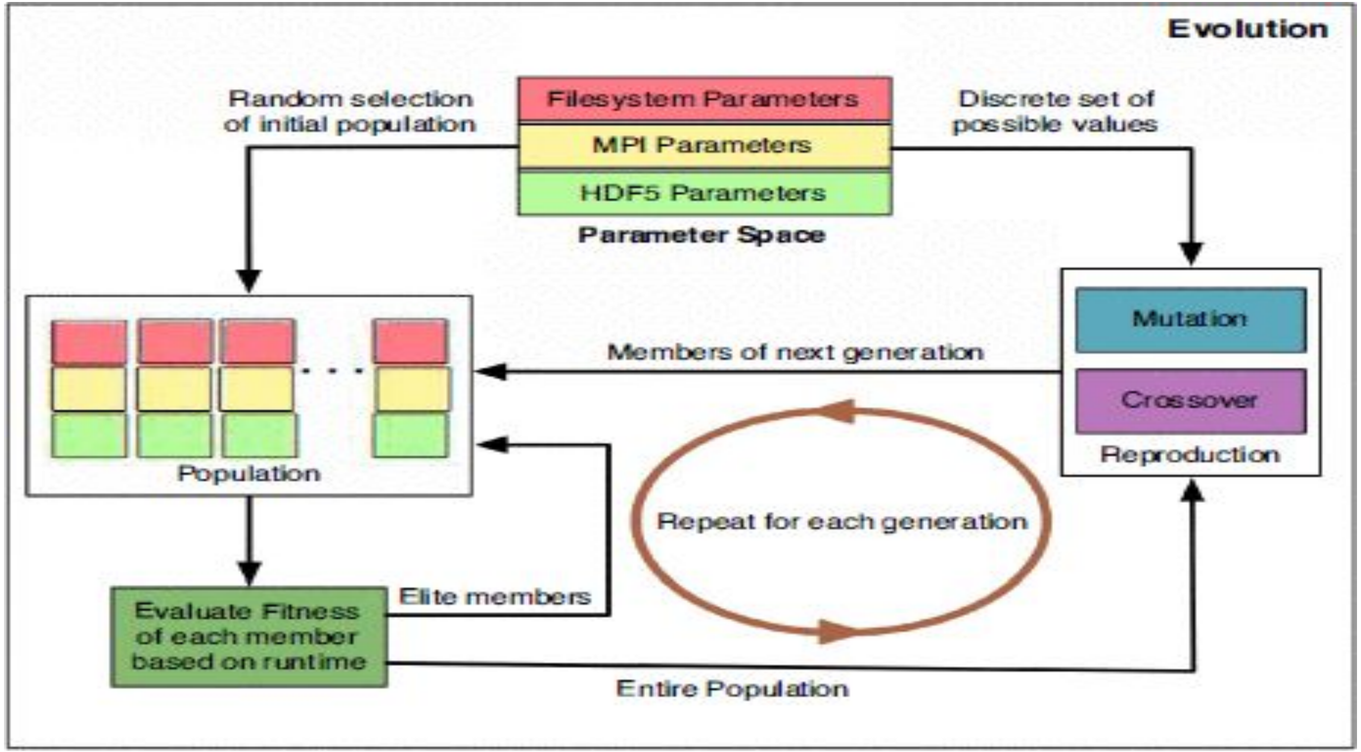
1. The I/O architecture;
2. The speed and amount of I/O hardware (disks, etc.);
3. How well the file system can handle concurrent reads and writes; and
4. How well the file system's caching policies (read-ahead, write-behind) work for the given application.

Previous Work



Src - **Taming parallel I/O complexity with auto-tuning**

Previous Work



Src - **Taming parallel I/O complexity with auto-tuning**

Challenges

- Large number of parameters interdependent on each other.
- Some of the parameters are real valued hence doesn’t allow brute forcing the parameter space to find optimal parameters.
- Even if the parameters are constrained in a range, the number of possible combinations are several which makes the process of auto tuning very time consuming.

Our Approach

Bayesian Optimization

limit expensive evaluations of the objective function by choosing the next input values based on those that have done well in the past

Mathematically we can represent our problem as :

$$x^* = \operatorname{argmax}_{x \in X} f(x)$$

- f(x) represents our objective function to minimize which in our case is run time of application/ inverse of bandwidths of the applications,
- x is the value of parameters
- x* is best value found for each of parameters in sample space X.

Bayesian Optimization Steps (Active learning model)

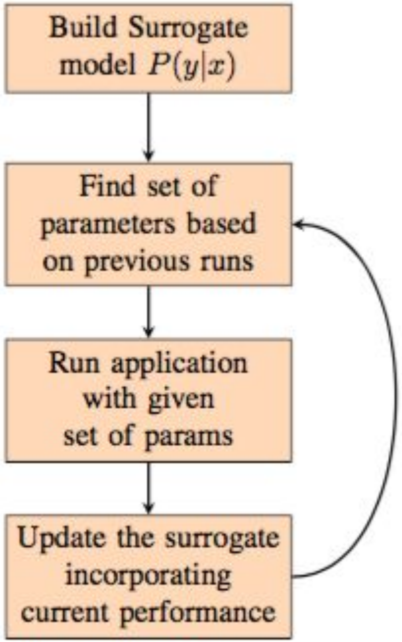


Fig. 1: Steps Taken By Bayesian Optimization

Steps Overview

- “Objective function” is the function that takes input as value of parameters and outputs read and write bandwidth
- “Selection function” chooses which values to choose in next iteration
- Run the active learning for optimum iterations.
- Output the best parameters for the configuration.
- We have come up with 3 Models as we will discuss now to model the objective function

Improvising Loss

- The loss was taken as **cube** of I/O Time.
- This enabled us to reach to good parameters in limited iterations as the difference between loss increased by cube.

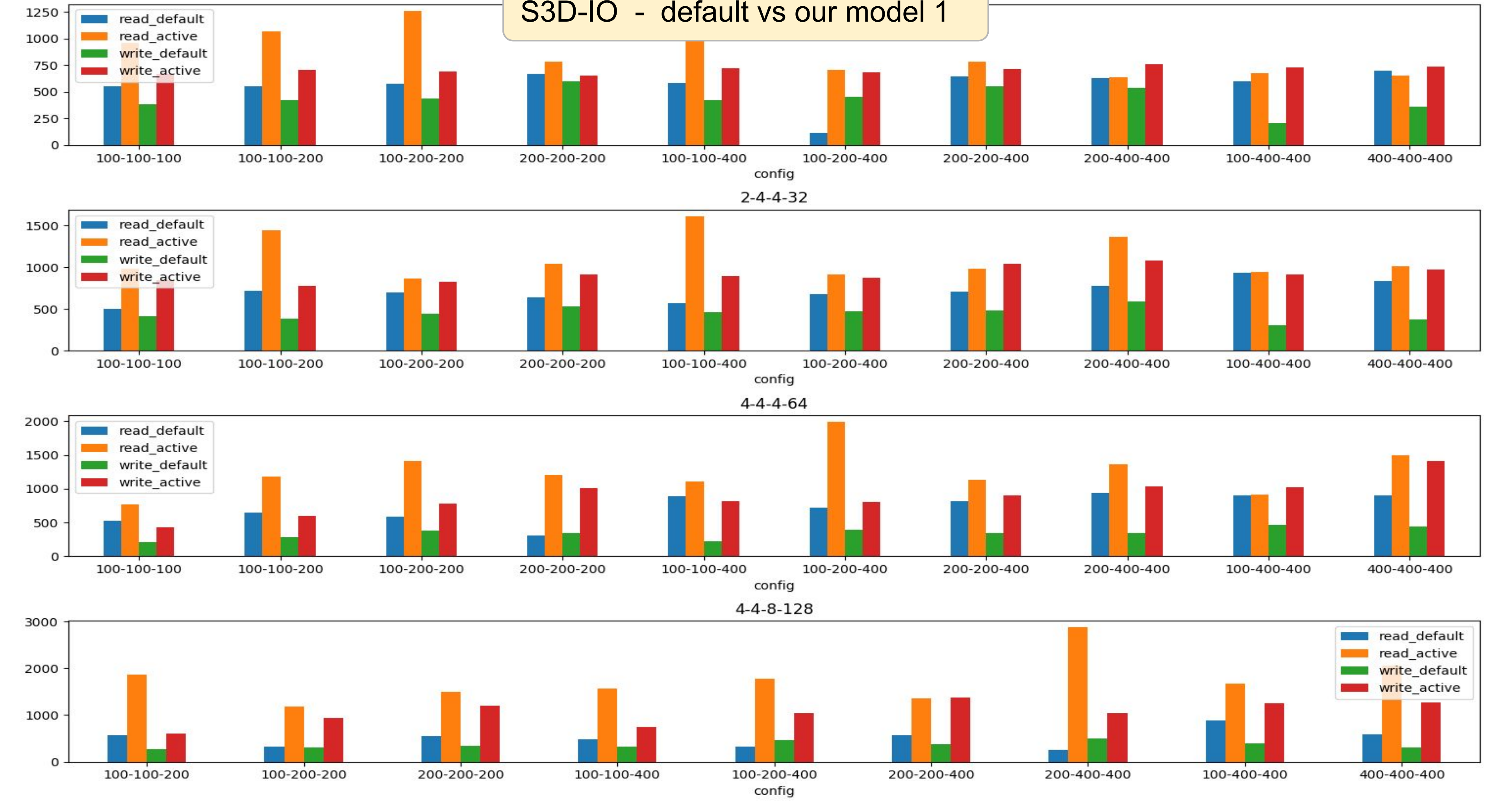
Results of Model 1

Application I/O Kernels for benchmarking

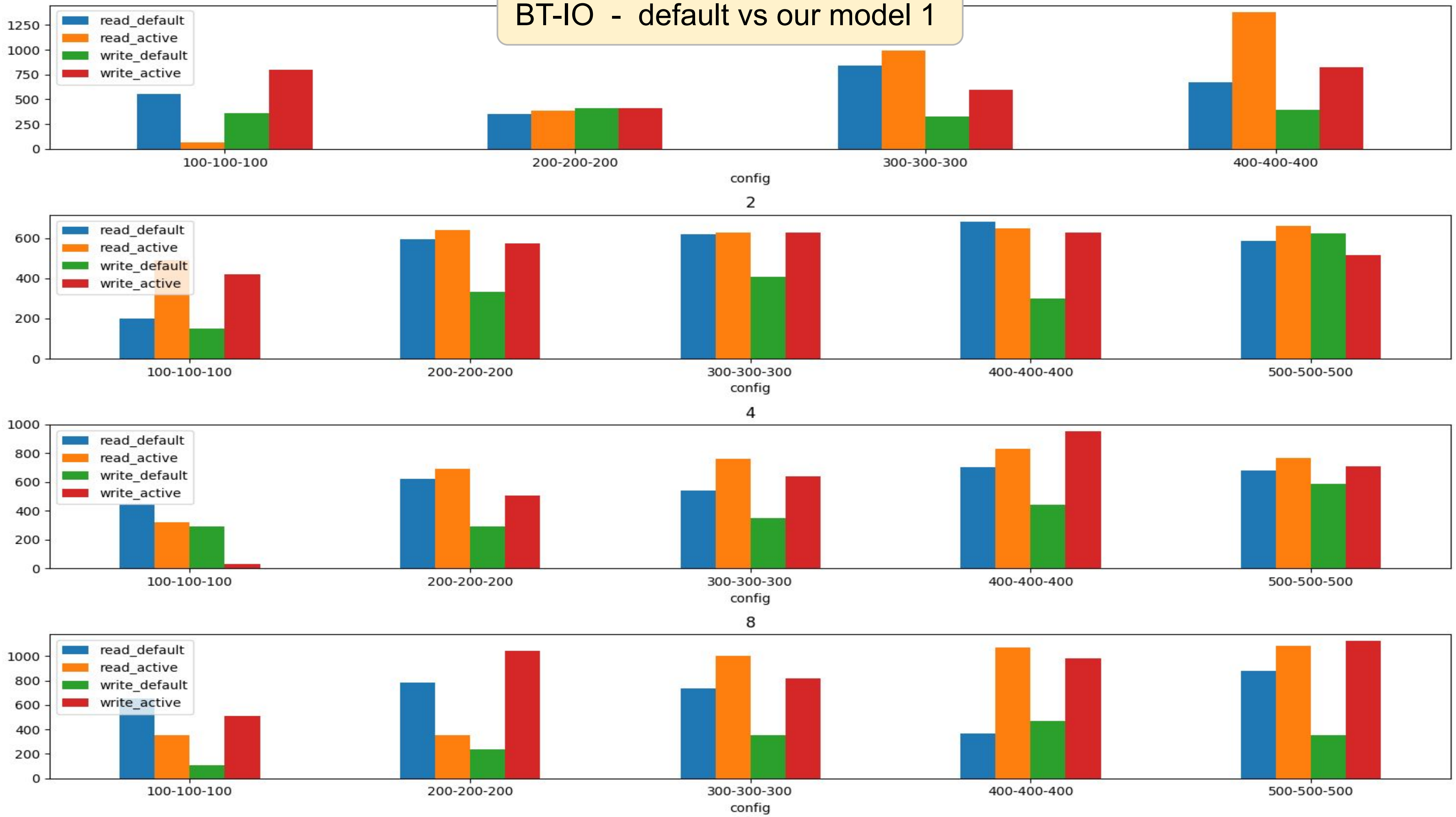
- S3D-IO

- BT-IO

S3D-IO - default vs our model 1



BT-IO - default vs our model 1



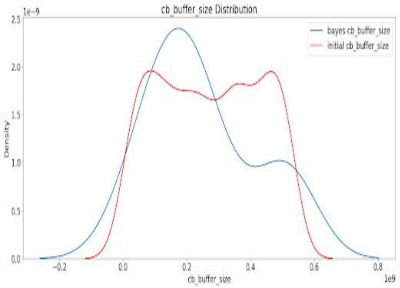
Observation

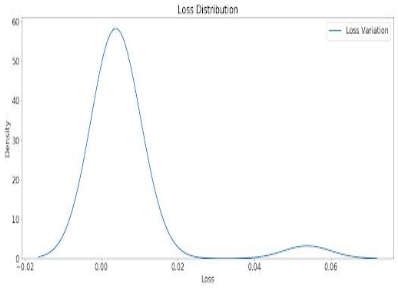
- For large dataset improvement is significant
- Model1 runs for 20 iterations so runtime is also less compared to other machine learning based auto tuning of parallel IO

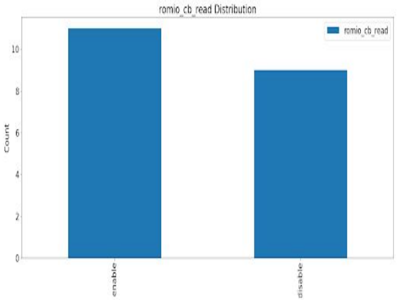
How it is working ?

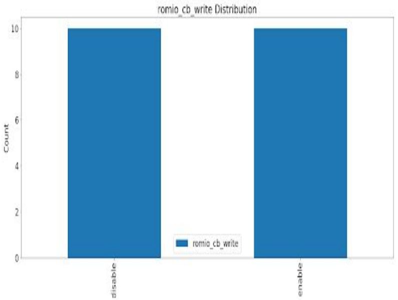
```
space = {
    'romio_ds_read' : hp.choice('romio_ds_read',['enable','disable']),
    'romio_ds_write' : hp.choice('romio_ds_write',['enable','disable']),
    'romio_cb_read' : hp.choice('romio_cb_read',['enable','disable']),
    'romio_cb_write' : hp.choice('romio_cb_write',['enable','disable']),
    'cb_buffer_size' : 1048576*hp.quniform('cb_buffer_size',1,512,1),
    'setstripe-size' : 65536*(hp.quniform('setstripe-size',0,512,1)),
    'setstripe-count' : hp.qloguniform('setstripe-count',0,5,1)
}
```

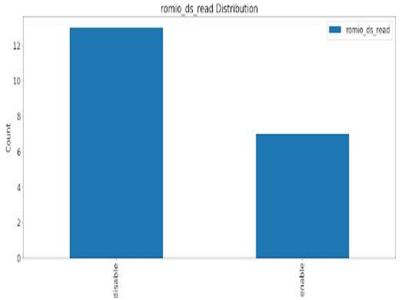
Bias and Learning Plots

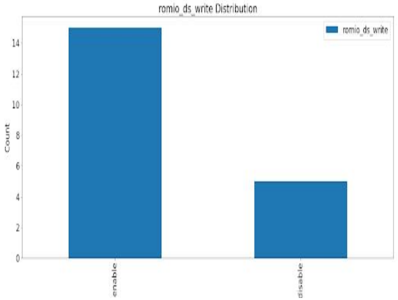


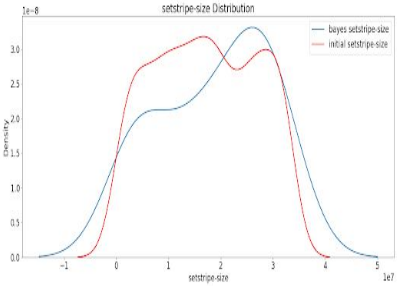


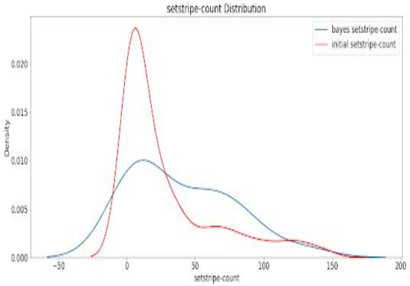




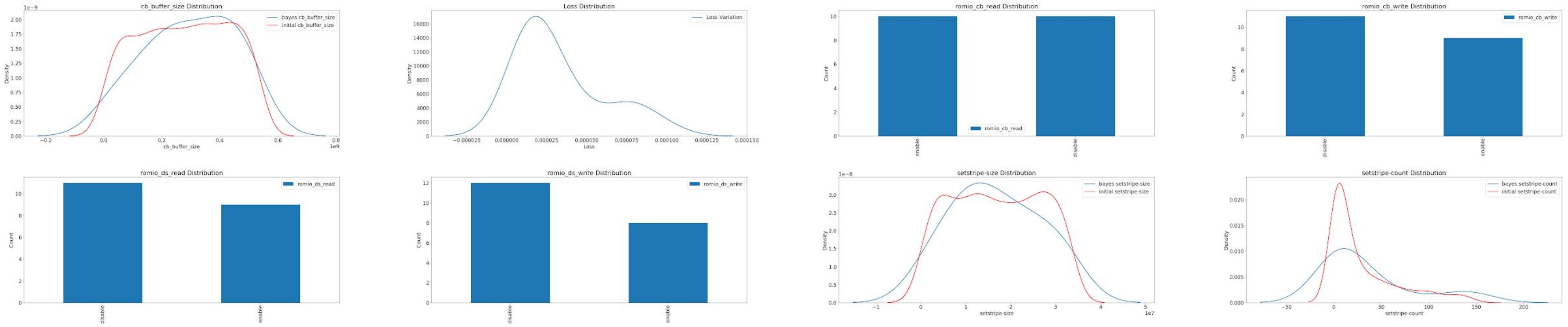




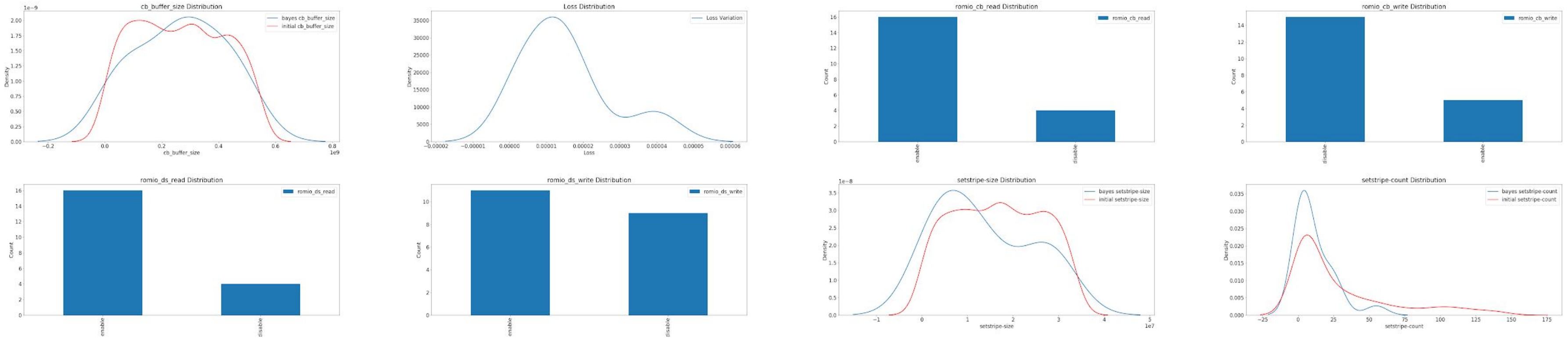




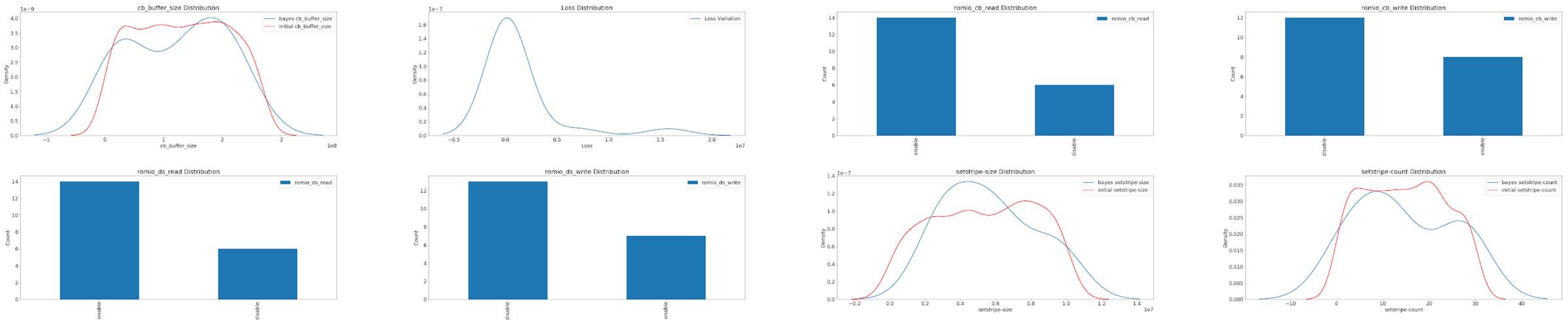
200 400 400 4 4 8 S3DIO



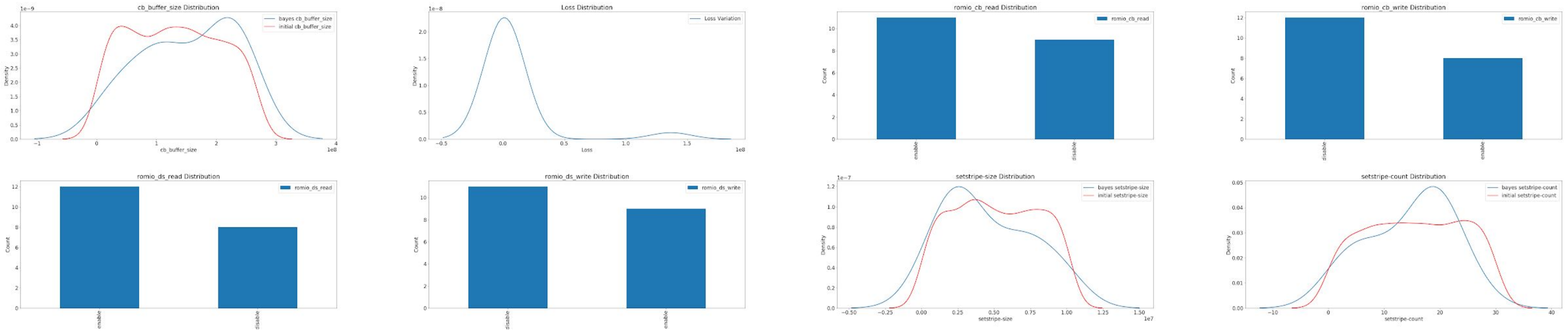
100 200 200 4 4 8 S3DIO



100 100 200 4 4 4 S3DIO



400 400 400 8 BTIO



500 500 500 8 BTIO

Automation

Just change the script to run any new benchmark.

```
def runthebenchmark(hyperparameters):
    os.chdir(project_dir+'active/../')
    storeinfile(hyperparameters)
    out=subprocess.Popen(["python3","read_config_general.py","-n 2","-c200 200 400 2 2 4 1"], shell=False,
stdout=subprocess.PIPE)
    logging.basicConfig(level=logging.DEBUG)
    output=out.stdout.read().decode('utf-8')
    print("output"+output)
    if len(output.split(" ")) > 5:
        values = output.split(" ")
        value = float(float(values[6])*1024)/float(values[5]) + float(float(values[3])*1024)/float(values[2])
        value = float(value)
        print(value)
        return float((value/100)**3),output
    return 0,0;
```

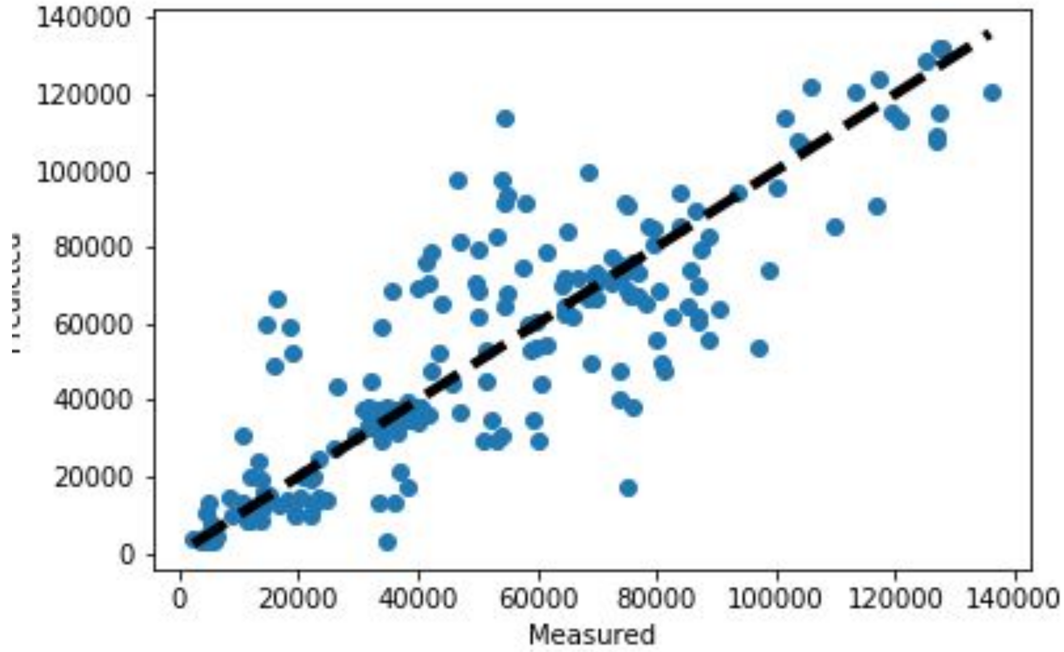
Model 2 Motivation

- If one has logged data for various configurations, run time can be further reduced drastically as compared to Model 1
- Predicts best parameter set for any general *input configuration*

Model 2 details

- An extremely randomized tree trained on logged data and predicts runtime and bandwidth on given set of parameters
- This trained model is used in objective function to predict time and bandwidth for given input configuration

Plot of measured vs Predicted time on test set



ERT on S3D-IO. R2 Score = 0.76 on 70/30 train/test split

Model 2 R2 Scores

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: ETR results trained on S3D-IO

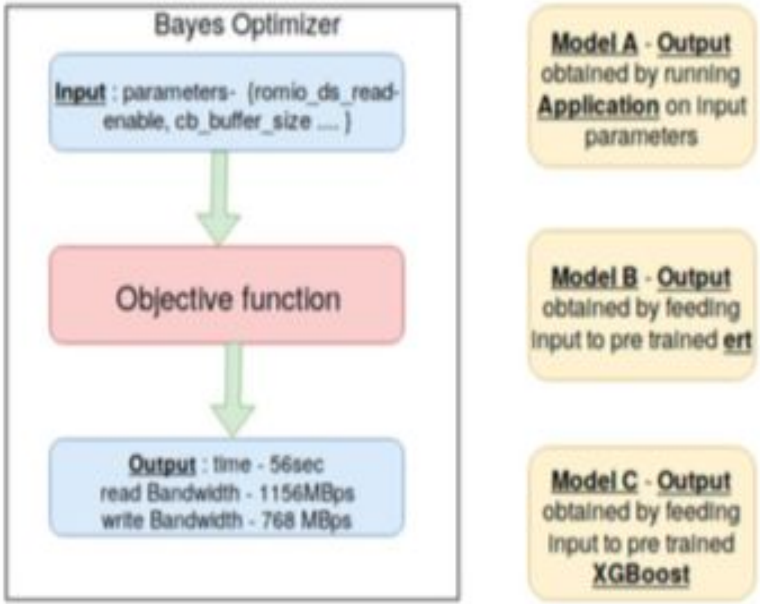
Model 2 drawbacks

- Prediction does not change with small change in values of input configuration
- Thus Bayesian Optimization performs not very good
- So replaced ert with XgBoost regression to overcome these drawbacks

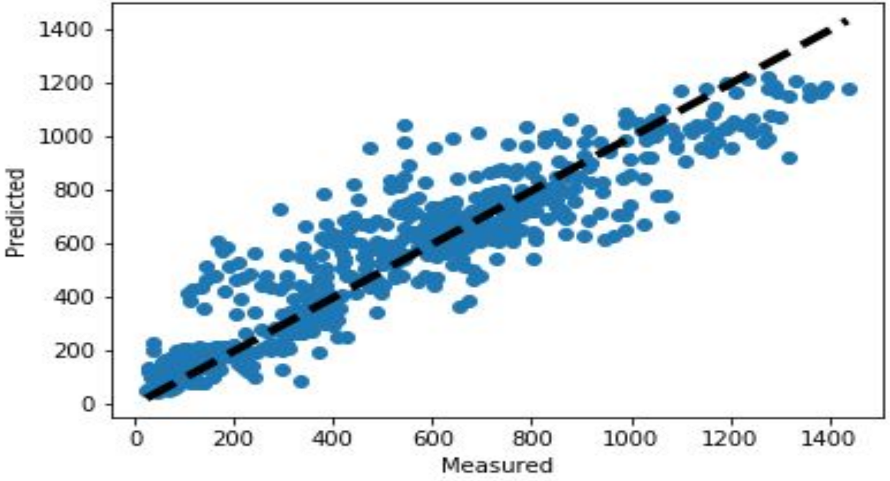
Model 2 drawbacks

- Prediction does not change with small change in values of input configuration
- Thus Bayesian Optimization performs not very good
- So replaced ert with XgBoost regression to overcome these drawbacks

Overview of 3 models



Comparatively better!

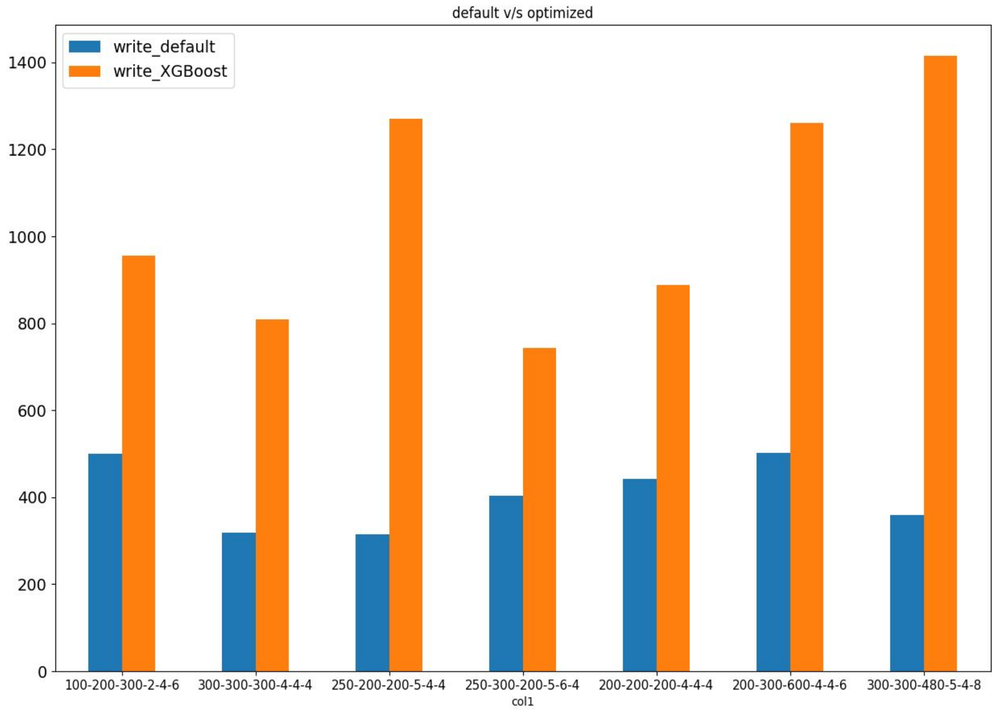


XGBBoost S3D-IO R2Score = 0.85 on 30/70 Train/Test split

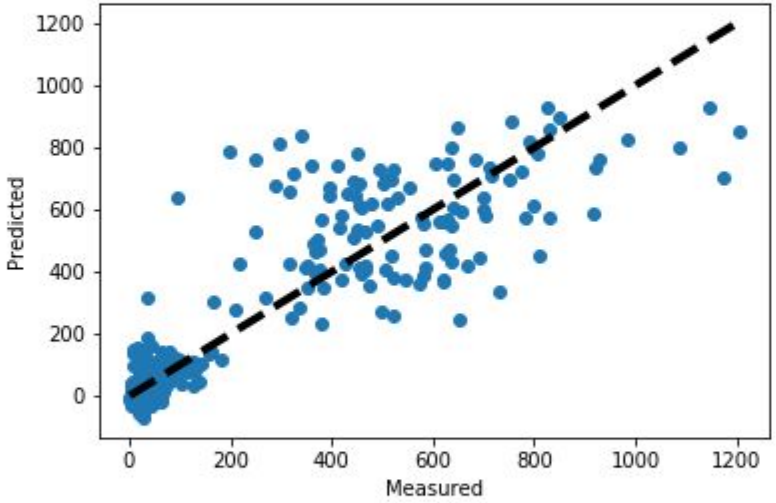
After feeding Model 3 in Objective function

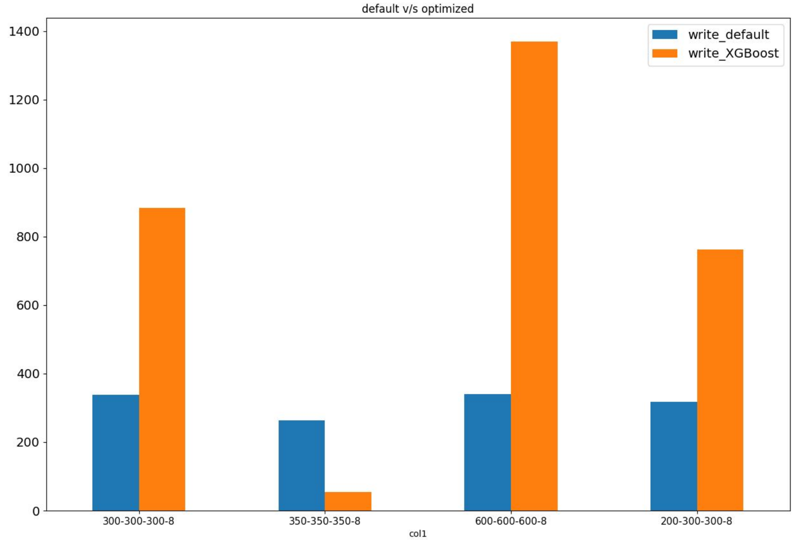
Represents comparison of write bandwidth of default with the one optimized by Model 3.

Blue Bars are default write Bandwidth while the red bars are Model 3's optimized write bandwidth Obtained on S3DIO



Results for BTIO with just 300 data-points





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

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 II: XGBoost results trained on S3D-IO(left) and BTIO(right)

Conclusions

- We developed an autotuning framework with a unique approach that is extremely fast with very good results