DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

TAIL LATENCY ADMISSION CONTROL FOR RAID STORAGE SYSTEMS

YIYAN WEI
SPRING 2020

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Timothy Zhu
Professor of Computer Science and Engineering
Thesis Supervisor

Danfeng Zhang
Professor of Computer Science and Engineering
Honors Adviser

* Electronic approvals are on file.
ABSTRACT

Admission control is an important system component for limiting the number of workloads accessing a server to ensure good performance for the existing workloads sharing the system. With the growth of shared computing infrastructures like clouding computing and datacenter computing, admission control is the one of the most critical parts since it prevents the system from becoming overloaded. This thesis addresses tail latency admission control for RAID storage systems. We use the tail latency to measure the response time for the most delayed work in order to better understanding the pattern of the workloads and thus provide accurate admission control. Redundant Array of Independent Disk (RAID) is a storage technique that provides faster speed, larger storage and fault tolerance by applying multiple disks at the same time. Due to its advantages, many datacenters have use it as their storage device, so our work will focus on this common system.

Prior work has investigated multiple approaches for admission control and tail latency reduction, such as the trace-based admission control algorithm and the Few-to-Many parallelism technique, but these do not apply to this problem because they do not consider the inner structure of RAID that affect the admission control. The closest line of prior work, PriorityMeister / SNC-Meister / WorkloadCompactor, uses network calculus to analyze workloads and make admission control decisions for networked storage workloads, but doesn’t support RAID storage. In this thesis, we demonstrate that the prior work cannot effectively support RAID storage, and we extend this previous work for RAID storage. The key challenge in supporting RAID is being able to properly model and account for how workload access patterns are distributed across the storage drives. For example, workloads that primarily access a single drive behave very differently from workloads that access all the RAID drives in a balanced manner. Thus, modeling RAID storage
with black box models is insufficient, and this thesis demonstrates that a RAID-aware storage model is necessary for building effective admission control systems for RAID storage.
# TABLE OF CONTENTS

LIST OF FIGURES ........................................................................................................iv

LIST OF TABLES .......................................................................................................... v

ACKNOWLEDGEMENTS ............................................................................................... vi

Chapter 1 Introduction ................................................................................................. 1

1.1 Problem definition ................................................................................................. 2
1.2 Challenge ................................................................................................................ 2
1.3 Preliminary results ................................................................................................. 3

Chapter 2 Background ................................................................................................ 4

2.1 RAID 0 system Introduction .................................................................................. 4
2.2 Tail Latency ............................................................................................................ 5
2.3 Prioritization and Rate limits ............................................................................... 6
2.4 Leaky Bucket Model ............................................................................................ 7
2.5 Profiling disks ....................................................................................................... 8
2.6 Scheduler ............................................................................................................... 9
2.7 Traces ................................................................................................................... 10
2.8 NFS Enforcer ....................................................................................................... 10
2.9 Related Works ..................................................................................................... 10

Chapter 3 System architecture .................................................................................. 13

3.1 System Design ....................................................................................................... 13
3.2 System Working Procedures ............................................................................... 15
3.3 Disk Profiling ....................................................................................................... 15
3.4 Work Estimator ................................................................................................... 17
3.5 Configure generator ............................................................................................ 18
3.5 Configure calculator ............................................................................................ 19

Chapter 4 New Ideas .................................................................................................. 19

Chapter 5 Results and ExpSetup ............................................................................... 20

3.1 Experiment setup ................................................................................................. 22
3.2 Experiment result ............................................................................................... 23

Chapter 6 Conclusion .................................................................................................. 26

BIBLIOGRAPHY .......................................................................................................... 27
LIST OF FIGURES

Figure 2.1. “Inserting” leaky bucket model .......................................................... 8
Figure 3.1. Configure Stage ............................................................................. 14
Figure 3.2. Application Stage .......................................................................... 14
Figure 5.1. Experiment results a ................................................................. 23
Figure 5.2. Experiment results b ................................................................. 23
LIST OF TABLES

Table 5.1. Admission control and SLO guarantee comparison for singel access.......45
Table 5.2. Admission control and SLO guarantee comparison for full access..........45
ACKNOWLEDGEMENTS

There are many people who helped me to finish this thesis paper. First and foremost, I want to express my gratefulness to my supervisor, Professor Timothy Zhu, who has spent tons of time helping me understand concepts, solve problems and even debug my code. Every time when I was in need, Professor was always willing to help. Without his help and guide, there is never a chance for me to finish this thesis. I also would like to thank my advisor, who gave me ideas on how to search for a research topic. Finally, I need to thank my family, especially in this hard time of Coronavirus. Even though they are on the other side of the earth, they always give me courage and hope to carry on the study.
Chapter 1
Introduction

Nowadays, shared computing and storage resources are prevalent in many domains, such as large datacenters and cloud computing. Sharing can reduce the IT management costs significantly since it allows users to access tremendous resources as if they are on their computers for only the times when the resources are needed. However, effective sharing comes at the cost of performance variability, which could lead to bad user experience. Thus, shared systems often have performance goals, which are specified as service level objective (SLOs). These can apply to a variety of metrics such as throughput, average latency, tail latency, etc. In this thesis, we consider the tail latency metric, the slowest percentile of requests in the latency distribution (e.g., the 99th percentile). Tail latency is important because even though it only affect a small amount of workloads, these clients would possibly be significantly slow and cause the bad user experience.

A key component for meeting tail latency SLOs is admission control, which limits the number of workloads allowed to share the system. While it is good for a system to serve more workloads at the same time, it is also important that the existing workloads still receive good performance. In this case, admission control is critical for providing SLO guarantees by ensuring that the newly admitted workloads will not affect the current ones too much.

This thesis addresses tail latency admission control for RAID storage systems. Storage is one of the slowest components within a system and often is a bottleneck. One approach for
improving storage performance is RAID [2,3], which is a classic technique for using multiple storage drives for increased performance and reliability. Prior research has shown that RAID provides a cost effective option to improve storage performance [3].

### 1.1 Problem definition

This thesis addresses the admission control in a networked storage environment with RAID as the storage device. On the server side, we have a RAID 0 storage system; on the client side, we have multiple clients, each running one workload. The clients and the server are connected via Network File System (NFS). Each workload sends a sequence of requests over time and has a performance goal presented as a SLO. Our admission control system measures the work done by the workloads, and decides how many workloads it should admit based on both the client’s SLO, the client’s workload behavior and the RAID storage performance. This research focuses on how the workload access patterns affects the performance on RAID storage. That is, if the workload access pattern is more centralized in one of the disks in the RAID system, our admission controller should detect this and ensure the disk is not overloaded with too much work.

### 1.2 Challenges

Many recent papers have investigated techniques for meeting tail latency SLOs using approaches that dynamically react to workload behavior [10]. However, these do not apply to admission control since the admission decision occurs before the workload runs on a system. Thus, some recent work has taken an analytical modeling approach to performing admission control [cite
PriorityMeister/SNC-Meister/WorkloadCompactor]. These works have been demonstrated to work in networked storage systems with Solid State Drives (SSDs) and conventional magnetic hard drives, but not in RAID storage. In this thesis, we will show that the black box storage modeling approach is insufficient for RAID, and that a more detailed RAID model is needed for accurate and efficient admission control.

The key challenge in working with RAID storage is managing the potential load imbalance between the multiple disks in the RAID storage. It is possible that some of the disks get overloaded with high request rates even when the average request rate is low. Therefore, our system takes a white box approach in modeling how the requests to the RAID storage get routed to the individual disks to more accurately characterize any load imbalance.

### 1.3 Preliminary results

Our preliminary results show that by accurately modeling the RAID storage, we are able to accurately perform admission control to admit many workloads when possible while not accidentally overloading any individual disk within the RAID storage. By comparison, a conservative black box approach provides performance guarantee to fewer admitted workloads, and an optimistic black box approach sometimes admits more workloads than it can handle, which causes the system to become overloaded.

While our preliminary results are applicable to the RAID 0 variety of RAID known as striping, we envision this work to be applicable to other types of RAID with some additional work. For instance, for RAID 4 and 5, we can have one more layer in our modeling that decides the storage block that is for the parity; for RAID 01 and 10, further research is required to understand the effect of data mirroring to the storage performance.
Chapter 2

Background

In this chapter we introduces the basic concepts that builds the foundation of this thesis. In chapter 2.1, we generally discuss the RAID 0 system that our system is build on. Then in chapter 2.2 we introduce the definition of the tail latency; Chapter 2.3 introduces the working procedure of Priority Meister using priority and rate limiter. In Chapter 2.4, we specifically discuss the Leaky Bucket Model that is used to build the rate limiter. In Chapter 2.5, we talked about the method to profile the disk; In chapter 2.6, we introduce the scheduler; Chapter 2.7 will define what is a trace and what it is used for; 2.8 introduces the NFS enforcer; Finally, chapter 2.9 talked about other related works.

2.1 Introduction of RAID 0

Redundant Array of Independent Disks (RAID) system serves as one of the most important components in today’s storage systems. It combines multiple physical disk drives and serves them as one, thus achieving the goals of increasing storage performance and/or data security. By definition, RAID systems can be divided into different categories or levels based on their different organizations, each with its unique performance and restrains. Specifically, RAID 0 system provides the highest performance increase as well as the total space available. This chapter introduces the concept of a RAID 0 system.

Normally RAID 0 is recognized as data striping, meaning to split data into multiple data chunks, and perform operations on these chunks simultaneously. Though this looks simple and
attractive, the inner mechanism is more complicated. Before we introduce the procedure, there are few terms that need to be defined first:

**Stripe size**, also known as stripe unit size by some researchers, refers to the size of each chunk when striping data. Usually, this number is between 64 KB to 128 KB. For example, a data with 512 KB will be stripe into four chunks in a RAID 0 with 128 KB of stripe size. This is a configurable parameter. In our experiment, we set the stripe size to 64 KB.

**Request** refers to the command issued by the client. Each request contains the necessary information for the file system to locate the data needed, this includes arrival time, request type (read or write), request size (how large the data is) and request offset. Among them, the request size plays a critical role for the Priority Meister to calculate the work of each workload and then build the rate limiter.

**Request Offset** refers to the location where the requested data starts. In our design, there are two types of offset: RAID offset and disk offset. The RAID offset refers to the real offset that is carried with request. Since the client views the whole RAID system as a one storage device, they should only consider the offset for the whole disk. On the other hand, the disk offset refers to the offset that is specific to each of the disk. By using some calculation, we are able to convert the RAID offsets to disk offsets, which is critical to the storage profiling as well as the work estimator.

### 2.2 Introduction of Tail Latency

Latency refers to the delay of the requests due to the processing limit of the system. Tail latency is the latency of the slowest requests. Although such tail latency may only occur in a short time and thus does not affect the system performance on average, it will still cause templar delay
of some requests in a certain time, which may cause bad using experience. Sometimes, a high tail latency may be caused by burstiness. This may occur when many workload send intensive requests in a short time, and thus lead to a long queuing. One solution is to reduce the request queue by analyzing the previous trends of each workload, and then use a model called leaky bucket to decide whether a new workload should be permitted (i.e. can be added to the total works while still meeting the SLOs for the rest of the workloads). So far many researches have been done to reduce the rail latency. Some of them focus on parallelization of requests,

2.3 Prioritizing and Rate Limits

In the admission control model, each client is represented as a workload to the server. Each workload contains a flow of requests that either read or write some sizes of data from or to the disk. In a realistic situation, there are many clients or applications that access the server at the same time, which means many workloads. In order to determined which request will be processed next, we prioritize these workloads depending on their SLOs. That is, the workload with the most intensive SLO has the highest priority and thus should be processed first. This method, however, will cause starving problems to those lower-priority workloads if the current high-priority workload keeps sending request to the servers. In order to solve this, we apply an idea called rate limits. For each workload, there is a limitation of how many work it can processed at a certain period of time. Once the total size of requests of the workload exceeds this limitation, its request will not be picked until the next period. To calculate the rate limit of each workload in the storage stage, we perform analysis on the trace of these workloads, each contain many requests that represents the actual workflow of workloads. After the analysis, the system uses a model called
leaky bucket to enforce the rate limits on the workloads. The leaky bucket will be introduced in chapter 2.5. With the help of the priority and rate limits, the system will be able to figure out how to pick request among all workloads while meeting their SLOs.

2.4 Leaky Bucket Model

This chapter generally discuss the model the system uses to calculate the rate limits for each workload. The leaky bucket is a bucket with two parameters: b and r. The parameter b refers to the size of the bucket, and the parameter r refers to the rate of the tokens to be added or removed. Generally, there are two types of leaky buckets. To understand the first version, consider the bucket, with size of b, is initially full, and the token is added to the bucket in a rate of r. In this version, the processing of the requests is considered as taking the corresponding tokens from the bucket. If the tokens that in the bucket are enough for the consumption of the request, then we say this request is feasible now. The second version of the leaky bucket model is the opposite to the first version. The bucket was initially empty, and every time a request in being processed, the corresponding token will be added to the bucket. If the tokens size is still within the size of the bucket after the addition, then the request is considered as feasible. In our system, we adopt the first version. That is, when a request is under consideration of processing, the work of it should be calculate first and presented as token. Then it tries to remove the token from the bucket, if the bucket has enough tokens, then the request is permitted and thus can be processed (either read or write.) Otherwise, the requests will be queued and wait until the tokens get increased in a rate of r. In order to reduce the time for queuing, a estimator will analysis all the traces and find the suitable bucket size and rate limit, called r-b curve. Based on the r-b curves of all the workloads, the scheduler will be able to properly place all the workloads so that
most of the SLOs will be met. In order to perform the analysis, the performance of the disk as well as the predefined calculation of estimator are necessary.

![Leaky Bucket Model Diagram](image)

**Figure 2.1** "Inserting" leaky bucket model: Token in the bucket leaky in a rate of r, the leaking stops when the bucket is empty; Any tokens added represent the amount of work done.

### 2.5 Profiling Disks

The implementation of leaky bucket model needs a pretested bandwidth table to indicate the storage performance. The bandwidth table, which is generated by the profiler, contains a list of request size and a list of bandwidth that corresponding to them. If the storage system uses magnetic disks, then the seeking time is also included.

To generate the bandwidth table, the profiler needs to randomly generate some offsets within the range of the whole storage system, then by recording the duration to read/write certain sizes of data, the profiler will calculate the bandwidth of different request sizes. For a distributed storage system like RAID 0, additional work is needed to split the request size as a whole to the request size for each individual disk. More details about the RAID0 profiler will be introduced in chapter 3.3.
2.6 Scheduler

In the practical use of the system, scheduler plays an important role in determining which request should be processed next by using the leaky bucket model the priorities of the requests. Once scheduler detects that there is multiple requests from different workload coming together, it will firstly update all the leaky buckets models that represents each of the workload. During the updating, the sizes of each request will be firstly transferred to corresponding tokens, which will be compared with the token that are currently in the bucket, if the current tokens is greater, the request will be considered as feasible, otherwise it will be considered as infeasible. After updating all the workloads, the scheduler will compare each of the two workloads to determine which should be processed next by the following rules: if one client’s request is feasible and the other one is not, then the feasible one has the higher preference; if both of the requests are feasible, then their SLOs will be compared as the priority, and the one with lower SLO should have higher preference; if both of the requests have the same priority, then the common scheduling policies, such as FCFS, LCFS or RR, should be implemented accordingly. After the determined request is processed, the corresponding tokens will be taken from the bucket. By applying the scheduling policies described above, the scheduler will be able to pick up the most urgent request while still avoid starvation of other workloads. When a new workload is added to the system, the scheduler will be updated by setting the new rate limiter for the workload.
2.7 Traces

The traces are a series of requests that represent the pattern of the workload. Each request in the traces contains the following information: request size, arrival time, request offset and request type. It is the source of modeling the clients.

2.8 NFS Enforcer

The NFS-enforcer is responsible for scheduling requests to the storage devices based on their priorities and the rate limits. The enforce hook into the NFS network at the remote procedure call steps. Generally speaking, it simulates the leaky bucket model and creates waiting queues for each of the workload, but unlike the analysis stages, the bucket size is determined and fixed until the system gets reboot. The each of the workload, the request enter the queue and get checked in a first in first out order. Each time a request gets processed, the corresponding tokens will be taken from the bucket, if there is not enough token left, the workload is considered as infeasible and should be treated with the lowest priority.

2.9 Related Works

There are three major topics that related to this thesis: the admission control policy, the tail latency and the RAID performance modeling.

Admission Control: Some previous paper mainly focus on providing statistical service guarantee control by exploring the variation in access time of the blocks in the disk [6]. In these paper, a set
of traces that present various workloads is getting analyzed before the actual use of the system. Specifically, it focus on the access times of different block in the disk, then use a distributed admission control algorithm that is in the middle way of a deterministic algorithm and the observation based algorithm. This algorithm takes advantage of both the strict performance guarantee and the fairly reliable service and is able to provide possibilities of overflow, but it does not address the previous RAID problem. There is still possibilities that all the requests get centralized and causing overloaded issue; Another question is that it does not test any burst cases, so the tail latency will still be an unsolved problem.

**Tail Latency**: For the tail latency, some recent works came up with an algorithm called Few-to-Many (FM), which dynamically adjust parallelization to reduce the tail latency [1]. In order to avoid using up all system resources, this algorithm checks the request size of each request, and only perform parallelization on those longer requests. This algorithm assumes that the service demand is various and hard to predict, thus it will perform analysis on the coming requests. Then based on the request sizes, it progressively increases the number of threads. In this case, the short requests get processed sequentially, while longer requests get processed in parallel. My careful analyze the requests, this algorithm is able to decrease the tail latency by 26% [1].

FM has two limitations. The first one is that it assumes that the bottleneck of the system is the CPU. If this is not the case, only added more parallelism will not benefit the tail latency reduction. Secondly, FM is a reactive method, which does not work well for the burst workloads, which will cause many SLO violation before the method take action [1].

**RAID modeling**: There are also other researches focusing on the RAID modeling. One paper address the RAID 0 modeling question by constructing a split-merge layer on the top of an existing simulator that mathematically calculates the different disk latencies in the RAID system [11].
Based on these calculation, it was bale to find the requested service time for the system. Although this paper does not consider any pattern from the clients, it actually provide an idea on how to deal with the request distribution problem. That is, we can add one more layer on the top of an existing storage estimator (described in chapter 2), this layer will mathematically calculate the desired disks of the request and corresponding request sizes. In chapter 5, we developed this idea to support our RAID 0 specific estimator.

Chapter 3

System Architecture

In this chapter, we introduce the architecture of our new tail latency admission control system that performs tail latency Service Level Objectives (SLOs) guarantee based on the RAID storage system. This system allows more accurate modeling of the work done in the RAID storage devices than the previous works, thus provides better SLOs guarantees in a tradeoff of possibly less admitted workloads.

This chapter specifically describes the mechanism that maps client requests to each of the individual disks and how this mechanism can be combined with the scheduler to allows better modeling of the storage device and thus perform better SLO guarantees. Since the request might access different size of data from different disks in the RAID system, we need to specify these exact sizes in order to better detect whether some of the workloads can be compacted together without impacting other workloads too much. For example, if two of the workloads access only to the first disk in the RAID, then it is possible that only they can not be admitted together even though the total bandwidth of the whole RAID system still has more capacity.
We first introduce the overall design of the system in section 3.1. Then in section 3.2 we describe the working procedure of the system, starting from disk profiling to the scheduler picking up appropriate request to execute. Then we specifically introduce two important components, the profiler and the estimator in section 3.3 and 3.4.

For simplicity, we used RAID-0 system in this illustration, but the idea behind it is more general and can be implemented in other distributed storage system with slight modifications of the method that splits the requested work to each individual disks.

3.1 System Design

The system is built upon the PriorityMeister [7] work and is extended to work under a RAID storage system with NFS. The work uses the similar approach to deal with the workload and provide better latency for the workloads, but add additional attributes to ensure a better performance under the RAID 0 system. The system can be divided into two major stages. The first one is the configure stage, which includes the profiling of the storage system and calculate the arrival curve and the rate limiter for each of the workloads based on the traces that represents the practical situations; This stage will only be performed once before the system starts to process real requests, since all the workloads’ patterns have been analyzed. Then the analysis results will be collected and used as information to perform the admission control.
**Figure 3.1**: Configure stage. The work will be split by the RAID estimator and analyzed by the single estimator. The result will be used to determine whether to accept a new workload.

**Figure 3.2**: Application stage. The admitted workload will send requests to the storage through the NFS. Based on the estimation from the configure stage, there should be no overloaded and all the SLOs should be met.

The second stage is the actual use of the system. In this stage, when requests from different workloads come to the server, based on the calculation, the tail latency will be lower than their
SLOs. The major components of the system are the Bandwidth generator and the work estimator, which will be discussed in section 3.3 and 3.4.

### 3.2 System Working Procedures

The system starts by performing the disk profiling. In this step, the profiler will generate a bandwidth table that includes the a set of request size and the bandwidth of read and write. A configure generator will then use the bandwidth table along with other setting information like the number of disks, the stripe size as well as type of disk (i.e. SSD or magnetic disk) in the RAID to calculate the arrival curve as well as the rate-limiter for workloads based on the traces. That is, for each workload, there will be an array of rate-limiters generated to represent the pattern of the workload on each of the disks in the RAID system. Then based on the deterministic network algorithm, the system will determine whether the workload is permitted.

### 3.3 Disk Profiling

Disk profiling is the process to measure the performance of the storage device, and record the result in term of bandwidth. In the previous work, the assumption is that there is only one disk for each server. In this case, the profiler will not consider any inner data distribution (i.e. which disk does the data go), but instead view the whole disk as one black box. Since there is no need to consider the inner distribution, the steps of the profiling will be: (1) Generate a series of request size in an uniform distribution pattern. These are the size for the testing read and write; (2) Based on the total size of the storage device, generate a series of random offsets, which represent the location of each read and write; (3) for each of the request size, record the duration it takes to finish
(in order to be more accurate, take every ten thousand read as one set), then use the corresponding request divided by the duration to represent the bandwidth of at that request size. After finish all the request sizes, the bandwidth table is generated.

In a single disk server system, this method works fine. But in our new system that uses distributed RAID system, it fails to give detailed result on every single disk in the storage devices. Therefore, we need to add one more layer between the offset generation and the testing read or write. Before the offset has been generated, we first set the max block of the offset to the number of disk times lesser than the original max block. Then based on this max block number we generate the new random offset; Finally, to ensure that the request only goes to one of the disk, we multiple the offset by the number of disks. With this layer, the random offset now only access a single disk, thus provide profiling one disk in the RAID system.

Currently, we use the one disk version to represent the bandwidth of the whole RAID, this is based on the assumption that all the disks in the system has very similar performance. Although in the most of the time this will hold true, there is still space for the improvement. For example, we can profile on every single disks, and then use the least one to represent the RAID system based on the theory that the overall performance is limited by the lowest performed disk. We can even create different bandwidth table for each of the disks, though this may require many other modification of the rest of the system, especially the estimator part.

3.4 Working Estimator

Estimator transfers the request size and the offset into tokens, which are then used for the leaky bucket model in both configuration stage and the actual use. In the previous work, the tokens
that servers receive will be the number of the token going into the disk, since each server will only have one disk as the storage device. Now on the distributed storage system, a single estimator is not enough anymore, so we added another layer on the estimator, which deal with the request splitting part, and then this estimator will use the original estimator to perform the actual estimation.

### 3.4.1 Single disk estimator

Single estimator focus on calculate the work for each individual disk based on the assumption that all the request coming into the estimator will only go to one of the disk. Here, the work is measured as tokens with the unit of time. For the estimate work procedure, the system first takes the request size and match it with the bandwidth table generated in the profiling stage. If there is no exact matched request size in the bandwidth table, we use the linear interpolation to calculate the approximate bandwidth. Then the token can be calculated by using the request size divided by the corresponding bandwidth.

### 3.5 Configure Generator

The configure generator generates the arrival curves for each workload, which then are used to calculate the rate-limiters. These curves are generated based on the given traces. There are four major steps for the configure generator. Firstly, the generator constructs estimator based on the storage devices. If it is a RAID system, then it will also construct estimators for each disk. Secondly, the generator goes through all the requests in the trace file, and calculate the total amount of work for each disk using the generated estimators. Then find the average rate of the most heavily loaded disk as the minimum load rate:
Minimum load rate = \( \text{MAX(total work for one disk) / duration} \)

Here, the duration is calculated using the arrival time of the last request subtract the arrival time of the first request in the trace file.

\[ \text{Duration} = \text{Arrival time (last request)} - \text{Arrival time (first request)} \]

The third step is to decide the load rates that need to be recorded to form the arrival curve (i.e. the x-axis of the curve.) In our system, we take the rates based on the uniform distribution between the minimum load rate and the highest rate (i.e. total use of the bandwidth, which is 1.)

Then the fourth step is to calculate the burst. In this step, we need to address the worst case for each loads. That is, for a certain load of bandwidth, what is the maximum of the work in the whole duration. To address this question, we need to calculate the work for each disk in each load. In order to simulate the request processing, we use the following formula:

For any disk \( n \) in the RAID system:

\[ \text{Current Burst at load } i = \text{current work} - \text{load} \times \text{duration} + \text{new work} \]

Here, we simulate the consume of work by subtract the load \( \times \) duration. The new work represents the work of the new coming request.

In the end, we choose the maximum burst among all disks in each load and use it as the y-axis of the arrival curve.

3.6 Configure Calculator

The configure calculator has two main jobs: To calculate the latency of each workload based on the arrival curve, and to generate the priorities of workloads based on their SLOs. The new system uses the deterministic network calculus mentioned in the previous work.[7]. Once the
arrival curve and the priority have been calculated, we are now able to come up with the rate-limiter for each workload, and transfer it to the NFS enforcer in the server side through remote procedure call.

Chapter 4

New Idea

In this chapter, we introduce the new idea to properly model the access pattern distribution based on the RAID system. The general idea is to add one more layer at the top of the estimator that evaluate the work on an individual disk. This new layer is called RAID estimator.

The RAID estimator is an upper layer of many single estimators. It does not focus on the actual calculation of the tokens of disks, but instead it will split the request sizes to different ration based on the given offset. and assign them to each single estimators. The working procedure is:

1. Based on the stripe size and the offset, calculate the index of the start disk where this request will access:

   \[ \text{Start Disk Index} = \left( \frac{\text{offset}}{\text{stripe size}} \right) \% \text{disk number} \]

2. Starting from the start disk index, calculate the start index for each of the disk. For example, a RAID offset of 100 in a RAID system with 4 disks and stripe size of 10, the starting position of the request will be the second disk. And for that starting disk, the local offset will be 30.

3. Calculate the request size for each single disk based on their local offset and the stripe size:

   \[ \text{Number of full disk} = \frac{\text{request size} - \text{start disk size}}{\text{stripe size}} \]

   \[ \text{Disk request size}\{\text{start disk}\} = \text{start disk size} \]

   For the number of the full disk:

   \[ \text{diskRequestSize}\{(\text{startDisk} + \text{diskIndex})\%\text{diskNum}]++ = \text{stripeSize}; \]
(4) With the formula above, we are able to calculate the request size for each disk, then we passed the offset to each of the individual estimator, and collect the estimated tokens.

Chapter 5

Experiment setup and Results

In this chapter, we will introduce the experiment that demonstrate the new idea in Chapter 4. We will first introduce the basic setup of the experiment, like the device that we run our tests on; then I will introduce the steps to take the experiments. In the end we display the result and compared it with the result from the previous work. In the previous work, the system views the storage device as a whole. For instance, when measuring the performance of the storage device, the system does not care which disk the request asks to access, but instead test the total time to finish that request. Also, in the case of estimating requested work, the estimator will also treat the device as one. In this case, the bandwidth that represents the disk performance will be the overall bandwidth, and the work done by processing requests is considered averagely as well.

In the scenario that the requests access all the disks in an uniform distribution, the system works fine, since the work is flatten averagely as the profiling stage assumes. However, in a more strict case, in which all the requests are actually accessing only one disk in the RAID system, this method might not work well enough. Suppose the overall bandwidth of the RAID system with four unique disks is 400 MBps, this means 100 MBps for each individual disk. If for certain workload, all of its request go to the one of the disks with the total use of the bandwidth, then the average bandwidth it takes will be 400 MBps. For a system that focus on only the general use of the storage, it seems feasible, since the total bandwidth the storage can provide is around 400 MBps. However,
if we take a close look at what happens in the disk one, which is the disk that actually handles all the request, we can find that the load is far exceeded (400 MBps compared with 100 MBps). Though this situation could be rate in actual use, if this really happens, the exceeded requests will be queued and causing violation of the SLOs.

Our new approach, on the other hand, can handle this situation more properly by analysis the workload on each disk. Suppose we use the same example introduced above, this time we consider the limit for each disk, which is 100 MBps. When a new request gets analyzed, if the total use of the bandwidth exceeds limit of any disks, the workload will be infeasible and given the lowest priority. That means our new approach will accept relatively lower number of workloads, and give higher performance guarantee to those admitted workloads. Compared with the previous work, the result provided by the new system is more helpful, since it reflects more accurate usage of the storage device.

5.1 Experiment Setup

The server and the clients are running on the physical machines from Penn State University’s Computer Science and Engineering department. Below is the setup detail for the server and the clients:

**Server setup**: we use a two-disk formed RAID 0 system as the testing storage device. For simplicity we use the same disk with stripe size of 64 KB. The networked file system we use is nfs 3. Before the experiment, we filled 200GB random data into the test file.

**Client setup**: we tests for 1, 2 and 3 workloads, so we have three identical client machines, each with 1 CPU. We also tested different number of threads that is used to run the experiments to
ensure that the number of threads would not be the bottleneck. The traces used in the experiments are generated with a 0.5 bandwidth utilization of single disk. That means for each of the workload, it will take 50% of the total bandwidth while performing read test. For simplicity, we specify the tail latency to be the 99 percentile and the latency number to be 0.2 seconds.

5.2 Experiment Result

There are two set of experiments, each tested three different situations. The first set is the cases that all the request targets one single disk in the RAID system. And the three situations are for 1, 2 and 3 workloads. Figure 5.1 shows the results of this set of experiment. Figure 5.1(a) is for the 1-workload version, and the 5.1 (b) is the 2-workload version. The results shows that as more workload get added to the system, the tail latency gets increased dramatically, and only the 1-workload version can meet its SLO. This indicates that in the situation that all the request gets concentrated, the system can only admit one workload while maintain its performance goal.

The second set of the experiment simulates the situation that the data access get distributed into all the disks in the RAID system. Figure 5.2 (a) (b) (c) show the three scenario for 1, 2 and 3 workloads sending requests to the server. Compare the (a) (b) and (c), we found that the latency only increases slightly when more workloads get added to the system. And even the 3-workload version meets all clients’ SLOs. This indicates that for the situation that the request data gets distributed to all disks, the system can admit up to three workloads and still maintain their SLOs.
We also recorded the estimation from the three versions of the admission control system. Table 5.1 and 5.2 shows their prediction on how many workloads they can admit in each of the experiment and whether they still meet their performance goals. From the table, we can see that for centralized data access, both the pessimistic and our new system meets the SLO goals. The optimistic version fails to meet the SLO because it admits more workloads than it can handle. For
the fairly distribution data access version, all of the three system meet the performance goals. However, the optimistic and our new system admit more workloads than the pessimistic version. In this case, we can conclude that no matter how the request data access distributed, our new system can always meets the performance goal while admit as many workloads as possible.

<table>
<thead>
<tr>
<th></th>
<th>Admit</th>
<th>Meet SLO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimistic</td>
<td>3 workloads</td>
<td>✗</td>
</tr>
<tr>
<td>New System</td>
<td>1 workload</td>
<td>✓</td>
</tr>
<tr>
<td>Pessimistic</td>
<td>1 workload</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Table 5.1**: Admission control and SLO guarantee for centralized data access for three system

<table>
<thead>
<tr>
<th></th>
<th>Admit</th>
<th>Meet SLO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimistic</td>
<td>3 workloads</td>
<td>✓</td>
</tr>
<tr>
<td>New System</td>
<td>3 workloads</td>
<td>✓</td>
</tr>
<tr>
<td>Pessimistic</td>
<td>1 workload</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Table 5.2**: Admission control and SLO guarantee for fairly distributed data access for three system
Chapter 6

Conclusion

This thesis presents tail latency admission control based on RAID system. It provides method to combine the tail latency SLO admission control and the RAID system. It does so by profiling single disk in the RAID system, build a model to calculate the request distributions to each of the disk and then analyze the traces to determine whether it should permit a new workload. In the experiment, we compare three different systems – the optimistic, the pessimistic and our new system. By showing our system performs better in both the single-disk access workflows and the full-disk access workflows, we demonstrate that our system has made progress on the RAID admission control system. Future work will be to expand the idea of our system to other RAID system like RAID 5 or RAID 10.
BIBLIOGRAPHY


European Conference on Computer Systems (EuroSys ’17). Association for Computing Machinery, New York, NY, USA, 95–110. DOI:https://doi.org/10.1145/3064176.3064209


ACADEMIC VITA

Yiyan Wei

EDUCATION BACKGROUND
Pennsylvania State University Sep 2016 - May 2020
Bachelor of Engineering in Computer Science, Minor in Math

Areas of courses taken: Computer Organization and Design, Introduction to System Programming, Data Structure and Algorithm, Discrete Math, Elementary Probability, Database management system, Artificial Intelligence

WORK EXPERIENCE
Software developer, Tencent, Shenzhen, China Jun 2019 – Aug 2019
- Optimized the existing engine to reduce and stabilize the loading time for PUBG Mobile Game
- Established a standalone version PUBG with AI robotics as the enemies
- Used multi-thread and module loading method to load the faraway part of the map
- Implemented request for adding functions to any character in the game and tested its feasibility

Software developer, Invell Inc., Washington DC, USA Jun 2018 - Aug 2018
- Investigated the possibility to generate government work report automatically by using Deep Neural Network in TensorFlow with Python
- Used Natural Language Processing method to parse Chinese language and train model
- Set up frontend website with JavaScript and html language

Schreyer Honor Thesis Research, Penn State University, University Park, USA Jun 2018 - present
- Transplanted one tail-latency reduction algorithm previously used in SSD to data center (i.e. RAID) with distributed system to add latency reduction function to data center
- Compared the performance metrics of tail-latency reduction algorithm in different RAID and measure each RAID’s advantages and disadvantages
- Chose the optimal RAID system from different RAID systems based on the experiment results including the latency, throughput and stability

SKILLS AND AWARDS
Computer Skills: Microsoft Office Suite (Word / Excel Certified / PowerPoint), Databases, C, C++, Java, Python, SQL, Verilog

Mathematic Skills: Vector analysis, Ordinal differential equation, Calculus, Statistics and discrete math, game theory

Awards: President's Freshman Award, President’s Sparks Award, The Evan Pugh Scholar Junior Award