Dynamic I/O Model Recommendation System With Machine Learning

*Note: Sub-titles are not captured in Xplore and should not be used

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—In a typical database and file system, using asynchronous I/O is generally a good way to optimize processing efficiency. However, each process and application mainly focus on its own performance, rarely consider the global performance of the system. In this work, in order to balance the I/O performance and resources in a system, we use Machine Learning(ML) techniques to learn I/O model's performance, and set up a client/server system based on gRPC to recommend the more efficient I/O model under different system loads. Which is high performance, scalable, cross-platform and easy to adapt. The experimental result shows that our system has a 15% performance improvement compared to using asynchronous I/O alone and only cost little system resources.**

***Keywords*—asynchronous I/O, synchronous I/O, Machine Learning, performance prediction, gRPC, PMML**

I. Introduction

With revolution of "Big Data" and "Cloud Computing", data center has expended to a large scale and data has grown exponentially. Therefore, data center has to process hundreds of millions of pictures and hundreds of billions of messages each day. How to improve processing efficiency is a hot issue of a data center. Due to the huge speed gap between CPU and I/O device, I/O is one of the bottlenecks of the issue. Using asynchronous I/O is a common way to boost I/O speed. Synchronous and asynchronous I/O are two types of I/O synchronizations as ***figure 1*** shows. In a synchronous I/O job, it starts a thread for I/O operation, and it would hang immediately until the operation is finished. While in an asynchronous I/O job, it would start a thread to send a I/O request to Kernel by calling a function, if the request is accepted successfully, it continues to process other jobs. The kernel signals the calling thread when the operation is finished, then the thread interrupts its current job and processes the data from the I/O operation as soon as possible. However, using asynchronous I/O frequently requires much CPU resources and the I/O's latency would become higher when the I/O's depth is longer.

Identify applicable funding agency here. If none, delete this.

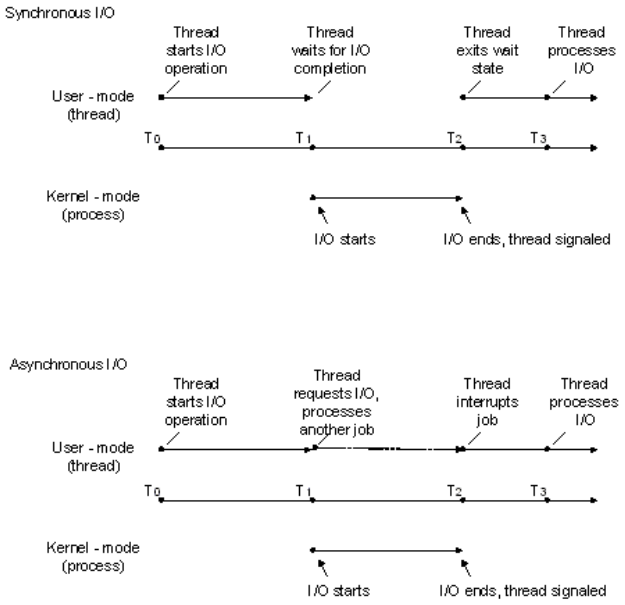


Fig. 1. Asynchronous and Synchronous I/O

To help improve system I/O and processing efficiency, we purpose to combine I/O with Machine Learning(ML). The I/O synchronizations of each I/O's engine have different performance when it is facing different kinds of I/O job and different system load. Collecting these I/O data and training into a Decision Tree model which can predict the most suitable I/O method in current situation. And setting up a Client/Server-based recommendation system using gRPC framework to decide using which kind of I/O for each I/O job.

The main challenge of our system is performance, since each I/O job have to call the server for the better I/O method which cost much time on process communication and prediction. To solve the related problems, we use memory cache and gRPC framework. The data is serialized

to protocol buffers and passed by stream, meanwhile, store the hot data into cache. Therefore, the communication's time will be shortened and reuse the prediction result. After the improvements applied in our system, Running the same task has an efficiency increase of nearly 15%.

In summary, we have made the following contributions in our paper:

1) Train a Decision Tree model which can predict the best I/O method base on the current system load.
2) We build a Client/Server-based, lightweight and scalable system using gRPC to help each I/O job improve efficiency.
3) Run a script to make the system self-optimization daily.

The rest of the paper is organized In Sect. 2 we show the design of recommendation system and propose its implementation in Sect. 3. In Sect. 4 we introduce the experiment configuration and result. The related work is discussed in Sect. 5. Finally, we conclude the paper and talk about the future work in Sect. 6.

II. Design

In this section we talk about the goals of our system first and then show the overview architecture.

A. Goals

According to the motivation of this paper, we should obtain the following goals on system design:

- High-Performance. Our system should calculate the prediction in a short time and return it to the client on time, otherwise it would become a burden.
- Cross-Platform. The system should have the ability to support variable of programming language, because the clients are written in different language.
- Scalable. The system should be easy to add feature to afford different scenarios.
- Self-Optimization. The system should have the ability to optimize itself daily for hot data.

B. Overview

Figure 2 show the architecture of the recommendation system. The system is built based on gRPC to archive the goals of High-Performance, Cross-Platform and Scalable. The redis is deployed to store the hot query data.

A detail procedure of user applications get the predictions from the system is described as follow by referring Fig.2

1) Application prepares the characteristic parameters for the I/O jobs.
2) Remote call predict method through gRPC, waiting for the result or continue to other jobs if using asynchronous call.
3) The method would first check the query if it's in cache, if in, return it to the application first and update it in cache.

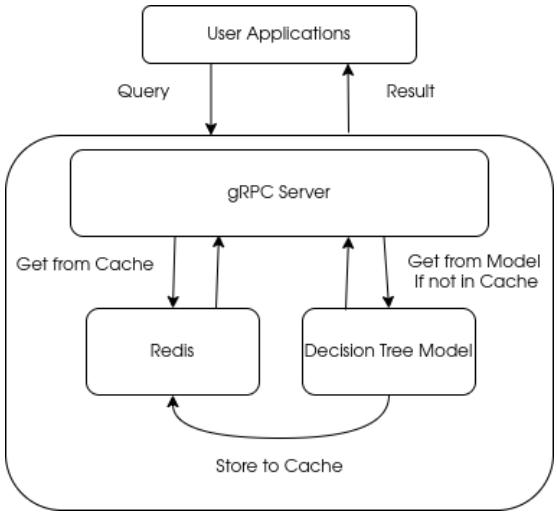


Fig. 2. Architecture

4) If it is not in cache, pass the parameters into Decision Tree model to predict the result. Then, return the result, and store it into cache.
5) Check the hottest queries in a day, collect them as the training data set to update the Decision Tree model.

C. Techniques

- *Fio*
  Fio was written by Jens Axboe <axboe@kernel.dk> to enable flexible testing of the Linux I/O subsystem and schedulers. We use fio to simulate different I/O jobs in actual production environment, which we can chang it I/O engine, I/O depth, file size and so on.
- *Machine Learning*
  Machine Learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention. It is very useful to predict the best I/O method in our system.
- *Decision Tree*
  A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning. It is easy to understand and implement, and can make feasible and effective results for large data sources in a relatively short time. At the same time, it is highly efficient. The decision tree only needs to be constructed once and used repeatedly, and the maximum number of calculations for each prediction

does not exceed the depth of the decision tree. By using Decision Tree can archive the goal of High-Performance.

- *gRPC framework*
  To archive the goals of High-Performance, Cross-Platform and Scalable, using gRPC framework is one of the best choice. gRPC is a modern open source high performance RPC framework that can run in any environment. It can efficiently connect services in and across data centers with pluggable support for load balancing, tracing, health checking and authentication. It is also applicable in last mile of distributed computing to connect devices, mobile applications and browsers to backend services. UDP socket may higher transmission speed, but it is hard apply features and may lose the data.
- *Atomic*
  To speed up the system, we use atomic variables instead of locks, which have better performance in high concurrency scenarios compare to locks. It is very easy to deploy the redis to archive High-Performance.
- *Redis*
  Redis is an in-memory data structure store, which is usually used as cache.
- *PMML*
  pmml pmml pmml pmml pmml pmml pmml pmml pmml pmml pmml pmml

```
                        <DataDictionary>
<DataField name="y"
                    optype="categorical"
                    dataType="integer">
        <Value value="0"/>
        <Value value="1"/>
        <Value value="2"/>
        <Value value="3"/>
        <Value value="4"/>
        <Value value="5"/>
        <Value value="6"/>
        <Value value="7"/>
</DataField>
        <DataField name="x1"
               optype="continuous"
               dataType="float"/>
        <DataField name="x2"
               optype="continuous"
               dataType="float"/>
        <DataField name="x3"
               optype="continuous"
               dataType="float"/>
        <DataField name="x4"
               optype="continuous"
               dataType="float"/>
        <DataField name="x5"
               optype="continuous"
               dataType="float"/>
        <DataField name="x6"
               optype="continuous"
               dataType="float"/>
        <DataField name="x7"
               optype="continuous"
               dataType="float"/>
</DataDictionary>
```

D. Discussion

- How to improve the performance?
  First, gRPC is a High-Performance and lightweight framework especially in intranet environment.
- How to Optimize itself?
  While running the system, a script will run daily to look for the data in redis, to check if there is any hot data that has called for most time. That is one of the most import data for this environment. Then, collect it as the training data set, and re-train the model again. By doing this, the system has been updated without taking it offline, just like hot repair.

III. Implementation

We implement all above improvements on a Linux system, which is similar to the data center environment, and the disk we use is a West Data 256G SSD.

- We used Fio and cpulimit tool to test the I/O performance of different I/O methods in our system, we can change different parameters to simulate single task or multitasking and random read or random write. And we mainly collect 4K random read/write performance of the system, one of the most important indicators is iops(I/O per second). After collecting the data, we preprocess it. We use one-hot encoding to convert the string into a vector form, and select the data with the highest iops as our training data.

```
; -- start job file --
[global] ; global parameters
rw=randread
bs=4k
direct=1
[job1]
filename=./mytest
iodepth=4
ioengine=io_uring
size=1G
numjobs=1
```

- After preprocessing the data, we combine the ioengine and iodepth as the classification label, then import the data into Decision Tree Model. We save the trained model as a pickle file, so we don't need to train it every time we use it, besides, we can deploy it in similar environment.
- After the model is ready, we build a gRPC server and load the model to memory waiting for clients' calls.

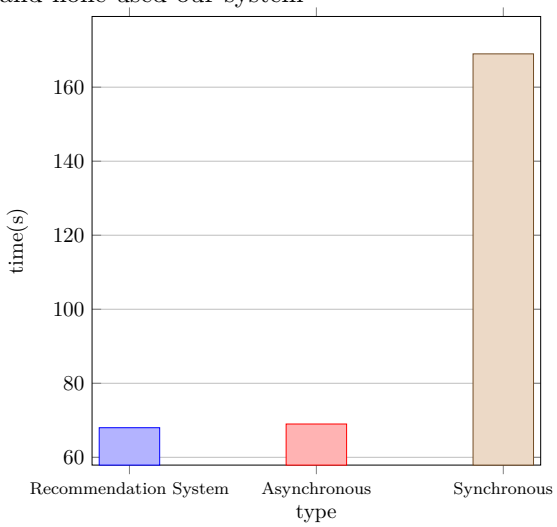
IV. Evaluation

In this section, we evaluate our recommendation system by comparing it performance in different situations with only using synchronous or asynchronous I/O. And we have prepared 20 I/O task for the evaluation.

- We used Fio and cpulimit tool to test the I/O performance of different I/O methods in our system, we can change different parameters to simulate single task or multitasking and random read or random write. And we mainly collect 4K random read/write performance of the system, one of the most important indicators is iops(I/O per second). After collecting the data, we preprocess it. We use one-hot encoding to convert the string into a vector form, and select the data with the highest iops as our training data.

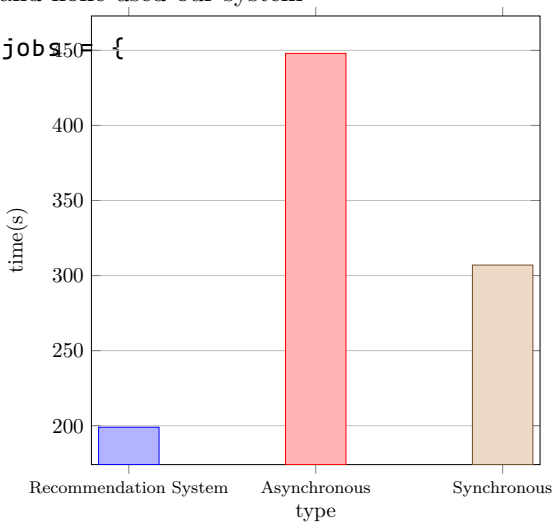
```
const std::vector<std::vector<std::string>> jobs = {
//size(KB)  rw
{"31174", "read"},
{"96871", "write"},
{"94486", "randread"},
{"40694", "randwrite"},
{"46993", "read"},
{"76150", "write"},
{"76967", "randread"},
{"128517", "read"},
{"473399", "write"},
{"338556", "randread"},
{"130019", "randwrite"},
{"334198", "write"},
{"31174", "randread"},
{"96871", "randwrite"},
{"94486", "read"},
{"46993", "randread"},
{"2309", "randwrite"},
{"76150", "read"},
{"76967", "write"},
{"323408", "randread"},
{"128517", "randwrite"}
};
```

- compare single I/O work performance between used and none-used our system



As the figure shows, when the system is idle and there is less competition for I/O, our system has the same performance as all asynchronous I/O, and has a great improvement with all synchronous I/O. This proves that our system will bring I/O performance improvement, and will not generate a lot of additional system overhead. At the same time, it can be seen from this figure that asynchronous I/O has a very large performance improvement compared to synchronous I/O when idle

- compare multi I/O works performance between used and none-used our system



When the system is busy and there is more I/O competition, our system has a greater performance improvement. Compared with all asynchronous I/O, there is a 55% performance improvement, and compared with all synchronous I/O, there is a 35% performance improvement. In addition, the chart also shows that in this case, synchronous I/O also has a great performance improvement compared to asynchronous I/O. And our recommendation system can recommend different I/O paths in real time, can use synchronous I/O when the system is busy, and use asynchronous I/O when the system is relatively idle, combining the respective advantages of these two types of I/O , In order to achieve the effect of improving system performance.

- io-uring performance
- evaluate the cost of gRPC
- evaluate the cost of Decision Tree

V. Related Work

- hot issue in I/O

VI. Conclusion

improvement of our system and future usage scenario

Acknowledgment

References

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove the template text from your paper may result in your paper not being published.