UNIVERSITY OF CALIFORNIA,
IRVINE

Low-Latency MapReduce

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF THESIS

in Electrical Engineering and Computer Science

by

Xiaoran Li

Dissertation Committee:
Professor Zhiying Wang, Chair
Professor Ender Ayanoglu
Professor Zhou Li

2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

My study experience at UCI has been rewarding, challenging, and full of fun. It is almost impossible for me to generate a complete list of the people to whom I want to extend my gratitude. I am not in the best time to meet all of you, but it is because of all you the time becomes memorable.

First, I would like to thank my thesis advisor Professor Zhiying Wang for her constant guidance and support. I appreciate her teaching of distributed storage problem in the very beginning. Without her, this work would never exist. Zhiying is not only a great mentor in science and research, but also a mentor in life philosophy. I feel lucky to join her group and learn from a broad range of research topics.

I also would like to thank Professor Paul Asimow, whom I believe is the most amazing researcher and best life mentor as dreamed. Since I first joined his lab, I have always learned from him about how to communicate and express my idea to others, and improve my work efficiency. One of my biggest reasons to do a master thesis is because he opened the door of research to me, encouraged me to do research and be a kind of person like him.

This thesis could not have been completed without generous help from my committee. I thank Prof Zhiying Wang for providing many great resources to help me learn and enjoy the topics. Thank Prof Ender Ayanoglu who introduced the field of communication to me in different classes from undergraduate to graduate school. Thank Prof Zhou Li to help me think about the security problem in the field of communication.

For my research, I benefited greatly from everyone in Zhiying's group, including Marwen Zorgui, Weiqi Li, Zhen Chen, Peng Fei, and Alireza Javani. Thanks for being so resourceful and helpful all the time. I do appreciate all the suggestions and comments on my work. I also appreciate Tianren (silver) Zhang and Yuxin Zhang for the help of software support and figure creating. Thanks to all of you, the road of research is not lonely and full of joy.

I am deeply grateful for Dr. Jin Yang for friendship, for sharing the passion for good food, and for inviting me to hiking. I thank Dr. Cheng Li for always sharing me the topics of the new technology and science. I thank Dr. Chun-lin Liu for bringing me from geology field to communication in Electrical Engineering, and for taking me to concerts at Walt Disney Concert Hall many times and re-encourage me to continue the research. I thank Yinan Ke, Kai Li, Xiyao Tang, Jiaqi Li, Yuan Zhong, Yuwei Xiao, Wei Zhao, Munan Li, Daming Dong, and Aiqiu Zhang for the life of joy. I would also thank who helped me in the past.

Finally, I thank my dad Prof. Maoqiang Song, my mom Lei Li, my sister Dr. Rui (Amber) Song, and my brother in law Dr. Rui Song for love and support over the past of my life. I would not have the chance to be here without their constant trust, love and support. I thank my grandparents Minggeng Li and Jiaorong Zhang for having taught me to be an honest person and showing me what a good engineer looks like by using themselves as examples. This thesis is dedicated to them.

ABSTRACT OF THE DISSERTATION

Low-Latency MapReduce

By

Xiaoran Li

Master of Thesis in Electrical Engineering and Computer Science

University of California, Irvine, 2019

Professor Zhiying Wang, Chair

We investigate the problem of MapReduce and coded MapReduce. MapReduce is a programming model for the website search engine. It has three phases: Map, Shuffle, and Reduce. Even though it is an extensively used tool for distributed computing, the communication time involved in the shuffle phase can be a bottleneck for the overall system delay. Coded MapReduce was recently proposed that trades replicated computation for shorter communication. In this thesis, we formally define the model and algorithm of MapReduce and coded MapReduce. Moreover, we implement them with identical Map and Reduce functions. The communication time and the overall system time is analyzed theoretically and measured experimentally. Finally, as an example, reverse indexing is implemented under MapReduce and coded MapReduce framework. We analyze and measure the delay performance for this case as well. We find that when each Map computation task is replicated twice, the coded MapReduce scheme is almost twice as fast as the naive MapReduce scheme.

Chapter 1

Introduction

The programming model designed in 2004 named as MapReduce enables the distributed processing for large amount of datasets on a cluster of commodity servers [6]. As we are living in a network society, everything is based on the computer. We expect immediate results once we enter a word in the Google search bar. To save time when they are traveling and waiting in a line for the cashier, people start shopping online. Therefore, companies like Alibaba, Amazon, eBay become popular due to their high convenience. The main reason why people have chosen online shopping is that they do not need to wait for a long time and wait for their order delivered to their homes. The server for those online markets they use is called distributed file system. In 2003, Google developed the first distributed file system named global file system (GFS)[2] by using MapReduce. Three years later, another company, Oracle, developed a software named Hadoop[1]. Based on Hadoop they created the second distributed file system called Hadoop Distributed File System (HDFS). Another idea of the distributed file system is called Ceph, the idea was designed at the same year of 2006 [7]. Each distributed file system works in different areas, but the core of those distributed file systems is MapReduce. MapReduce is well designed in the HDFS, used in Amazon Web Services and then developed in the Ceph system later in industry. One of the main reasons

of the popularity of MapReduce is it provides a convenient programming framework for distributed computing. For example, it automatically handles server failures, as well as the distribution of input files and the computation tasks among the servers.

MapReduce, by its name, includes two computation steps which are mapping and reducing; In between these two computation steps, data is exchanged among the servers, called the Shuffle phase. Mapping is to find a way to uniformly distribute stored files into different users. After files are mapped into different users, these stored filed are sent (shuffled) to receiver users, so that the Reduce step can be performed at the receiver.

Word counting is one of the most commonly used example for MapReduce [6]. Consider a book with 12 chapters, and we are interested in obtaining the number of occurrences of letters A, B, C, D in the book. Assume that we have 4 servers to complete this task. We assign 3 chapters to each server in the beginning. In the Map phase, each server counts the numbers of appearances of A, B, C, D in the assigned chapters, respectively. In the Shuffle phase, the counts of letter A are transmitted to server 1. The counts of letter B are transmitted to Server 2, and so on. Each server receives 3 counts from the other servers regarding its letter, and it also has 1 count from its own computation in the Map phase. In the Reduce phase, server 1 aggregates the counted numbers and gives the overall number of occurrences of letter A. And similarly each of the other letters is aggregated at one of the servers. The MapReduce framework makes it easy to write the computation application. In particular, one only needs to program the Map function (counting the letters in the assigned chapters), and the Reduce function (aggregation of the letters from different chapters). The assignment of the tasks to available servers, the shuffle phase, and the handling of possible server failures are all done invisibly to the application.

However, the shuffle phase may correspond to a large communication delay, which drastically affect the overall speed of MapReduce. As a result, the coded MapReduce was proposed. The coded MapReduce is a way to communicate the information from one user to others[2].

The communication time is very important due to the speed of the search engine[5]. The idea of coded MapReduce is inspired by the result of cache network[8]. To reduce overall delay, it is important to design an efficient shuffle phase algorithm. Key idea: trade computation for communication. Each original file is replicated in r servers, and the Map phase is thus computed r times compared to the original naive MapReduce. As an advantage, the communication time in the shuffle phase can be improved by a large factor, depending on r, using ideas similar to index coding. Another important assumption is that, the server can broadcast or multicast to many other servers. Under this assumption, the communication cost for multicast a file is the same as unicast. In particular, we assume that to transmit a file to multiple users takes the same time (or obeys the same distribution) as to transmit to a single user. Under such assumption, the coded MapReduce scheme allows several receivers to decode their desired file in just one transmission. Hence the overall communication cost is substantially reduced.

For example, We have three servers named as server 1, server 2 and server 3. The server 1 contains all the information that server 2 and server 3 want. If server 2 wants file A and server 3 wants file B because server 2 contains file B and server 3 contains file A. Then both servers as receiver request server 1 as transmitter to send file A and file B to them. The naive scheme is server 1(transmitter) based on other servers needs and sends a different file to both server 2(receiver) and server 3(receiver). The request files are different from both servers. Server 1(transmitter) needs to send the file one by one. If the single file transfer cost t unit of time, the naive scheme would cost $2t$ for total file transfer. Instead of transfer two different files the coded scheme only need to transfer one file, by doing A $\oplus$ B where $\oplus$ means $'XOR'$ for file A and file B. After compress the file then send the new file to the server 2 and the server 3. The server 2 and the server 3 can decode the new file by using what they already have. The server 1(transmitter) transferred one file to both server 2(receiver) and server 3(receiver). The communication cost is t time which is half time of the naive scheme.

This thesis focuses on two schemes of shuffling: the naive scheme and the coded scheme with $r = 2$. The naive scheme is to send the file one by one, as in the original MapReduce framework. The coded scheme, as explained above, is to combine two files into one file and send the file simultaneously to the users who need the file. After the new users receive the combined file, they will decode the receiving file to get the file they have expected to receive.

MapReduce is applied in many different areas, such as word counting, GPS coordinates. In particular, the reverse index is an important one because it is essential in search engines. Define reverse index first finds all the source links and it's target links in a domain. Then reverse links from (source, target) to (target, source). By using the MapReduce method to transfer the file which contains links information to different users by their request. As an illustration of coded MapReduce, we will investigate the coded reverse index problem, and measure the corresponding performance.

The MapReduce of distributed storage will be more thoroughly in Chapter 2. The summary of communication cost for MapReduce through different communication parameter is stated in Chapter 3. Chapter 4 explains the detail about how to accomplish the reverse index problem and summary the result in Chapter 5 for different communication parameters. Finally, Chapter 6 concludes the thesis with a summary and future directions.

Chapter 2

MapReduce and Coded MapReduce

The MapReduce programming model designed by the Google engineer Jeff Dean for multiple purpose of website search engine[2]. The MapReduce has three phases: Map, Shuffle, and Reduce. In this chapter we implemented our own program based on the Coded MapReduce paper[6] which inspired from the cache network[8].

2.1 Problem Setting

In this section, we explain the phases of naive(MapReduce) scheme and coded MapReduce scheme. In particular, for the case of $r = 2$ replication factor of each Map task, we explain the number of files to be transferred during the Shuffle phase.

We start by describing the last phase, Reduce phase. Let Q be the number of Reduce functions to be computed, which corresponds to the number of final results. In this work, we assume that $Q = N$. Each server is responsible to compute $Q/N = 1$ Reduce functions.

Now we move to the first phase, Map phase. The input data is first split into many files, to

be assigned to the servers. In the Map phase, a map computation is performed at the server for each of its input files, resulting in the output of Q intermediate files. Each intermediate file is used for one reduce function in the last phase.

In the Shuffle phase, the intermediate files are communicated between the servers. In particular, server i desires all the intermediate files related to its reduce function.

The key in the coded MapReduce is that, each input file is replicated in $r = 2$ servers. As a result, the Map phase has twice the computation load, but through index coding and multicast, the Shuffle phase takes only half the communication time, compared to the naive MapReduce. Some details of the phases and choices of parameters are explained below.

2.1.1 Step 1: Map.

To start the MapReduce execution, the very first step is to have a master split the input data into files among the available N servers. We set the total amount of files to be $N^2(N-1)$. We assume that each file is replicated in two servers. The file assignment is symmetric among the servers. Thus each server has exactly $2N(N-1)$ files. Namely, each server needs to perform $2N(N-1)$ Map tasks. Note that in our implementation, each Map task is replicated twice in the naive scheme and the coded scheme. However, a different naive implementation can be made such that only one Map task is performed for each input file. The corresponding communication cost would be different from our implementation, and is left as a further direction.

2.1.2 Step 2: Shuffle.

The server needs to obtain intermediate files related to its own Reduce function. Note that each server already has cached $2N(N-1)$ desired intermediate files from its Map phase, and

the total number of intermediate files is $N^2(N-1)$ The total amount of file for transfer is shown in the equation 2.1.

$$f_{total} = N^2(N-1) - 2N(N-1) = (N-1)(N^2-2N) \tag{2.1}$$

From the equation 2.1, the total amount of file is waiting for transfer. For the naive scheme each user will transfer a total amount of $(N-1)(N-2)$ files. For the coded scheme, each transmission corresponds to two desired intermediate files at two receivers, respectively. Thus we will transfer a total amount of $\frac{(N-1)(N-2)}{2}$ files as expected. The equation can be written as (2.2)

$$M = \begin{cases} N(N-1) & Naive \\ \frac{N(N-1)}{2} & Coded \end{cases} \tag{2.2}$$

2.1.3 Step 3: Reduce.

The focus on reducing task is to evaluate the final output in a distributed fashion, from the intermediate files. In our formulation, we assume that the number of Reduce functions is $Q = N$. Thus, each server outputs 1 final result.

2.2 Implementation

As the primary goal of this research, the transmission time is very important. Hadoop is well designed for MapReduce and easy to transmit all files to the destination, but finishing the transmission does not mean the ability to control each transmission time. Moreover, it does not allow the control of the shuffle phase. In particular, one cannot determine which Reduce jobs are performed at which receiver servers. However, as explained, in coded MapReduce,

we need to have precise control the transmission process in terms of the transmitters and receivers. As a result, a new python implementation of MapReduce is developed from the scratch and the simulation is performed on one local machine.

2.2.1 File Creator

Assume there are multiple users in this design where subfiles(N) are sufficiently large. N is a variable depending on the need. The goal is then to find the relation between the overall delay of different amount of users and the delay for a single user.

Assume the four users are searching on websites to find what they need. Through the equation 2.1 it is known that the total amount of files for transfer is 24. First generate $N(N-1)$ files and store them into the master. Set Filename A=$\alpha\_\beta\_\gamma\_\theta$ for all the filename in the master folder where file A is located at folder α,A(map1), and folder β,A(map2). The range of α, β is $[1, 2, 3, \cdots N]$. γ represents the copy of the file A(copy), so the range of γ is $[1, 2]$. θ in the filename stands for the folder expected to receive the file from folder α and folder β denoted as A(reduce). Thus θ has the same range as α and β.

Because the file is stored at two different locations, the file needs to have $\binom{N}{2}$. For each file, two copies with N types are needed. Thus, the master server needs to create $\binom{N}{2} \times 2 \times N$ files.

2.2.2 Mapping

Assume $\binom{H=gN}{pN}$ for some integers g. The master controller partitions the subfiles into $\binom{N}{pN}$ equal-sized subsets, where each subset contains g unique subsets, called subfiles. For the motivating example, $Q = N = 4, pN = 2$, thus we have $g = 2$. Every 2 servers are assigned 2 unique category.

2.2.3 Map Tasks Execution

Desire to have same amount of files in each user. Thus, after map tasks are assigned, each server starts to map all of its assigned subfiles simultaneously. Assume that the times the N servers spend mapping their assigned subfiles are i.i.d. across servers and subfiles. Thus the probability that any subset of rN servers in a finished mapping Subfile n by the end of Map tasks execution. Before proceeding the reduce tasks define an important quantity H_u where $H_u \in \{1, 2, 3, \cdots, k\}$.

Naive Scheme

During the shuffling phase, for the naive scheme each server(folder) contains $(N-1)(N^2-2N) = 24$ files if there are 4 users. Each of the files is shared by different servers. After the shuffling phase each folder should contain $(N-1)(N^2-2N)+(N-1)(N-2) = 30$ files including the file not deleted in the server. The reason why there are 6 increased files is because the original folder has $H \times N$ files, where $H = 6$ and $N = 4$. The total amount of files is $N^2(N-1) = 48$. Thus, we can find $H_{total} = 12$. For each of the transfer, we are assuming the model based on long distance travels. The communication cost from one server to another server for each file is 0.1 second.

Coded Scheme

Similar to condition in the naive scheme, all the servers in the coded scheme initially transfer a total amount of $(N-1)(N^2-2N) = 24$ files. During the shuffling phase, each server first creates additional $M = \dfrac{(N-1)(N-2)}{2}$ files which are the combination for mutil-servers request called encoding. After the encoding steps, each server transfers those files into two new servers. The new servers should receive H amount of new files from different users. All

the received file can be decode at each server and generate the dream files which are another H amount of new files, then added to the list called this step as decoding.

2.2.4 Data Shuffling(Encoding)

This section will discuss in detail how the files are transferred from one user to another user. Assume there are two files named file A and file B. File A is named as $\alpha\_\beta\_\gamma\_\theta$, knowing that in general $\alpha \neq \beta \neq \theta$ and $\gamma = 1$ or $\gamma = 2$. Section 2.2.1 has shown that file A exists at location α and location β. Another file B is named as $\alpha\_\theta\_\gamma\_\beta$ and located at location α and location θ.

For the naive scheme each server will check the last part of the filename. Thus file A will transfer to location θ and file B will transfer to location β and both of them will be transferred from locationα.

Unlike the naive scheme, the coded scheme will not use brute force to transfer the file. In the coded scheme, both file A and file B are stored at location A. First, compare the value of β and θ. If $\theta > \beta$, then create a new file named as B__A($\alpha_\theta_\gamma_\beta$__$\alpha_\beta_\gamma_\theta$). Otherwise, the filename will become A__B($\alpha_\beta_\gamma_\theta$__$\alpha_\theta_\gamma_\beta$). Name the new filename as file C, and then use brute force to transfer the file C to locationβ and location θ and each transfer cost is 0.1 seconds. The naive scheme will spend 0.1 second per file as the coded scheme, but the coded scheme has a fewer amount of files to transfer. Thus, as shown in the result, the communication cost of the coded scheme should be half of that of the naive scheme.

2.2.5 Data Shuffling(Decoding)

The naive scheme uses brute force to transfer files, regardless of the decoding process. By comparison, the coded scheme needs the decoding process because the coded scheme does not use brute force method to transfer files. In this process, user β and user θ receive file C from userα. As mentioned in section 2.2.1 file C is located at both locations. So for file A located at userβ also located at userα. Luckily, the user β wants the last bit is β. After receiving file C, user β can decode file C by using file A to get file B through the $'XOR'$ method.

2.3 Algorithm

This section will talk about the algorithm implemented for the MapReduce problem. As mentioned previously. MapReduce has three steps file creating, mapping, encoding, and decoding, but in the actual implementation, we have another step called "shuffle" showed in the figure 2.2. This step is the key step that to generate all the possible pairs for the filename to use for the encoding and decoding which save lots of waiting time to find the file name to do encoding and decoding.

The first steps as known are the file creating which describe more in section 2.2.1. This step, based on the total amount of user(N) ,worker_num in the figure 2.4, to create $N^2(N-1)$ amount of files.

Then based on the filename send all the files to different user location as expected. The pseudo code mentioned in the figure 2.2.

After all the pre-process finished, the encoding for the two schemes will have different pseudo code as showed in figure 2.4 for coded scheme and figure 2.3 for naive scheme. For the both

```
procedure FileCreate:
    for i, j, l ∈ {1, 2, … worker_num}, k ∈ {1, 2} and i ≠ j ≠ l :
        Create File(Concatenate(i, j, k, l))
end procedure
```

Figure 2.1: Based on section 2.2.1, this is the the process for file creating

```
procedure Shuffle:
    Collection = {[i, [j,k for j, k in Random(worker_num) and k ≠ j ≠ i ]] for i in {0, 1, …
    worker_num}}
    return Collection
end procedure
```

Figure 2.2: This is filename generator to help encoding and decoding to find the name of the file

scheme mentioned in section 2.2.3.

```
procedure Encoding(Collection)      [for Naive Scheme]
    merged_files = empty array of files
    for each pair in Collection do:
        file_1, file_2 = {files with the name containing both pair.first and pair.second}
        rec1, rec2 = get_receivers(file_1), get_receivers(file_2)
        Send File(xor_file, rec1)
        Send File(xor_file, rec2)
end procedure
```

Figure 2.3: Based on section 2.2.3, this is the the process for the encoding steps' naive scheme

As known that for MapReduce the coded scheme need the decoding process which described more in section 2.2.5. The algorithm stated in the figure 2.5.

```
procedure Encoding(Collection)      [for Naive Scheme]
        merged_files = empty array of files
        for each pair in Collection do:
                file_1, file_2 = {files with the name containing both pair.first and pair.second}
                rec1, rec2 = get_receivers(file_1), get_receivers(file_2)
                Send File(xor_file, rec1)
                Send File(xor_file, rec2)
end procedure
```

Figure 2.4: Based on section 2.2.3, this is the the process for the encoding steps' naive scheme

```
procedure Decoding:
        xor_files = Receive Files()
        for Xor File f1_f2 in xor_files:
                file = file {file.name = f1 or file.name = f2}
                decoded_file = Create File( Xor(xor_file, file))
end procedure
```

Figure 2.5: Based on section 2.2.5, this is the the process for the decoding

Chapter 3

Communication Cost Analysis for MapReduce

In this chapter, we measure the different condition via communication parameter changes. From Chapter 2, the expectation of the result of communication cost for coded MapReduce is half of the cost for MapReduce for the case of $r = 2$ replication factors of each Map task.

The simulation is performed on the local machine. As a further direction, to simulate real condition one can potentially use amazon EC2[6] or add time delay in the program and run locally. Amazon EC2 can give a real feedback of the condition for the file transfer, but chosen time delay on the local machine can change the time various via different condition and the reduce the unknown condition. For the experiment the less condition changes the easier we can get a result. In our simulation, we use the pause function of python to emulate the communication time between computation nodes. Local pause time is chosen to be fixed time, exponential distribution, or uniform distribution. Moreover, to emulate the constant overhead that may occur during communication, we also simulated with the pause time to be a constant plus a exponential or uniform random variable, which are called the

practical exponential distribution and the practical uniform distribution, respectively. For each condition, we set the mean pause time as 0.1 second/file. We measured the average communication cost, and the maximum communication cost among the transmitters. The maximum communication cost represents the overall system time spent on communications. We denote by i the server (transmitter) index and j the index of the file given a server. To complete the experiment three random variables are in the list of consideration which are single file transfer time (X_{ij}), total file transfer time for a single user (Y_i), overall transmission time (Y_{max}). Moreover, we consider the expectation of the file transfer time for each user (EY_i), and the expectation of file transfer time for all the users (EY_{max}). As stated, the random variable X_{ij} are chosen to be constant, exponential, and uniform, respectively. The mean is set to be

$$EX_{ij} = 0.1. \tag{3.1}$$

The relation between Y_i and # users showed in the (3.2). The meaning of the Y_i is for each file from a user cost Y_i seconds:

$$Y_i = \sum_{j=1}^{N} X_{ij}. \tag{3.2}$$

Overall, having N files to transmit at transmitter i. The expression for the overall communication time among the users can be expressed as (3.3),

$$Y_{max} = \max\{Y_1, Y_2, \cdots, Y_N\}. \tag{3.3}$$

In order to evaluate the performance, we consider the expectation of the average and the overall communication time. The expectation of the communication time for user i can be

presented in the (3.4).

$$EY_i = NE(X_{ij}). \tag{3.4}$$

The expectation of the total amount of communication time can be represented as (3.5).

$$EY_{max} = \int_{-\infty}^{\infty} x \times f_{Y_{max}}(x)dx \tag{3.5}$$

Here $f_{Y_{max}}$ is the probability density function of Y_{max}. We note the following relation:

$$\begin{aligned} f_{Y_{max}} =& P((X_{max} < x)) = 1 - P(X_{(n)} > x) \\ =& P(X_1 < x, \cdots X_n < x) \\ =& P(X_1 < x) \cdots P(X_n < x) \\ =& Y_{max}(x)^n \end{aligned}$$

3.1 Fixed Communication Time

In our fist setup, the communication time is set to be the constant 0.1 seconds for each file. For table 3.1 and table 3.2 are the result of the naive scheme and coded scheme for MapReduce problem. The first row of the table which lead by #users means N servers(users) are handling the problem of MapReduce. The second row #files/user stands for how many files that each user will transfer to other users. The row of EY_i stands for the theoretical value of the communication cost via the total amount of file will do the transfer work for each user. The row for EY_{max} is based (3.5) to find the maximum of communication cost for different amount of users. Next row is for every user when dealing with file transfer how would take

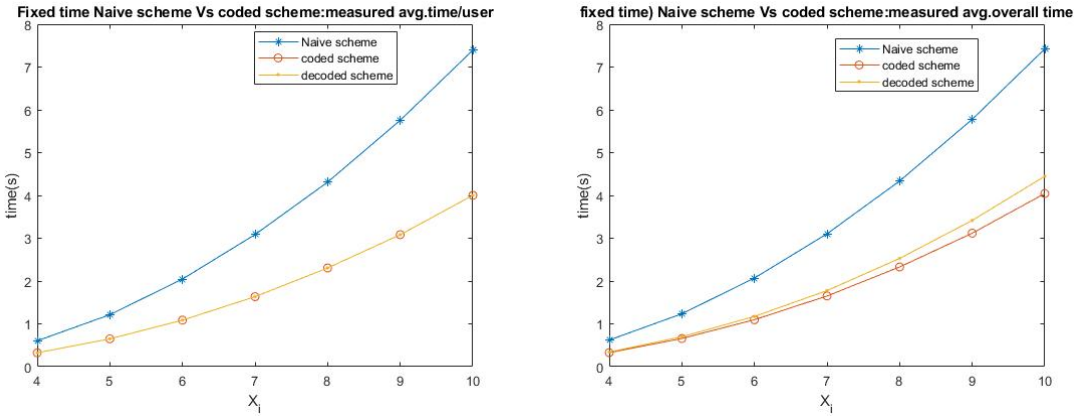
for N users. the last row is for the server which did longest time. We choose the number of users to be from 4 to 10. For a given number of users, the experiment is done for 350 times, and the measured result is averaged over these times. By viewing the tables, we can find the theoretical communication cost for naive is twice as coded scheme. $EY_i = EY_{max}$ when the pause time is fixed. After measured the actual condition the coded scheme is twice as naive scheme as well. Thus we draw the plot as showed below figure 3.1 where compared between the naive scheme and coded scheme. The figure 3.1a is the comparison graph between naive scheme and coded scheme for every user how long it cost for the communication. The figure 3.1b is the overall time comparison between naive scheme and coded scheme. As result, the ratio between naive scheme and coded scheme is 1.85. Because computer has threshold so the communication cost does not reach the factor of two.

# users	4	5	6	7	8	9	10
# files/user (N)	6	12	20	30	42	56	72
EY_i	0.6	1.2	2	3	4.2	5.6	7.2
EY_{max}	0.6	1.2	2	3	4.2	5.6	7.2
Measured avg. time/user	0.6	1.21	2.05	3.09	4.31	5.76	7.39
Measured avg. overall time	0.61	1.23	2.07	3.1	4.34	5.78	7.42

Table 3.1: Naive Scheme fix time condition for MapReduce.

# users	4	5	6	7	8	9	10
# files/user (N)	3	6	10	15	21	28	36
EY_i	0.3	0.6	1	1.5	2.1	2.8	3.6
EY_{max}	0.3	0.6	1	1.5	2.1	2.8	3.6
Measured avg. time/user	0.32	0.65	1.09	1.63	2.31	3.08	4
Measured avg. time/user with decoding	0.32	0.65	1.09	1.63	2.31	3.08	4
Measured avg. overall time	0.33	0.66	1.1	1.65	2.33	3.12	4.05
Measured avg. overall time with decoding	0.34	0.7	1.17	1.78	2.53	3.42	4.45

Table 3.2: Coded Scheme fix time condition for MapReduce



(a) Measured avg. time/user (b) Measured avg. overall time

Figure 3.1: Fixed Pause time(0.1 seconds)

3.2 Exponential Distribution Pause Time

We desire to find what happens if we change the pause time to exponential distribution after testing the fixed communication time as showed in section 3.1. To compute the mean for exponential distribution by using the probability density function(PDF) written as (3.7) and the mean of the equation is λ^{-1}. To keep consistency from section 3.1 the, mean should be equal to 0.1 and thus $\lambda = 10$.

We follow [10] to generate an experiential random variable. Let U be a uniform random variable distributed between 0 and 1. Then it can be shown that the random variable below is exponential with parameter $\lambda = 10$:

$$X = -\frac{\ln(U)}{10}. \tag{3.6}$$

In particular, the probability density function of X is

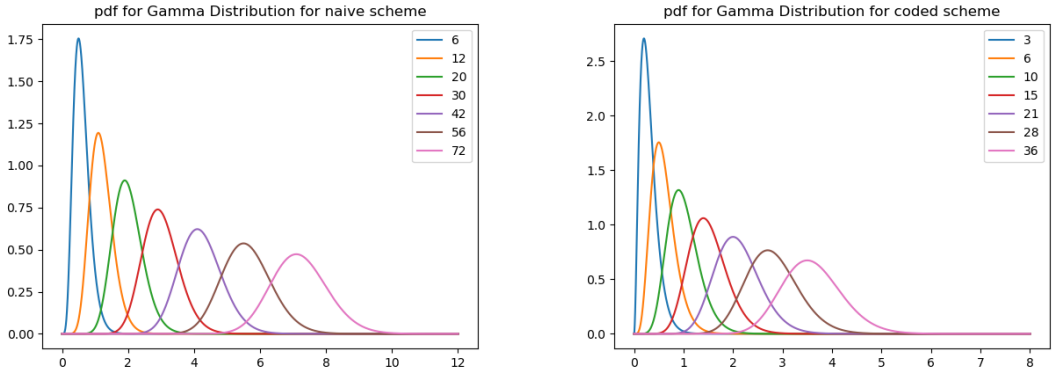
$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{3.7}$$

Because the simulation is not transfer one file, for each user will transfer multiple files at one time. Before discuss multiple users, first discuss the distribution if single user transfer multiple files to another user. The sum of N exponential random variables is a Gamma random variable, $\Gamma(\alpha, \beta)$, where $\beta = \frac{1}{\lambda}$. The λ is the parameter of exponential distribution α and λ can be found due to β and the mean of the exponential distribution equation[3]. The mean of the exponential distribution equation is the theoretical time, and β is the 0.1 which helps us to find the gamma distribution $\Gamma(M, 0.1)$ where M is the total amount of files that will transfer to other users in (2.2). The possibility of density function(PDF) can be expressed as (3.8) where $\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-\alpha} dx$.

$$f_{Y_i}(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \tag{3.8}$$

Based on the (3.8) one can plot the figure 3.2. The legend in each graph is the amount of file per each user (N) for the transfer. The number of users varies from 4 to 10. The left figure 3.2a is the naive MapReduce and the right figure 3.2b is coded scheme for exponential distribution. The legend on the figures are present due to each scheme how many files are transfer from one user to another user. The x-axis stands for the time in seconds and y-axis is the density corresponding the x-axis. As the figures to present that the coded scheme is faster than the naive scheme.

After we find the PDF of the gamma distribution then we look for the cumulative distribution



(a) Naive Transmission PDF Graph (b) Coded Transmission PDF Graph

Figure 3.2: The above two figures are the PDF for gamma distribution based on equation 3.8.

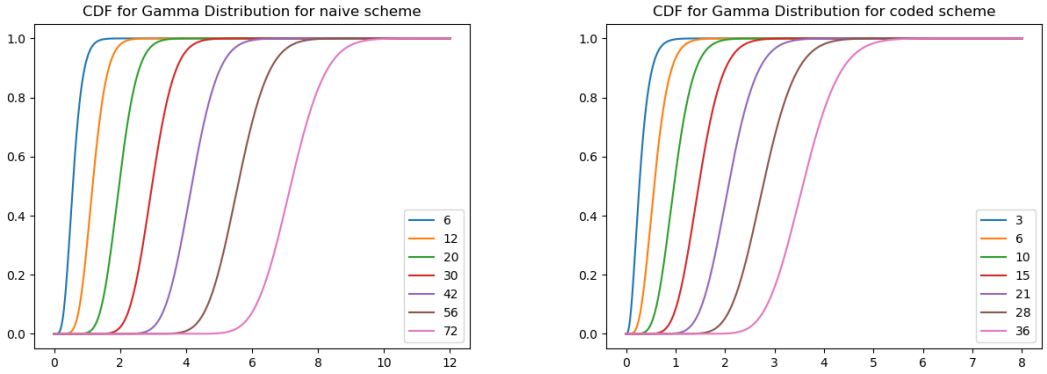
function ($F_{Y_i}(x)$) that can write as (3.9) where $\gamma(\alpha, \beta) = \int_0^x t^{\alpha-1}e^{-\alpha}dt$.

$$F_{Y_i}(x) = \int_{-\infty}^{x} f_{Y_i}(x) = \int_{-\infty}^{x} f_{Y_i}(U)dU = \frac{1}{\Gamma(\alpha)}\gamma(\alpha, \beta) \tag{3.9}$$

Due to (3.9) can plot the figure 3.3 the legend is the total amount of file during the transfer from one user to another user. The x-axis stands for the time and y-axis is the sum of the function from 0 to the current position's height. From the graph found the slope for coded transmission process are rising faster. Similarly, the left figure is naive scheme and right figure is coded scheme.

Theoretical max pause time (EY_{max}) is different with theoretical average pause time unlike the case with constant communication time per file. Let function of the distribution be independent and identically distributed (iid) then the density of f(x) is given by (3.5), After finding the CDF function of Y_i, one can find the CDF $F_{Y_{max}}$ for the maximum communication time. The $F_{Y_{max}}$ can be expressed as (3.10) where N present as number of users.

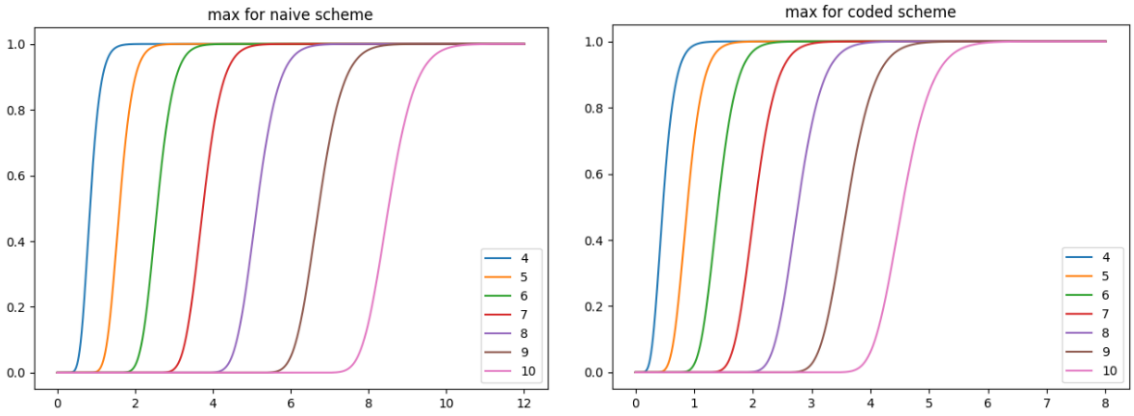
$$F_{Y_{max}} = F_{Y_i}(x)^N = (\frac{1}{\Gamma(\alpha)}\gamma(\alpha, \beta))^N \tag{3.10}$$



(a) Naive Transmission CDF Graph (b) Coded Transmission CDF Graph

Figure 3.3: The above two figures are the CDF for gamma distribution based on (3.9).

Then take the derivative for the (3.10) can generate a figure which showed in figure **??**. The coded scheme is faster than the naive scheme from the figure. The legend represents the number of users, M. X-axis is the time as seconds, y-axis



(a) Max function for the naive scheme (b) Max function for the coded scheme

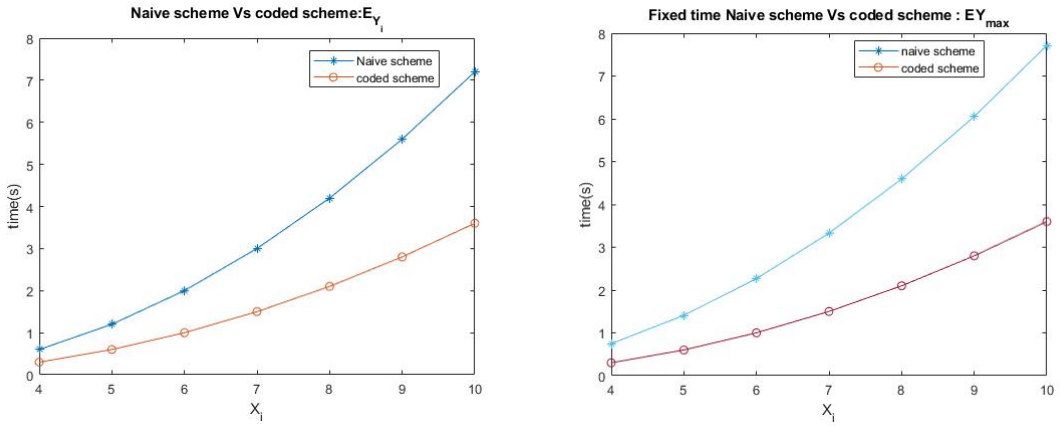
Figure 3.4: The above two figures are the probability density of the maximum variable among M gamma random variables based on 3.10.

Thus the expectation of the max transfer time calculate by take the gradient by (3.10) and times x, and integrate from 0 to ∞.

$$E_{Y_{max}}(x) = \int_{-\infty}^{\infty} x f_{Y_{max}} dx = \int_{-\infty}^{\infty} x \frac{dF_{Y_i}(x)^N}{dx} dx = \int_{-\infty}^{\infty} xN f_{y_i}(x) F_{Y_i}(x)^{N-1} \tag{3.11}$$

The $E_{Y_{max}}$ can be found in the fourth row at both table which are table 3.3 and table 3.4. The plot showed in the figure 3.5b.

After calculate the $E_{Y_{max}}$ it is time to compare between the E_{Y_i} vs. $E_{Y_{max}}$, from the section 3.1 knowing that the E_{Y_i} for any of the distribution same as the $E_{Y_{max}}$ at fixed pause time condition. Thus, comparing the figure as shown figure 3.5. The legend represent the scheme of chosen, x-axis represent the amount of users and y-axis is the time to spent in second.



(a) fixed time for $E_{Y_{max}}$ (b) $E_{Y_{max}}$ for Exponential pause time

Figure 3.5: $E_{Y_{max}}$ comparison between exponential pause time and fixed time

The total amount of file delivery to others for naive scheme is $N = (n-1)(n-2)$ and coded scheme is $N = \dfrac{(n-1)(n-2)}{2}$. Expecting to see the factor of 2 since in coded scheme we transfer a file to two different users.

To find the difference between theory and real environment thus two tables created as table 3.3 and table 3.4. The first row of the table which lead by #users means N servers(users) are handling the problem of MapReduce. The second row #files/user stands for how many files that each user will transfer to other users. The row of EY_i stands for the theoretical value of the communication cost via the total amount of file will do the transfer work for each user. The row for EY_{max} is based (3.5) to find the maximum of communication cost for different amount of users. Next row is for every user when dealing with file transfer how

would take for N users. the last row is for the server which did longest time. For the coded scheme has two more lines with named of decoding which means measured time not only with encoding(file transfer), also with decoding process which we assume took zero times. We choose the number of users to be from 4 to 10. For a given number of users, the experiment is done for 350 times, and the measured result is averaged over these times. The table can be plot in the figure 3.6

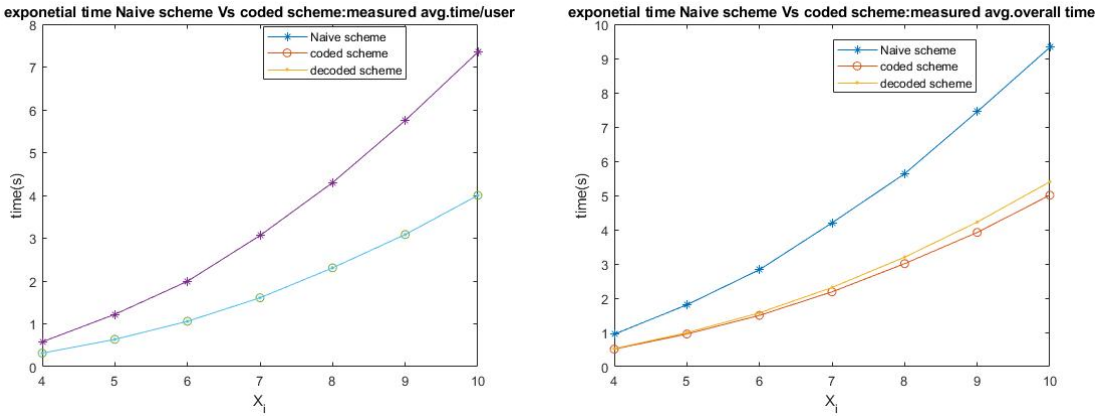
# users	4	5	6	7	8	9	10
# files/user (N)	6	12	20	30	42	56	72
EY_i	0.6	1.2	2	3	4.2	5.6	7.2
EY_{max}	0.87	1.63	2.6	3.78	5.17	6.76	8.56
Measured avg. time/user	0.58	1.22	1.99	3.06	4.3	5.75	7.35
Measured avg. overall time	0.95	1.81	2.83	4.2	5.63	7.45	9.32

Table 3.3: Naive Scheme exponential distribution condition for MapReduce

# users	4	5	6	7	8	9	10
# files/user (N)	3	6	10	15	21	28	36
EY_i	0.3	0.6	1	1.5	2.1	2.8	3.6
EY_{max}	0.49	0.91	1.43	2.06	2.79	3.63	4.58
Measured avg. time/user	0.35	0.647	1.06	1.62	2.32	3.09	4
Measured avg. time/user with decoding	0.35	0.64	1.06	1.61	2.31	3.09	4
Measured avg. overall time	0.51	0.95	1.49	2.18	3.01	3.92	5.01
Measured avg. overall time with decoding	0.53	0.99	1.57	2.31	3.19	4.22	5.4

Table 3.4: Coded Scheme exponential distribution condition for MapReduce

The measured time is longer than the expectation time($E_{Y_{max}}$) from table 3.3 and table 3.4 because the computer has unknown need to process. Such an email notification, RAM cache, etc. All those factors caused the measured time is longer than the expectation. If checking the detail the threshold is not too large to affect the result. The expectation ratio for the coded scheme and naive scheme is 2, but the cache reasons and distribution has longer waiting time for the file transfer made the factor to 1.47.



(a) Measured avg. time/user (b) Measured avg. overall time

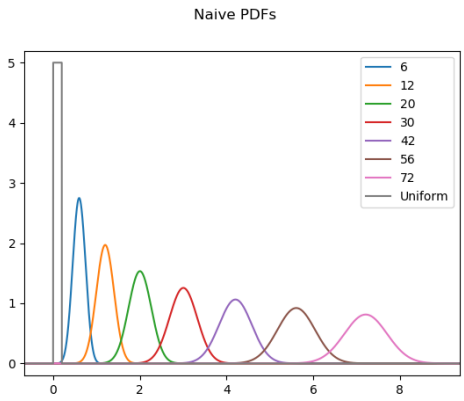
Figure 3.6: Exponential time comparsion between each user and overall

3.3 Uniform Distribution Pause Time

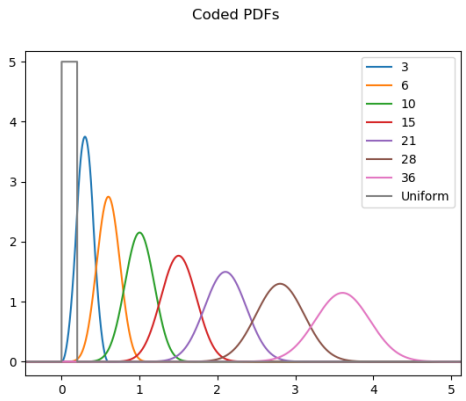
The experience for exponential distribution in section 3.2 helped to develop a uniform distribution function as delivery time(pause time). To get a mean value of 0.1, we set the uniform distribution to be in the interval 0 to 0.2. Since many files will transmit from one user to other users. The communication time of each user will be no longer as uniform distribution, but the sum of N uniform random variable, which will become Irwin–Hall distribution[9].

$$f_{Y_i}(x) = \frac{1}{(n-1)!} \sum_{k=0}^{\lfloor x \rfloor} \binom{n}{k} (x-k)^{n-1} \tag{3.12}$$

Checking the convolution for uniform distribution and generate image showed in figure 3.7 where the x-axis stands for the amount of time cost during the file transmission, y-axis stands for the total amount of file size can be transfer by the channel. Also the probability density function(PDF) can be written as (3.12) where n is the amount of uniform distribution, x in the range of $[0, 1, 2, \cdots, n]$. The plot for irwin-hall distribution showed in the figure 3.7 where the left figure is naive scheme and right is coded scheme. Numbers in the right top are the total amount of file for each user will transfer to other user. The number also indicate to the total amount of users (M) from top to bottom matched from the list $[4, 5, 6, \cdots, 10]$.



(a) Naive Transmission PDF Graph



(b) Coded Transmission PDF Graph

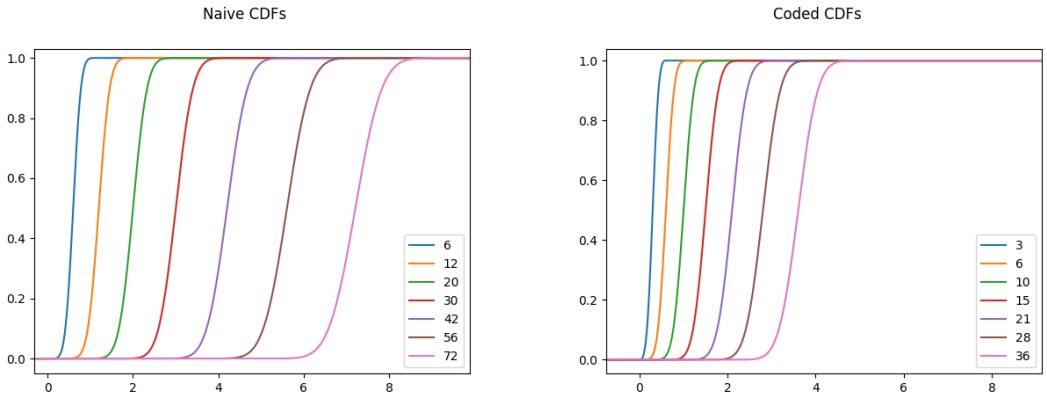
Figure 3.7: The above two figures are the PDF after convoluted multiple uniform distributions based on the (3.13)

The cumulative density function(CDF) can be derived as (3.13) after we did the integration of the PDF function.

$$F_{Y_i}(x) = \frac{1}{n!}\sum_{k=0}^{\lfloor x \rfloor}(-1)^k \binom{n}{k}(x-k)^n \tag{3.13}$$

The mean value of the Irwin-hall distribution is $\frac{n}{2}$. After knowing (3.12), continue to plot the CDF function and generate a figure 3.8 where the x-axis stands for the amount of time cost during the file transmission, y-axis stands for the total amount of file size can be transfer by the channel. Keep all the variable same which checks the amount of files transmitted. As we can discovery that both figure 3.8a(left) and figure 3.8b(right) showed while increase the total amount of file transmitted(the legend) the longer time needed.

To find the max of the CDF of the max Y_{max} take the power of the CDF function where the power is the total amount of file, showed as the legend in figure 3.9, being transfer to other user. The figure 3.9a is for the naive scheme and the x axis is the total amount of time spent for the scheme. The figure 3.9b is coded scheme for the max value for different amount of user working on the same project. Comparing the figure 3.8 and figure 3.9 can



(a) Naive Transmission CDF Graph (b) Coded Transmission CDF Graph

Figure 3.8: The above two figures are the CDF for Irwin-hall distribution based on (3.13)

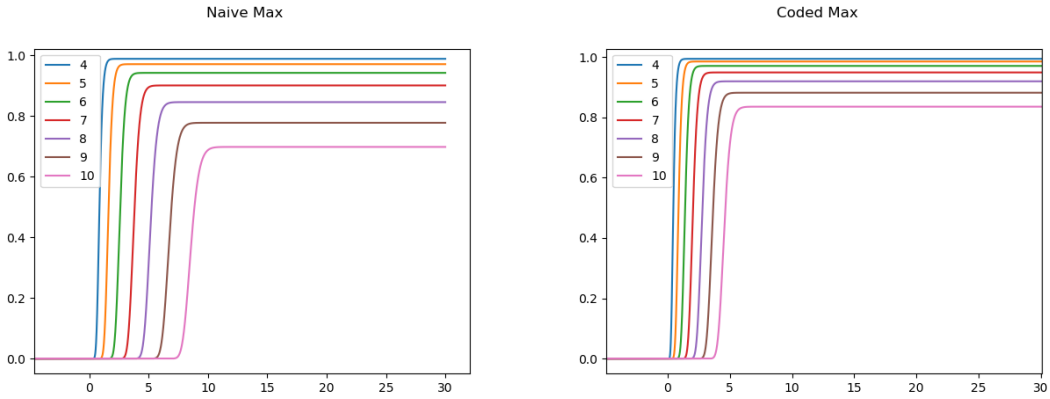
see the slope is getting sharp. After this step, since the original CDF function is $F(x) = \frac{x}{0.2}$ for the range x between 0 and 0.2, the function turn out to be (3.14). Plotting the max of the CDF showed in the figure 3.9.

$$F_{Y_{max}} = F_{Y_i}(x)^N = (\frac{x}{0.2})^N \tag{3.14}$$

The figure 3.9a(left) is the naive scheme condition and the figure 3.9b(right) is the coded scheme condition. Comparing both figure can find the coded scheme rising it's slope faster than the naive scheme, but the error got increased many times. Thus the final answer for the average expectation will be not accurate as we expect.

To find the changes due to the figure 3.9 take the derivative of the max function can see more detail of the changes of the function. Due to previous steps the derivation function can be expressed as (3.15) where N is the total amount of the users and x is the range of the function which interval is (0,∞).

$$f_{y_{max}}(x) = \frac{d(F_{Y_{max}})}{dx} = nf_{Y_i}(x)F_{Y_i}(x)^{N-1} = \frac{1}{(0.2)^N}nx^{N-1} \tag{3.15}$$



(a) Naive Max Transmission CDF Graph (b) Coded Max Transmission CDF Graph

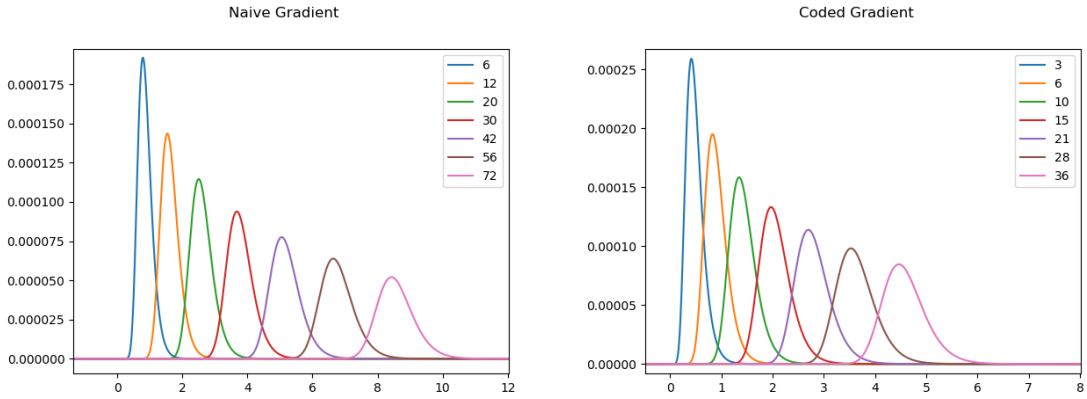
Figure 3.9: The above two figures are the max of CDF after change the power based on N.

If check the information that provide from figure 3.10 can find the peak value is closer and bit larger than the expected value. The detail information showed in the figure 3.10 by the (3.15). Can find the peak and interval for the new distribution by look at the figures. The x axis stands for the how long the file needed for transmitted for amount of users where users are in the range from 4 to 10. The figure 3.10a(left) is naive scheme's condition the figure 3.10b. The legend stands for the total amount of file for the transfer. The x-axis stands for the time for the file transmission, the y-axis stands for the peak value when it is increased the fastest time.

After finding (3.15) of the max function is to prove the peak is the mean or not. Drawing the figures for the new CDF based on the derivation. we can receive the image below which in the figure 3.11. The function to find the expected value is in (3.16) where the x is in the range of $(-\infty, \infty)$

$$E_{Y_{max}}(x). = \int x f_{y_{max}}(x) dx = \int x \frac{1}{(0.2)^N} n x^{N-1} dx \tag{3.16}$$

The result of the figure showed that the mean is at the peak. The result of the max mean value showed in the both table which are table 3.5 and table 3.6. The legend stands for the

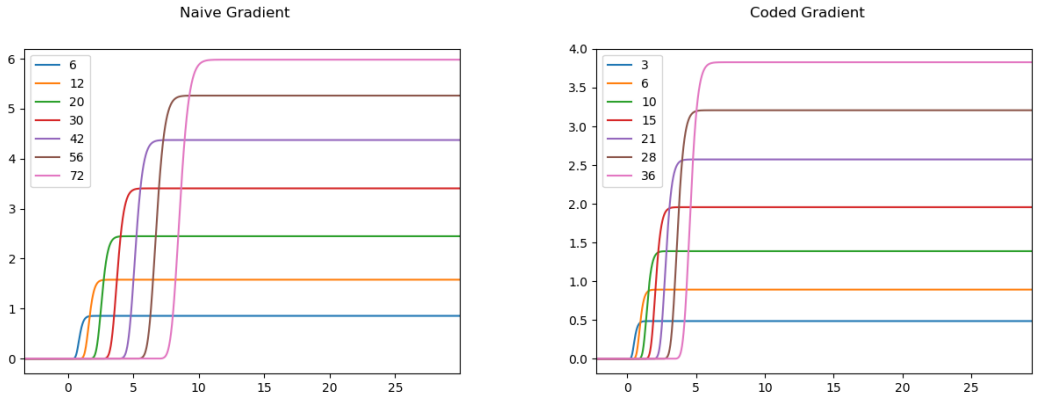


(a) Naive Max Transmission derivation Graph

(b) Coded Max Transmission derivation Graph

Figure 3.10: Two figures above are the derivation of the max function for both naive and coded scheme.

total amount of file for the transfer. The y-axis is the $E_{Y_{max}}$.

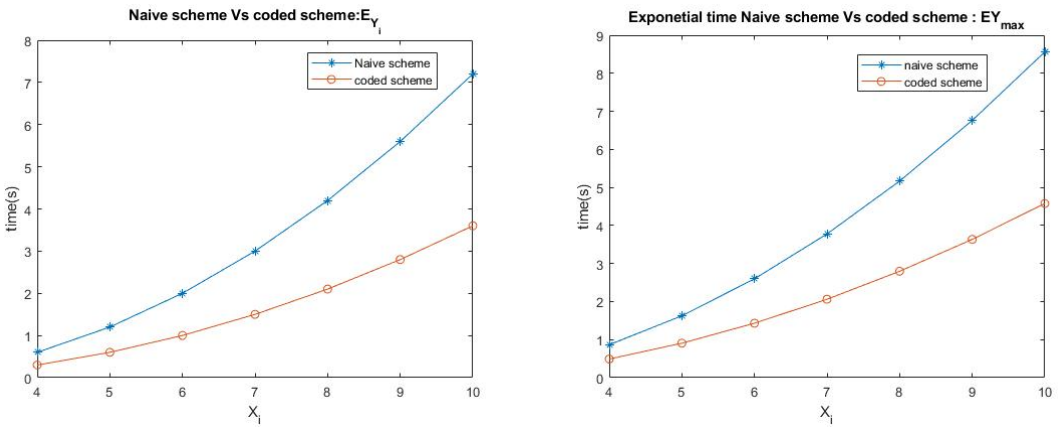


(a) Naive CDF Graph for multiple users

(b) Coded CDF Graph for multiple users

Figure 3.11: The above two figures are the derivation of the max function. The numbers in the legend are the amount of files transmitted during the shuffling steps. The figure 3.9a is the naive condition and the figure 3.9b is the coded condition.

After calculate the $E_{Y_{max}}$ it is time to compare between the E_{Y_i} vs. $E_{Y_{max}}$, from the section 3.1 knowing that the E_{Y_i} for any of the distribution same as the $E_{Y_{max}}$ at fixed pause time condition. Thus, comparing the figure as shown figure 3.12. After calculate the theoretical max value of the uniform equation, and known theoretical pause time with all the measured



(a) fixed time for $E_{Y_{max}}$ (b) $E_{Y_{max}}$ for Uniform pause time

Figure 3.12: $E_{Y_{max}}$ comparison between exponential pause time and fixed time

value can create two table for naive scheme and coded scheme showed in table 3.5 and table 3.6. The first row of the table which lead by #users means N servers(users) are handling the problem of MapReduce. The second row #files/user stands for how many files that each user will transfer to other users. The row of EY_i stands for the theoretical value of the communication cost via the total amount of file will do the transfer work for each user. The row for EY_{max} is based (3.5) to find the maximum of communication cost for different amount of users. Next row is for every user when dealing with file transfer how would take for N users. the last row is for the server which did longest time. For the coded scheme has two more lines with named of decoding which means measured time not only with encoding(file transfer), also with decoding process which we assume took zero times. We choose the number of users to be from 4 to 10. For a given number of users, the experiment is done for 350 times, and the measured result is averaged over these times. Then comparing the measured time/user and measured time for overall time and plot in the figure 3.13. As figures showed that the measured overall time is bit longer than the time/user because the system has some overhead cause some node will run faster some will run slower. The overall time is based on the longest time that spent for the program, but time/user is take the average for all the users process time. The final ratio for the communication cost is 1.98

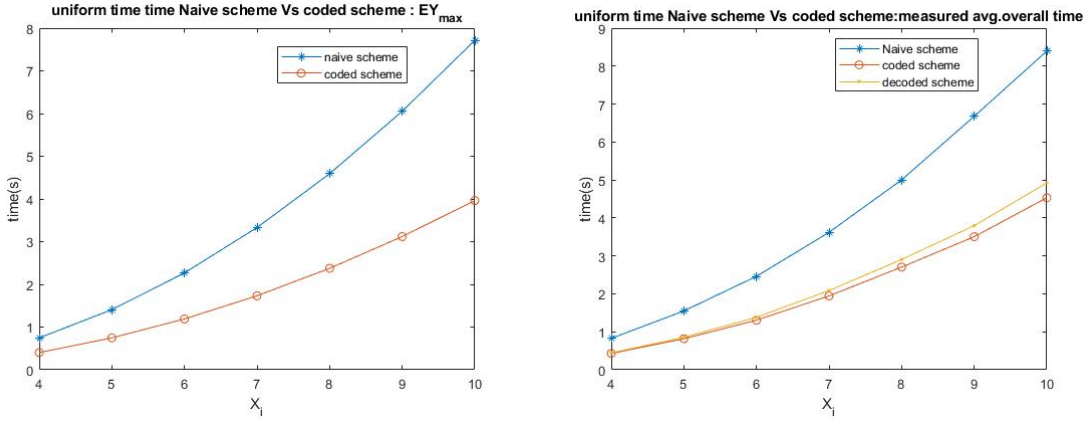
between naive scheme and coded scheme.

# users	4	5	6	7	8	9	10
# files/user (N)	6	12	20	30	42	56	72
EY_i	0.6	1.2	2	3	4.2	5.6	7.2
EY_{max}	0.7	1.39	2.28	3.36	4.68	6.17	7.88
Measured avg. time/user	0.63	1.21	2.01	3.03	4.28	5.71	7.34
Measured avg. overall time	0.83	1.55	2.46	3.62	5	6.68	8.4

Table 3.5: Naive Scheme uniform distribution condition for MapReduce

# users	4	5	6	7	8	9	10
# files/user (N)	3	6	10	15	21	28	36
EY_i	0.3	0.6	1	1.5	2.1	2.8	3.6
EY_{max}	0.38	0.74	1.21	1.76	2.43	3.2	4.08
Measured avg. time/user	0.32	0.65	1.07	1.62	2.31	3.05	3.98
Measured avg. time/user with decoding	0.32	0.65	1.07	1.62	2.31	3.05	3.98
Measured avg. overall time	0.43	0.82	1.31	1.95	2.7	3.51	4.53
Measured avg. overall time with decoding	0.44	0.86	1.38	2.09	2.91	3.8	4.92

Table 3.6: Coded Scheme uniform distribution condition for MapReduce

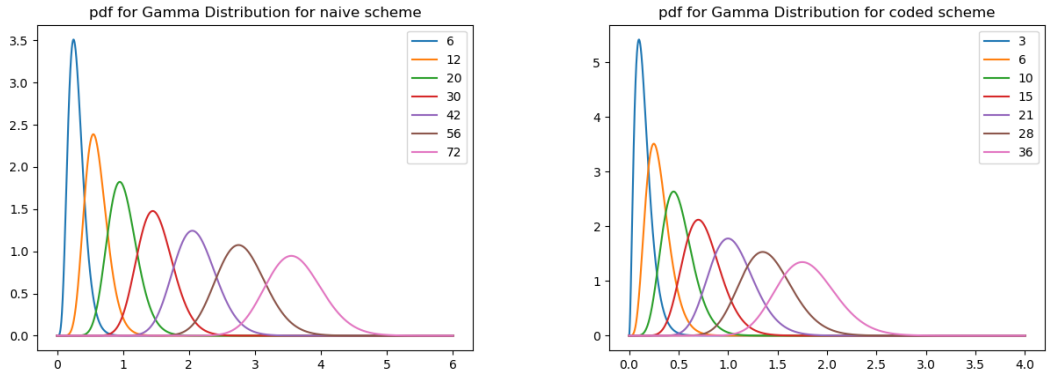


(a) Measured avg. time/user (b) Measured avg. overall time

Figure 3.13: Measured Uniform Pause time between time/user and overall time

3.4 Practical Exponential Distribution

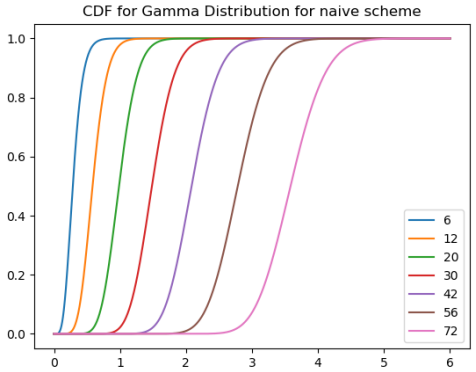
For exponential distribution it may have a chance to transfer the file taking 0 seconds, but real world the communication cost never equal to zero and always has time to count down. Due to section 3.1 state that the mean value for the communication for each file is 0.1 seconds. Thus we assume the communication time per file is the fixed constant 0.05 plus an exponential distribution function from section 3.2 with expectation 0.05. By using the equation to calculate the λ, $\frac{1}{\lambda} = 0.05$ so $\lambda = 20$. The gamma distribution changed due to lambda changed, the new gamma distribution can be written as $\Gamma(mean \times 0.05, 0.05)$. The total communication cost for practical exponential distribution changed to $X_j \times 0.05$ (seconds)+ $E_{Y_{max}}$ where $E_{Y_{max}}$ is showed in (3.11). Then plot the PDF function of the distribution as shown in the figure 3.14 where the legend stands for the total amount of the file for the transfer. The x-axis is the time need for the communication as unit of second.



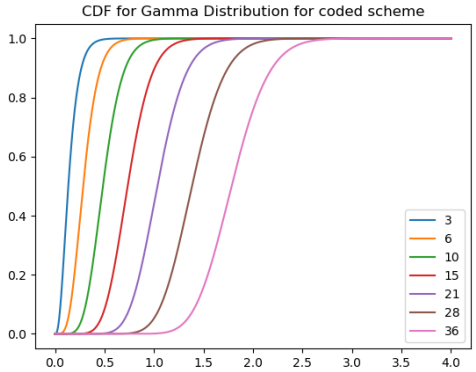
(a) Uniform piratical naive Graph(f_{Y_i}) (b) Uniform piratical coded Graph(f_{Y_i})

Figure 3.14: The above two figures are the PDF for practical condition and the graph only present for f_{Y_i}.

Then plot the CDF based on the PDF function from figure 3.14 and generate figure 3.15 where the legend is the total amount of file during transfer. The x-axis is the communication cost corresponding to the legend as unit of second.



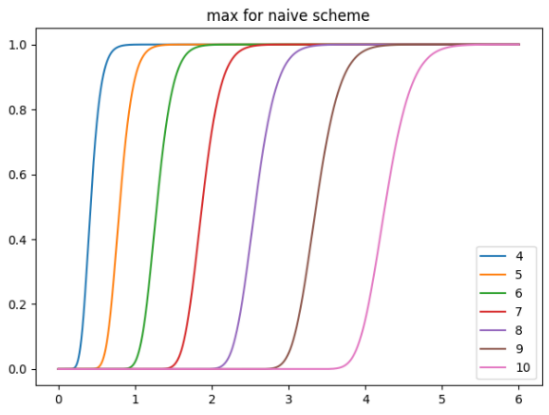
(a) Uniform piratical naive Graph(f_{Y_i})



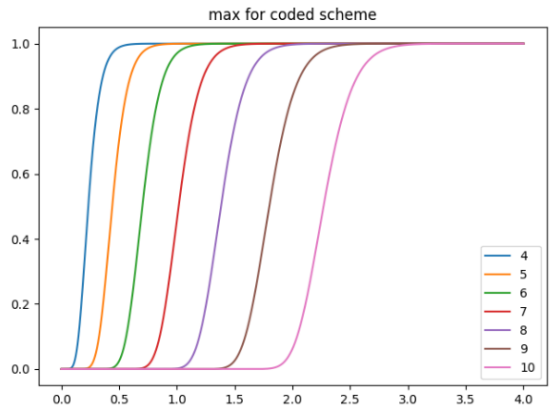
(b) Uniform piratical coded Graph(f_{Y_i})

Figure 3.15: The above two figures are the CDF for practical condition and the graph only present for F_{Y_i}.

The maximum function can be calculated by taking the power regarding the total amount of users(N) by using (3.5). For the figure 3.16 is to find the $F_{Y_{max}}$. The legend is the total amount of file for each server to the transfer(M) the x-axis is the time in seconds for the function to reach the summation of the y-axis.



(a) Uniform piratical naive Graph($F_{Y_{max}}$)

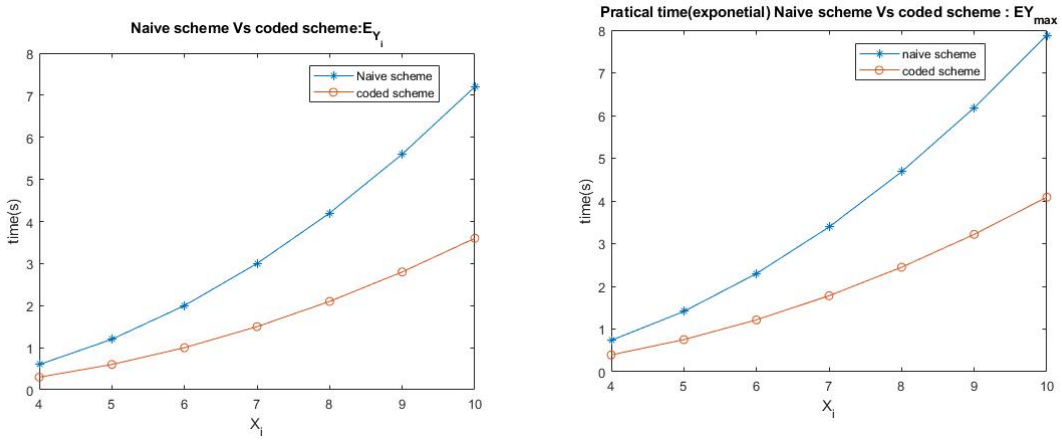


(b) Uniform piratical coded Graph($F_{Y_{max}}$)

Figure 3.16: The above two figures are the Max funtion for practical condition and the graph only present for $F_{Y_{max}}$.

After calculate the $E_{Y_{max}}$ it is time to compare between the E_{Y_i} vs. $E_{Y_{max}}$, from the section 3.1 knowing that the E_{Y_i} for any of the distribution same as the $E_{Y_{max}}$ at fixed pause time

condition. Thus, comparing the figure as shown figure 3.17. The legend represent as the scheme to use and the x-axis is the total amount of users, the y-axis is the how long it will take for all the server showed in the right figure 3.17b. Checking the figure 3.17, the plot



(a) fixed time for $E_{Y_{max}}$

(b) $E_{Y_{max}}$ for practical exponential pause time

Figure 3.17: $E_{Y_{max}}$ comparison between practical exponential pause time and fixed time

showed that the Naive scheme will spent nearly 1 seconds more than ideal result. Taking the experiment to see the naive scheme and coded scheme. Showing the table 3.7 and table 3.8 below are the theoretical and experimental result. The first row of the table which lead by #users means N servers(users) are handling the problem of MapReduce. The second row #files/user stands for how many files that each user will transfer to other users. The row of EY_i stands for the theoretical value of the communication cost via the total amount of file will do the transfer work for each user. The row for EY_{max} is based (3.5) to find the maximum of communication cost for different amount of users. Next row is for every user when dealing with file transfer how would take for N users. the last row is for the server which did longest time. For the coded scheme has two more lines with named of decoding which means measured time not only with encoding(file transfer), also with decoding process which we assume took zero times. We choose the number of users to be from 4 to 10. For a given number of users, the experiment is done for 350 times, and the measured result is averaged over these times. Based on the result, can generate a new comparison between

measured avg. time/user and overall time and showed in the figure 3.18. The overall time is longer than the measure time/user as expected like 1 seconds longer when total amount of user is 10. The expectation of the ratio by the assumption is 2, after the experiment created two tables: table 3.8 and table 3.7. The final ratio of the practical exponential distribution between coded scheme and naive scheme is 1.877.

# users	4	5	6	7	8	9	10
# files/user (N)	6	12	20	30	42	56	72
EY_i	0.6	1.2	2	3	4.2	5.6	7.2
EY_{max}	0.73	1.41	2.98	3.39	4.68	6.18	7.88
Measured avg. time/user	0.62	1.23	2.01	3.07	4.29	5.73	7.38
Measured avg. overall time	0.83	1.56	2.42	3.62	4.98	6.55	8.37

Table 3.7: Naive Scheme practical exponential distribution condition for MapReduce

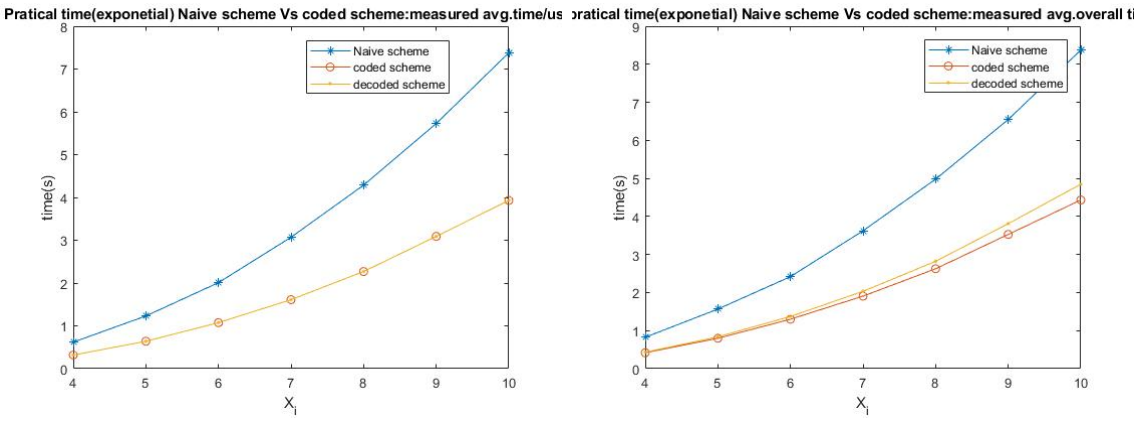
# users	4	5	6	7	8	9	10
# files/user (N)	3	6	10	15	21	28	36
EY_i	0.3	0.6	1	1.5	2.1	2.8	3.6
EY_{max}	0.395	0.75	1.22	1.78	2.45	3.22	4.09
Measured avg. time/user	0.32	0.63	1.07	1.61	2.27	3.09	3.93
Measured avg. time/user with decoding	0.32	0.64	1.075	1.61	2.27	3.09	3.93
Measured avg. overall time	0.42	0.79	1.29	1.9	2.62	3.52	4.43
Measured avg. overall time with decoding	0.43	0.84	1.37	2.03	2.82	3.81	4.85

Table 3.8: Coded Scheme practical exponential distribution condition for MapReduce

In the end, the figure 3.18 is the comparison for measured time of single user and overall time between the coded scheme and naive scheme. The legend means the scheme, the x-axis stands for the number of users(N), and y-axis stands for the time that spend during each communication steps.

3.5 Practical Uniform Distribution

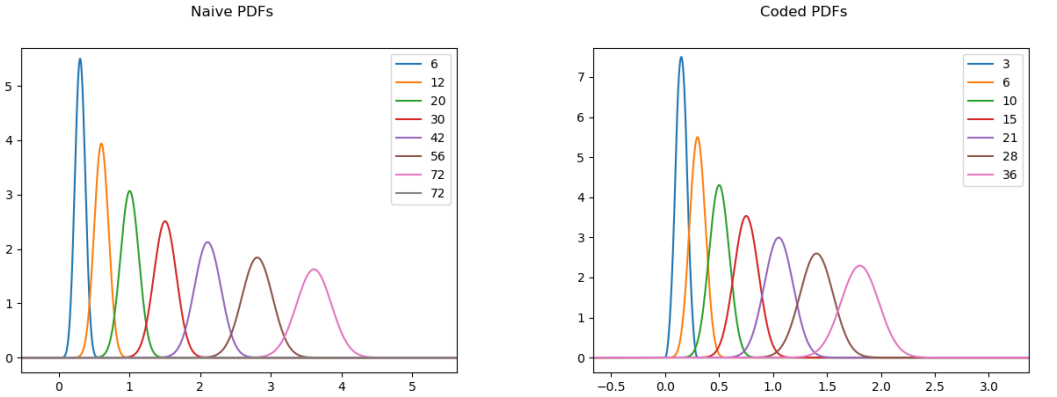
Similar to section 3.4, here want to simulate real condition under the uniform distribution where some fixed overhead time occur during the transmission of every file. The pause



(a) Measured avg. time/user (b) Measured avg. overall time

Figure 3.18: $E_{Y_{max}}$ comparison between practical exponential pause time and fixed time

function for practical uniform distribution is $X_j \times 0.05 + E_{Y_{max}}(x)$ where $E_{Y_{max}}$ from (3.16). The PDF showed in the figure 3.19 where the legend is the size of total amount files for each user will transfer to others, the x-axis is the time spent corresponding to the amount of users.

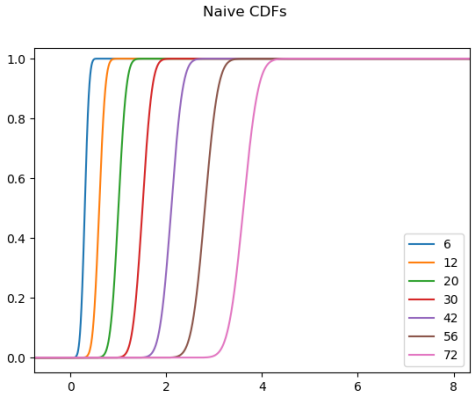


(a) Uniform piratical naive Graph(f_{Y_i}) (b) Uniform piratical coded Graph(f_{Y_i})

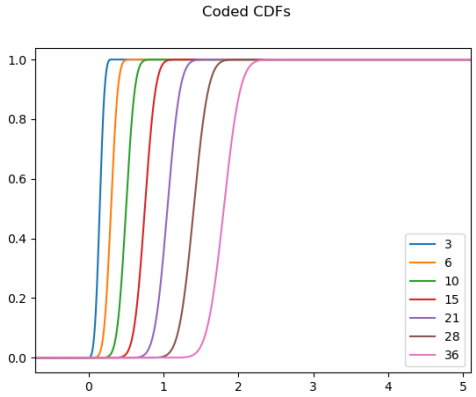
Figure 3.19: The above two figures are the PDF for practical condition and the graph only present for f_{Y_i}.

After plot PDF then draw the CDF which showed in the figure 3.20b where the legend also is the size of total amount of file transfer from one server to other servers. The x-axis is the

time that spent for the transfer.

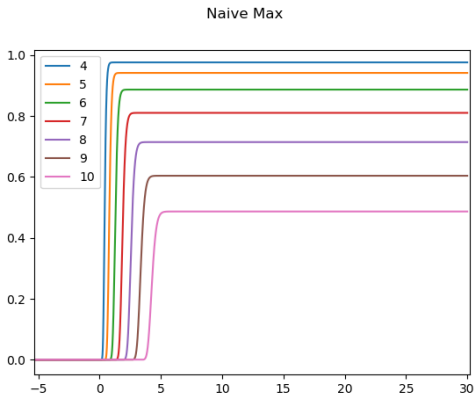


(a) Uniform piratical naive Graph(f_{Y_i})

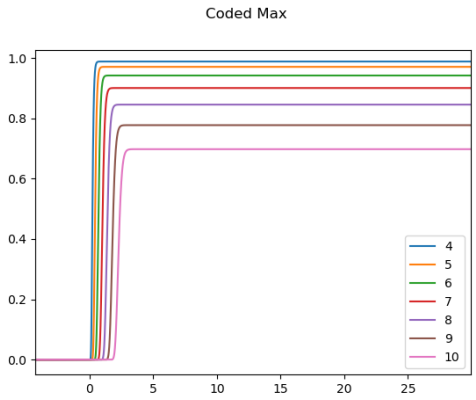


(b) Uniform piratical coded Graph(f_{Y_i})

Figure 3.20: The above two figures are the CDF for practical condition and the graph only present for F_{Y_i}.



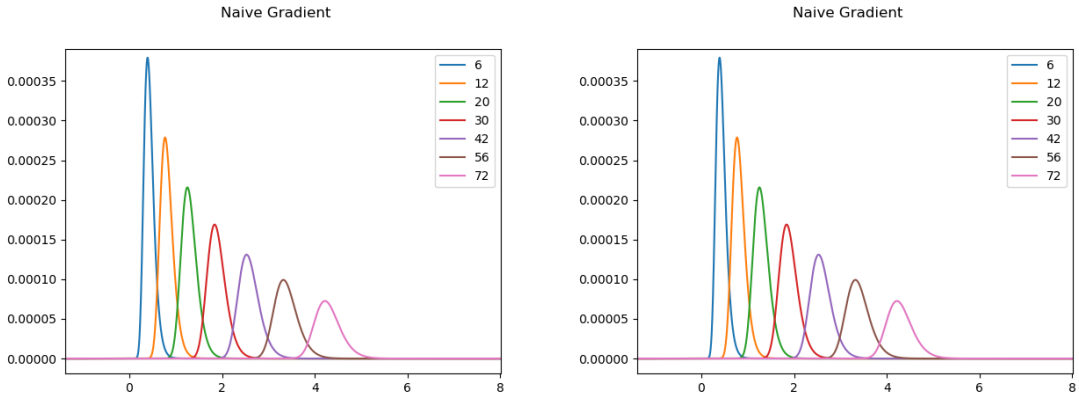
(a) Uniform piratical naive Graph($F_{Y_{max}}$)



(b) Uniform piratical coded Graph($F_{Y_{max}}$)

Figure 3.21: The above two figures are the Max funtion for practical condition and the graph only present for $F_{Y_{max}}$.

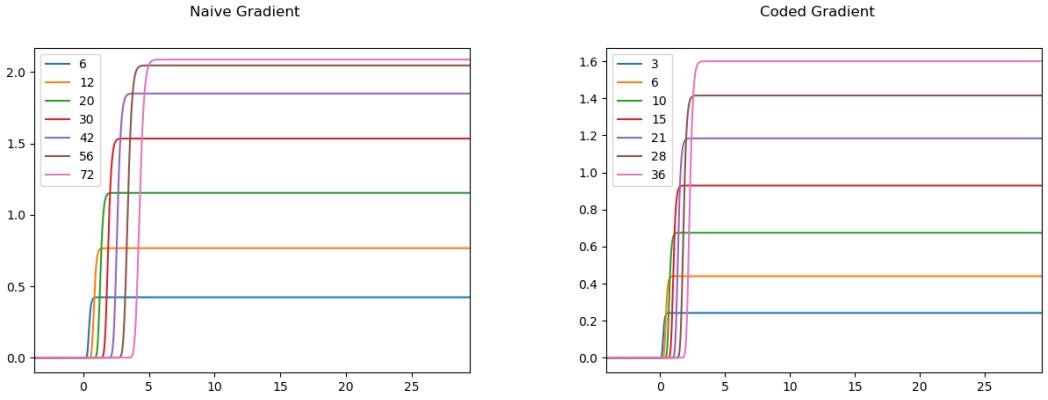
After find the maximum power of the function take the derivative find the figure 3.22. The figure 3.22a(left) is the naive scheme and the figure 3.22b(right) is the coded condition. Legend is the total amount of file for the transfer. x-axis is the communication cost. y-axis is to find the peak value to match with time to find when the plot increased the fastest. The $E_{Y_{max}}$ can be find after find the derivative function. The figure plot in the figure 3.23. The



(a) Uniform piratical naive derivation graph (b) Uniform piratical coded derivation graph

Figure 3.22: Two figures above are the derivation of the max function for both naive and coded scheme.

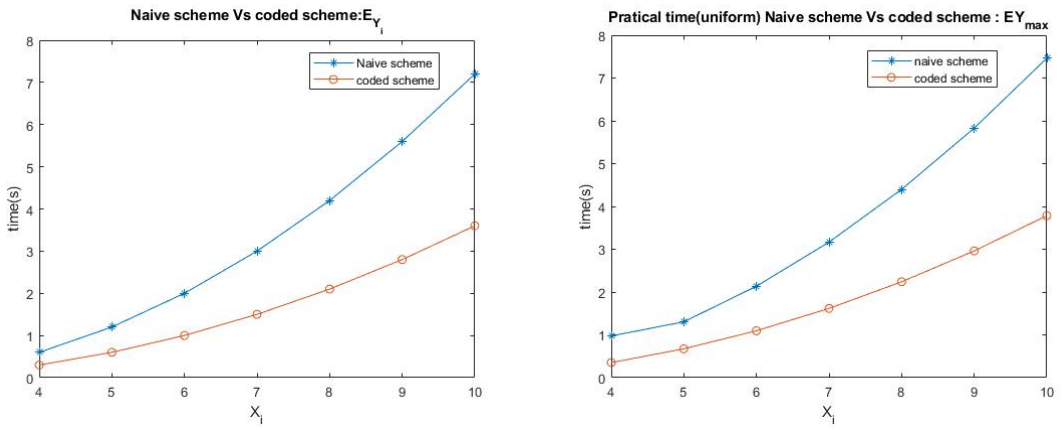
figure 3.23a(left) is the naive scheme, and the figure 3.23b(right) is the coded scheme. The legend is the total amount of file per each user will transferred the file. and y-axis is the value of $E_{Y_{max}}$. After calculate the $E_{Y_{max}}$ it is time to compare between the E_{Y_i} vs. $E_{Y_{max}}$,



(a) Naive CDF Graph for multiple users (b) Coded CDF Graph for multiple users

Figure 3.23: The above two figures are the derivation of the max function. The numbers in the legend are the amount of files transmitted during the shuffling steps. The figure 3.9a is the naive condition and the figure 3.9b is the coded condition.

from the section 3.1 knowing that the E_{Y_i} for any of the distribution same as the $E_{Y_{max}}$ at fixed pause time condition. Thus, comparing the figure as shown figure 3.24.



(a) fixed time for $E_{Y_{max}}$ (b) $E_{Y_{max}}$ for practical uniform pause time

Figure 3.24: $E_{Y_{max}}$ comparison between practical uniform pause time and fixed time

Taking the experiment and written in the table below as shown as table 3.9 and table 3.10. Based on the table can plot as figure 3.25 to check if the realistic matched with our calculation. Based on the graph can see it matched with the original design of the data.

# users	4	5	6	7	8	9	10
# files/user (N)	6	12	20	30	42	56	72
EY_i	0.6	1.2	2	3	4.2	5.6	7.2
EY_{max}	0.97	1.3	2.14	3.17	4.4	5.83	7.46
Measured avg. time/user	0.61	1.22	2.26	3.06	4.27	5.72	7.34
Measured avg. overall time	0.72	1.38	2.26	3.36	4.64	6.18	7.89

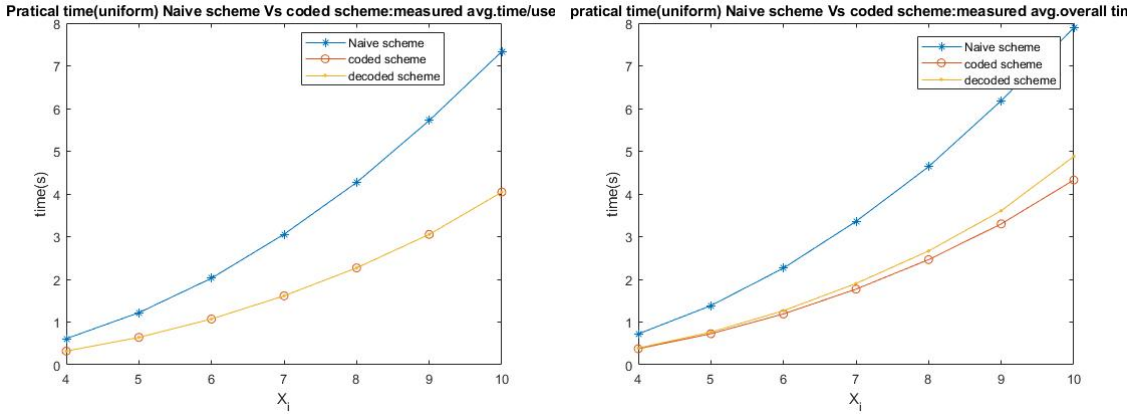
Table 3.9: Naive Scheme practical uniform distribution condition for MapReduce

# users	4	5	6	7	8	9	10
# files/user (N)	3	6	10	15	21	28	36
EY_i	0.3	0.6	1	1.5	2.1	2.8	3.6
EY_{max}	0.35	0.67	1.1	1.62	2.24	2.96	3.78
Measured avg. time/user	0.32	0.64	1.07	1.61	2.27	3.05	4.04
Measured avg. time/user with decoding	0.32	0.64	1.07	1.61	2.27	3.05	4.04
Measured avg. overall time	0.38	0.73	1.19	1.77	2.46	3.3	4.33
Measured avg. overall time with decoding	0.39	0.76	1.27	1.9	2.67	3.61	4.88

Table 3.10: Coded Scheme practical uniform distribution condition for MapReduce

The figure 3.25 is the comparison between the measured time for single user and for overall

time. The legend is the scheme choice, x axis is the total amount of servers, y-axis is the time(s) to spend.



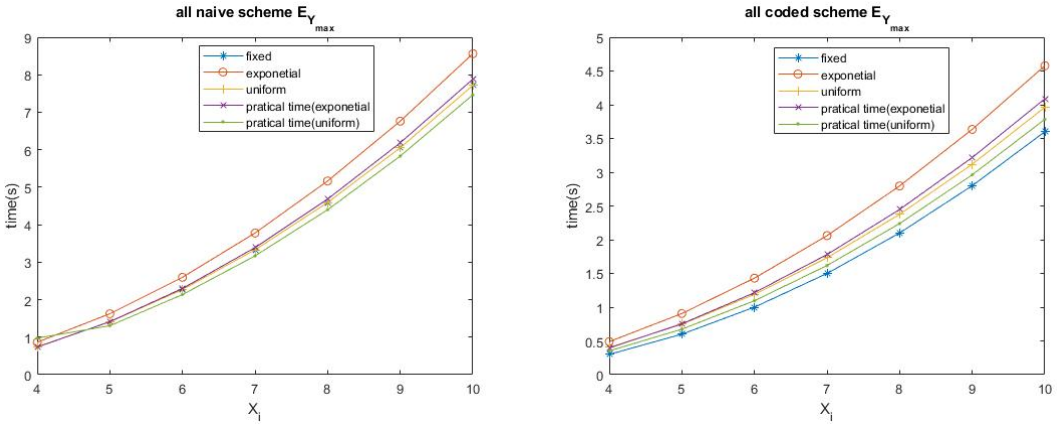
(a) Measured avg. time/user (b) Measured avg. overall time

Figure 3.25: $E_{Y_{max}}$ comparison between practical uniform pause time and fixed time

3.6 Discussion

Comparing all the coded table can found the cost for decoding took no time, but max time decoding took longer time because each server start time are different which means the max time with decoding will look like a slight longer than max time. First want to check the $E_{Y_{max}}$ for all the condition for naive scheme and coded scheme showed in figure 3.26. By looking at the graph knowing that both naive scheme and coded scheme the exponential distribution cost the most time. The practical uniform distribution is the one most close to the fixed time. The legend stands for the communication parameter, the x-axis stands for the number of users, y-axis stands for the communication cost.

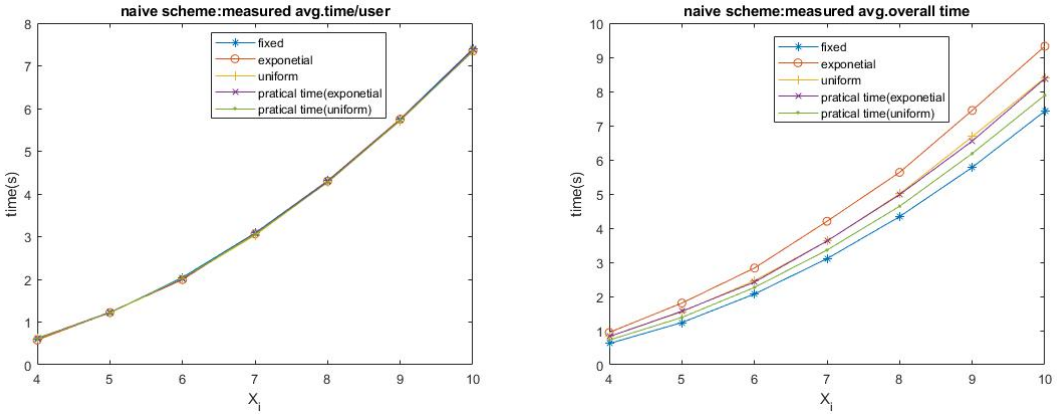
Knowing the theoretical value is not enough, therefore check the measured average time per user is very important. So plot in figure 3.27. The naive scheme for every single user cannot find the difference, but naive scheme overall running time satisfy the ideal model, $E_{Y_{max}}$.



(a) Naive Graph($E_{Y_{max}}$) (b) Coded Graph($E_{Y_{max}}$)

Figure 3.26: The two images are for naive scheme and coded scheme $E_{Y_{max}}$.

The legend stands for the communication parameter, the x-axis stands for the number of users, y-axis stands for the communication cost.

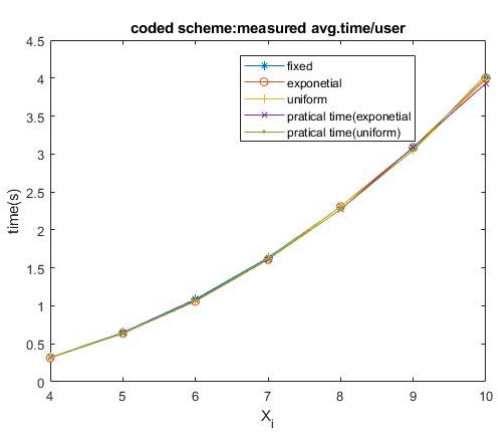


(a) Naive scheme measured avg. time/user (b) Naive scheme measured avg. overall time

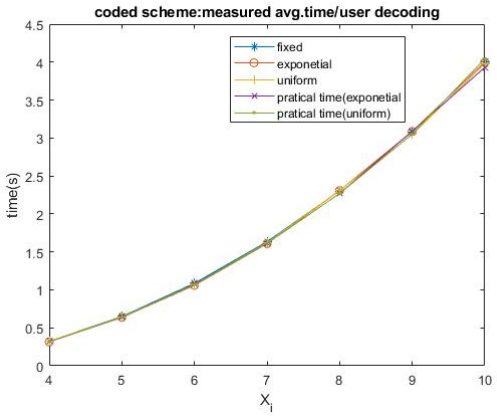
Figure 3.27: Naive scheme measured time.

After plot the naive scheme why not to check the coded scheme. Thus, plot the graph as shown in the figure 3.28. Similar as naive scheme. The coded scheme showed all the curves are about same and this is to prove the coded scheme are more stable than the naive scheme during the file transmission steps. The legend stands for the communication parameter, the x-axis stands for the number of users, y-axis stands for the communication cost.

Last, check the overall time if the coded scheme also satisfy as the previous result from



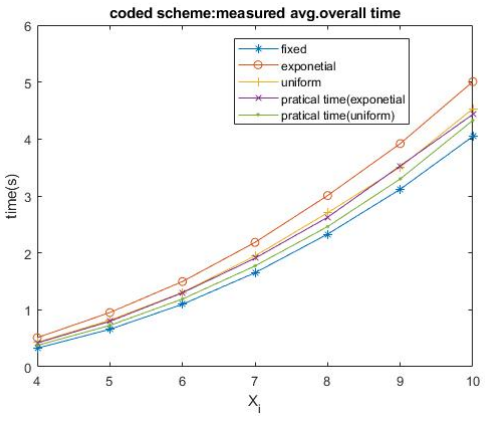
(a) Coded scheme measured avg. time/user



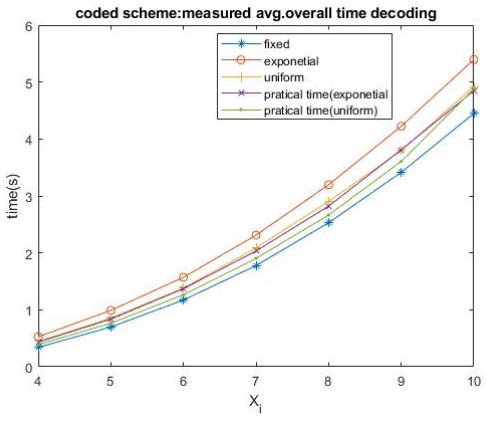
(b) Coded scheme measured avg. time/user with decoding

Figure 3.28: Coded scheme measured avg. time/user.

figure 3.28. This means after multiple user doing the file transfer. The function from exponential distribution and uniform distribution to gamma distribution and Irwin-hall distribution. Also meet the expectation as we want. The legend stands for the communication parameter, the x-axis stands for the number of users, y-axis stands for the communication cost.



(a) Coded scheme measured avg. overall time



(b) Coded scheme measured avg. overall time with decoding

Figure 3.29: Coded scheme measured avg. overall time

Chapter 4

Reverse Index Coding

Searching engines such as Google, Bing, Yahoo are important tools for daily life. Users are expected to find results as fast as possible. For example, when typing University of California, Irvine(UC Irvine) in the search bar, one would expect to find all the information by searching. The database is unbelievably large so that users can find all the resources they need. Users expect the search is fast enough to demonstrate the result to save their time. Users need the search engine companies to build their searching program fast enough with a large database to provide convenience to users. The MapReduce, created by Google's engineer Jeff Dean and Sanjay Ghemawat, creates a programming model for fast search [2]. Coded MapReduce saves the communication cost based on the data communication[8]. The reverse index is one of the key steps of search engines, whose goal is to help users find all the webpages that refer to a given webpage. In fact, reverse indexing can be conviniently implemented using MapReduce [2].

4.1 Algorithm Overview

The reverse index coding framework is designed to find the most important website that is among the UC Irvine webpages ending with ".uci.edu". The reverse index coding has three simple steps conceptually: get all the data, analyze the data and then output the data. The steps matched with crawler, map, and reduce

4.2 Crawler

Firstly, we gather all the subdomains under "uci.edu" which is the domain of the University of California, Irvine, to do this, we use an open source Python tool called Sublist3r. We end up crawling 1155 sub-domains and they are saved as a list in the Master Server, we also build up a link-to-number hash table that maps each link to a unique number[12]. so that we only need to operate numbers in the later stages.

By using the Requests and lxml libraries in Python, we are able to fetch the HTML code of each web page and extract all the URL links, also called target links, referred in that page. For each link, after fetching all the links it refers to, a file named after this link's mapped number is created and all the mapped numbers of those referred links are written to that file. Through this process, some links out of UCI's domain can also be found. For those ones we still add them to our link-to-number hash table.

Finally, after an iteration of crawling, 1155 files are created, each of them represents a source link and its content indicates the target links.

4.3 Mapping for Reverse Index Coding

After clawer collects all the web links, store them into a file. Name all the links by numbers($\bar{x}$) first used hash table[12] to find all the keys and values for the source and target website and then use designed ideas which will introduce in the section 4.1 to convert all the links to the file format such as $\alpha\_\beta\_\bar{x}$. Fill out the file by the sources link($\bar{x}$) and this is the target link.

Assume 1155 files exists in the folder with 4 active servers. Due to MapReduce example from chapter 2. Known that there will be 12 chapters which can be calculate as $N \times (N-1)$ where N is the amount of servers. Thus, 1155 cannot be the multiplier for 12 for four servers. Rounding up the total amount of value to make it become multiplier. Thus, we are going to have 1160 files which named from 1 to 1160.

The mapping for Reverse Index Coding is similar to MapReduce question as showed in Section 2.2.2. Here all the file after index to pair named as α_β_S. As mentioned in the section 4.4 the file located at both location α and location β which controlled by user α and user β. After storing all the file from master to different locations. The map task end.

4.4 Conversion between Pair and Index

After collect the data by clawer steps from section 4.2 it is time to dealing the file naming. Hash table linked all 1155 website address(S) to a dictionary. Because want to continue using the model for MapReduce[6]. From section 2.2.1 knowing that the file name is based on the amount of users. Set N users are trying to request from the server. The filename want to generate as Filename A=α_β_S where the file located at folder α,A(map1), and folder β,A(map2). The range of α is $[1, 2, 3, \cdots N]$, so does β. To meet requirement of filename from S to Filename A first need follow the steps that will introduced in section 4.4.1. After

all the work finished then use section 4.4.2 to complete the work.

4.4.1 Index to Pair

To continue for the idea of mapreduce. An algorithm has been implemented named as index2pair. The table 4.1 is a demonstrate the index2pair when only has four servers. α and β stands for the server shared for these data. The calculation of α is shown in the equation 4.1. In the case of the example the value of α is the separate the 12 values to 4 different parts. Second server that shared the file with alpha denoted as β which shown in the equation 4.2 where t=index mod(N-1).

$$\alpha = \lfloor \frac{index}{N-1} \rfloor \tag{4.1}$$

$$\beta = \begin{cases} t & t < \alpha \\ t+1 & t \geq \alpha \end{cases} \tag{4.2}$$

4.4.2 Pair to Index

Similar to the section 4.4.1, here we want to covert the pair back to index value. Knowing α and β. Thus, we can calculate the index value. Based on the equation 4.2 we can easier find the value of t as shown in the equation 4.3. Then the recover of the value for index(i) is i=$\alpha(N-1)+t$

Pair Value	Index Value	α	β	t
01	1	0	1	0
02	2	0	2	1
03	3	0	3	2
10	4	1	0	0
12	5	1	2	1
13	6	1	3	2
20	7	2	0	0
21	8	2	1	1
23	9	2	3	2
30	10	3	0	0
31	11	3	1	1
32	12	3	2	2

Table 4.1: The table for index2pair

$$t = \begin{cases} \beta & \beta < \alpha \\ \beta - 1 & \beta > \alpha \end{cases} \tag{4.3}$$

4.5 Encoding for Reverse Index Coding

The encoding for reverse index coding has three steps which are Filename Changing(section 4.5.1), Transfer File to Binary file(section 4.5.2), and File Transfer(section 4.5.3). From table 3.1 to table 3.10 knowing that there are two important factors which are measured average time per user, and measured overall time. The measured overall time is include both section which are section 4.5 and section 4.6. The measured time from this part called as the measured average time per user.

4.5.1 Filename Changing

The filename should follow as the file name in MapReduce from section 2.2.1 where filename A=$\alpha\_\beta\_\gamma\_\theta$ for all the file located at folder α,A(map1), and folder β,A(map2). The range of α is $[1, 2, 3, \cdots N]$, so does β. γ represent for the copy of the file A(copy), so the range of γ is $[1, 2]$. θ in the filename stands for which folder expects to receive the file from folder α and folder β denoted as A(reduce). Thus θ has the same range as α and β.

The content of each file A has source and target link which converted as numbered system by using a hash table. Different user would have different value of *alpha* or β. If the user at locationα means all the β are different for different files. To calculate gamma first count for each file A=α_β_S has how many target for same beta value, the target index called as Ω where Ω in the range of $[1, 2, 3, 4, \cdots, 1155]$. The γ value can be calculated based on the equation 4.4. The Θ value can be calculated as $\theta = \Omega \ mod(N)$

$$\Omega = \begin{cases} 1 & \Omega = odd \\ 2 & \Omega = even \end{cases} \tag{4.4}$$

After the calculate both γ and θ can generate new file A, named as $\alpha\_\beta\_\gamma\_\theta$. The benefit for doing such complicated work because in this way of calculate the θ value can guarantee the file with same name in both locationα and locationβ.

4.5.2 Word to Binary

It is almost impossible to do $'XOR'$ for words or numbers. Thus, keep the filename and change the content from letter or number to the form of binary. The length of each character as 7. After changing the content of files can save time combine files and save the size of the files. The ideal model is all the file size is same so do not care about file size affects the

communication cost. In real life, if a file is large enough the communication time is longer than the file size which is only few bits.

4.5.3 File Communication

After the file become the one we expected then can use the same method from section 2.2.4. Because the reverse index coding based on the MapReduce scheme[2]. The benefit of reverse index coding is using the MapReduce scheme to transfer the file and make it faster.

Naive Scheme

The Naive Scheme is same as MapReduce for each file transfer from one user to another user cost 0.1 seconds. The way of transferring the file by checking the value of θ and transfer the file to the location θ. The design idea is 0.1 seconds per each file, but for reverse index coding has previous steps such word to binary, and filename changing. Thus for each of the file will run a little bit longer time as expected. For naive scheme will transfer total (N-1)(N-2) files to others. For each of the design has threshold off from 0.1 a little, but overall will be a lot more.

Coded Scheme

The Coded Scheme also is same as MapReduce, If file A located at user α will look for partners of β and θ. After finding the β and θ, can direct to know which can be transferred. Then create a new file C named as $\alpha\_\beta\_\gamma\_\theta\_\_\alpha\_\theta\_\gamma\_\beta$ and transfers the file to both locations at β and θ. The coded scheme will transfer the total amount of file is $\frac{(N-1)(N-2)}{2}$ compared with naive scheme is relatively closer to the design idea since coded scheme has less threshold than the naive scheme.

4.6 Decoding for Reverse Index Coding

The decoding process for the reverse index coding separates by two steps: decoding and sorting. The decoding process same as MapReduce problem as mentioned in the section 2.2.5. The main goal for the decoding process is to find the each servers' desired file. The sorting is to find the top links that website trying to promote

4.6.1 Decoding for Reverse Index Coding

The naive scheme uses brute force to transfer files, regardless of the decoding process. The coded scheme needs the decoding process because the coded scheme does not use brute force method to transfer files. In this process, user β and user θ receive file C from userα. As mentioned in section 4.5.3 file C is located at both locations. So for file A located at userβ also has a copy located at userα. Luckily, the user β wants the last bit is β. After receiving file C, user β can decode file C by using file A to get file B through the $'XOR'$ method.

4.6.2 Sorting for Reverse Index Coding

The sorting procedure has two steps. First, each server finds the top 10 ranking under their own location. Then servers send the file to the master server. After the master server received the file analyze the result to find the top overall ranking for the clawed website.

4.7 Algorithm for Reverse Index Coding

Here, we separate the algorithm into two parts which wrote in the figure 4.1 is the algorithm code for the master node. The main contribution for the master node is to collect all the data

from the website by using clawer from section 4.2, send the file to the new location where introduce from section 4.3, generate a file name by using pair to index from section 4.4.1, and generate a filename that can be used for each local machine to use for their encoding and decoding process. The program came to it is the most exciting part after master did

```
Class Master Node:
    procedure Main:
        Collection = Shuffle()
        for all Crawled Links crawled_link, Target Links target_links referred by
    crawled_link:
            Map(crawled_link, target_links, Collection)
    end procedure

    procedure Shuffle:
        Collection = {[i, [j,k for j, k in Random(worker_num) and  k ≠ j ≠ i    ]] for i in
    {0, 1, … worker_num}}
        return Collection
    end procedure

    procedure Map(Source Link src_link, Target Links tar_links, Collection):
        File file
        for all Target Links target_link ∈ tar_links do:
            src_link_file.write(target_link)

        for file and pair in the Master Node and H do:
            Translate (pair.first, pair.second) into file.filename
            SendFile(file, pair.first)
            SendFile(file, pair.second)
    end procedure
```

Figure 4.1: based on what the master sever does written the pseudo code

all the pre-process the algorithm showed in figure 4.2. For each server change the file name, after received file with the filename as α_β_S, by checking the target link. Because by doing this, we can guarantee the file in a different location has the same content. The convert the file from word to binary form which will be benefited from the $'XOR'$ method. Similar as MapReduce problem here transfers the file with controlling the file size by using $'XOR'$ method. The reduction step is the decoding step which decodes the file to find the dream files. As for fun, we have the function of sorting to find the most interest links from all the websites we have clawed

```
Class Worker Node:
    Number of Workers worker_num
    new_file_list = empty list
    procedure FileRename():
        rec_file = ReceiveFile(Master Node)
        src_link <- rec_file
        for target_link in rec_file do:
            Translate src_link into alpha, beta
            File new_file
            Translate (alpha, beta, target_link) into new_file.filename
            Write_To_File(tuple(target_link, src_link), new_file)
            new_file_list.append(new_file)
    end procedure


    procedure Reduce()
        PD <- new POSTING_DICTIONARY<String, List>
        new_file_list = {files generated by FileRename() function}

        for each new_file in new_file_list do:
            for each tuple(target_link, src_link) in new_file.links do:
                PD[target_link].append(src_link)
        sort(P, key = len of each list value)
    end procedure
```

Figure 4.2: This is for all the servers work controlled by the multi-process

4.8 Result

The goal for the reverse index coding is to find the top links that UC Irvine webpages ending with ".uci.edu" trying to promote as mentioned in the section 4.2. The table 4.2 in below is the top links that UC Irvine current promoted by the search engine. The result showed in the table 4.2.

Ranks	Website Address	Mentioned Times
1	www.uci.edu	122
2	som.uci.edu	25
3	www.som.uci.edu	24
4	fa.uci.edu	18
5	ip.ce.uci.edu	7
6	oit.uci.edu	7
7	open.uci.edu	7
8	strategicplan.uci.edu	7
9	studentaffairs.uci.edu	7
10	studentlife.uci.edu	7

Table 4.2: Top Links that UC Irvine homepage to Promote

Chapter 5

Communication Cost Analysis for Reverse Index Coding

The reverse index coding also based on the local machine as the simulation for MapReduce. In this simulation, the local communication time for each file take 0.1 second as ideal. The simulation chosen fixed time, exponential distribution, uniform distribution and the practical condition of exponential distribution and uniform distribution as the communication time for each file. For communication cost separately by regular communication cost and the max of the communication cost. Regular communication cost stands for the value after taking the average of each server communication cost. Max communication cost stands for all the server's responding time. The simulation is running on a local machine and desired all the server can run simultaneously, but cause of the delay not all the servers can open at one time and each server run time is about same. During the simulation total amount of user is i and for each of file transfer cost j time. To complete the experiment five variables are in the list of consideration to complete the experiment which are single file transfer time(X_{ij}), total file transfer time for single user(Y_i), overall transition time(Y_{max}), expectation of file transfer time for each user(EY_i), and expectation of file transfer time for all the users(EY_{max}). The

relation between Y_i and X_i showed in the equation 3.2. The meaning of the Y_i is for each file from a user cost Y_i time as seconds. Unlike MapReduce problem, the reverse index coding simulation has a threshold for file rename during encoding process. It is hard to find the threshold value then the $E_{Y_{max}}$ used the previous result from equation 3.5.

5.1 Fixed Pause time

For the fixed pause time, can set the transfer cost at 0.1 second per file. For table 5.1 and table 5.2 are the result of the naive scheme and coded scheme for MapReduce problem. The first row of the table which lead by #users means N servers(users) are handling the problem of MapReduce. The second row #files/user stands for how many files that each user will transfer to other users. The row of EY_i stands for the theoretical value of the communication cost via the total amount of file will do the transfer work for each user. The row for EY_{max} is based (3.5) to find the maximum of communication cost for different amount of users. Next row is for every user when dealing with file transfer how would take for N users. the last row is for the server which did longest time. For the coded scheme has two more lines with named of decoding which means measured time not only with encoding(file transfer), also with decoding process which we assume took zero times. We choose the number of users to be from 4 to 10. For a given number of users, the experiment is done for 350 times, and the measured result is averaged over these times. The table can be plot in the figure 5.1.

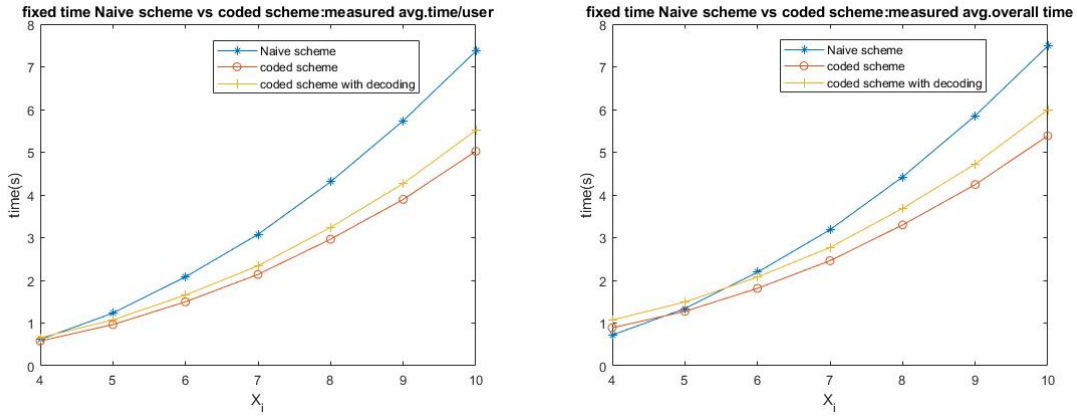
By viewing the table, we can find the theoretical communication cost for naive is twice as coded scheme. The $E_{Y_i} = E_{Y_{max}}$ when the pause time is fixed. After measured the actual condition the coded scheme is twice as naive scheme as well. Thus draw the plot as showed below figure 5.1 where compared between the naive scheme and coded scheme. As we can find the ratio for the communication between the naive scheme and the coded scheme is 1.43 when user=10.

# users	4	5	6	7	8	9	10
# files/user (N)	6	12	20	30	42	56	72
EY_i	0.6	1.2	2	3	4.2	5.6	7.2
EY_{max}	0.6	1.2	2	3	4.2	5.6	7.2
Measured avg. time/user	0.61	1.23	2.08	3.08	4.31	5.74	7.37
Measured avg. overall time	0.72	1.34	2.19	3.19	4.42	5.86	7.49

Table 5.1: Naive Scheme fix time for Reverse Index Coding

# users	4	5	6	7	8	9	10
# files/user (N)	3	6	10	15	21	28	36
EY_i	0.3	0.6	1	1.5	2.1	2.8	3.6
EY_{max}	0.3	0.6	1	1.5	2.1	2.8	3.6
Measured avg. time/user	0.58	0.96	1.5	2.14	2.96	3.9	5.02
Measured avg. time/user with decoding	0.66	1.07	1.65	2.34	3.24	4.37	5.51
Measured avg. overall time	0.89	1.27	1.81	2.46	3.3	4.24	5.39
Measured avg. overall time with decoding	1.07	1.49	2.08	2.77	3.69	4.73	5.99

Table 5.2: Coded Scheme fix time for Reverse Index Coding



(a) Measured avg. time/user (b) Measured avg. overall time

Figure 5.1: Fixed Pause time(0.1 seconds)

5.2 Exponential Distribution Pause Time

Recall section 3.2 compute the mean for exponential distribution by using the probability density function(PDF) which written as eq 3.7 and the mean of the equation is λ^{-1}. To keep consistency from section 3.1 the, mean should be equal to 0.1 and $\lambda = 10$. Let U be a

uniform random variable distributed between 0 and 1. The exponential distribution pause function written in the program is $X = -log(U(a,b))/\lambda$ by taking the advantage of uniform distribution. For the pause function λ=10, uniform function provide the range for a and b which is [0,1]. Hence, the function can be written as

$$X = -\frac{\ln(U)}{10}.$$

. Theoretical max pause time(EY_{max}) is different with theoretical pause time unlike the fix time. The theoretical max for distribution can be various of the interval changes. Let function of the distribution be independent and identically distributed(iid) then the density of f(x) is given by the equation 3.5, To find the expectation value need to check the exponential distribution function which can be written as equation **??** where N present number of users.

Because the simulation is not transfer one file, for each user will transfer multiple files at one time. Before discuss multiple users, first discuss the distribution if single user transfer multiple files to another user. The sum of N exponential random variables is a Gamma random variable, $\Gamma(\alpha, \beta)$, where $\beta = \frac{1}{\lambda}$. The λ is the parameter of exponential distribution α and λ can be found due to β and the mean of the exponential distribution equation[3]. The mean of the exponential distribution equation is the theoretical time, and β is the 0.1 which helps us to find the gamma distribution $\Gamma(M, 0.1)$ where M is the total amount of files that will transfer to other users in (2.2). The possibility of density function(PDF) can be expressed as (3.8) where $\Gamma(\alpha) = \int_0^\infty x^{\alpha-1}e^{-\alpha}dx$. Based on the equation 3.8 can generate plot the figure 3.2 for the exponential PDF function. For the legend at in each graph is the amount of file per each user(X_{ij}) for the transfer. The left figure 3.2a is the naive condition and the right figure 3.2b is coded scheme condition for exponential distribution. Use the result from section 3.2 because is hard to find the threshold for the reverser index coding. Thus, the plot for $E_{Y_{max}}$ and showed in figure 3.5b

Two table for the naive scheme(table 5.3) and the coded scheme(table 5.4) created based on $E_{Y_{max}}$ and other measured parameter. The first row of the table which lead by #users means N servers(users) are handling the problem of MapReduce. The second row #files/user stands for how many files that each user will transfer to other users. The row of EY_i stands for the theoretical value of the communication cost via the total amount of file will do the transfer work for each user. The row for EY_{max} is based (3.5) to find the maximum of communication cost for different amount of users. Next row is for every user when dealing with file transfer how would take for N users. the last row is for the server which did longest time. For the coded scheme has two more lines with named of decoding which means measured time not only with encoding(file transfer), also with decoding process which we assume took zero times. We choose the number of users to be from 4 to 10. For a given number of users, the experiment is done for 350 times, and the measured result is averaged over these times. The table can be plot in the figure 5.2.

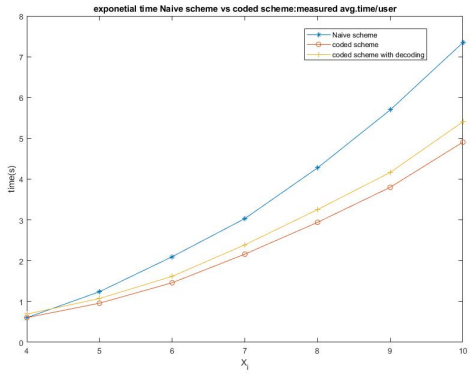
# users	4	5	6	7	8	9	10
# files/user (N)	6	12	20	30	42	56	72
EY_i	0.6	1.2	2	3	4.2	5.6	7.2
EY_{max}	0.87	1.63	2.6	3.78	5.17	6.76	8.56
Measured avg. time/user	0.6	1.24	2.1	3.03	4.27	5.7	7.35
Measured avg. overall time	1.1	1.96	3.07	4.21	5.74	7.51	9.4

Table 5.3: Naive Scheme Exponential Distribution for Reverse Index Coding

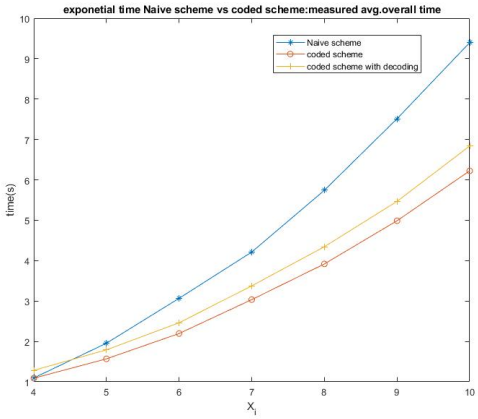
# users	4	5	6	7	8	9	10
# files/user (N)	3	6	10	15	21	28	36
EY_i	0.3	0.6	1	1.5	2.1	2.8	3.6
EY_{max}	0.49	0.96	1.46	2.16	2.94	3.8	4.91
Measured avg. time/user	0.6	0.96	1.46	2.16	2.94	3.8	4.91
Measured avg. time/user with decoding	0.69	1.07	1.61	2.39	3.25	4.16	5.4
Measured avg. overall time	1.09	1.57	2.2	3.04	3.92	4.99	6.22
Measured avg. overall time with decoding	1.28	1.79	2.16	3.38	4.34	5.47	6.84

Table 5.4: Coded Scheme Exponential Distribution for Reverse Index Coding

The measured time is longer than the expectation time($E_{Y_{max}}$) and the simulation for ex-



(a) Measured avg. time/user



(b) Measured avg. overall time

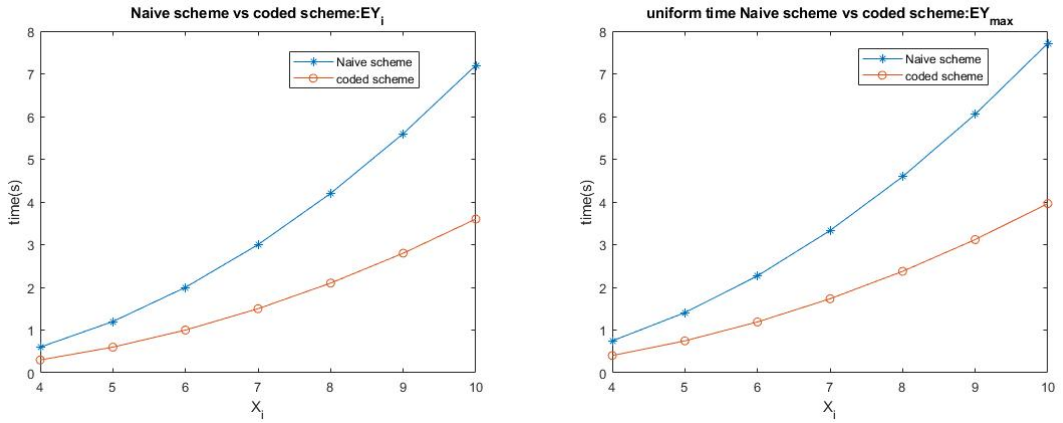
Figure 5.2: Exponential comparison between each user and overall for Reverse Index Coding

ponential distribution by comparing figure 3.6a vs figure 5.2a and figure 3.6b vs figure 5.2b. From table 5.3 and table 5.4 shown that the measured time result for reverse index coding is longer than the result in MapReduce which are table 3.3 and table 3.4. The reason why take longer because what mentioned in the section 4.5 which are file rename and transfer the file to binary. The over time should be the threshold value during the exponential distribution which is close to 0.034 second for each file. Also we can find the ratio for the communication between the naive scheme and the coded scheme is 1.49 when user=10.

5.3 Uniform Distribution Pause Time

As mentioned in the section 3.3, The mean of the uniform distribution is 0.1, also after the many files transfer from one user to others the distribution turn to be Irwin-hall distribution[9]. The PDF can be addressed in the equation 3.12 and CDF can be addressed in the equation 3.13. The equation 3.15 is to find the max of the expectation time which can be used to complete with the measured result. As mentioned in the section 5.2 the actual measured time will have threshold compare with the max time. So the expectation

of the measured time should be longer than $E_{Y_{max}}$. After running the simulation, two tables generated which are table 5.5 and table 5.6 for the naive scheme and the coded scheme. The first row of the table which lead by #users means N servers(users) are handling the problem of MapReduce. The second row #files/user stands for how many files that each user will transfer to other users. The row of EY_i stands for the theoretical value of the communication cost via the total amount of file will do the transfer work for each user. The row for EY_{max} is based (3.5) to find the maximum of communication cost for different amount of users. Next row is for every user when dealing with file transfer how would take for N users. the last row is for the server which did longest time. For the coded scheme has two more lines with named of decoding which means measured time not only with encoding(file transfer), also with decoding process which we assume took zero times. We choose the number of users to be from 4 to 10. For a given number of users, the experiment is done for 350 times, and the measured result is averaged over these times. The table can be plot in the figure 5.2. We



(a) fixed time for $E_{Y_{max}}$ (b) $E_{Y_{max}}$ for Uniform pause time

Figure 5.3: $E_{Y_{max}}$ comparison between Uniform Distribution pause time and fixed time

can compare between the E_{Y_i} vs. $E_{Y_{max}}$ after the $E_{Y_{max}}$ calculated. From the section 3.1, we know that the E_{Y_i} for any of the distribution same as the $E_{Y_{max}}$ at fixed pause time condition. Thus, comparing the figure as shown figure 5.3.

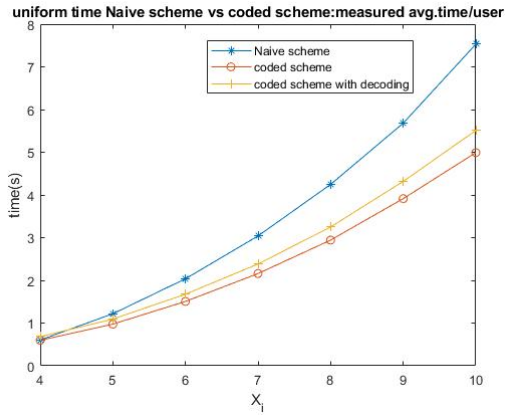
The plot 5.4 for measuring time per user and measured time for overall time can be generated

# users	4	5	6	7	8	9	10
# files/user (N)	6	12	20	30	42	56	72
EY_i	0.6	1.2	2	3	4.2	5.6	7.2
EY_{max}	0.75	1.41	2.27	3.33	4.59	6.05	7.72
Measured avg. time/user	0.6	1.22	2.04	3.04	4.25	5.68	7.53
Measured avg. overall time	0.92	1.65	2.61	3.74	5.14	6.7	9.18

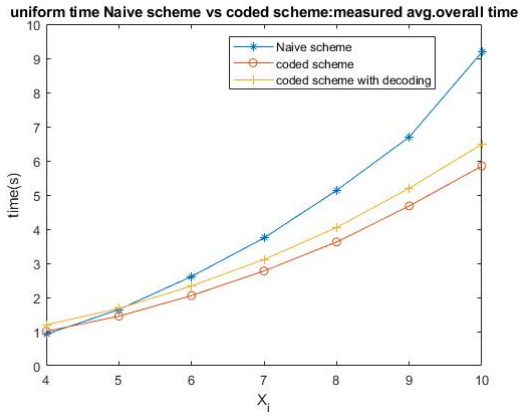
Table 5.5: Naive Scheme Uniform Distribution for Reverse Index Coding

# users	4	5	6	7	8	9	10
# files/user (N)	3	6	10	15	21	28	36
EY_i	0.3	0.6	1	1.5	2.1	2.8	3.6
EY_{max}	0.4	0.75	1.09	1.73	2.38	3.18	3.96
Measured avg. time/user	0.6	0.98	1.01	2.16	2.94	3.91	4.99
Measured avg. time/user with decoding	0.68	1.09	1.68	2.38	3.25	4.32	5.51
Measured avg. overall time	1.01	1.45	2.05	2.77	3.62	4.68	5.85
Measured avg. overall time with decoding	1.2	1.68	2.33	3.11	4.05	5.2	6.48

Table 5.6: Coded Scheme Uniform Distribution for Reverse Index Coding



(a) Measured avg. time/user



(b) Measured avg. overall time

Figure 5.4: Measured Uniform Pause time between time/user and overall time

based on two tables which are table 5.5 and table 5.6 as the naive scheme and the coded scheme. If we compared between two figures, the measured overall time for figure 5.4b is longer than the measured time per user in figure 5.4a. The reason why the overall time is longer is the nodes running time is different and overall time takes the longest time for the nodes, but time per user is finding the average of running time between different nodes. As

we can find the ratio for the communication between the naive scheme and the coded scheme is 1.5 when user=10.

5.4 Practical Exponential Distribution Pause Time

As mentioned in the section 3.4, we know that the total communication cost is $X_j \times 0.05$(seconds)+ $E_{Y_{max}}$ where $E_{Y_{max}}$ is showed in equation 3.11. The experiment measured for both the naive scheme and the coded scheme and showed in the table 3.7 and table 3.8 for the theoretical and experimental result. Two table for the naive scheme(table 5.3) and the coded scheme(table 5.4) created based on $E_{Y_{max}}$ and other measured parameter. The first row of the table which lead by #users means N servers(users) are handling the problem of MapReduce. The second row #files/user stands for how many files that each user will transfer to other users. The row of EY_i stands for the theoretical value of the communication cost via the total amount of file will do the transfer work for each user. The row for EY_{max} is based (3.5) to find the maximum of communication cost for different amount of users. Next row is for every user when dealing with file transfer how would take for N users. the last row is for the server which did longest time. For the coded scheme has two more lines with named of decoding which means measured time not only with encoding(file transfer), also with decoding process which we assume took zero times. We choose the number of users to be from 4 to 10. For a given number of users, the experiment is done for 350 times, and the measured result is averaged over these times. The table can be plot in the figure 5.5.

Based on the result, a new comparison between measured average time per user and overall time and plot in the figure 5.5. The overall time is longer than the measured time/user as expected near 1 second longer when the total amount of user is 10. Then plotting the measured time for the naive scheme from table 5.7 and the coded scheme from table 5.8 showed in the figure 5.5. As we expected the measured overall time is longer than the

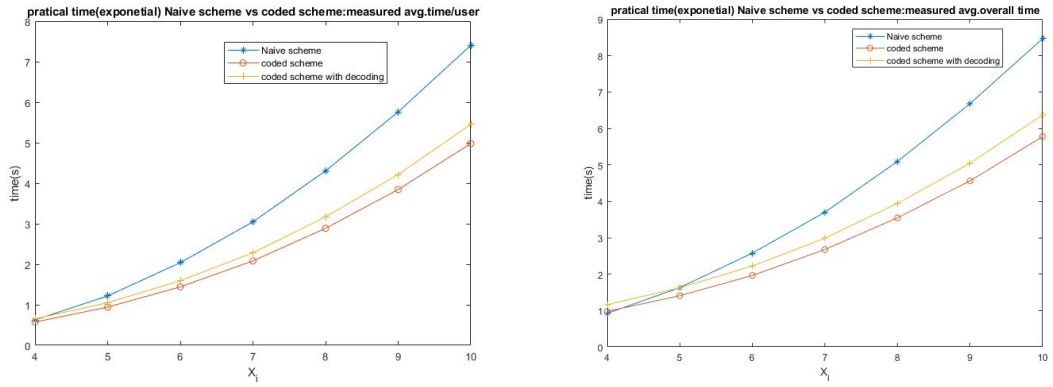
# users	4	5	6	7	8	9	10
# files/user (N)	6	12	20	30	42	56	72
EY_i	0.6	1.2	2	3	4.2	5.6	7.2
EY_{max}	0.73	1.41	2.3	3.39	4.68	6.18	7.88
Measured avg. time/user	0.63	1.22	2.04	3.05	4.31	5.76	7.41
Measured avg. overall time	3.92	1.63	2.58	3.7	5.09	6.68	8.46

Table 5.7: Naive Scheme Practical Exponential Distribution for Reverse Index Coding

# users	4	5	6	7	8	9	10
# files/user (N)	3	6	10	15	21	28	36
EY_i	0.3	0.6	1	1.5	2.1	2.8	3.6
EY_{max}	0.39	0.75	1.22	1.78	2.48	3.22	4.09
Measured avg. time/user	0.57	0.94	1.44	2.08	2.89	3.84	4.98
Measured avg. time/user with decoding	0.65	1.05	1.59	2.28	3.17	4.21	5.45
Measured avg. overall time	0.98	1.41	1.96	2.68	3.55	4.56	5.78
Measured avg. overall time with decoding	1.16	1.62	2.22	2.98	3.94	5.04	6.37

Table 5.8: Coded Scheme Practical Exponential Distribution for Reverse Index Coding

measured for average time for every users. As we can find the ratio for the communication between the naive scheme and the coded scheme is 1.48 when user=10.

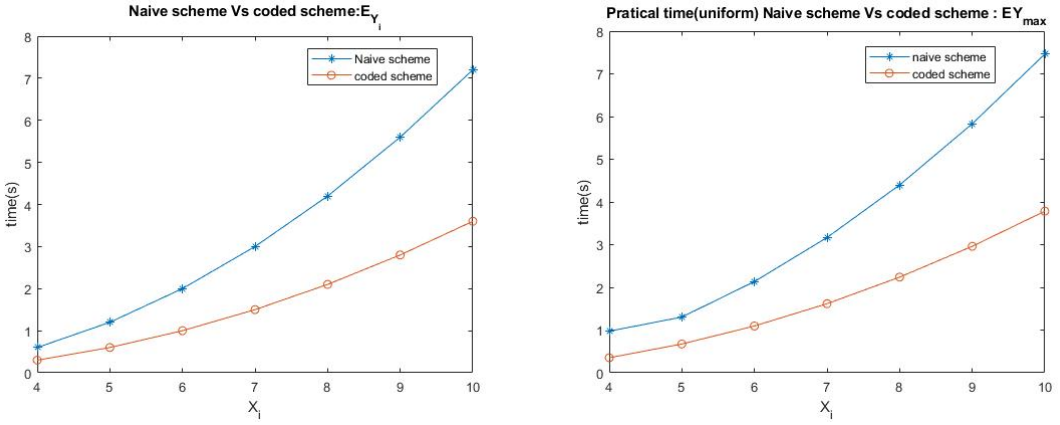


(a) Measured avg. time/user (b) Measured avg. overall time

Figure 5.5: Measured for avg. time/user and avg. overall time

5.5 Practical Uniform Distribution Pause Time

Same as what we mentioned in section 3.5, the $E_{Y_{max}}$ plot generated and make a simple comparison, The legend is the scheme that choosing, the x-axis is the amount of users, and the y-axis is the communication cost. As found that the plot are similar so we expect the result should be same as the fixed time $E_{Y_{max}}$. All the value has been generated or



(a) fixed time for $E_{Y_{max}}$ (b) $E_{Y_{max}}$ for practical uniform pause time

Figure 5.6: $E_{Y_{max}}$ comparison between practical uniform pause time and fixed time

measured, thus created two table below which are the naive scheme(table 5.9)and the coded scheme(table 5.10). The row for EY_{max} is based (3.5) to find the maximum of communication cost for different amount of users. Next row is for every user when dealing with file transfer how would take for N users. the last row is for the server which did longest time. For the coded scheme has two more lines with named of decoding which means measured time not only with encoding(file transfer), also with decoding process which we assume took zero times. We choose the number of users to be from 4 to 10. For a given number of users, the experiment is done for 350 times, and the measured result is averaged over these times. The table can be plot in the figure 5.7 where the legend stands for the different scheme, the x-axis is the total amount of user and y-axis is the time that for the communication cost.

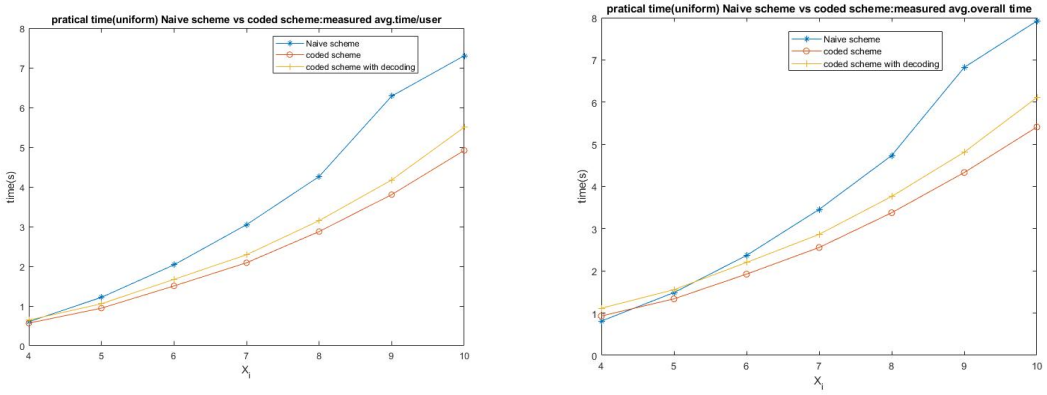
As we can find the ratio for the communication between the naive scheme and the coded

# users	4	5	6	7	8	9	10
# files/user (N)	6	12	20	30	42	56	72
EY_i	0.6	1.2	2	3	4.2	5.6	7.2
EY_{max}	0.67	1.3	2.14	3.17	4.4	5.83	7.46
Measured avg. time/user	0.6	1.22	2.04	3.05	4.26	6.29	7.3
Measured avg. overall time	0.81	1.49	2.37	3.46	4.74	6.82	7.92

Table 5.9: Naive Scheme Practical Uniform Distribution for Reverse Index Coding

# users	4	5	6	7	8	9	10
# files/user (N)	3	6	10	15	21	28	36
EY_i	0.3	0.6	1	1.5	2.1	2.8	3.6
EY_{max}	0.35	0.067	1.1	1.62	2.24	2.96	3.78
Measured avg. time/user	0.57	0.95	1.51	2.09	2.88	3.81	4.92
Measured avg. time/user with decoding	0.65	1.06	1.67	2.29	3.15	4.17	5.5
Measured avg. overall time	0.93	1.33	1.92	2.55	3.38	4.33	5.41
Measured avg. overall time with decoding	1.12	1.56	2.2	2.87	3.77	4.81	6.11

Table 5.10: Coded Scheme Practical Uniform Distribution for Reverse Index Coding



(a) Measured avg. time/user

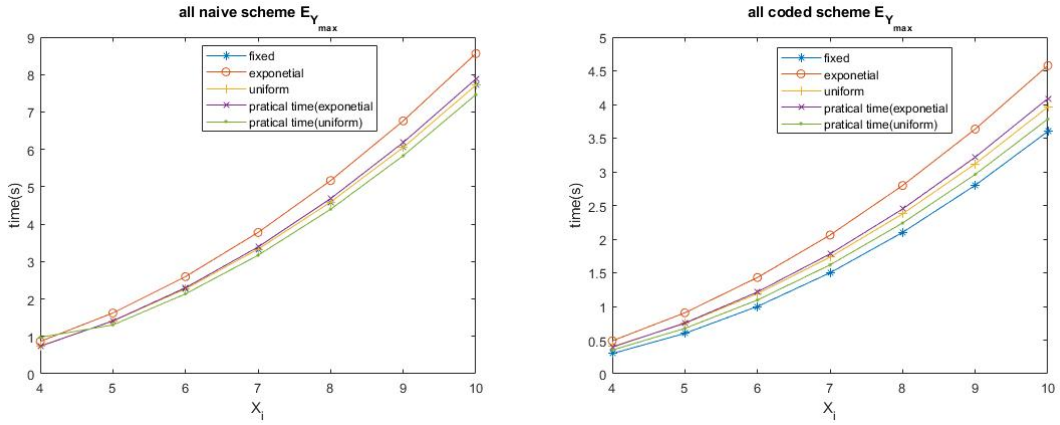
(b) Measured avg. overall time

Figure 5.7: Measured time between average time per user and overall time

scheme is 1.51 when user=10.

5.6 Discussion Between Pause Time for Reverse Index Coding

Quick summary the reverse index coding by compare all the coded table can find the cost for decoding took longer time than MapReduce, but max time decoding took longer time because each server start time is different which means the max time with decoding will look like a slightly longer than the max time. First, want to check the $E_{Y_{max}}$ for all the condition for the naive scheme and the coded scheme showed in figure 3.26. By looking at the graph knowing that both naive scheme and the coded scheme the exponential distribution cost the most time. The practical uniform distribution is the one most close to the fixed time. The legend are the communication parameters, x-axis is the total amount of users, and the y-axis is the communication cost.

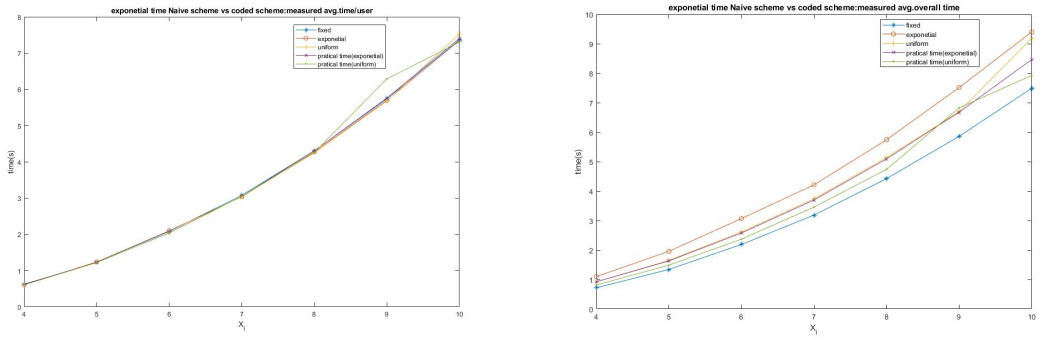


(a) Naive Graph($E_{Y_{max}}$) (b) Coded Graph($E_{Y_{max}}$)

Figure 5.8: The two images are for naive scheme and coded scheme $E_{Y_{max}}$.

Knowing the theoretical value is not enough, therefore check the measured average time per user is very important. So plot in figure 5.9. The naive scheme for every single user cannot find the difference, but naive scheme overall running time satisfy the ideal model, $E_{Y_{max}}$. The legend are the communication parameters, x-axis is the total amount of users, and the

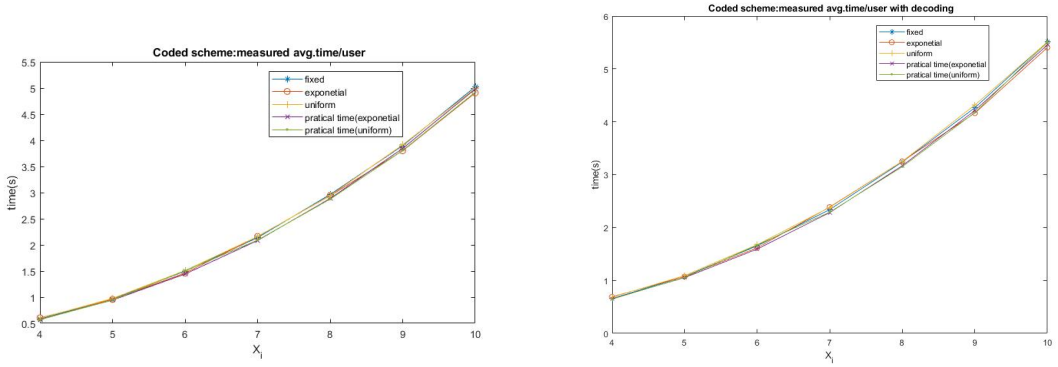
y-axis is the communication cost.



(a) Naive scheme measured avg. time/user (b) Naive scheme measured avg. overall time

Figure 5.9: The.

After plot the naive scheme why not to check the coded scheme. Thus, plot the graph as shown in the figure 3.28. Similar as naive scheme. The coded scheme showed all the curves are about same and this is to prove the coded scheme are more stable than the naive scheme during the file transmission steps. The legend are the communication parameters, x-axis is the total amount of users, and the y-axis is the communication cost. Last, check the overall

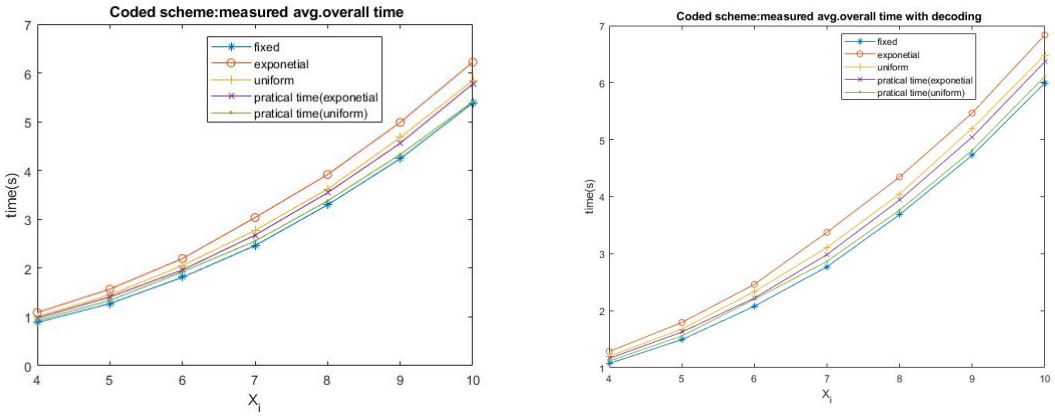


(a) Coded scheme measured avg. time/user (b) Coded scheme measured avg. time/user with decoding

Figure 5.10: Coded scheme measured avg. time/user.

time if the coded scheme also satisfy as the previous result from figure 3.28. This means after multiple user doing the file transfer. The function from exponential distribution and uniform

distribution to gamma distribution and Irwin-hall distribution. Also meet the expectation as we want. The legend are the communication parameters, x-axis is the total amount of users, and the y-axis is the communication cost.



(a) Coded scheme measured avg. overall time

(b) Coded scheme measured avg. overall time with decoding

Figure 5.11: Coded scheme measured avg. overall time

Chapter 6

Concluding Remarks

In this thesis, we mainly studied the coding techniques for information storage regarding the MapReduce problem. We have discussed MapReduce and coded MapReduce, and implement the MapReduce programming model to a reverse index coding to find the most popular websites for UC Irvine.

There are still a lot of open problems in storage. We only list a few possible directions as examples. For the reverse index coding problem, we want to implement the program more realistic and meet the real world by adjusting the communication time with transmitted file size. For a large file, size should take longer time than what is the threshold for the file size. If we success changes the communication time what happen for the communication cost for the average time per user and what happens for the overall time? Should we expect to see the measured average time per each server keep the same, but overall time increased. If so, what about the ratio between the coded scheme and the naive scheme? The asynchronous reverse index problem is another interesting problem besides of the communication time changes with transmitted file size. The idea can be implemented based on another programming model named Sandblaster L-BFGS[4]. In this case what happens for the communication

cost. Will we save more time for the file transfer between different servers?

In the network communication problems, two questions listed are waiting to be solved. If the map, unlike MapReduce, is not fully connected map how do we complete the task by using the coded scheme? Next, EC2 is the most popular cloud machine from Amazon Web Services(AWS)[11]. How do we finish all the task that previously stated can be finished and what is the communication cost for the real system will look like? Does the result of EC2 will generate the same the result that I found in the local machine? List problems above worth to continue discovery.

Bibliography

[1] D. Borthakur. HDFS architecture guide. Hadoop Apache Project. pages 1–14, 2008.

[2] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *JTN Weekly*, 22 Sept 6:6, 1996.

[3] R. D. Gupta and D. Kundu. Generalized exponential distributions. *Australian and New Zealand Journal of Statistics*, 41(2):173–188, 1999.

[4] K. C. Jeffrey Dean, Greg S. Corrado, Rajat Monga and A. Y. N. Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang. Large Scale Distributed Deep Networks. *Cambridge University Press*, pages 1–9, 2012.

[5] M. A. Y. Q. S. A. A. Li, Songze Maddah-Ali. A fundamental tradeoff between computation and communication in distributed computing. *IEEE Transactions on Information Theory*, 64:109–128, 2018.

[6] S. Li, M. A. Maddah-ali, and A. S. Avestimehr. Coded MapReduce. pages 1–16, 2015.

[7] S. A. W. S. A. B. E. L. M. D. D. E. Long. Ceph: A Scalable, High-Performance Distributed File System Sage. *The Journal of Hellenic Studies*, 30(01):109–132, 2006.

[8] M. A. Maddah-ali and U. Niesen. Fundamental Limits of Caching. (July):1–19, 2013.

[9] J. E. Marengo, D. L. Farnsworth, and L. Stefanic. A Geometric Derivation of the Irwin-Hall Distribution. *International Journal of Mathematics and Mathematical Sciences*, 2017:1–6, 2017.

[10] F. H. C. Oates and S. Ross. *A First Course in Probability*, volume 61. 2007.

[11] K. Rohloff and R. E. Schantz. High-performance, massively scalable distributed systems using the MapReduce software framework. pages 1–5, 2011.

[12] L. Song, C. Fragouli, and T. Zhao. A Pliable Index Coding Approach to Data Shuffling. (Isit 2017):1–41, 2017.