Performance of Repetition Codes in Latency Analysis in Distributed Storage

A Project Report

submitted by

ANIRUDDH VENKATAKRISHNAN

in partial fulfilment of requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

June 2018

THESIS CERTIFICATE

This is to certify that the thesis titled **Performance of Repetition Codes in Latency Analysis in Distributed Storage**, submitted by **Aniruddh Venkatakrishnan**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Pradeep Sarvepalli
Research Guide
Assistant Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 18th June 2018

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor Prof. Pradeep Sarvepalli for the continuous support of my B.Tech study and research, for his patience, motivation, enthusiasm, and immense knowledge.

Besides my advisor, I would like to thank Prof. Parimal Parag for his encouragement, insightful suggestions, and hard questions.

I also thank my fellow student in IISc, Ankit Dhiman, for the stimulating discussions and immense help.

ABSTRACT

KEYWORDS: Latency ; MDS codes ; partitioned repetition codes ; Sojourn Time

Modern communication systems store data across multiple nodes to improve the reliability and performance of the storage system. While the most widely used method of redundancy is the Maximum Distance Separable(MDS) codes, this paper compares the latency performance of the partitioned repetition codes in an attempt to discover if the difference is marginal or significant.

Marginal difference in the mean sojourn time can be compensated by the difference in decoding time to recover the original data. The paper also discusses the advantage of partitioning and the variation of mean sojourn time with varying parameters, while keeping the comparison fair through normalizing the mean download time.

The paper also presents an efficient method to arrange the data within a server system to increase the performance of the storage system.

TABLE OF CONTENTS

NOTATION and ABBREVIATION

IITM	Indian Institute of Technology, Madras
MDS	Maximum Distance Separable code
FEC	Forward Error Correcting code
λ	Mean inter-arrival rate of requests
μ	Mean service rate of each server
S	Number of Servers
K	Number of pieces the file is split into
N	Number of pieces stored in the server after coding
p	Number of Layers in which the file is split
m	Number of replication of original file

CHAPTER 1

INTRODUCTION

Data servers hold terabytes of information and face innumerable downloads every hour. The standard methods to safeguard against servers' fault or a temporary drop in its access speed are to include redundancies. These are often in the form of Forward Error Correcting codes (FECs), the most prominent of which are the MDS codes.

A file which is to be stored in a storage system, is split into k pieces. These pieces are then encoded into n pieces, inherently adding redundancies. These n pieces are then stored across S servers.

The MDS codes have an advantage in the fact that **any** k pieces of information stored in the servers can be used to obtain the entire file. This ensures that even if any $k-1$ pieces are downloaded, any of the remaining $n-k+1$ will still be independent of the downloaded data and hence will still have some useful information. However, the MDS codes imposes certain limitations in terms of the size of each of the pieces and the decoding time to retrieve the original file from its encoded version. The decoding complexity scales in the order of $O(n^2)$

Repetition codes, on the other hand, involves just repeating the data and storing them across the servers. This method is not as efficient as the MDS codes as when few pieces of data are downloaded, some of the remaining pieces are entirely redundant in that they have no extra information than that which is already downloaded. This fact renders certain servers to be entirely devoid of further use. However, the repetition code has no limitations in terms of size of the pieces and the decoding complexity is simply a matter of concatenation of the split pieces, which is of order $O(n)$.

Such methods of encoding are usually used to improve the robustness of the storage system and as an insurance against the failure of a few servers. However, the use of redundancies in multiple servers can also be used to decrease the time to download data which is very often used. This paper explores the methods of improving the latency performance of repetition codes and to bridge the download time difference between the repetition codes and the MDS codes.

The method proposed is to modify the structure of the servers by means of **partitioning**. Through partitioning, each server can store more data pieces, although of a smaller size. This, however, increases the re-usability of servers and will increase the parallel download capacity of the server system.

The paper also proposes an efficient method to arrange the data pieces within the server system to increase the server's usefulness beyond the partitioning gain. This is through a cyclic shift of data across the servers for a prime number of servers.

The paper shows numerical results for the partitioning gain and also theoretical calculations for calculating the mean download time for a given permutation of the data.

CHAPTER 2

Background and Related Work

2.1 Queuing Theory

2.1.1 System Model

The simulations to be run for testing the latency of the storage system is through the use of queuing systems. The queuing system, in this paper, consists of a collection of servers and some queues. The objects in the queues are requests to download the files stored in the servers.

The inter-arrival time between 2 requests is an exponentially distributed random variable with mean rate λ. Each server will provide a piece of the data after an exponentially distributed random time with mean rate μ. As soon as enough information, as needed to recover the entire file, is downloaded, the requests will leave the queue. The **mean sojourn time** is defined as the time between the arrival and departure of a request from the queuing system.

There are 2 models, the *pull model* and the *push model*. The basic idea in both the models is that requests arrive for the file to be downloaded and they are added to the queue.

2.1.2 Push Model

The push model assumes that there is a single queue and all the servers service the request in this queue simultaneously. The push model makes the mathematical analysis more tractable, although not actually implemented.

2.1.3 Pull Model

The pull model assumes that each of the servers have their own queues, and they serve only the requests in their respective queues. The pull model is widely used in applica-

tions and is the model used in this paper in the queuing systems.

2.2 Scheduling Policies

Scheduling policy is the order in which each server serves the requests which are present in the queue. The schemes used for scheduling is fundamental in the performance of the queuing system. Each scheme has its advantages in different queuing models and for different parameters to be optimized. Some of the methods are: *round robin scheme*, *greedy scheme*, *sharing scheme*. All these schemes assume that there is a **central controller** which knows the data that has been downloaded by the requests and directs the servers to behave accordingly.

2.2.1 Greedy Scheme

In this scheme, each server serves the request at the head of the queue. As soon as one server finishes serving all its data to the request, it proceeds to serve the next request. As soon as the request receives K pieces, the remaining servers terminate their current operations and begin to serve the next request. This scheme ensures that each request is given the maximum priority when it is at the head of the queue.

2.2.2 Sharing Scheme

In this scheme, at max only K servers handle one request and not necessarily simultaneously. As soon as one server finishes serving all its data to the request, it proceeds to serve the next request which **does not have** k servers serving it yet. In this scheme, each request is given the minimum resource needed to obtain the entire data.

2.2.3 Round Robin Scheme

In this scheme, the servers process the requests one after the other collectively. When a server becomes idle, it serves the next request in the queue which is not yet being served by any of the other servers. This method serves all the requests equally and in a

parallel manner.

2.2.4 Optimal Scheme

All the above schemes are *work conserving schemes*, in that all servers are in use as long as the queue is not empty. For a constant download time, the sharing scheme works best as there is no redundant downloads. In case the download time is exponentially distributed, as is most likely, the greedy scheme is delay optimal, i.e, has the least sojourn time. (2)

2.3 Encoding Schemes

2.3.1 Usage in Reliability

Usage of FECs have been used to provide reliability of service. For data which are sparingly accessed, these methods act as failsafe in the case of failure of a few servers. For a given measure of reliability, the MDS scheme has the least storage size. Conversely, for a given number of servers, the MDS scheme provides the maximum reliability, because of their intrinsic property of having the maximum distance.
While MDS is a storage-optimal encoding scheme, other encoding schemes, like raptor codes (9) and LDPC codes (7) have also been developed which are near optimal but have lower encoding-decoding complexity.

2.3.2 Other Performance Metrics of Coding

While lot of work has been done on the reliability of encoding schemes, another factor to be considered is the cost of repairing a faulty server (1). Studies from social networking servers show hundreds of Terabytes of cross-traffic is created to repair a server node failure. Here, the MDS scheme proves inefficient in repairing a faulty server and other codes like Locally Recoverable and ReGenerating codes are used (8). (2) also examines the effect of *download bandwidth* needed and the optimal scheduling scheme for it.

2.4 Latency Performance of MDS

For data which is very often used, or *hot data*, these encoding schemes act as tools to improve latency, which is just as critical as reliability. A trade-off study in (3) shows the variation of download time with storage space. Improvements in Latency performance using simple codes and queue models is done in (4). A theoretical study on the latency performance of the MDS codes is done in (5), which give a lower bound and upper bound on the throughput.

2.5 MDS and Repetition codes

The coding scheme used to distribute the data pieces across the servers is vital in determining the latency performance of the storage system. To compare the performance of different coding systems, the model used for queuing and scheduling must be the same. In (6), the push model is used with the greedy scheme. The analysis given in their paper proves that the MDS codes outperform the repetition code in all cases, and also provides theoretical bounds on the performance of the MDS codes. While plotting the performance of the schemes for different coding rates, which is the ratio K/N, it is observed that the difference in performance between the two codes is marginal in cases of very low code-rates (values near zero) and very high code-rates (values near unity), where the redundancies are sparser.

This paper is motivated to improve the performance of the repetition code and attempts to bridge the latency performance gap between MDS coding and repetition coding.

CHAPTER 3

Simulation Model

In this chapter, the exact details of the simulation model and the parameters used are described and explained. While varying the parameters such as number of servers (S), number of pieces the file is split into (K), file size, etc, the mean sojourn time for the download will also vary. To ensure that the comparison between results of simulations with different parameters is fair, the service time of the servers will have to be modified.

3.1 Queue Model

The paper assumes a *pull model*, where each server is attached to a separate queue. When a new request arrives, it joins the end of each queue. As soon as it gets all the relevant data from a server, it exits the queue of that server. As soon as the entire file is obtained, in simply fragmented or in encoded format, it exits from all the queues and leaves the queuing system.

3.2 Scheduling Scheme

As discussed before, and proved in (2), the greedy scheme is delay optimal in cases where the service time of the servers are exponentially distributed. However, the proof and simulation verification is done considering an MDS encoding, where each of the pieces have information. In such a case, all the parallel downloads are useful except in the case where *k* pieces are obtained and the other downloads are dropped. However, in the case of repetition, it is possible that two servers are simultaneously downloading the same piece. Even though the probability of such a case is low for a large number of pieces, it is still a redundant use of server usage. However, there is an increased speed of the download as two servers are downloading the same piece parallelly. Hence, in this study, greedy scheme is used and all servers parallelly serve the head of their queues.

3.3 Server Working

Any server whose queue is empty goes to an idle state. When a new request joins its queue, it starts its service. If a server has multiple pieces of data (possible in the case the file is split into many pieces), the download happens from the first piece to the last piece in order. In the case of repetition code, some of the pieces might be useless as they have already been downloaded from other servers. In such cases, the redundant pieces are skipped *instantaneously* and the next *useful* piece is downloaded. If a download is *interrupted*, as can happen if the piece being downloaded is obtained from another server or if the request has received the entire data and is exiting the queuing system, the current service is terminated and the next service is started. The next service might be to the same request or to the next request depending on the cause of interruption.

3.4 Service Time

For all the simulations, the file size is assumed to be very large and constant to ensure that any piece size criterion and divisibility criterion are met. Different runs of the simulations have varying *S, K, N*. To ensure fairness of comparison, the service rate of each server is varied according to $\mu = K/S$. This can be explained by considering *T*, the mean time to download a piece. As there are *S* servers working in parallel, the overall time of the system decreases by a factor of *S*. To compensate for this **Parallel Gain**, the mean time of download for each server is increased by *S*. As the file is split into *K* pieces, the download time for each piece must decrease by a factor of *K*. Hence the time to download a piece on any server is *T = S/K*. Since the rate of service is the reciprocal of the mean time, we get $\mu = K/S$. Considering this rate of each server and the parallel gain of multiple servers (*S*), we get the mean time to download a file (*K* pieces) by a request is

$$T_{mean} = T * \frac{K}{S} = 1$$

3.5 Arrival Rate and Stability of Queue

For any queuing system, the stability is decided by the length of the queue. If the length keeps increasing indefinitely, the queue is termed **unstable**. There are two ways to keep a queue stable. The first is to limit the length of the queue to a MAX limit. This method is physically used in real-time situation as the buffers in the servers have a maximum capacity beyond which requests cannot be handled. The requests which arrive after the buffer is full are dropped (not considered). The second method is to ensure that the service rate is higher than the arrival rate. This would ensure that the queue length is stable at a certain value and only very occasionally increase to a high value. In such cases, the value of the length of the queue, on average, is

$$L = \frac{\lambda}{\mu - \lambda}$$

While the average length formula is applicable only when the arrival rate and service rate are exponential, the stability criterion is always valid. Since the mean service time for a request is unity (as explained in the previous section), for the queue to be stable the arrival rate must be less than unity. Though it is not usually possible to control the arrival rate, this study considers λ to be less than unity (typically 0.3). There is no limit on the length of queue.

3.6 Simulation Runs

During the simulation, the storage system is modeled as S arrays, which represent the queues of each server, and requests arriving at one end and leaving from the other end when they are serviced. Since the inter-arrival time and the service time are exponential, the random values of the time are generated from the MATLAB random number generator. Every run consists of 5000 requests to average out the initial transient effects of the queue. Each run is repeated 10 times with different sequences of random numbers. The waiting time is measured and averaged over all requests in all 10 runs and the *mean sojourn time* is obtained.

CHAPTER 4

MDS and simple Repetition results verification

4.1 Theory

In this chapter, we verify and simulate the results in (6). The server system considered is such that $S = N$, i.e each server has only 1 piece after encoding. The paper also considers a *push model* with greedy scheme.

(a) MDS encoding

(b) Repetition encoding

Figure 4.1: Push Model with 4 servers and K = 2, N = 4
figure reference (6)

In the figure 4.1, we consider a case of 4 servers, i.e $S = 4$. The file stored is split into 2 pieces, i.e $k = 2$: *A* and *B*. In repetition case 4.1b, the pieces stored are *A, A, B, B*. In MDS case 4.1a, the pieces stored are *A, B, A+B, A+2B*. The piece size is assumed large enough to allow for MDS coding. Each server, in both cases, has data of the same size, i.e half of the original file size and hence the mean service rate of each server in both the cases is the same, $\mu = 1/2$.

4.2 Results

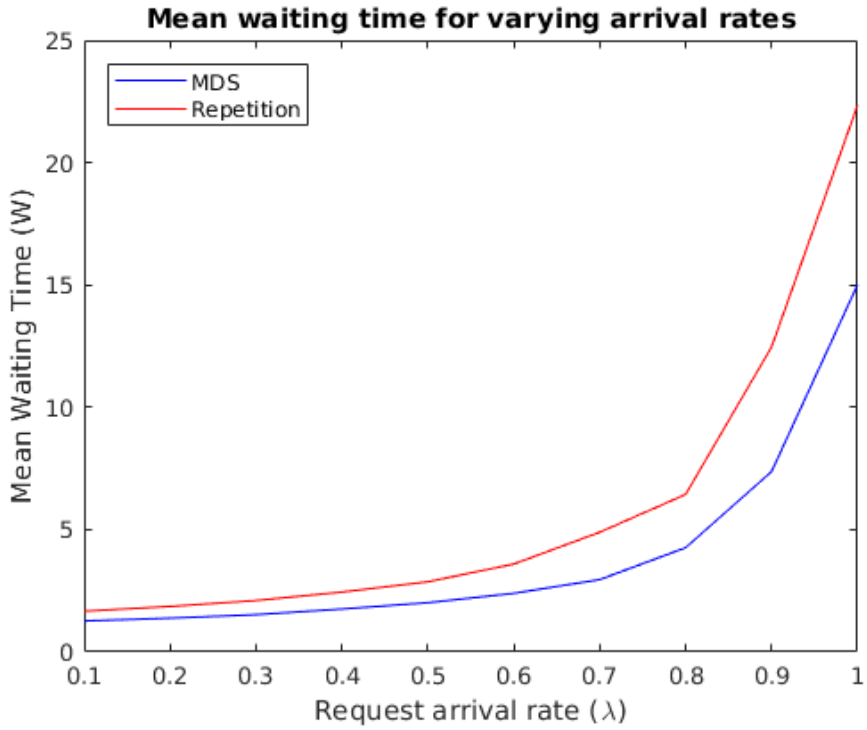

Figure 4.2: Variation of mean sojourn time of MDS and repetition encoding with λ

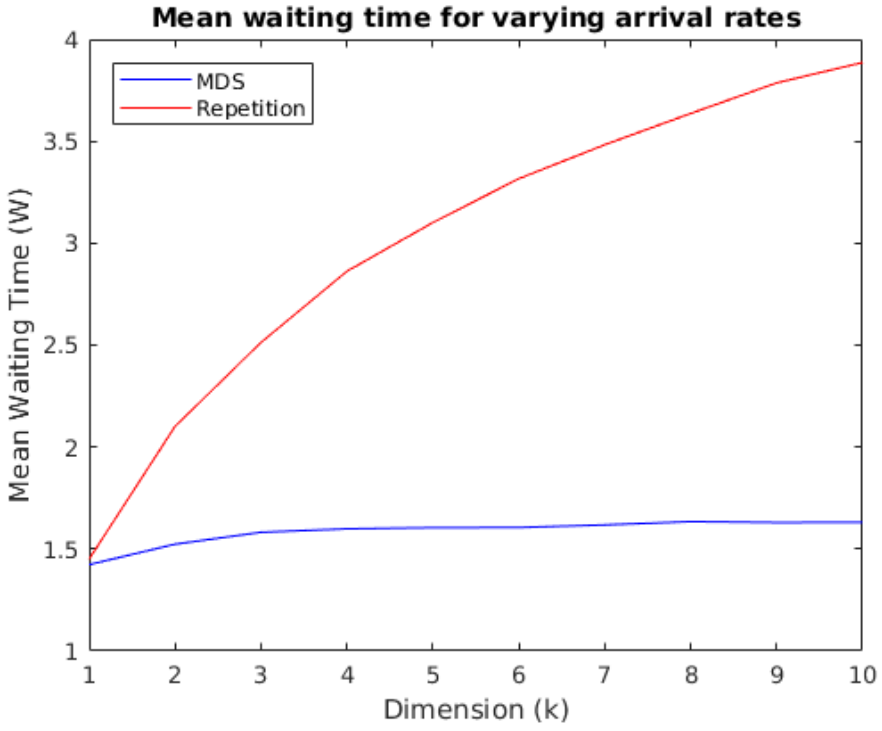

Figure 4.3: Variation of mean sojourn time of MDS and repetition encoding with N keeping coding rate = 0.5

A simulation was run based on the example case in 4.1. The arrival rate λ was varied from 0.1 till 0.95. The resulting values of mean waiting time, which is the mean sojourn time and which will be used interchangeably, are plotted in 4.2.
In a second simulation, λ was maintained at a constant value of 0.3, while the value of K was increased from 1 to 10. The value of N, and thereby S, was also correspondingly changed to ensure that the code-rate is 0.5. The resulting values of mean waiting time are plotted in 4.3.
The results obtained in the simulations closely match those in (6).

4.3 Observations and Inference

The MDS queue outperforms the repetition queue. This is because, in repetition queue, as more pieces are downloaded the information in other servers become useless. As the servers can no longer provide any useful information to the request, they no longer serve the request. This decreases the number of servers serving a request in a parallel manner, thus leading to a drop in the parallel gain. The MDS queue on the other hand has no useless servers as each server holds distinct pieces of information. Hence the parallel gain of the system holds always.

We aim to reduce this time difference in the following chapters. This is by increasing the non-recurring data in each server through the use of **partitions** in the servers. The proposed method maintains the parallel gain in the repetition encoding scheme.

CHAPTER 5

Partitioning

5.1 Definition

Partitioning is defined as the splitting of the storage capacity of each server into smaller pieces. Instead of storing one large chunk of data in a server, several smaller pieces can be stored. Each server is assumed to have the same capacity and are split into equal chunks. Each chunk of data is accessed in order but not necessarily continuously. Some pieces of data can be skipped and the next *useful* piece can be accessed.

Server	Layer 1
1	A
2	A
3	B
4	B

Table 5.1: Server pieces without partitions

Server	Layer 1	Layer 2
1	A1	A2
2	A2	A3
3	A3	A4
4	A4	A1

Table 5.2: Server pieces with two partitions

Here, we give an example of the data stored in a storage system with and without partitioning. In 5.1, the file is split into 2 pieces (A and B) each of size half the total file size. Each piece is repeated twice and stored across 4 servers. Hence, the storage capacity of each server is half the total file size. In 5.2,the file is split into 4 pieces (A1, A2, A3, and A4) each of size one-fourth the total file size. Each piece is repeated twice and stored across 4 servers in the permutation shown. Hence, the storage capacity of each server is half the total file size in this case.

5.2 Advantages

As we saw in the comparison of MDS queue and repetition queue 4.2, the speed of the repetition system decreases due to the occurrence of *useless servers*. This occurs as when one server offers its data, the data in other servers become redundant. To overcome this effect, multiple data can be stored in a server. In this way, even if one chunk of data in a server is redundant, the other chunks can still provide useful data. This method, thus, ensures that all the servers remain useful for most of the download time. Even after significant data has been downloaded, most of the servers still remain useful.

Let us consider the non-partitioning case in 5.1. If a request downloads piece A, servers *1* and *2*, become useless for the request. Hence the parallel gain of the storage system reduces to only 2 servers. Similar argument holds for the download of B.

In the partitioning case in 5.2, a download of 2 distinct pieces is equivalent to the download of A in the non-partitioning case. Let us consider pieces A1 and A2 are downloaded. This makes only server *1* useless. The other servers have atleast one useful piece of data and still hold a part in the parallel gain. If we consider pieces A1 and A3 are downloaded, **all** the servers are still useful.

Hence, we see that even though the number of servers, the data stored and the capacity of server are equal in both cases, the presence of partitions increase the number of useful servers after a few pieces are downloaded. This improves the latency performance of the queuing system.

5.3 Limitations

While partitioning increases the number of useful servers for the repetition queue, a server (or a file) cannot be partitioned indefinitely. Each piece of the file must have a header to identify the position of the piece in the original file. Increasing number of partitions increases the percentage overhead of headers. Increased partitions also increase the load on the central controller as mentioned in 2.

5.4 Simulation Results

In the simulation to demonstrate the gain due to partitioning, S is the number of servers, $K = p*S$ is the number of split pieces, N is the number of pieces after encoding. Since the encoding scheme is repetition, $N = m * K$ where $m \in \mathbb{N}$. The value of λ is assumed to be 0.3. The service rate $\mu = K/S$, as explained in 3, varies according to the parameters.

An example is given in 5.3, where we have considered $(S, K, N) = (5, 15, 45)$. Hence, we obtain $p = 3$ and $m = 3$. Here we see that each server stores 9 pieces of data, where each piece is $1/15$th of the original file. Hence, the storage size of each server is $9/15$ which is three-fifth of the file size. The data pieces are labeled from 0 to 14. and are arranged such that the first p layers contain the entire file data. The next p layers contain a group cyclic shift of the data in the first p layers. Similarly, the the 3rd set of p layers have a group cyclic shift of the data in the 2nd set of p layers.

Server	Lay. 1	Lay. 2	Lay. 3	Lay. 4	Lay. 5	Lay. 6	Lay. 7	Lay. 8	Lay. 9
1	0	5	10	4	9	14	3	8	13
2	1	6	11	0	5	10	4	9	14
3	2	7	12	1	6	11	0	5	10
4	3	8	13	2	7	12	1	6	11
5	4	9	14	3	8	13	2	7	12

Table 5.3: Group shifting

The simulations to validate the partition gain was run for 2 cases. In the first case, 4 servers are taken with a code rate of 0.5. The number of partitions(p) was linearly increased. The variation of mean waiting time with the change in p was plotted 5.1. A similar simulation was run with 5 servers and a code-rate of 0.6, the results of which was also plotted 5.2. The graphs show a decrease in the mean waiting time.

In another run, the repetition scheme and MDS scheme were run together to check the effects of partitioning. Here, 4 servers are taken with a code rate of 0.5. The results are plotted in 5.3. We see that both schemes show improvement, but the repetition scheme has a steeper decrease in waiting time.

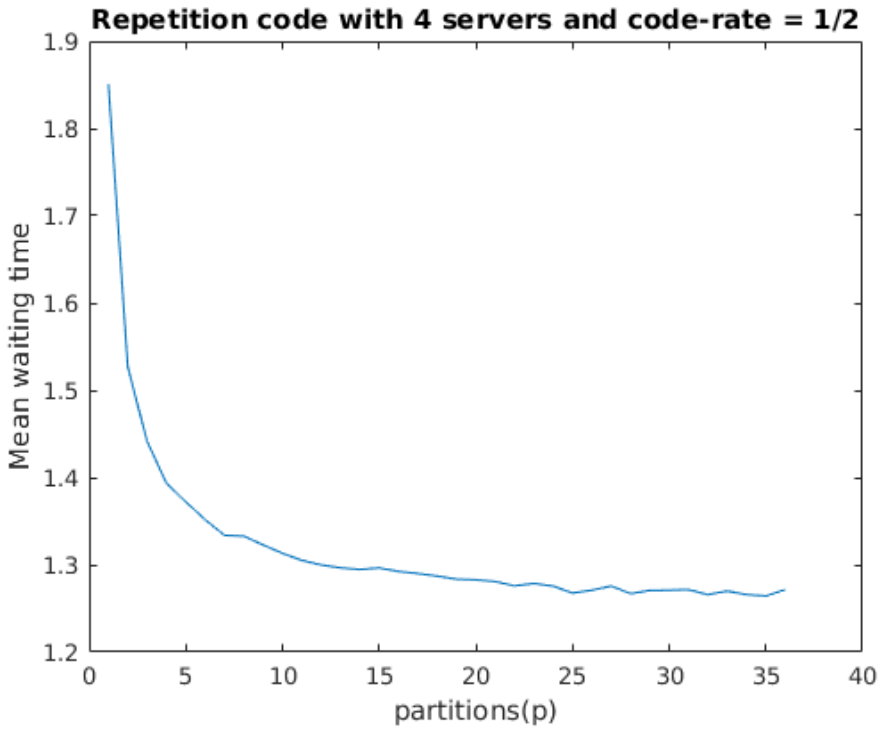

Figure 5.1: Variation of mean sojourn time of repetition encoding with partitioning for $S = 4$, $N/K = 2$

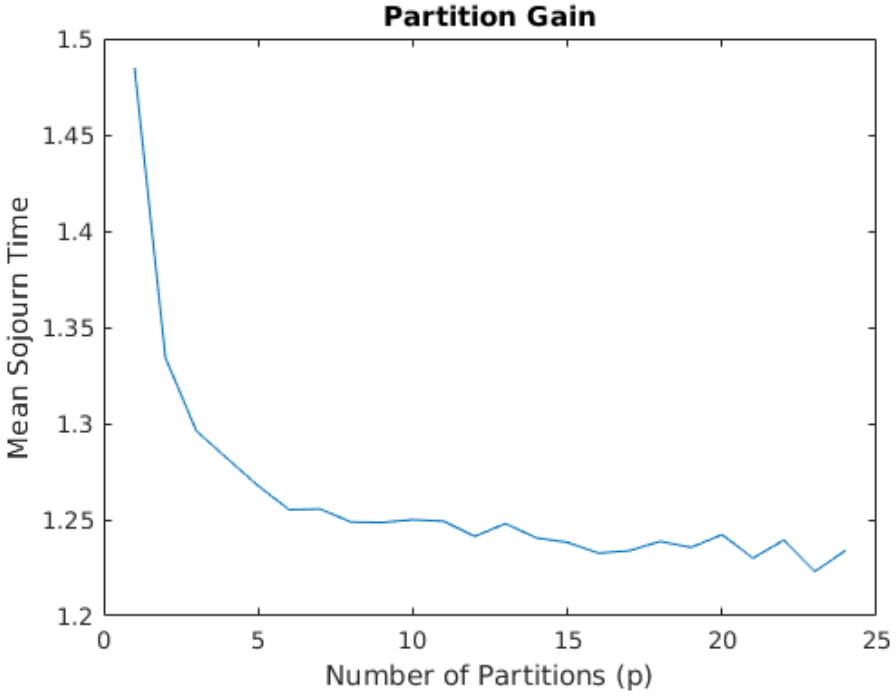

Figure 5.2: Variation of mean sojourn time of repetition encoding with partitioning for $S = 5$, $N/K = 3$

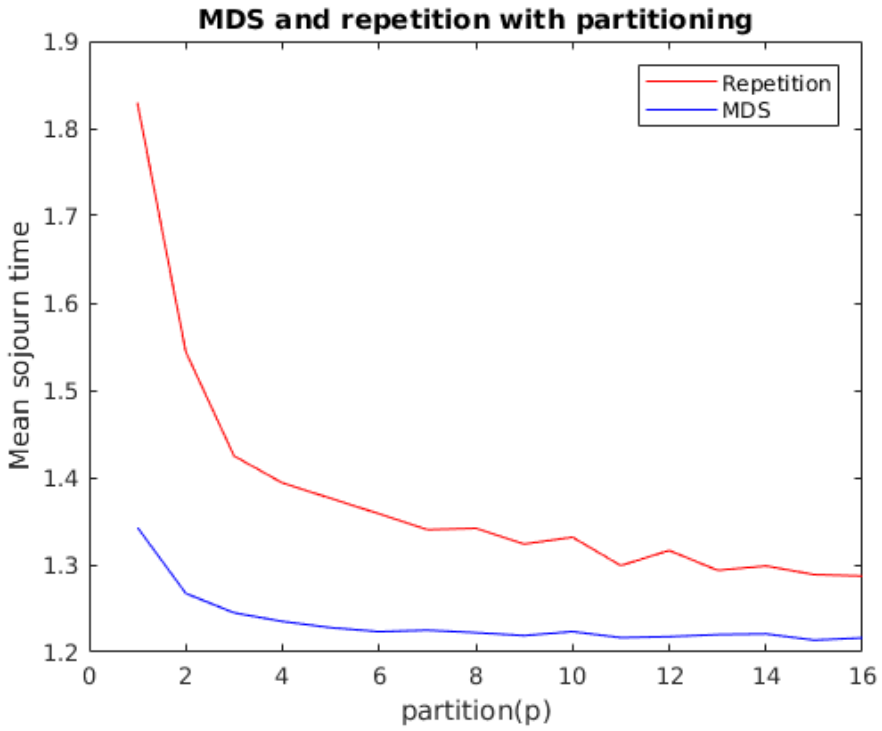

Figure 5.3: Comparison of mean sojourn time of repetition and MDS encoding with partitioning for $S = 4$, $N/K = 2$

5.5 Observation and Inference

The simulation results show that partitioning the servers improves the latency performance of the storage system. The mean sojourn time reduced by nearly 20% due to partitioning. If we consider the $(S, N/K) = (4, 2)$ case, the mean sojourn time of the repetition scheme reduced from 1.85 to 1.3 sec. If we consider the MDS scheme, the mean time reduced from 1.34 sec to 1.21 sec after partitioning. So we see that the difference in the download time reduces as we increase partitioning.

However, there is no significant improvement after a certain value of p. This value can be used to find the *optimal* p as increasing the partition has a penalty, as discussed in its limitations.

We have seen the improvement due to partitioning. However, the permutation of data used doesn't completely utilize the 2D nature of the storage system. It is possible to spread the data to a greater *distance*, within the same system. We now explain the merits of efficient data arrangement and propose a scheme for the same.

CHAPTER 6

Efficient Data Permutation

6.1 Theory

The aim of partitioning is to increase the number of useful servers. We can directly see that if each of the servers contain the entire file, there will be no useless servers. However, the study assumes that each server can store only a **fraction** of the total file and hence, the optimization must be done with this restriction.

In MDS scheme, all the pieces are distinct but in repetition scheme the pieces are repeated. Hence, beyond partitioning, the location of the pieces within the partitioned system is also vital for reducing latency.

6.2 Illustration

Server	Layer 1	Layer 2
1	A1	A3
2	A2	A4
3	A3	A1
4	A4	A2

Table 6.1: Second column is cyclically shifted twice

Server	Layer 1	Layer 2
1	A1	A2
2	A2	A3
3	A3	A4
4	A4	A1

Table 6.2: Second column is cyclically shifted once

In this illustration, we see that both the cases have identical parameters, including the partitions. However, they differ in the arrangement of the pieces. In the first case 6.1, the second column is a double cyclic shift of the first column. In the second case 6.2, the second column is a single cyclic shift of the first column.

In both the cases, we see that the download of a single piece doesn't decrease the number of useful servers. Let us consider that A1 is downloaded from the first server in

both the cases. However, when the second piece is downloaded, the useful server count depends on the piece which is downloaded. If we take A2 to be downloaded next, the number of useful servers in 6.1 is still 4, but the number of useful servers in 6.2 is only 3. On the other hand, if we take A3 to be downloaded next, the number of useful servers in 6.2 is still 4, but the number of useful servers in 6.1 is only 2.

6.3 Probability Calculations

It can be seen in the previous illustration that the position of the pieces can decide the number of useful servers. We see that $(A1, A2)$ gives 6.1 the advantage, while $(A1, A3)$ gives 6.2 the advantage. However, the probability of the sequence of download which leads to one data storage permutation being better than the other must also be considered.

If we consider the number of useful servers after i pieces are downloaded, for a particular download sequence, to be *n(i)* we get

$$T = \sum_{i=0}^{K-1} \frac{1}{n(i)} \tag{6.1}$$

$$P = \prod_{i=0}^{K-1} \frac{1}{n(i)} \tag{6.2}$$

where T is the average time of download and P is the probability of occurrence of that symbol sequence. Hence the mean sojourn time is given by

$$T = \sum_{c \in C} T_c * P_c \tag{6.3}$$

where C is the set of all possible download sequences, T_c, P_c are the average time of download and probability for download sequence c.

In order to minimize the average download time, the values of $n(i)$ must be increased. The logic of permuting the data must be optimal and must be applicable to a wide varieties of parameters (S, K, N).

6.4 Prime Cyclic shifts

6.4.1 Objective

The prime cyclic shifts is a proposed method to ensure that the data pieces are spread apart sufficiently to ensure that there is very little redundancy in a server and across servers in the same layer. This improves the performance of the queuing system in that most of the servers remain useful for most of the time needed to download the file. It is also a scalable method to include many values for number of servers (S), and code rate ($m = N/K$).

6.4.2 Principle

This method assumes that the S is a prime number, and the file is split into K pieces where K is an integral multiple of S, i.e $K = p * S$ where $p \in \mathbb{N}$. First, the original file is filed in the servers, first moving across the servers and then proceeding to the next layer of the servers. Then the redundant data, which are the duplicates of the original file are filled in the later layers.

The redundant data are also stored across the servers, first, and then across the layers of the servers. However, these layers are a cyclic shift of the original data arrangement. Within a *repetition set* (m), which is the set of server layers containing the entire data in a permuted logic, the first layer is cyclically shifted by $m - 1$ positions. The second layer is cyclically shifted by $2 * (m - 1)$ positions and so on till the k^{th} layer is shifted by $k * (m - 1)$ positions.

This logic works on the basis that the shift will never be a multiple of S as S is a prime number. Since the shifts are not a multiple of S, the same piece of data will not occur in the same server. This logic also ensures that 2 data pieces in the same server will be spread to different servers in different *repetition sets*.

Since the logic has a symmetric nature, most download sequences have the same $n(i)$. This ensures that the mean time is reduced for any order of downloads and not only for a target few possibilities.

6.4.3 Illustration

Server	Lay. 1	Lay. 2	Lay. 3	Lay. 4	Lay. 5	Lay. 6	Lay. 7	Lay. 8	Lay. 9
1	0	5	10	4	8	12	3	6	14
2	1	6	11	0	9	13	4	7	10
3	2	7	12	1	5	14	0	8	11
4	3	8	13	2	6	10	1	9	12
5	4	9	14	3	7	11	2	5	13

Table 6.3: Column shifting

In the illustration, we consider $(S, K, N) = (5, 15, 45)$. So we get $m = 3$ and $p = 3$. The file is split into 15 parts labeled from 0 to 14. We see that the first repetition set, which is the data in layers 1, 2 and 3, contains the entire file in the same order.

In the second repetition set (Layers 4 to 6), the fourth layer is a cyclic shift of the first layer by one position. The fifth layer is a cyclic shift of the second layer by 2 positions. Similarly with the sixth layer which is shifted by 3 positions.

In the third repetition set (Layers 7 to 9), the seventh layer is a cyclic shift of the first layer by 2 positions. The eighth layer is a cyclic shift of the second layer by 4 positions. Similarly with the sixth layer which is shifted by six positions.

Consider an example to check the performance of this permutation logic. If we assume that all the pieces in the first repetition set of the first 4 servers are downloaded by a request, we see that only pieces 4,9,and 14 are needed to complete the file download. From the illustration table we see that all the servers have atleast one of the piece needed making **all** the servers useful. The design of the servers ensures that the useless pieces are skipped and only the useful pieces are offered by the server. In this case, *server 1* offers the piece 4 from the fourth layer, *server 2* offers the piece 9, *server 3* offers the piece 14, *server 4* offers the piece 9, *server 5* offers the piece 4.

CHAPTER 7

Results

7.1 Partition Gain

The comparison between encoding schemes in (6) has shown that the MDS encoding performs better than the repetition coding. However, the usage of partitions in the structure of the servers has reduced, and nearly bridged, the time difference between the two encoding schemes as can be seen from 5.3. Increasing the partitions improves the performance of the queuing system significantly without incurring a large overhead in terms of header size.

7.2 Data Permutation

The prime cyclic shift method of data arrangement in the servers is shown to be efficient in spreading the data pieces to maximize the cardinality of the set of useful servers. The symmetric nature of the arrangement ensures that most data download sequences will have the same mean download time thereby reducing the mean time to download a file by a request as can be seen from *equation 6.1*.

CHAPTER 8

Future Work

8.1 Optimal Permutation

While the prime cyclic shift method works efficiently, it need not be optimal. The number of servers are also limited to prime numbers. Alternate permutation logics can be found to work with any number of servers.

8.2 Real World Analysis

While the analysis and the simulations are done to examine the performance of the schemes, the results do not show the actual values which are seen in the real world. This is because the values used for the arrival and service time are normalized. By using actual values of the distribution of the random time, an actual comparison can be carried out.

8.3 Total Time Comparison

The simulations illustrates the download time difference between the encoding schemes. To compare the total time, the decoding time must be added to the real world download time.

8.4 Optimal Partitioning

As discussed earlier, it is not advantageous to keep partitioning beyond a certain point. The parallel gain increases only marginally while the data overhead keeps increasing. Finding the optimal level of partitioning is needed for best performance in terms of latency and data overhead.

REFERENCES

[1] **Alexandros G. Dimakis, Y. W. C. S., Kannan Ramchandran** (). A survey on network codes for distributed storage.

[2] **Chen, S.** and **Y. Sun** (2014). When queuing meets coding : Optimal-latency data retrieving scheme in storage clouds.

[3] **Gauri Joshi, E. S., Yanpei Liu** (). On the delay-storage trade-off in content download from coded distributed storage systems.

[4] **Longbo Huang, H. Z. K. R., Sameer Pawar** (). Codes can reduce queueing delay in data centers.

[5] **Nihar B. Shah, K. R., Kangwook Lee** (). The mds queue: Analyzing the latency performance of erasure codes.

[6] **Parag, P.** and **A. Bura** (). Latency analysis for distributed storage.

[7] **Richardson, T.** and **R. Urbanke** (2008). Modern coding theory.

[8] **S. B. Balaji, M. V. V. R. B. S., M. Nikhil Krishnan** and **P. V. Kumar** (). Erasure coding for distributed storage: An overview.

[9] **Shokrollahi, A.** (2006). Raptor codes.