UNIVERSITY OF CALIFORNIA

SANTA CRUZ

Storage Workload Characterization and Performance Prediction for Better I/O Traffic Management

A thesis submitted in partial satisfaction
of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Xiaoyuan Lu

December 2015

The Thesis of Xiaoyuan Lu
is approved:

Professor Ethan L. Miller, Chair

Professor Darrell D. E. Long

Professor Jishen Zhao

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Storage Workload Characterization and Performance Prediction for Better I/O Traffic Management

by

Xiaoyuan Lu

In distributed high-performance computing storage systems, contention is a problem that usually occurs when the volume of client I/O requests exceeds the capability of storage servers; it degrades the whole system's performance. The Storage Traffic Control System (STCS) is a critical component of a large system, which reduces the contention between clients, alleviates congestions in the system, and improves network and storage system efficiency. In a rule-based STCS, the building process for an optimal Storage Traffic Control Rule Set (STCRS) usually requires repetitious and lengthy benchmarking and tweaking cycles. Furthermore, most STCRSs are workload-specific and the established rule sets only matches to a limited subset of the actual workloads a storage system needs to handle. To add adaptiveness to the rule-based STCS, we present a performance prediction approach to help choosing existing STCRS for new workloads. This approach simplifies the traffic management for large-scale systems: instead of redesigning traffic control rules for each new workload the system receives, the proposed method extracts and measures the features of new workload and uses a regression tree pipeline to predict the performance improvement each control rule set bring to the new workload. The rule set that can best improve the performance of a workload will be chosen and applied to the new workload. Evaluation results show that most of the regression models we build have a high correlation coefficient (0.87) and low mean absolute error (3.8%). Using these performance prediction models, the suggested approach added adaptability to a rule based STCS by greatly increasing the amount of workloads that STCRSs can deal with.

ACKNOWLEDGMENTS

Thanks to Prof. Ethan L. Miller for advising me in this project. Thanks to Prof. Darrell D.E. Long and Prof. Jishen Zhao for extensive reviews and comments on the project. Thanks to Yan Li for starting this project and his invaluable advice that helps me through various research challenges. Also thanks to people in SSRC and all my friends who have supported me in the course of this study.

1 Introduction

High performance distributed storage systems power our modern digital world, providing high capacity and speed for a wide range of applications, from High-Performance Computing (HPC) systems to large-scale web services that support millions of users. A distributed storage system achieves high performance and reliability by allowing clients to distribute I/O requests to multiple servers concurrently. The I/O operations from multiple clients are highly random when they are mingled together. As a result, a storage server needs to serve multiple clients simultaneously. Without a proper traffic control system, each client tends to issues requests as fast as they can, hoping to grab as many shares of resources as possible. The contention between clients creates congestions in network and storage servers, and thus affects system performance.

In a rule-based Storage Traffic Control System (STCS), every Storage Traffic Control Rule Set (STCRS) needs to strike a balance between several contradictory goals: maximizing the overall system efficiency (work-conserving), providing fair resource allocation between contenting clients, and solving congestions quickly when they occur. If well optimized, a traffic control rule set can greatly improve the workloads performance by accurately adjusting the I/O rate limit for each client. The rule sets will tell the client how to slow down the deliver rate of I/O requests when congestions are detected, and how to increase the maximum rate when resources become available; in some cases the performance boost brought by traffic control mechanism is reported as high as 30% [1].

The effect of a workload can be thought of as a kind of pressure applied to the storage system. Different workload patterns represent different processing pressure to the storage system. Traffic control rules are closely related to the type of workload. The storage traffic control rule set usually needs to be optimized for each different type of workload in order to achieve the optimal performance. A sequential write workload, as an example, needs a different traffic control rule set from that of a random read/write workload. Using an inappropriate traffic control method may adversely affect the workload's performance

by either being too aggressive on sending requests and causing congestion, or being too conservative and not fully utilizing the server's capability. Discovering the optimal traffic control rule set for a specific workload is a challenge since it usually needs running multiple trial-and-error cycles, and benchmarking different combinations of parameters. The time consumption may range from a day to a week to build the optimal STCRS. Due to the randomness of I/O requests and various parameter changes, there may exist more than thousands of different workloads. It is not feasible to build a huge bunch of rule sets to cope with the situation. Existing rule-based STCS uses a database to store rule sets for discovered common workloads, but this would only covers a subset of the workloads a storage system actually dealing with. Lack of support for a variety of new workloads remains a big issue.

In order to solve the above problem, we evaluated the effectiveness of existing rule sets to various derived workloads, and proposed a workload characterization and performance prediction approach, which uses machine learning method to help choosing proper STCRS for new workloads. The monitor designed on each client side keeps track of all I/O requests and a centralized management node characterizes the current workload. A regression model pipeline evaluates every rule sets effectiveness on current workload and selects an optimal one.

The contributions of this research include:

- Proposed a carefully defined feature set which extracts key attributes from storage I/O streams to characterize the aggregated storage workloads. It includes finer grained features that help to identify user group which contend for same file resources.

- Provided a method for building performance prediction models and constructing the models into a pipeline. This approach provides a simple and efficient way approach for making decision and adapting the rule-based traffic control system to new workloads.

- Discovered and analyzed the correlation between performance predictability of the derived workload and the effectiveness degree of traffic control action to the original common workload, and the correlation between effectiveness of a rule set and the derived workload's characteristics.

2 System Design

To add adaptability to the rule-based system, we take a supervised learning approach to do the performance boost prediction and decision support. We first set an experiment environment to evaluate the effectiveness of rule sets to the changed workloads. This is based on our previous work of how to efficiently build a traffic control rule set for a specific storage workload [1]. In our previous study, we constructed a group of commonly used typical workloads based on HPC benchmarks statistics, and then established traffic control rule sets for these workloads. In this study, we call these rule sets associated workloads as common workloads. For each known common workload, we generate many derived workloads by changing one feature at a time and see how the original traffic control rule set works for derived workloads. When a new workload enters the system, we characterize it by measuring several key features, such as the request size, access type, the read-write ratio and its throughput. These features are used to characterize derived workloads and they will be further discussed in section 3. We then calculate the efficiency of the existing traffic control rule on every derived workload. Based on these evaluation results, we build a training data set, which will be used to train a regression tree model. The whole process is repeated for every common workload. By combining these regression tree models into a pipeline, we provide an automatic method for online storage workload evaluation and rule set searching. With this solution, there is no need to run a lengthy process to customize the rule sets for each new workload.

Figure 1 shows the process of building a regression tree model and how it is used when new workload enters the system. In the offline training process, I/O requests

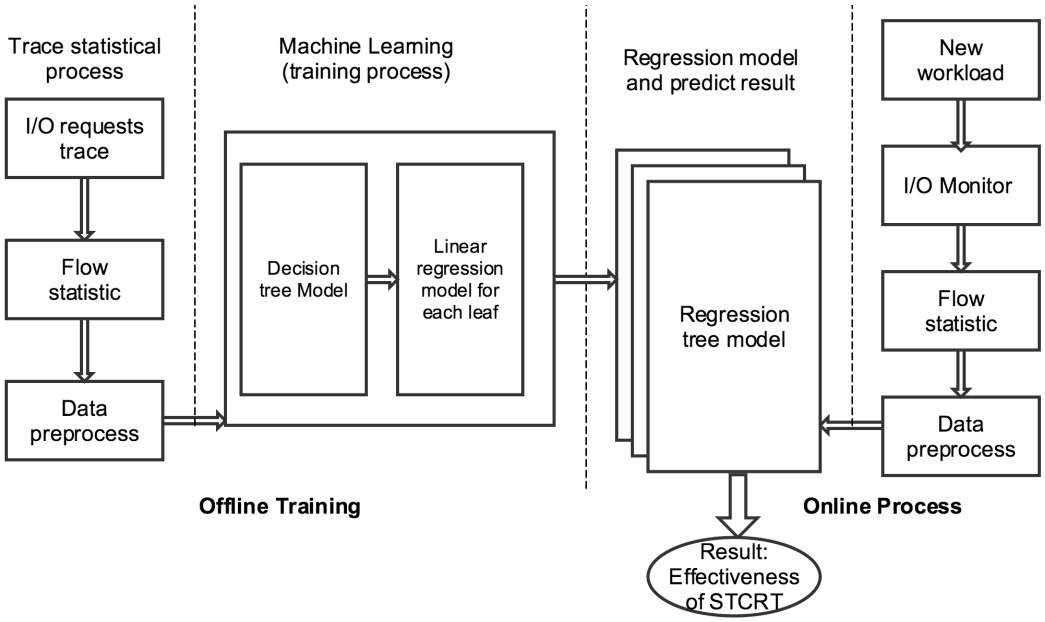
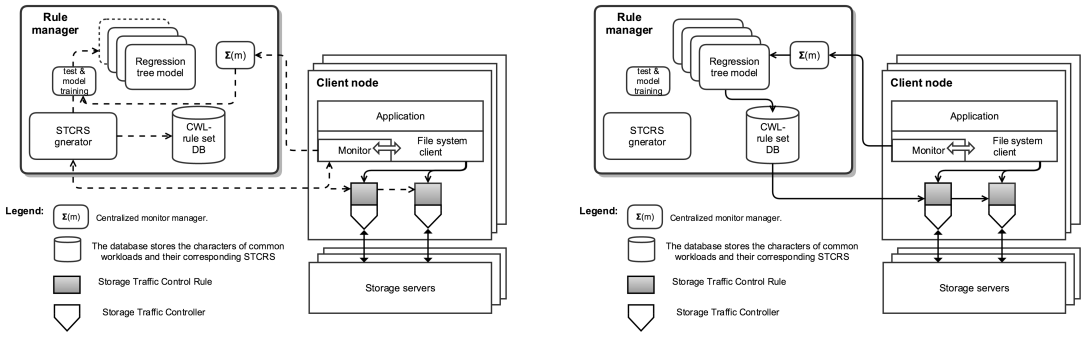


Figure 1: Offline training precess and online prediction process

and performance results for each common workload and its derived workloads will be analyzed. These statistical data will be gathered together and used as a training data set to train a tree based regression model. The training process is done offline when the system is idle. After training process, we get a series of prediction models associated with each type of common workload. We group these models into pipeline to be used in the online process. In the online process, when a new workload enters the system, the distributed monitors gather the workload's feature values every several seconds. The statistical data is used as input to our regression tree model and get the performance prediction result. The characterized workload (feature vector) will be input to every single regression tree in the pipeline and the rule set with the highest predicted boost rate will be selected. If all predicted throughput boost is negative, then no rule set is selected.

The system takes a distributed structure, which consists of a central rule management node and the client nodes. Figure 2 shows the whole system design of our rule-based STCS. Traffic controllers run on each storage client and each controller regulates one



(a) offline training process (b) online predicting process

Figure 2: System Structure

and only one data streams between clients and servers. In our case, each client has two traffic controllers because one client needs to communicate with two servers and each with a dedicated congestion window.

A system level monitor on client side plays a very important role in our design. It is used to keep track of the system clock, file descriptor and request size for every read and write related system call. Using filename along with its directory as the key, the monitor extracts and stores these values in file basis. It calculates the I/O parameters in accordance with the features set for each file operation as well as the overall read-write ratio every 3 seconds. We assume that the access pattern remains unchanged during this relatively short period.

The dashed lines in Figure 2a indicate process involved for common workload. Rule set generator works with distributed monitors to build an optimal traffic control rule set for a common workload, which will be delivered to client side controller later. In our early work, a rule set is produced in an unsupervised manner by systematically exploring the solution space of possible rule designs and evaluating the target workload under the candidate rule sets. The whole process usually takes 30 to 60 hours since it requires many benchmarking and tweaking cycles [1]. The generated rule set is stored in a database along with the common workloads feature set. The test and training unit evaluates how the new rule sets work on derived workloads and the resulting data is

used as training data set for a new regression model. The built regression model will be appended to the model pipeline.

The solid lines show the interactive process when a new workload enters the system. It corresponds to the online process in Figure 2b. The monitor extracts and computes workloads features using I/O log every several seconds and the processed data is sent to the regression tree pipeline. Decision is made on the prediction basis. A selected rule set is retrieved from database and dispatched to client side storage traffic controller. Distributed monitors also watch the aggregated features of local workloads and notify a centralized management node when a change in any feature parameter exceeds a preset threshold. The management node summarizes the features reported from client monitors and chooses the right traffic control rule set as the result of regression models pipeline process.

3 Workload Characterization

The process of workload characterization measures the I/O features of a given workload. Choosing the proper features is essential because all of the following calculation and modeling process will be based on the characterization. Researchers often use parameters like I/O request size, read-write ratio, random or sequential, response time and number of outstanding I/Os to create the feature space of the workloads and these characteristics are often used to describe traces over a long time period [2, 3, 4, 5, 6, 7]. However, workload characterization that serves traffic management should also describe the dynamic state change. An ideal feature set is able to capture all the important characteristics of a workload. It extracts relevant parameters not only from long term traffic records but also from monitoring over short periods. In our research, we use real time monitors to keep track of all operations. Since traffic management is about controlling the level of pressure the workload puts on the storage system, any features that affect the pressure have to be included.

3.1 Discussion of workload patterns

The workloads discussed here are aggregated workloads for a storage system to process, not a single I/O stream generated from one application. The aggregated workload usually combines the effects of multiple workloads. Therefore, we place more emphasis on the patterns of the aggregated workload. For instance, a typical enterprise office storage server may need to process a combination of workloads at any given moment. To illustrate this point, consider this scenario: one user is streaming a video from the server (sequential read) while another user is backing up her laptop (sequential write), and the third user might be compiling a Linux kernel (random read and write) all at the same time. If we focus on storage bandwidth allocation, we may say that the combined workload is composed of 33% sequential reads, 33% sequential writes, plus roughly 23% random reads and 10% random writes.

However, just the percentage of I/O requests by their type and size is not adequate to depict the characteristics of the workload. Taking the two workloads shown in Figure 3 as an example, both workloads have three random requests among the total eight requests, but the pressures they put on the storage system are different. The bottom one contributes a much lighter pressure. As a result, it will get a higher throughput than the top one because its first five requests form a long sequential request which is much easier for the storage system to process than random requests. Similarly, Figure 4 shows three workloads that all have 60% read requests and 40% write requests, but they produce different pressures on the underlying system as the arrangement of read and write requests are different. Therefore, the defined features must be able to accurately describe the comprehensive effect of the workload patterns. This is considered in our definition of workload features as well as their computational methods, which will be introduced in section 3.2.

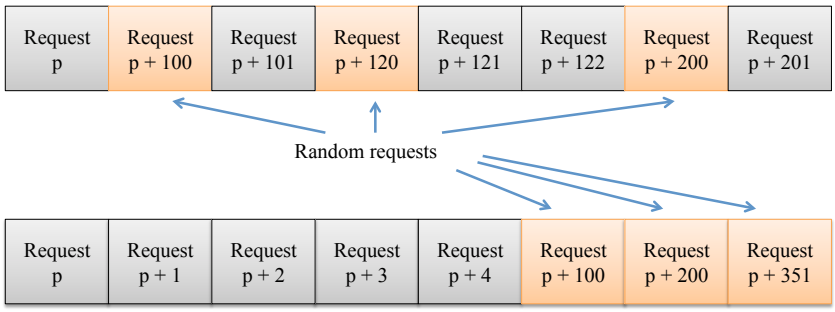


Figure 3: Two different patterns of random workloads

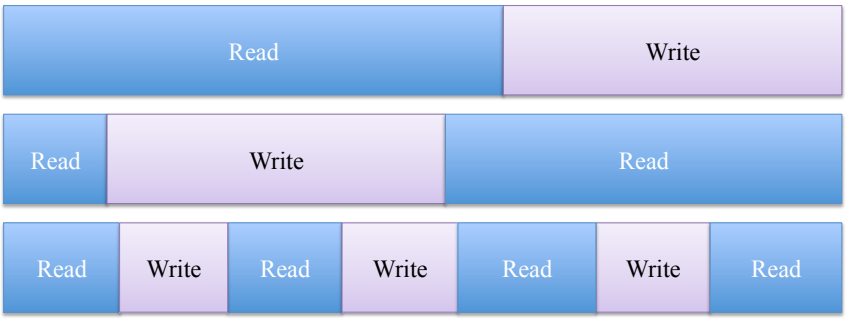


Figure 4: Three different patterns of mixed read and write workloads

3.2 Building the feature set

We designed the following features in order to further quantify the pressure effect that different workload patterns brought to the underlying storage system. A workload consists of several different kinds of I/O operations: read, and write, as well as metadata operations such as those for file creation, file deletion, etc. As a first step, we only consider non-metadata operations in this paper (read and write). We also assume that different types of operations will not execute on the same file within a limited monitor window (for example, 3 seconds, as this is the refresh period of our monitor). In real system, if different operations occur on the same file, this period will be skipped and no data will be collected. The window size can also be adjusted so that only one type of operation will be executed on each file within a monitor window.

Since the pressure on storage system is largely due to heavy demands for limited resources,

and one file may be more intensively requested than another, we also measure a workload in file basis. To describe a specific workload, we define the following features. The sequential request size and access type are measured per file per operation type, read-write ratio and the original workload throughput are measured on the basis of aggregated workload.

- Sequential request size (sreq_size)
- Access type (acs_type)
- Read-write ratio (rw_ratio)
- Original workload throughput (ori_throughput)

3.2.1 Sequential request size

A sequential request is a conceptual request that consists of one or more actual I/O requests accessing consecutively located data in the storage. The sequential request size does not describe the size of the real I/O requests issued by the application, but the aggregated size of the sequential requests as seen by the file system. For example, if the application issues five 1 MB I/O requests that read consecutively-stored data from storage before issuing another request that accesses a non-consecutive location, the file system actually reads 5 MB sequentially. This is the value of sequential request size in our model. The reason to define the request size this way is to measure the gross effect of a series of operations as shown in the pattern of Figure 3.

3.2.2 Access type

The type of file access reflects the randomness of the file I/O operation position in a workload. File access type is an enumerator with two values, random and sequential. When doing reads and writes to a file, the operating system keeps track of the location by using a counter generically known as the file pointer. For every read/write/seek

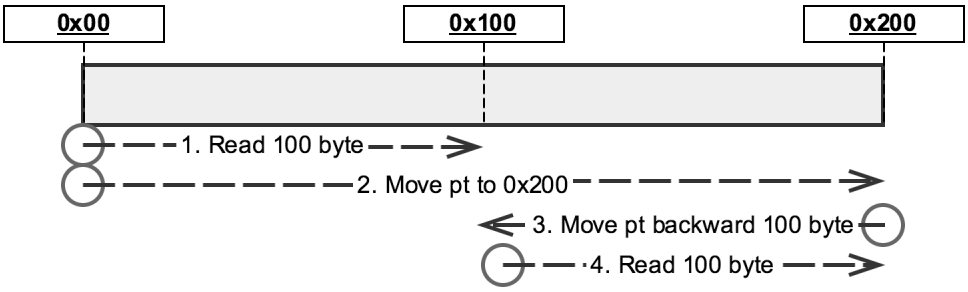


Figure 5: An example of file pointer operations

related system call, the file pointer will move according to a requested offset length or to a specific point. The access type is measured by monitoring the gap between the current access point and the file pointer position after last file access. For example, taking the process shown in Figure 5, if the file pointer points at 0x000 initially, after a system call of reading a 0x100 byte block, the file pointer will be at the 0x100 position. If the next lseek call first moves the file pointer to the position with a 0x200 offset from the start of the file, and another lessk call moves the file pointer 0x100 backward from the current position, and finally the system issues another read operation, the position gap between access point of the second read and the file pointer position after the first read is zero. Although the file pointer moves twice between the two accesses, we treat the two accesses as sequential in this case. Therefore, if every gap between the operations is zero, then it is sequential access; otherwise it is a random access.

3.2.3 Read write ratio

Read-write ratio is the ratio between the total bytes of read and the total bytes of write in a certain period of time. It is widely used as a workload specific characteristic in many storage workload researches [2, 3, 4, 5, 6].

Table 1: Relation between boost rate (br) and original workload throughput ($ori\_throughput$)

$ori\_throughput$	item with negative br	numbers of item	percentage of negative br
< 337.3	19	44	43%
> 337.3	3	33	9%

3.2.4 Original workload throughput

The original workload throughput is a feature that indicates the throughput of a workload running before any traffic control rule set is activated. In our study, we group workloads based on the effectiveness of a traffic control rule set working on the workloads. Although the original throughput is not a distinguished I/O feature to identify a workload, it provides an important baseline in measuring the performance improvement of the workload controlled by a given traffic control rule set. We found that there is a strong correlation between original throughput and the performance boost. As indicated in Figure 6 and Table 1, where these throughputs are measured for derived workloads from one common workloads running without any rule set activated, when the original throughput for a derived workload is high, the likelihood of the boost rate being negative is much larger than that of the workloads with relatively smaller original throughput values.

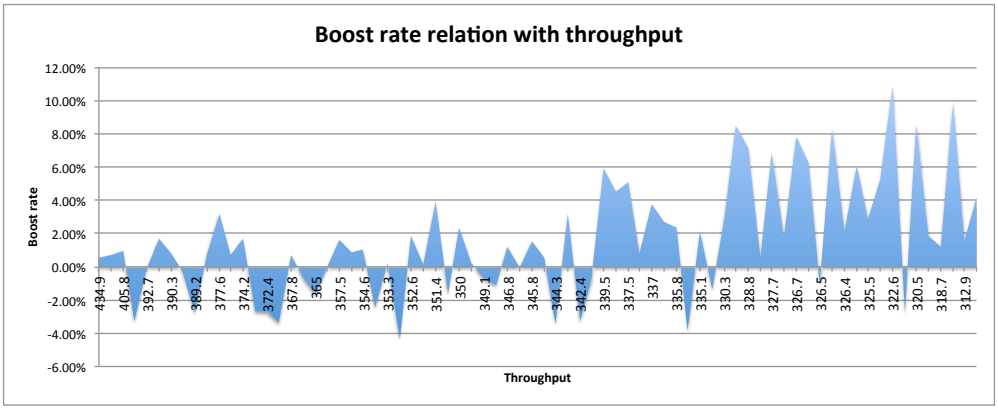


Figure 6: Relation between boost rate and workload's original throughput

3.3 Monitored operation parameters

A workload in a storage system consists of various file access requests, therefore certain features are measured and computed in a file operation basis. As mentioned in section 2, we use distributed system level monitors to track every I/O related system call. To compute the value of features, we need to monitor every read and write request and the related parameters for each file. In this study, we assume that the read and write operations will not occur on the same file at the same time (actually within our monitor window). The monitored parameters of the access operations include request type, request size, request time, and file pointer.

We use the file pointer to track and compute the position gap between the current access point and the file position after the last file access. The resulting value is used to decide the access type for the combined operations of each file. We also use the request time to compute the temporal gap between requests of same operation type for each file. The average value of temporal gap is generally inversely proportional to the amount of work that a storage system needs to handle. Longer gaps indicate a longer idle time between the I/O requests and that there is a need to schedule the storage traffic growing more aggressively; shorter gaps indicate the need to regulate the I/O requests from clients more cautiously to avoid congestion. Since we are dealing with I/O intensive workloads, the temporal gaps are relatively short most of the time (from 1ms to 300ms). In this case, temporal gap is used along with the position gap and request size to compute sequential request size. If position gap is zero and the temporal gap is almost zero(< 10ms) between a continual set of pure read or write requests, we will consider these requests as one sequential request.

3.4 Using feature vector to describe a workload

Now with the defined features, we will describe a workload in our feature space. A workload can be described as a feature vector. As we mentioned earlier, in the feature set, access type and sequential request size are measured per file and they remain unchanged

in the monitor window. A feature tuple C is used to describe the access pattern of each file.

$$C^{op} = \langle op\_sreq\_size, op\_access\_type \rangle \tag{1}$$

where superscript *op* is the operation type of the file. For example, a read process will be

$$C^{r} = \langle read\_request\_size, read\_access\_type \rangle \tag{2}$$

If a workload operates on n files within the monitor window, the feature vector $\boldsymbol{wv}$ used to describe the workload will be

$$\boldsymbol{wv} : \{C_1^{op}, \cdots, C_i^{op}, \cdots, C_n^{op}, rw\_ratio, ori\_throughput\} \tag{3}$$

The actual feature vector is workload pattern dependent. Pure read or write workloads do not have a read-write ratio feature. The number of file based features pairs is related to the number of files that the workload operates on, so the size of a feature vector for a common workload may be different from another. However, the vector structure of all derived workloads keeps the same as their original common workload.

4 Regression Tree Models for Performance Prediction

Our ultimate goal is to select a proper known rule set for every new workload instead of generating a new one. We already know how to characterize a workload using the feature set we proposed in section 3.4. We now introduce how to build a regression model to predict how a new workload can be benefit from the rule sets. Before building an analytical model, we will evaluate the effectiveness of established rule set to changed workloads and prepare the training set. Several series of experiments have been designed for these tasks.

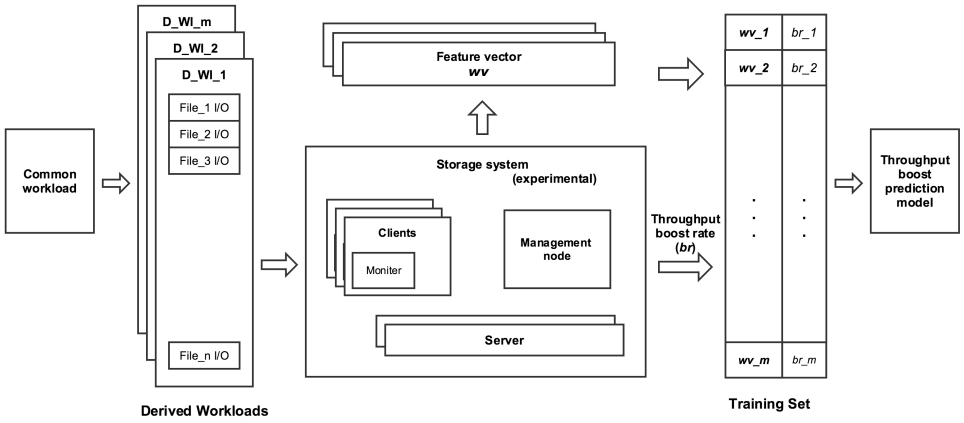


Figure 7: Building training set

4.1 Building a training set

The core idea of our method is to take one workload along with its corresponding traffic control rule set, change one feature at a time, and measure if the known traffic control method is still effective. The effectiveness is measured by how much the known traffic control method can improve the performance of the changed workloads. For instance, we can change a common workload's read-write ratio from $1:10$ to $1:2$, $1:5$, $2:1$, and $3:1$, then measure if the old method can still improve the new workload's performance. For each derived workload, we apply two experiments. One is the base line experiment running without a rule set activated while the other one runs with the rule set enabled. We run the workload ten times for every experiment and then calculate the average of the throughput for each. Finally, we compute the throughput boost rate for every workload.

As indicated in Figure 7, the process of building a training set is as following:

- First, input the derived workloads into storage system. The client side monitors and central traffic management node cooperatively extract various file access parameters and compute the values of workload features. The feature vector set (WV) is then constructed.

- Next, the traffic management system tests and calculates throughput boost rate (br) for each derived workload.
- Finally, the training set is created by combining feature vector set and corresponding throughput boost rate.

One derived workload is used to build one sample in a training set and is described as a feature vector as mentioned in 3.3.2. If there are m derived workloads from a common workload, then we have a workload feature vector set

$$WV : \{\boldsymbol{wv_1}, \boldsymbol{wv_2}, \cdots \boldsymbol{wv_i}, \cdots, \boldsymbol{wv_m}\} \tag{4}$$

We combine the feature vector and its corresponding throughput boost rate into a sample

$$s_i : (\boldsymbol{wv}_i, br_i) \tag{5}$$

where the subscript i means the ith derived workload. All these pairs together form the required training set S where

$$S : \{s_1, s_2, \cdots, s_i, \cdots, s_m\} \tag{6}$$

An example of the training set can be found on section 5.3. We usually generate 70-80 derived workloads for one training set and then the built training set is used to train a regression model to predict the performance boost of future workloads.

4.2 Machine learning algorithm choosing

After data preprocessing, we build a learning model to predict how a rule set works for a new workload. Normally, kernel models such as Kernel Ridge Regression, Support Vector Machine, Kernel Logistic Regression and Gaussian Process Classifiers work well for small dataset like our case [8]. In order to use the throughput boost rate as the label of training set for these classifiers, we can set boundaries to divide it into several

classes. The classes are: negative boost rate ($< 0\%$), poor boost rate (0-5%), medium boost rate (5-10%) and large boost rate (10%+). However, dividing the boost rate into several segments is not always a good idea since the scale of boost rate for each different common workload and its derived workloads can be different. Some training sets achieve a boost rate ranging from 10% to 30% while for others the max boost rate is as low as 6%. Thus using the same metric to divide them into classes is not always a best way.

To solve this dilemma, we predict throughput boost rate directly. There are several methods that can be used to build regression model. The most straightforward statistical learning method is the Ordinary Least Square (OLS) linear regression [9]. Conceptually the effectiveness of the workload can be pictured as a n-dimensional hypercone, with the apex being the original workload. Since OLS generates hyperplanes, we will need to slice the training data into regions separated by the hyper planes, which is not an easy task in this case.

Because of the complexity and non-uniform of different storage workloads' feature space, building a single global model would not be a suitable method. An alternative approach for a single global model is to partition the space into smaller regions. We partition the regions recursively until we get spaces that can be fit into simple models. So our global model consists of two parts: one is a recursive partition, the other is a simple model for the partition. A decision tree is used to represent the recursive partition. Each leaf of the tree represents a cell of the partition, and has a simple regression model attached to it. A regression tree combines the advantages of the decision tree model and the linear regression model. There are several obvious advantages of using a regression tree algorithm:

- It can make fast prediction. Instead of executing complex calculations, our approach just needs to look up constant along the tree, which provides a fast processing speed. Since a new workload's feature vector will be input to every regression model in the pipeline, so the speed of the prediction algorithm is important.

- It is easy to identify the significant variables for performance prediction. Understanding what features of a storage workload are crucial in the Quality of Service (QoS) aspect is essential. It can help us understand how a storage system reacts to different workloads.

- It is able to build sub-tree or branches independently. Getting performance result for a storage workload is time consuming. It would be impossible to get hundreds of samples for one training set. If some branches are missing, we might not be able to go all the way down the tree to some leafs, while under this approach we will still be able to build sub-trees for every branch independently. This is the most important reason for choosing this learning model.

4.3 Regression tree model

We use the M5 Model trees [10, 11] as our regression tree algorithm, and each leaf node of the tree has a Linear regression Model (LM) [10].

4.3.1 Building the Initial Tree

The basic idea behind building a model tree is quite straightforward. The decision tree induction algorithm is used to build the tree. A split criterion is used at each node to minimize the intra-subset variation down each branch. The feature and its threshold which maximize Standard Deviation Reduction (SDR) are chosen. The SDR is calculated by the formula:

$$SDR = sd(T) - \sum_{i=1}^{n_t} \frac{|T_i|}{|T|} \times sd(T_i) \tag{7}$$

Where sd is the function to compute standard deviation, T is the set of samples that reach the node and $|T|$ is the number of the set members. T_i is the subset resulted from splitting the node according to the chosen feature and n_t is the number of branches at the node. Also during the splitting procedure, M5 stops splitting a node if it represents

very few samples (less than 4 samples) or their values vary only slightly (less than 5% of the previous) [10, 11] .

4.3.2 Building linear regression model

For a particular leaf in the tree with feature vector $\boldsymbol{x} : \{x_1, x_2, \cdots, x_p\}$ containing p features, the model at that node is a linear combination of these attributes. The linear regression model takes the form:

$$y = \beta_1 \times x_1 + \beta_2 \times x_2 + \cdots + \beta_p \times x_p + \varepsilon = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{\beta} + \varepsilon \tag{8}$$

Where y is predicted output, $\boldsymbol{x}$ is the feature vector. The training task now is to use the training samples of the leaf node to deduce a p-dimensional parameter vector $\boldsymbol{\beta}$. Error variable ε is an unobserved random variable that adds noise to the linear relationship between the dependent variable and regressors $\boldsymbol{x}$. In our case, the output is predicted boost rate and the feature vector is the workload feature vector ($\boldsymbol{wv}$) mentioned in section 3.4.

The training process of a linear regression model uses least squares approach [12, 13]. The linear model has the form $f(\boldsymbol{x}, \boldsymbol{\beta})$. In our case, if the sample data set consisted of n data pairs $(\boldsymbol{wv_i}, br_i)$ where i varis from 1 to n, the linear model will be $f(\boldsymbol{wv}, \boldsymbol{\beta})$. The goal is to find the parameter values that best fits the data set. The least squares method finds its optimal parameters when the error sum ES reaches minimum.

$$ES = \sum_{i=1}^{n} [br_i - f(\boldsymbol{wv}_i, \boldsymbol{\beta})]^2 \tag{9}$$

After building the regression tree and finding the optimal linear model for every leaf node, the tree will be tested and evaluated. Those passed 10 fold cross validation with good results will be appended to the prediction regression tree pipeline.

5 Experiment and Result

The evaluation experiments presented in this section show how our method works in practice. The experiments objectives are:

- To build training set by evaluating the effectiveness of the established rule sets for changed workloads.
- To build the prediction models using the data set collected from the evaluation experiments.
- To test the prediction models.

5.1 Experiment environment

The evaluation system contains five dedicated servers and five dedicated clients that run the Lustre 2.4.0 file system. Our client and server nodes use the same hardware configuration: Intel Xeon CPU E3-1230 V2 @ 3.30 GHz, 16 GB RAM, one Intel 330 SSD for the OS. Each node has one 1 Gb network connection. For the Lustre cluster, each storage server node uses one 7200 RPM HGST Travelstar Z7K500 hard drive, whose raw I/O performance is measured at 113 MB/s for sequential read and 106 MB/s for sequential write. The Lustre cluster has one dedicated metadata node and four storage nodes, which match the default stripe count 4. The Lustre file system uses default settings: 1 MB I/O size, 1 MB stripe size, stripe count 4.

The workload is generated by using a modified version of FileBench [14]. Our modification focused on supporting Lustre's Direct I/O in order to remove the effect of client-side cache, and generating workloads with fixed read-write ratio. For evaluations that use synthetic workloads, it is very important to generate the required fixed read-write ratio, which is often overlooked. Real applications usually generate workloads with fixed read-write ratios because the ratio reflects the amount of work the application has finished. For instance, a Hadoop data analysis program has to read a certain amount of

input data before it can generate a certain amount of output data. It is relatively easy for a traffic optimization mechanism to boost one kind of operations, like writes, and give the impression of achieving a higher overall throughput. But it may not work for real applications that have a fixed read-write ratio, because only making writes faster will not make the application run faster if read is the bottleneck and is as slow as before. Our evaluation shows that the read-write ratio feature actually plays a vital role in determine rule sets efficiency.

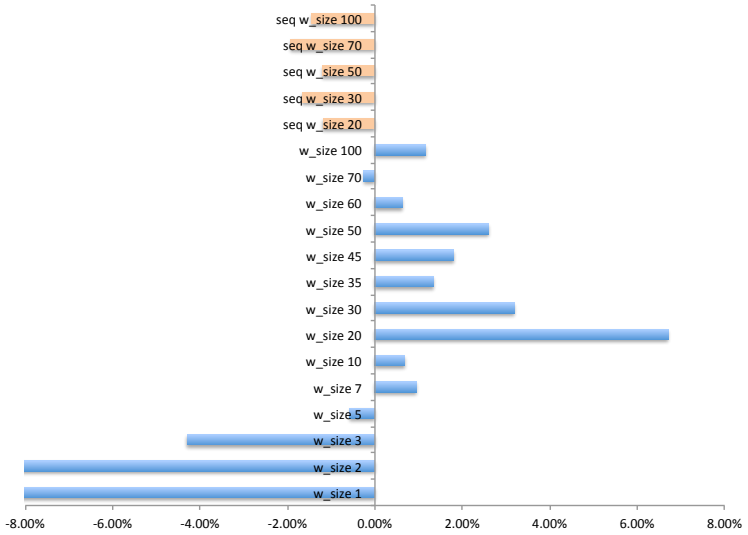
5.2 Rule sets effectiveness evaluation for derived workloads

To generate derived workloads and test the rule set's effectiveness, we took one common workload along with its corresponding traffic control rule set, changed one feature at a time, and measured if the known rule set is still effective. We organized the common workloads into two groups, pure workloads and read write mixture workload, because we cannot measure read-write ratio for pure workloads.

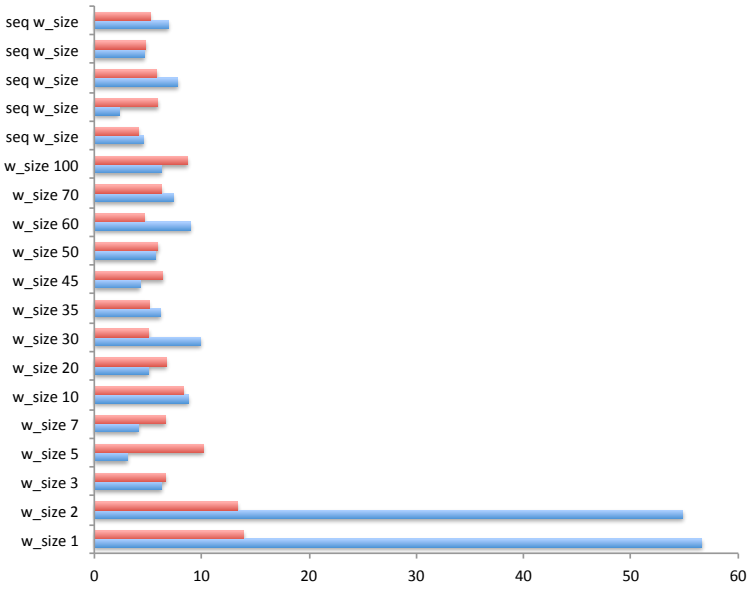
5.2.1 Pure read or write workloads

For pure read or pure write workload, it is easy to generate changed workloads. For each common workload, we derived a series of changed workloads by varying the request size and the access type. For example, to derive workloads from a 1MB random read common workload, we change the request size to 2MB, 3MB, 5MB, and even larger like 20MB and 30MB, and then tested the traffic control rule sets effects on them. We will also change the access pattern from random to sequential with different sizes and test the rule sets effectiveness.

The pure workload patterns fall into four categories: pure sequence read, pure random read, pure sequence write and pure random write. The first set of experiments we chose to present is a 1MB pure random write workload. As indicated in Figure 8, the traffic control rule set did not increase the throughput boost rate for the original 1MB random write common workload. But it decreases its throughput variance of multiple



(a) Throughput boost rate for changed request size and access pattern



(b) Variance before and after changed request size and access pattern, blue line is the original value

Figure 8: Rule set effect on derived workloads, original: 1MB random write workload

runs by 75%, so we still consider this rule set has a positive influence on the common workload. From Figure 8a, we can see that a random access type with large request size can obtain throughput boost, while with sequential access or small request size, the derived workload does not get a positive influence from the control rule set. For pure read or write workloads, when the access type changes from random to sequential, the sequential request size increases to a large number. Since some derived workloads differ a lot from the common workload, it is not surprise that the same traffic control rule set would performed differently.

The variance decreased to 10 when request size increased to 3MB and it stays under 10 as request size continues to increase. The same rule works on increasing throughput other than decreasing variance when a random access I/O stream increases the request size, since the performance tends to be more stable. This might be the reason why the traffic control rule set works better on increasing throughput for workloads with larger request size.

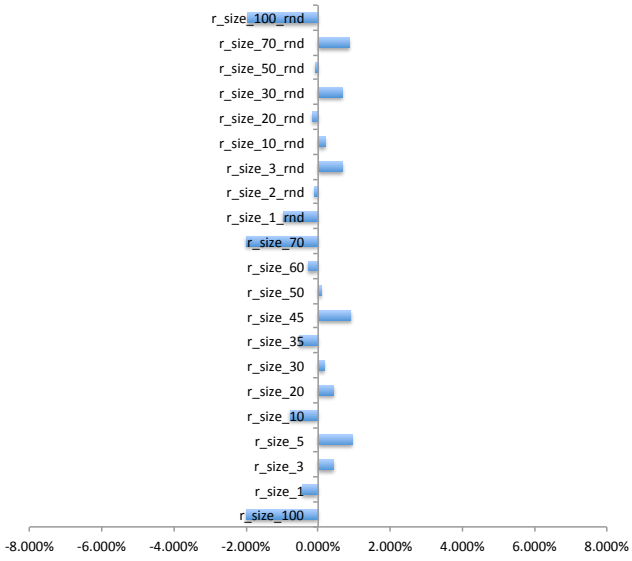


Figure 9: Rule set effect on derived workload, original: 100 sequential read workload

We also evaluated the other three pure patterns: random read, sequential write and sequential read common workloads. Different from our previous random write workload,

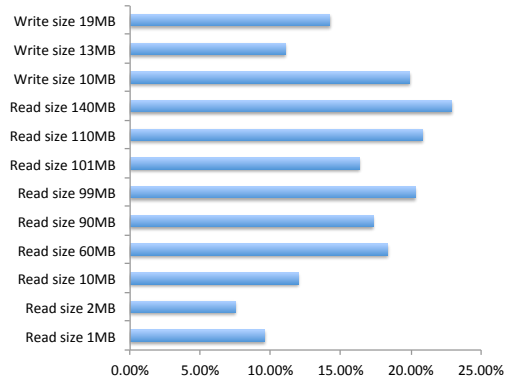
we found the rule set effectiveness for the derived workloads of these three common workloads are relatively poor. Figure 9 shows how the traffic control rule for common workload with 100MB pure sequential read works on its derived workloads. It is obviously that the whole throughput boost rates range from -2% to 2%. We therefore conclude the rule set do not have any noticeable influence on either the original common workload or the derived workloads. Experiments on the other two pure patterns have the similar results. The correlation between negative or positive influence on these workloads is unpredictable. These cases will be further discussed in section 5.6.

5.2.2 Read and write mixture workload

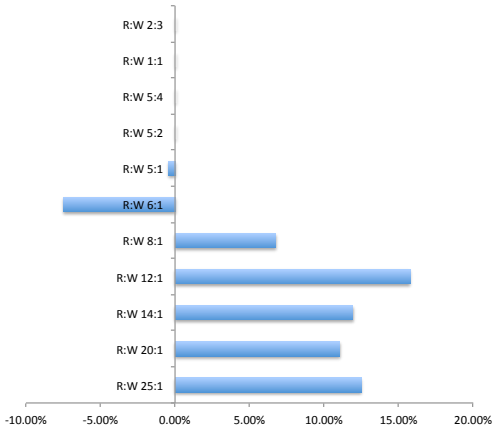
The cases for read and write mixture workload patterns are a little bit complex. We had to consider the read-write ratio in addition to other features. Each common workload in our experiment contains two types of operation and operates on two different files. The experiment result and discussion are in this section.

The first set of experiments presented a group of positive evaluation results as shown in Figure 10. It is for a mixed workload with 100MB sequential read and a 1MB random write with the read-write ratio being $10:1$. The original throughput is 25.6MB/s. To produce derived workloads from it, we changed the request size and read-write ratio parameters one at a time and keep others the same. We also changed sequential read to random read and do the same process again. The rule set's effectiveness is high when read-write ratio is larger than $8:1$ and low when read-write ratio is less than $6:1$ as indicated in Figure 10b. The rule set's effectiveness is obviously satisfactory and stay stable as shown Figure 10a when request size changes as the access type remains unchanged. When we changed sequential read to random read while the read size is small, the boost rate drops below zero as it is in Figure 10c. The evaluation results show that the read-write ratio is an important factor for the workload evaluation.

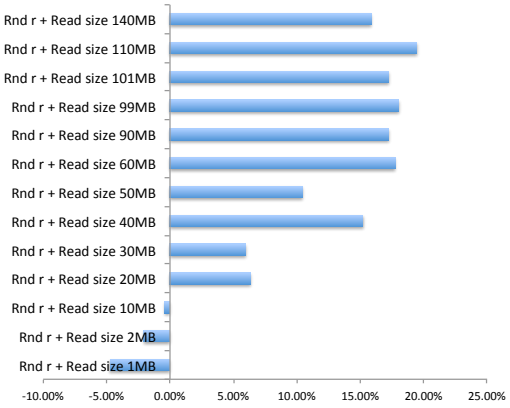
The results of the second set of experiments is indicated in Figure 11. It is based on common workload with 9MB sequential read and 10MB sequential write and has similar



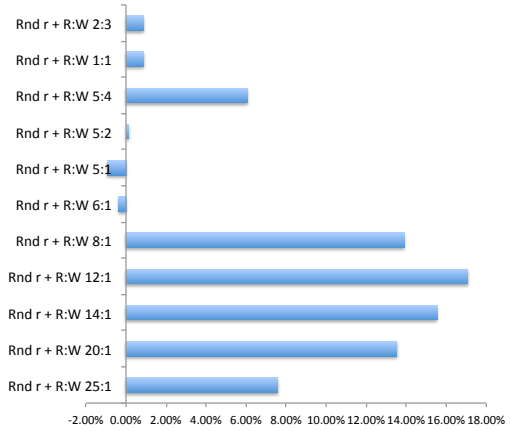
(a) Changed read request size



(b) Changed read-write rato



(c) Changed sequential read to random, and change read size



(d) Changed sequential read to random, and change read-write ratio

Figure 10: Rule set effect on derived workloads, original: read-write ratio 10:1, 100MB sequential read and a 1MB random write mixture common workload

results as the first group of experiments. Since the access type for both read and write are sequential, we cannot only change one feature at a time. We kept the read-write ratio the same and changed the read and write size at the same time (Figure 11a), then changed request size to get different read-write ratio (Figure 11b). We also changed the access type for both read and write from sequential to random and then changed request size or read-write ratio one at a time (Figure 11c and Figure 11d). The base line performance is around 335MB/s, which is much higher than it is in our first group of experiments. As discussed earlier, the baseline throughput becomes an important aspect when predicting STCRS effectiveness. A traffic control rule set tends to have a positive boost rate on a workload if the workload's original throughput is not high.

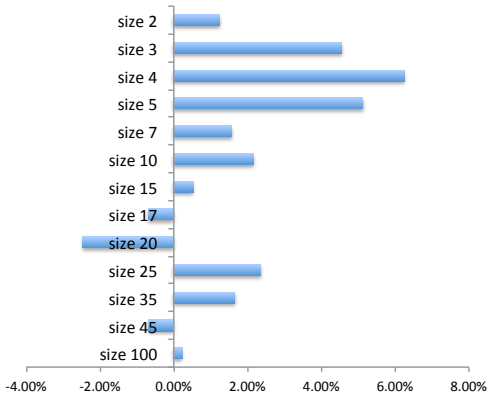
We also did a set of experiments for 1MB random read, 1MB random write with read-write ratio 10:1 common workload and got similar results. All these experiment results show that the effectiveness of using an existing traffic control method on a new workload has the following properties:

- There is a strong correlation with the read-write ratio for all kinds of read and write mixture workloads.
- There is a strong correlation with read or write size, it differs from one workload to another workload.
- There is a strong correlation with original workloads throughput if its high.

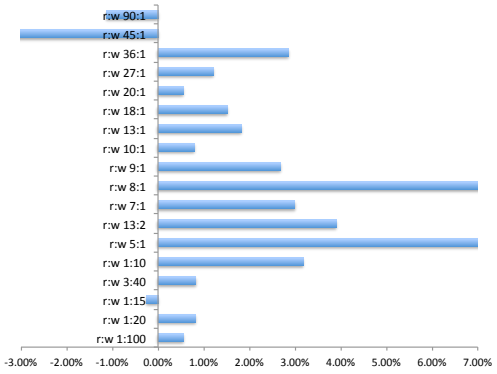
As for the last set of experiments, which is for the common workload with 1MB sequential read, 1MB random write and $6:1$ read-write ratio, the results are shown in Figure 12. There is no clear correlation observed between feature change and the boost rate. This will be further discussed in section 5.6.

5.3 Training set from evaluation result

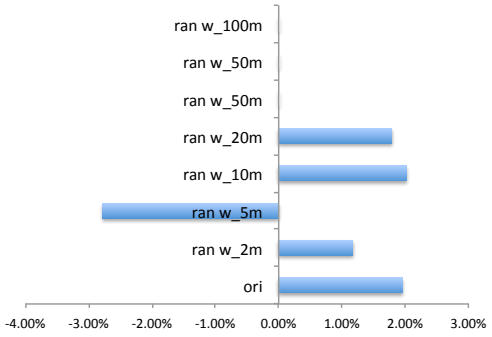
The main purpose of the evaluation is to generate a training data set for the prediction model. We usually 65-80 derived workloads for a training set. One derived workload



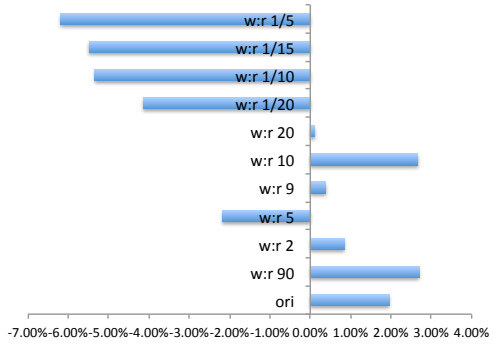
(a) Changed request size



(b) Changed read-write ratio

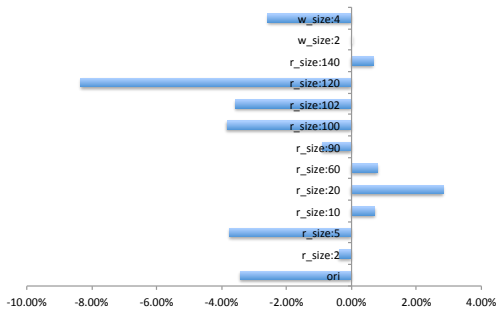


(c) Changed both read and write to random, and write request size

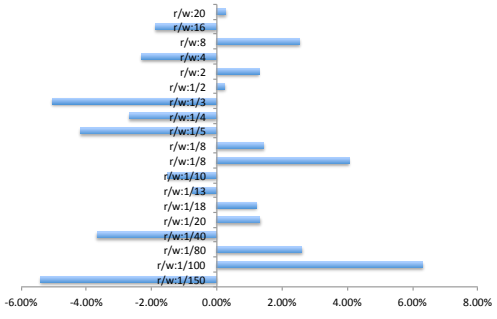


(d) Changed both read and write to random and r_w ratio

Figure 11: Rule set effect on derived workload, original: read-write ratio ratio 9 : 10, 1MB sequential read and a 1MB sequential write mixture workload



(a) Changed request size



(b) Changed both read and write to random and r_w ratio

Figure 12: Rule set effect on derived workload, original: read-write ratio 6 : 1, 1MB sequential read and a 1MB random write mixture common workload

is used to build one sample in a training set, which is consists of the feature vector and related throughput boost rate as described in 4.1. For example, we generated 68 derived workloads for a 100MB sequential read and a 1MB random write mixture with 10 : 1 read-write ratio common workload. Then we have a workload feature vector set $WV : \{\boldsymbol{wv}_1, \boldsymbol{wv}_2, \cdots, \boldsymbol{wv}_i, \cdots, \boldsymbol{wv}_{68}\}$. Combining each workload's vector with its boost rate, we have the training sample set $S : \{s_1, s_2, \cdots, s_i, \cdots, s_{68}\}$ as described in 4.1. For this specific workload, the samples have the following structure:

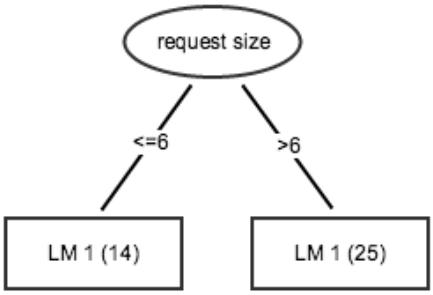
$$\begin{aligned} s : (\{&read\_request\_size, read\_access\_type, \\ &write\_request\_size, write\_access\_type, \\ &rw\_ratio, ori\_throughput\}, throughput\_br) \end{aligned} \tag{10}$$

Where *read_request_size* is an abbreviation for read sequential request size and *read_access_type* is abbreviated for the access type of the read, and the same for write operation.

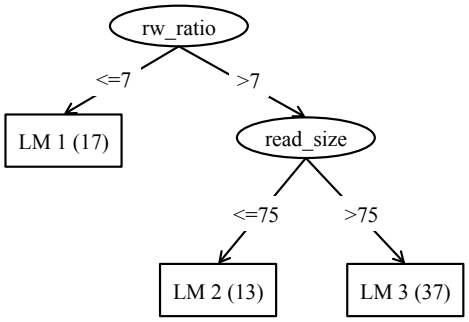
5.4 Regression tree model training process

As mentioned in section 4.3, we use the M5 Model trees [10, 11] as our regression tree algorithm. Figure 13 shows the learning result of regression tree models for the common workload we mentioned earlier. LM is abbreviated for Linear Model and the number in the parentheses is the number of cases covered by each node.

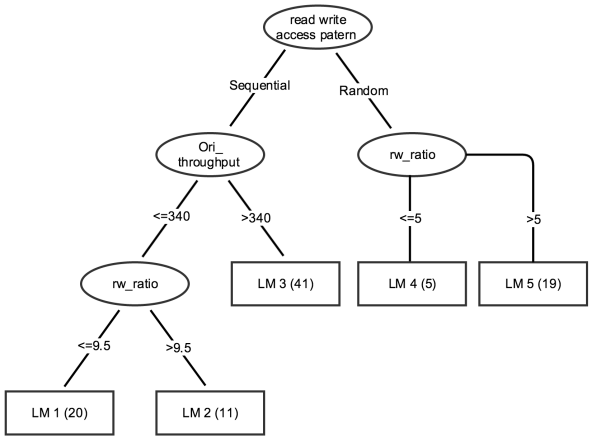
As shown in Figure 13a, the regression tree for the pure pattern workload (e.g. a random write workload) is pretty simple, where request size is the decisive feature in the regression models. The trees become more complex for read and write mixture workloads with more feature nodes and layers included. We can conclude from the tree models that the request size feature matters for both pure workloads and mixture workload, and read-write ratio feature is a key factor for mixture workloads.



(a) Regression tree for 1MB pure random write



(b) Regression tree for 100MB sequential read and a 1MB random write mixture with 10 : 1 read-write ratio



(c) Regression tree for 1MB sequential read and a 1MB sequential write mixture with 9 : 10 read-write ratio

Figure 13: Regression trees for common workloads

Table 2: Regression tree evaluation

Common workloads	Training set size	derived workloads running time (hr)	regression tree training time (s)	model evaluation		
				CC	*MAE*	*RMSE*
1MB random write	35	12	0.04	0.61	0.027	0.045
100MB sequential read 1MB random write R/W ratio 10:1	68	22	0.03	0.86	0.038	0.046
9MB sequential read 10MB random write R/W ratio 9:10	73	24	0.04	0.62	0.021	0.026
1MB Random read 1MB Random write R/W ratio 10:1	38	13	0.03	0.64	0.024	0.031

5.5 Prediction models evaluation

A key criterion on which to differentiate between regression models is the accuracy of the prediction. We use Correlation Coefficient (CC), Mean Absolute Error (MAE) and Root Mean Square Error ($RMSE$) to measure the accuracy of a model. CC measures the correlation between the target values and the model's predicated values. The errors describe the deviations between the predicted and the actual values. The MAE measures the average magnitude of the errors in a set of predictions. The $RMSE$ is a quadratic scoring rule that measures the average magnitude of the error. The MAE and the $RMSE$ can be used together to diagnose the variation in the errors of a model. The greater difference between them, the greater the variance in the individual errors between predicted and true value in the sample.

The evaluation result of all regression models is shown in Table 2. For all the models that we have successfully built, we got a relatively high correlation coefficient and a relatively low error rate. In addition, comparing with the time consumed (30 to 60 hours) for generating a rule set, building a regression tree model is pretty fast. The most time consuming part of our approach is building a training set. We can only evaluate three derived workloads performance in an hour, thus it takes almost a day to build a mini training set. Even including the time consuming for building a training set, the total process time is still less than the time needed for generating a traffic control rule set. Furthermore, the model can benefit many new workloads. Constructing these models into a pipeline, we will be able to predict the throughput boost rate for existing rule sets on a new workload and finally choose the best one.

5.6 Correlation factors of model building effectiveness

It is worth to note that the traffic control rule sets performance for some derived workloads is unpredictable. This means that we cannot build a model with good correlation coefficient and a low error rate like we did for other workloads sets.

Table 3 shows the relationship between STCRS effectiveness on the common workloads

Table 3: Correlation factors of model building effectiveness

Common workloads	w/o traffic control rule			with traffic control rule		Model trained
	TP	Var	Var/TP	TP	VAR	
100MB sequential write	104.2	9.32	8.9%	103.5(-0.6%)	5.61(-39%)	No
1MB random write	367.1	56.6	15.4%	319.1(-13%)	13.9(-75%)	Yes
100MB sequential read 1MB random write R/W ratio 10:1	25.6	2.7	10%	31.2(+21.8%)	3(+11%)	Yes
9MB sequential read 10MB random write R/W ratio 9:10	355	68.29	19%	356(+0.3%)	17.86(-73%)	Yes
1MB Sequential read 1MB Random write R/W ratio 6:1	53.1	1.51	2%	50.6(-4%)	0.8(-46%)	No
1MB Random read 1MB Random write R/W ratio 10:1	77.8	3.37	1.3%	80(+2.83%)	0.63(84%)	Yes

and model training result. These values are computed from 10 runs for each workload. The ThroughPut average (TP) and the throughput Variance (Var) are computed using results of 10 runs with one minute for each run. The reason for the lack of accurate models is one of the following as we analyzed.

First, the traffic control rule set does not have a noticeable positive influence on the original common workload. Thus, there is no surprise that the rule set does not function well on its derived workloads due to the similarity of their patterns. These can be further analyzed according to the causes. For example, for pure sequential read and write workload, it is easy to get good performance from the storage system, so the optimization space left for rule set is limited. This is the case of our experiment environment, which has only five clients, plus a mature file system like Lustre that generally handles this kind of simple sequential write workloads well enough. Another cause is the opposite as with the random read (like the second case in section 5.2.1). Generally, random workloads cannot be optimized since there's no order. According to Lustre's manual, random read, is considered one of the worst cases for Lustre system because the reads

Table 4: Long-term evaluation of rule set effect on common workload. original: read-write ratio 6 : 1, 1MB sequential read and a 1MB random write mixture common workload, result computed from 120 runs

w/o traffic control rule		with traffic control rule			
TP	Var	TP	Var	TP_bst	bst_Var
51.24	1.86	51.73(+0.49)	2.09(+0.23)	0.49	4.15

from clients may come in a different order and need a lot of seeking to get a read from the disk [15]. Therefore, it is not easy to find an effective traffic control rule set for pure random read workloads. Since the traffic control rule sets do not work well with these common workloads originally, not being able to find an effective rule for derived workloads is not a critical flaw of the model training process per se.

The second reason is time related. In this case, the traffic control rule does have a positive influence on the original common workload in a long-term view. The average throughput with the rule set may be stable after 100 runs but the average for every 10 runs differs a lot. As mentioned earlier, we used the average value of 10 runs when building training sets and running tests for experiments results. So if it takes much longer than 10 runs to converge, we will get different values from every 10 runs and consoder it unpredictable.

We tested and verified the long-term evaluation result for a certain case in Table 4. Taking the workload with 1MB sequential read, 1MB random write and 6 : 1 read-write as an example. The throughput boost rate became stable until 120 runs. Although throughput boost (TP_bst) eventually converged to 0.49, it differed a lot every time. In our experiment, running a workload once take one minute. The variance of throughput boost (bst_Var) is 10 times of its average. In order to get 120 results for both with and without rule, we need to run 240 times to get a stable result which takes 4 hours. If we want to acquire 60-80 samples to build a training set, it will take at least 10 days. On the other hand, building a rule set usually takes 1 or 2 days, so even if the result rule set is effective, we cannot afford to run one workload hundreds of time and wait it to

converge to a stable value if the converge time is way too long.

6 Related Work

Modeling is widely used in storage system related filed. In order to maintain elasticity and quality of service, many researchers use machine-learning technique to build model for system resources management in order to make good decisions to improve performance. Chameleon [16] is a self-evolution proposed by Sandeep, Li and other researchers in 2005. It is a fully adaptive resource arbitrator for shared storage resources. It relies on a combination of self-refining model to make decisions. Chameleon contains 4 models, a knowledge based sampler, a reasoning engine to make decisions, a policy set to store policies and a feedback model to take action based on the available knowledge. These models are built using Support Vector Machine (SVM), where the system is measured in different status and represented as a best-fit curve. The aim of chameleon is not proposing a perfect model, but solving problems using many simple models. It models the system performance and resources to make decisions to improve performance.

We adopted the idea of having multiple simple models as a whole. However, different from the idea in Chameleon which uses different models for different tasks; we divided the problem space by building one model for each type of common workload and combine these models into a pipeline for performance prediction. Every model is used to predict whether the corresponding traffic control rule set can be used for a new workload.

Traffic control can also be deployed for hardware [17, 18]. Researchers from TUdals built a system that modeling the system to suggest the re-configuration policy. The system first listens to low-level system statistics, and makes prediction based on performance model. Rather than software level reconfiguration, they design a dynamically reconfigurable hardware to reply policies.

Instead of using a centralized management mechanism, SCADS is a scalable distribute system [19]. To have an accurate view of the system, it uses linear regression model

to predict the rate of copy data from one node to another. It uses a feedback loop to refine and deploy policy. In our work, we combine the advantages of distributed client side monitors and centralized statistical and management mechanism. This means that the proposed architecture and mechanism can also be easily adopted for other storage systems.

7 Conclusion and Future Work

This study is a first (but significant) step in understanding how different I/O workloads respond to traffic control methods. Our work provides important background knowledge for designing future autonomous and self-optimizing storage systems, which would be too complex or too expensive for manual optimization. Our evaluation results are helpful in understanding which features are most important in a QoS system. We found that read-write ratio and request size have a strong correlation with a rule set's effectiveness. The efficiency also has a strong correlation with the workloads' original throughput when the throughput is high (> 300MB/S).

The performance prediction approach we proposed successfully adds adaptiveness to a rule-based storage traffic control system. This approach expands the applicability of the established rule set. The regression tree models and the pipeline provide an efficient prediction mechanism. Compared to the time consuming process of constructing a rule for every single workload, our method is better for online use in massive system.

Future work will include evaluating more complex common workloads. Because the workloads evaluation experiments are a time consuming process, we have only conducted evaluations for several typical workloads, and there is still a need to investigate other workloads with complex patterns. In addition, for the workloads that are not sensitive to the intervention of the traffic control rules, we should look into other methods as complements to the rule based traffic management system. We will also test the approach on different storage systems and see if our result still holds.

References

[1] Y. Li, X. Lu, E. L. Miller, and D. D. Long, "Ascar: Automating contention management for high-performance storage systems," in *Mass Storage Systems and Technologies (MSST), 2015 31st Symposium on*, pp. 1–16, IEEE, 2015.

[2] A. Riska and E. Riedel, "Disk drive level workload characterization.," in *USENIX Annual Technical Conference, General Track*, vol. 2006, pp. 97–102, 2006.

[3] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production windows servers," in *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, pp. 119–128, IEEE, 2008.

[4] E. L. Miller and R. H. Katz, "Input/output behavior of supercomputing applications," in *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pp. 567–576, ACM, 1991.

[5] P. Malkani, D. Ellard, J. Ledlie, and M. Seltzer, "Passive nfs tracing of email and research workloads," 2003.

[6] A. Gulati, C. Kumar, and I. Ahmad, "Modeling workloads and devices for io load balancing in virtualized environments," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 3, pp. 61–66, 2010.

[7] A. Gulati, C. Kumar, and I. Ahmad, "Storage workload characterization and consolidation in virtualized environments," in *Workshop on Virtualization Performance: Analysis, Characterization, and Tools (VPACT)*, 2009.

[8] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[9] J. D. Hamilton, *Time series analysis*, vol. 2. Princeton university press Princeton, 1994.

[10] R. J. Quinlan, "Learning with continuous classes," (Singapore), pp. 343–348, World Scientific, 1992.

[11] Y. Wang and I. H. Witten, "Induction of model trees for predicting continuous classes," in *Proc. of the 9th European Conference on Machine Learning Poster Papers*, Springer, 1997.

[12] A. Charnes, E. Frome, and P.-L. Yu, "The equivalence of generalized least squares and maximum likelihood estimates in the exponential family," *Journal of the American Statistical Association*, vol. 71, no. 353, pp. 169–171, 1976.

[13] S. P. Otto and P. Yong, "16 the evolution of gene duplicates," *Advances in genetics*, vol. 46, pp. 451–483, 2002.

[14] SUN Microsystems and File system and Storage Lab (FSL) at Stony Brook University, "FileBench." `http://filebench.sourceforge.net/`, 2014.

[15] F. Wang, S. Oral, G. Shipman, O. Drokin, T. Wang, and I. Huang, "Understanding lustre filesystem internals," *Oak Ridge National Laboratory, National Center for Computational Sciences, Tech. Rep*, 2009.

[16] S. Uttamchandani, L. Yin, G. A. Alvarez, J. Palmer, and G. A. Agha, "Chameleon: A self-evolving, fully-adaptive resource arbitrator for storage systems.," in *USENIX Annual Technical Conference, General Track*, pp. 75–88, 2005.

[17] P. Bodik, M. Armbrust, K. Canini, A. Fox, M. Jordan, and D. Patterson, "A case for adaptive datacenters to conserve energy and improve reliability," *University of California at Berkeley, Tech. Rep. UCB/EECS-2008-127*, 2008.

[18] J. Wildstrom, P. Stone, E. Witchel, R. J. Mooney, and M. Dahlin, "Towards self-configuring hardware for distributed computer systems," in *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pp. 241–249, IEEE, 2005.

[19] M. Armbrust, A. Fox, D. Patterson, N. Lanham, B. Trushkowsky, J. Trutna, and H. Oh, "Scads: Scale-independent storage for social computing applications," *arXiv preprint arXiv:0909.1775*, 2009.