DATA REPLICATION STRATEGIES IN CLOUD COMPUTING

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Yang Liu

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

July 2011

Fargo, North Dakota

North Dakota State University
Graduate School

Title

DATA REPLICATION STRATEGIES

IN DISTRIBUTED SYSTEMS

By

YANG LIU

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

ABSTRACT

Liu Yang, M.S., Department of Computer Science, College of Science and Mathematics, North Dakota State University, July 2011, Data Replication Strategies in Cloud Computing. Major Professor: Dr. Weiyi Zhang.

Data replication is a widely used technique in various systems. For example, it can be employed in large-scale distributed file systems to increase data availability and system reliability, or it can be used in many network models (e.g. data grid, Amazon CloudFront) to reduce access latency and network bandwidth consumption, etc. I study a series of problems that related to the data replication method in Hadoop Distributed File System (HDFS) and in Amazon CloudFront service. Data failure, which is caused by hardware failure or malfunction, software error, human error, is the greatest threat to the file storage system. I present a set of schemes to enhance the efficiency of the current data replication strategy in HDFS thereby improving system reliability and performance. I also study the application replication placement problem based on an Original-Front sever model, and I propose a novel strategy which intends to maximize the profit of the application providers.

ACKNOWLEDGMENTS

First, I wish to express my sincere gratitude to my supervisor, Professor Weiyi Zhang, for his patience, enthusiasm and encouragement. His logical way of thinking and his knowledge have been of great value for me in these two years. I could not have imagined having a better advisor for my Master's study.

Besides my advisor, I would like to thank all of my thesis committee members: Dr. Kong Jun, Dr. Chao You, and Dr. Changhui Yan, for their insightful comments and encouragement. I owe my deep gratitude to Dr. Jun Zhang for her valuable advice and friendly help for my Master's thesis. I also want to thank my lab-mates in Visual Computing And Advanced Network (ViCAN) Laboratory: Shi Bai, Farah Kanda and Yashaswi Singh for all of your help in these two years.

Last but not the least, my deepest gratitude goes to my family for their unflagging love and support throughout my life. And I also owe my deepest loving thanks to my bride Xiaoqing Luo. Only six days after we got married I had to come back to the U.S. Without her encouragement and understanding, it would have been impossible for me to finish this work.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1. INTRODUCTION

Cloud computing, as the newest technology, might change the way we work and the way we use hardware, software and the internet. From governments to public and private organizations, cloud computing is significantly and fundamentally influencing the IT landscape. Combining lots of technologies, such as distributed computing, distributed file storage, virtualization, etc., cloud computing venders intend to provide the computing and storage resource as a kind of utility, such as water and electricity, to us. Amazon, one leader of the cloud computing providers, has found a commercial model of cloud computing successfully. In this work, I study the data replication strategies which have been used in two of the Amazon products, Amazon Elastic MapReduce [15] and Amazon CloudFront [16], which are two commercial products that have been launched by the department of Amazon Web Service. Briefly speaking, Amazon Elastic MapReduce is a distributed computing framework which could support the customers by processing vast amounts of data easily and cost-efficiently. Amazon CloudFront service uses data replication technique to deliver the content to the clients with low latency, and high data transfer speed by using a global network of edge locations which is also named front servers.

Data replication is a widely used technique in various systems and networks for distinct purposes. For example, data replication can be employed in large-scale data storage system to protect the data from data loss; it can also be used in many network models or web services to reduce access latency and increase data availability, such as in Amazon CloudFront service or in Peer-to-Peer systems.

In Chapter 2 and Chapter 3, I study the data replication problem in the Hadoop system which is one of the core platforms in Amazon Elastic MapReduce. As a widely used distributed storage system and distributed computing platform, Hadoop consists of two major components: 1) Hadoop distributed file system (HDFS) [3], the primary distributed storage system used by Hadoop; 2) MapReduce, a programming framework developed by Google [2] for processing large amounts of data in parallel. To the best of our knowledge, Hadoop system is employed by many big companies such as Yahoo!, Facebook, Amazon, etc.

Hadoop Distributed File System is designed for running on large-scale lost-cost commodity hardware. However, one of the greatest problems for the commodity hardware is frequent and permanent hardware failure. Intending to alleviate permanent data loss, in HDFS, data replication scheme is adopted.

Previous works have proved that, a delicately designed replica placement strategy can not only improve the reliability of the file system, but also enhance the performance of the MapReduce application in Hadoop system [7], [8]. However, the current data replication policy in HDFS has a wide space for research studies. The replication factor is decided by the users subjectively which could cause data redundancy or unsatisfied reliability. Moreover, when the current replica placement strategy in HDFS selects computers for storing the data, the condition of the computers have not been considered sufficiently. Therefore, in Chapter 2 and 3, I intent to find a more efficient date replication strategy which could improve system reliability and performance.

In Chapter 4, I focus on an application replication placement problem based on an

original-front sever model which is used by Amazon CloudFront [16].

1.1 HDFS and MapReduce

Intending to store and manage enormous data sets reliably and efficiently, Google developed a distributed file system named Google File System [1] and a distributed computing software model named MapReduce [2]. Based on Google File System and MapReduce software framework, Apache Software Foundation develops the open source project named Hadoop [5]. Fig. 1 illustrates the architecture of the Hadoop Distributed File System.

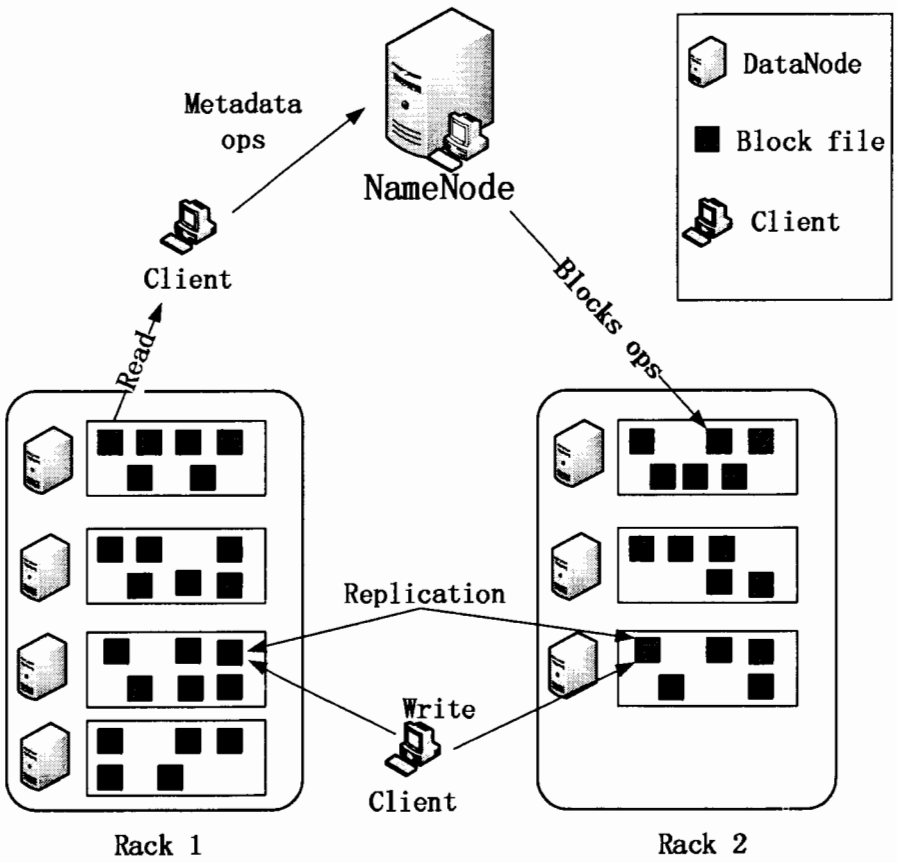


Fig. 1 Hadoop Distributed File System Architecture

There are mainly two types of nodes in HDFS architecture operating in

master/slave model: a single NameNode and multiple DataNodes, where the NameNode is in charge of the management of the file system metadata and namespace, and the DataNodes are the physically container of the data. The files in HDFS can be very large, typically from gigabytes to terabytes. Intending to provide high throughput access for the clients, the files are first divided into sequences of blocks and stored on different nodes. All blocks of a file are the same size (e.g. 64 MB, 128 MB or more) except the last block. A single DataNode can host millions of blocks files. Data coherency issues are solved by using a write-once-read-many access model in HDFS, which means once a file is written down and closed, it cannot be changed. The NameNode maintains the physical location of each block file on DataNodes and the namespace of the system.

The MapReduce framework adopts a *master/slave* model, in which there is a single master named *JobTracker* and multiple slaves named *TaskTracker*. The master is in charge of the schedule of the tasks on the slaves, while the slaves will perform the tasks assigned by the master. In Hadoop system, because HDFS is running on the same set of machines with MapReduce framework, the tasks are scheduled efficiently on the DataNodes where data is already stored.

1.2 Data Replication

Because an HDFS instance may have thousands of commodity machines, hardware failure happens frequently. Therefore, one of the core architectural targets of HDFS is to design an efficient scheme to protect data from data loss. The general idea of data replication is to store multiple copies of the file in distinct locations; when one replica or multiple replicas of a file fail, as long as there is still one replica of this file survived,

the rebuilding process can be initiated to create the copies of this file on other survived nodes. The data replication policy which is used in HDFS needs to solve two problems: 1) how many replicas are necessary? 2) what is the best placement strategy for placing these replicas?

1.2.1 Replication Factor

Replication factor does not only affect the reliability of the files, but also impact the performance of the MapReduce applications in Hadoop. As shown in [5], when the replication factor is changed from *3* to *2*, the performance of the MapReduce application is degraded. In contrast to the current replication factor scheme used in HDFS, In Chapter 2, I propose a dynamic replication factor decision strategy that the NameNode could calculate and dynamically adjust the replication factor for each file.

1.2.2 Replica Placement Strategy

A reasonable replica placement policy could reduce access latency, facilitate load balancing, as well as saving network bandwidth consumption. At present, the files in HDFS are replicated on multiple DataNodes with a *rack-aware replica placement* policy [3], [5]. The computers in a large cluster which is running HDFS normally spread across lots of racks. The communication between two nodes in different racks could cost more network bandwidth compared to the consumption between two nodes in the same rack. On the other side, the replicas spreading in multiple racks could prevent data loss when the entire rack fails during some unexpected situations. The current policy achieves the tradeoff between the network bandwidth utilization and the reliability of the system. However, it does not fully consider the workload balancing of

the DataNodes. In Chapter 3, I propose a file type aware replica placement strategy for improving the placement policy currently employed in HDFS.

1.3 Original-Front Server Model

The network model used in Amazon CloudFront service can be described as an original-front server model. Data is initially stored on the original server, and the replicas of the data are sent to the front servers which are located close to the customers. In this model, the applications which are ordered by the customers are delivered to the suitable front servers. Particularly, in our work, three parties are involved in this model: 1) cloud service vender, 2) application provider, and 3) application customer. Application providers create the applications and put them on the original storage server which is provided by the cloud service vender. Customers purchase the applications from the application providers, and they always expect the applications could be delivered with low latency. However, it is difficult for the original server to satisfy all requests due to the long distance or the network congestion. As a consequence, the provisioning time could be longer than the expected time. Intending to solve this problem, data replication strategy is adopted by the cloud service vender.

The contribution of our work is in three respects. Firstly, comparing with the current replication factor scheme used in HDFS, our dynamic replication factor decision strategy could improve both the system reliability and resource utilization balancing. Furthermore, the performance of the MapReduce application in Hadoop is enhanced due to the suitable replication factor used by the system. Secondly, I present a file type aware replica placement strategy which is operated in two steps. The first is to evaluate

the value of the DataNodes with Analytic Hierarchy Process. The second is to select the DataNode by using the method of Roulette Wheel Selection. The analysis results show that the resource unitization of the system tends to achieve balancing by using our new strategy. Thirdly, based on the original-front server model, the application replication problem is presented from a novel aspect which is to maximize the profit of the application providers. Two efficient heuristic algorithms are proposed to solve this problem.

1.4 Work Overview

In this work, I have three major topics. In Chapter 2, the replication factor decision problem in HDFS is studied and a dynamic replication factor decision strategy is proposed. In Chapter 3, I study the replica placement strategy in HDFS, and I propose a file type aware replica placement strategy intend to achieve resource utilization balancing. In Chapter 4, I study the application replication placement problem based on an Original-Front sever model, and I propose a novel strategy which intends to maximize the profit of the application providers.

All of the conclusions of these three problems are made in Chapter 5.

CHAPTER 2. REPLICATION FACTOR DECISION

2.1 Introduction

Data failure, which is caused by hardware failure or malfunction, software error, or human error, is the greatest threat for any data storage system. In order to achieve data reliability and durability, various redundancy schemes have been used in large-scale data system. Data replication is one of the widely used schemes [1], [8], [9]. A data replication policy essentially studies two major issues: 1) the replication factor and 2) the replica replacement strategy.

As previously mentioned, the replication factor could be configured by the HDFS users file to file subjectively. The default replication factor in HDFS is *3*. If the users realize that some files are more important than others, they would set the replication factor of these files to a higher value. Not surprisingly, in this way whether replication factor is adequate for the files or not depends on the experience of the users; in other words, this strategy is too subjective to achieve high precision.

Traditionally, estimating the replication factor of a file is a tradeoff between space consumption and reliability; whereas in Section 2.3 I analyze the replication factor problem in HDFS from a novel aspect, and then in Section 2.4 I present a method to calculate the replication factor dynamically and periodically.

2.2 Related Work

In [18], by using the particular measure of reliability of the storage system, named probability of data loss during rebuilding process, the upper and lower bounds are defined. The numerical results in [18] proved that this measurement is reasonable for

estimating the reliability of the system.

In [31], the authors proposed a dynamic content distribution system named dissemination tree based on a peer-to-peer location service. The number of replicas and the network bandwidth consumption are significantly reduced.

In [5], the authors show us replica placement in HDFS does not only affect the reliability of the system, but also affect the MapReduce processing efficiency in the Hadoop project. From the experiment results, we can see the MapReduce processing rate is higher when the replication factor of the files is 3 than when the factor is 2.

2.3 Replication Factor Decision Problem

Given a large cluster deployed with HDFS, there are a single NameNode, and n DataNodes mapped into k racks. By considering the availability and reliability of the system, block files will be placed on multiple DataNodes. For the block file with r_f replicas, these replicas are distributed in r_f out of n DataNodes when $r_f < n$.

The determination of the reasonable replication factor requires an assumption: each file has an assigned level of importance on a scale of 1 to 5. The levels have corresponding probabilities of data loss. The probability of data loss of a file with r replicas deployed on r DataNodes could mean the probability of all of the r DataNodes fail before this file is successfully copied to an available DataNode.

The authors in [2] demonstrate the policy exploited by the MapReduce programming framework in Hadoop for arranging *map* and *reduce* tasks. When the file f_i is requested for analysis by a MapReduce job, the *JobTracker* [2] in Hadoop system attempts to schedule *map* tasks on the DataNodes which hosted the block files of f_i.

However, among these DataNodes, some of them may already have many other tasks been scheduled. Due to the competition for CPU or memory, the processing time for some of the new tasks becomes unacceptable probably. These tasks are marked as *"stragglers"* [2]. The mechanism that has been used in MapReduce is these *"straggler"* tasks would be rescheduled and processed on other DataNodes. For example, if there is a block file bf_i replicated on three DataNodes, but all three DataNodes are occupied when the *map* task calls, bf_i should be copied to another idle DataNode to complete the operation. This process does not only increase the processing time, but also increases network bandwidth consumption which is recognized as a scarce resource. Obviously, if the file f_i could have more replicas $r_f > 3$ distributed on different machines, for a single block file, the probability that all r_f DataNodes are occupied simultaneously is less.

In contrast to the replication factor decision problems which address the tradeoff between the reliability and disk space consumption, I do not only consider the reliability, but also consider the tradeoff between the disk space consumption rate and the network bandwidth consumption rate.

Given a file f_j, if replication factor of f_j is relatively low compared with the optimal value, the disk space is saved, but on the other side, the transmission of f_j among the DataNodes may happens more. By contrast, if the replication factor is set to a relatively high value, more disk space is occupied instead of network bandwidth consumption. Therefore, our objective in this Chapter is to find the optimal replication factor to satisfy the reliability requirement and achieve the resource utilization balancing.

2.4 Dynamic Replication Factor Decision Strategy

In this section, I propose a two-phase dynamic replication factor decision strategy. Briefly speaking, in the first phase, the NameNode evaluates the replication factor for each file based on the corresponding reliability requirement; in the second phase, at each interval, the NameNode adjusts the replication factor for each file intend to achieve resource utilization balancing.

2.4.1. The First Phase

Assuming that each DataNode fails independently, we get the following two circumstances which could cause data loss:

1) The permanent failures occur simultaneously on r DataNodes. The files whose replicas are all distributed among these r DataNodes, then these files are identified as data loss.
2) Only one of replica out of r is available, in the other words, $(r-1)$ DataNodes which host the replicas of file f fail simultaneously. The DataNode DN_i which hosts the last replica of file f would rebuild this file by transferring it to the other available nodes. Data loss could occur if DN_i fails before the rebuild process is finished successfully.

The probability of the second condition is no less than the first condition, which means:

$$P\{\pi|(r-1)\ nodes\ failed\} \geq P\{r\ nodes\ failed\}$$

Where π is the event that data loss happens because the last DataNode which hosted f failure before the rebuilding process finished.

The second condition has been studied in [18]. The replica placement model in our work is the same as the de-clustered placement model mentioned in [18]. Hence I adopt

the results in [18] to calculate the replication factor which is able to satisfy the reliability requirement. The probability of data loss during the rebuilding process is bounded by the following function:

$$\frac{\lambda nc}{b}\frac{1}{\binom{n}{r-1}}\frac{1}{(1+\frac{\lambda c}{b})} \le \xi < \frac{2\lambda nc}{b}\frac{1}{\binom{n}{r-1}} \tag{2.4.1}$$

Where $\xi = P\{\pi | (r\text{-}1)\ nodes\ failed\}$, λ is the practical values of the failure rate, b is the node rebuild bandwidth, c is the node capacity, and n is the amount of nodes.

From formula 2.4.1 we can see that if we try to make the practical reliability no less than the expected reliability, we need to satisfy the upper bound of ξ in formula 2.4.1 for calculating the replication factor. If each importance level has a corresponding value of the probability of data loss, we can calculate the value of replication factor for each importance level by using formula 2.4.1. Table 1. shows us the practical probability of data loss for each importance level.

Table. 1 Probability of Data Loss to each Importance Level

Importance Level	Probability of Data Loss
1	$1*10^{-3}$
2	$1*10^{-4}$
3	$1*10^{-5}$
4	$1*10^{-6}$
5	$1*10^{-7}$

For instance, in a system, given the amount of DataNodes $n=100$, storage capacity of each node $c=12\ TB$, rebuild bandwidth available at each node $b=96\ MB/s$, mean time to failure of a node λ=1000 hours, if the importance level of the file f_i is 3, which means we expect that the probability of data loss of file f_i $\xi_{fi} \leqslant 10^{-5}$, replication factor

of f_i can be set as *4*. Because when r_{fi} = 4, the upper bound of ξ in formula (2.4.1) equals to $2.38*10^{-6}$, and *4* is the minimum number that can satisfy the expected probability of data loss of file f_i.

2.4.2. The Second Phase

In the first step, the NameNode has calculated the replication factor of the files independently according to the importance level. In the second step, I theoretically analysis the replication factor based on the resource utilization balancing between disk space and network bandwidth.

The resource utilization balancing is described as the tradeoff between the disk space consumption rate and the network bandwidth consumption rate. The tradeoff can be formulated as the following formula:

$$\frac{R_{DS}^{f_i}}{R_{NB}^{f_i}} = \mu \tag{2.4.2}$$

Where μ is a constant, $R_{DS}^{f_i}$ is the disk space proportion which is consumed by file f_i, and $R_{NB}^{f_i}$ denotes the network bandwidth proportion which is consumed by file f_i.

Given a file f_i, α_{fi} is the amount of blocks of f_i, bs is the size of each block, and r_{fi} is the current replication factor of f_i, storage consumption rate of f_i is calculated as:

$$R_{DS}^{f_i} = \frac{\alpha_{fi} \times bs \times r_{fi}}{\sum_{j=1}^{n} S_{DNj}} \tag{2.4.3}$$

Where $\sum_{j=1}^{n} S_{DNj}$ is the amount of the storage space of the file system.

Let us introduce several important concepts in term of MapReduce program based on the data replication strategy. In HDFS, most CPU resources of the DataNodes are

consumed by *map* and *reduce* tasks for data analysis. For a DataNode DN_i, if the CPU utilization rate remains at high, it is rarely able to respond to new tasks during this period. By using a practical threshold χ, when the CPU utilization rate of the DataNode DN_i over χ, we define this DataNode is in the *occupied-status*. Otherwise, it is defined in *idle-status*. For simplicity, during interval T_{in} (e.g., one day or one week), the probability that DN_i has enough CPU resource to respond to new tasks is calculated as:

$$P_{occu}^{DNi} = \frac{\sum t_{occu}^{DN_i}}{T_{in}} \tag{2.4.4}$$

Where $t_{occu}^{DN_i}$ represents the time when DN_i is in *occupied -status*.

Moreover, $\omega_{fi}^{T_{in}}$ represents the times of file f_i that called by the MapReduce jobs during the interval T_{in}. The amount of the blocks of f_i which are transferred to the other DataNodes to execute the *map* tasks in T_{in} is calculated as:

$$N_{trans}^{T_{in}} = \sum_{k=1}^{\alpha_{fi}} \left(P_{occu}^{DN_k} \right)^{r_{fi}} \times \omega_{fi}^{T_{in}} \tag{2.4.5}$$

The network bandwidth consumption rate $R_{NB}^{f_i}$ during T_{in} is calculated as:

$$R_{NB}^{f_i} = \frac{\sum_{k=1}^{\alpha_{fi}} \left(P_{occu}^{DN_k} \right)^{r_{fi}} \times \omega_{fi}^{T_{in}} \times bs}{B \times T_{in}} \tag{2.4.6}$$

Combining the formula 2.4.3 with the formula 2.4.6, we get the following function:

$$\frac{\alpha_{fi} \times bs \times r_{fi}}{\sum_{j=1}^{n} S_{DNj}} = \frac{\sum_{k=1}^{\alpha_{fi}} \left(P_{occu}^{DN_k} \right)^{r_{fi}} \times \omega_{fi}^{T_{in}} \times bs}{B \times T_{in}} \times \mu \tag{2.4.7}$$

By using formula 2.4.7, the replication factor of the file f_i is calculated during the time interval T_{in}, and then the NameNode will compare the value of r_{fi} with the value of r calculated in the first phase. If $r_{fi} > r$, then the replication factor of f_i is adjusted to

r_{fi}, otherwise, the replication factor of f_i will still be r.

2.5 Numerical Results

In this section, I evaluate the performance of our dynamic replication factor decision strategy. Given a HDFS cluster with *100* DataNodes, there are *10000* new files need to be uploaded to the cluster. Because the availability of the threshold of the probability of data loss during rebuilding process has been proved in [18], we did not evaluate the reliability of the system which is ensured by the first phase of the *DRFD* strategy. I compared the performance of our *Dynamic Replication Factor Decision (DRFD)* strategy and the *Current Replication Factor (CRF) strategy* in HDFS, in terms of *the ratio of the network bandwidth consumption rate and the disk space consumption rate*. I expect the ratio can achieve the constant μ as we set in different conditions. Fig. 2 illustrates that with the average CPU utilization rate of the system increasing, this ratio keeps on the value we set approximately by using our strategy. In this simulation, the value of the constant μ is *1*. I also observe that the ratio is very far from μ *by using the* current replication factor strategy. It means that our *DRFD* strategy has a better performance in terms of the resource utilization balancing. This conclusion can also be proved by Fig. 3, in which the Y-axis is the *Minimum Mean-Square Error (MMSE)* of the consumption rate of the network bandwidth and disk space.

Fig.4 shows that with the system gets busier, the average replication factor gets higher by using our DRFD strategy. By contrast, the current replication factor strategy in HDFS could not adjust the value dynamically.

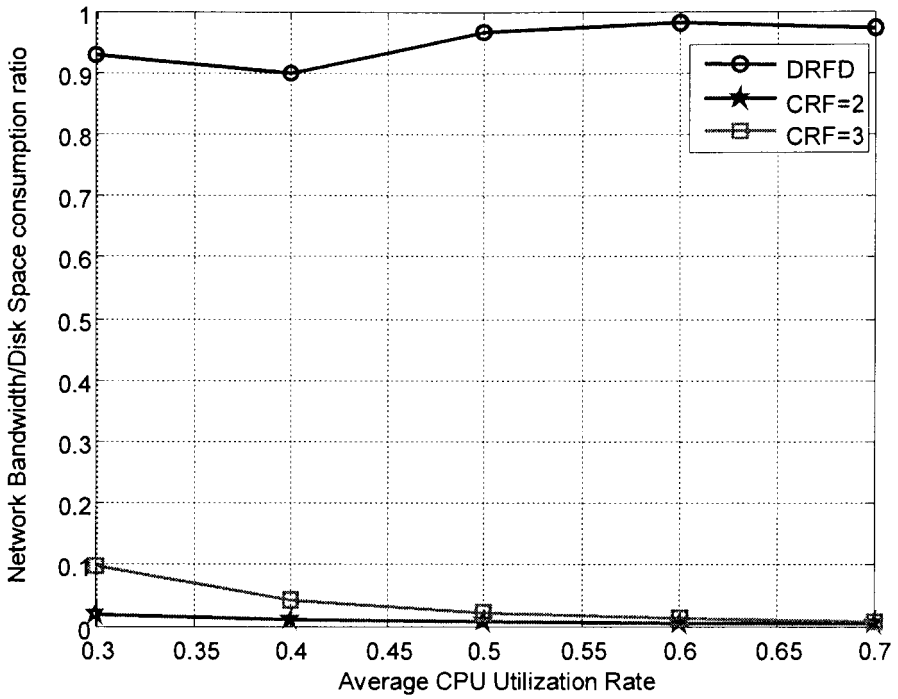


Fig. 2 Network Band/Disk Space Consumption Ratio

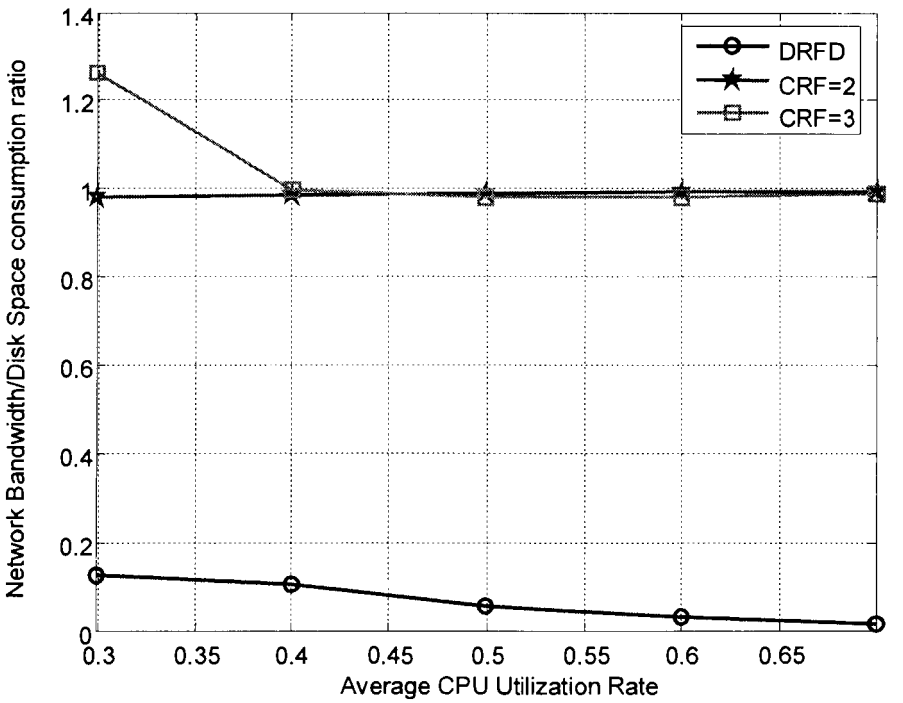


Fig. 3 MMSE of Network Bandwidth/Disk Space Consumption

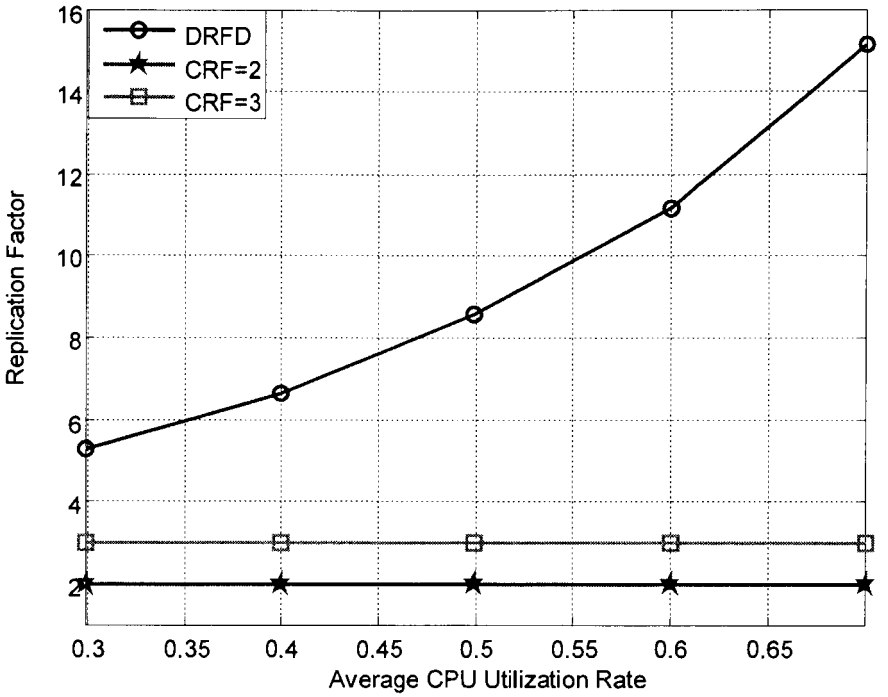


Fig. 4 Replication Factor

CHAPTER 3. REPLICA PLACEMENT POLICY

3.1 Introduction

The data replication policy needs to solve two problems: one is "how many replicas are necessary", and the other is "what is the best placement policy for placing these replicas". In Chapter 2, I have presented a dynamic strategy for estimating the replication factor; in this Chapter, I propose a novel replica distribution policy for improving the availability and the performance of HDFS.

The current replica placement policy in HDFS is called rack-aware replica placement policy. The nodes in a large cluster which is running HDFS normally spread across lots of racks.

The communication between two nodes from different racks could cost more network bandwidth compared with the nodes in the same rack. On the other side, the replicas of a file spreading in multiple racks could prevent data loss when the entire rack fails due to the unexpected situations. The following replica placement strategy is a tradeoff between the network bandwidth utilization and the reliability of the system:

1) The first replica is deployed randomly on the DataNode DN_1 in rack ra_i.
2) The DataNode DN_2 which is randomly chosen to host the second replica is also in rack ra_i. DN_1 streams the replica to DN_2.
3) The DataNode DN_3 which is randomly chosen to host the third replica is in the rack ra_j where $(i \neq j)$. The replica is streamed from DN_2 to DN_3.
4) The subsequent replicas are randomly placed in the DataNodes and should always obey the rule that no two replicas are placed in the same DataNode and

no more than two DataNodes host the same replica in the rack if there are enough DataNodes.

This replica placement policy is a simple way for achieving the tradeoff between the reliability and network bandwidth cost. However, the main issue in the whole procedure is that the random selections are used frequently without considering the condition of the selected DataNodes. As a consequence, unbalanced workload of the DataNodes occurs in the cluster. Although HDFS develops a tool *Balancer* to balance disk space usage when some DataNodes become full or new nodes join the cluster, it takes lots of time and costs extra network bandwidth. Therefore, intending to overcome the disadvantages of the current replica placement policy, I propose a file type aware replica placement strategy for selecting the optimal candidate nodes for the coming files.

By investigating the usages of the HDFS [19] [3], we find three main features:

1) HDFS is designed to store large-scale data reliably.

2) HDFS provides fast access to the data when a large number of clients launching reading requests.

3) HDFS cooperates well with MapReduce framework to process complex data analysis.

The types of the files stored in the HDFS have been studied in previous works [2], [3]. After analyzing the file types mentioned in these works, we recognize that the types can be divided into three categories based on the computer resources they mainly consumed:

1) Computing-type file

This type of file, such as crawled documents, web request log, etc. [2], needs to be executed by MapReduce application to analysis different derived data, such as inverted indices, graph structure of web documents, etc. The processing time of many large-scale files can last for several hours even several days.

2) I/O throughput-type file

Some companies employ HDFS for storing streaming media files, such as videos and figures, because HDFS provides high throughput access to the data. This type of file does not occupied CPU resources, but it consumes lots of I/O throughput resource of both the disk and the network. As a well designed distributed file system, HDFS overcomes the I/O throughput bottleneck issue which occurs in the single disk. For example, the I/O throughput of a standard SATA hard disk is 60 MB/S, which has became a big bottleneck for the computer. HDFS could solve this limitation by dividing the files into multiple block files and placing them on multiple nodes. When the reading requests are coming, data can be accessed in parallel from multiple nodes.

3) Storing-type file

For many companies, there are a vast amounts of data need to be stored reliably. The data may rarely be operated. But the reliability of the storage system is very important. In this situation, the main motivation to use HDFS is that HDFS could provide low-cost storage space with high reliability and durability.

For a typical computer, we could separate its capacities into three types: *computing capacity, I/O throughput capacity,* and *storage capacity.* Correspondingly, different types of files as I mentioned could primarily consume one of these three capacities of a computer. Intending to utilize the computer resource reasonably, we should avoid the unbalanced capacity consumption of the DataNodes. For example, if the storage idle rate of a DataNode DN_i is *10%*, but the average CPU idle rate of DN_i is *80%*, it indicates that there is a waste for DN_i's computing capacity. To avoid this situation, I aim to utilize the various capacities of a single DataNode as balanced as possible. For choosing the optimal DataNodes to host the block files, we should estimate the various capacities of the DataNodes comprehensively. In this work, a powerful tool for analysis of complex decision problems which called *Analytic Hierarchy Process (AHP)* is applied for evaluating the DataNodes in terms of three capacities. Then the values of all DataNodes are sent to the NameNode for decision making.

3.2 Related Work

Data replication is a widely used strategy to protect the data storage systems, especially the large ones, from data loss. By reasonably deployed multiple replicas of a file into different instances in the system, the lost data can be rebuilt quickly as long as there is one available instance which hosts the data file.

In [3], the researchers state the current widely-used replica placement policy of HDFS, and they also show the analysis reports collected from a cluster using HDFS to manage *25* petabytes of enterprise data at Yahoo!

To defend against frequent failure in the large cluster, various approaches have been

adopted by different enterprises and distributed file systems. Such as Redundant Arrays of Inexpensive Disks [11], which is well-known as a single disk failure tolerant technology, has been used by some distributed files systems like PVFS [12] and Lustre [14]. DiskReduce [15] is an application of RAID in HDFS to save storage capacity. In other words, it is a modification of the HDFS to asynchronously replace replicas of block files with RAID 5 and RAID 6 encodings.

However, to the best of our knowledge, RAID could only protect against failure that happened on the disk, which denotes that if the machine becomes unavailable due to some others reasons, the data hosted by this machine will be non-functional.

The *Analytical Hierarchy Process (AHP)* is a decision-aiding method developed by Saaty [20]. AHP is most widely used in analyzing feasible alternatives when multiple criteria need to be considered on a rational basis. Based on the judgment of the decision-maker, which may be one people or a group, it aims at evaluate a set of alternatives on a ratio scale [24].

3.3 Replica Placement Policy

In Chapter 2, I have studied the problem of how many replicas are necessary for a single file. In this section, I study the problem that where these replicas should be placed. The current replica placement policy in HDFS is considered to be a sound policy in terms of reliability and network bandwidth consumption; however, it does not achieve resource utilization balancing when arranging the files to the DataNodes due to the random selection. Hence, I present a file type aware replica placement policy. Comparing with the current policy used in HDFS, a major benefit of our new policy is

that the system could achieve resource utilization balancing which could further improve the availability of the MapReduce application.

With deploying a scalable distributed monitoring application called *Ganglia Monitoring System* [17] in HDFS, the NameNode can gather the following information of each DataNode periodically:

1) CPU idle rate $R_{cpu}^{DN_j}$;

2) I/O idle rate $R_{I/O}^{DN_j}$;

3) Storage idle rate $R_{space}^{DN_j}$;

The average CPU idle rate R_{cpu}^{Avg} of all the DataNodes is calculated by:

$$R_{cpu}^{Avg} = \frac{\sum_{j=1}^{n} R_{cpu}^{DN_j}}{n} \tag{3.3.1}$$

Similarly, the average I/O idle rate $R_{I/O}^{Avg}$ can be calculated by the formula 3.3.2 and the average storage idle rate R_{space}^{Avg} could be calculated by the formula 3.3.3.

$$R_{I/O}^{Avg} = \frac{\sum_{j=1}^{n} R_{I/O}^{DN_j}}{n} \tag{3.3.2}$$

$$R_{space}^{Avg} = \frac{\sum_{j=1}^{n} R_{space}^{DN_j}}{n} \tag{3.3.3}$$

Our objective is to achieve resource utilization balancing which could further improve the efficiency of the whole Hadoop system. Our objective in this section is formulated as follows:

Object:

$$Min\left\{Max\sqrt{\left(R_{cpu}^{DN_i}-R_{cpu}^{Avg}\right)^2+\left(R_{I/O}^{DN_i}-R_{I/O}^{Avg}\right)^2+\left(R_{space}^{DN_i}-R_{space}^{Avg}\right)^2}\right\} \quad (3.3.4)$$

Subject to:

$$R_{cpu}^{DN_i}\le 1,\ R_{I/O}^{DN_i}\le 1,\ R_{space}^{DN_i}\le 1,\quad i=1,\ldots\ldots,n. \quad (3.3.5)$$

$$R_{cpu}^{Avg}=\frac{\sum_{i=1}^{n}R_{cpu}^{DN_i}}{n},\quad i=1,\ldots\ldots,n. \quad (3.3.8)$$

$$R_{I/O}^{Avg}=\frac{\sum_{i=1}^{n}R_{I/O}^{DN_i}}{n},\quad i=1,\ldots\ldots,n. \quad (3.3.8)$$

$$R_{space}^{Avg}=\frac{\sum_{i=1}^{n}R_{space}^{DN_i}}{n},\quad i=1,\ldots\ldots,n. \quad (3.3.8)$$

3.4 File Type Aware Replica Placement Policy

In this section, I adopt two powerful tools Analytical Hierarchy Process (AHP) [20] and Roulette Wheel Selection (RWS) [30] to improve the candidate nodes selection process. AHP is most widely used in analyzing feasible alternatives when multiple criteria need to be considered on a rational basis. Here AHP is employed to evaluate the DataNodes based on the three features we mentioned above: *CPU idle rate, I/O idle rate, storage idle rate.* It is worth noticing that because HDFS utilizes a single-master pattern, the computing resource of the NameNode is a kind of scarce resource. The NameNode may process hundreds of thousands operations every second. Therefore, it is unpractical for the NameNode to update the information of all DataNodes and execute the AHP evaluation process frequently. This period can be set to one day or more.

3.4.1. DataNodes Evaluation with AHP

The perceived importance of multiple criteria is utilized by AHP as pairwise comparisons on a scale of 1 to 9. Saaty's scale of relative importance is shown in Table. 2.

Table. 2 Saaty's Scale of Relative Importance

Intensity of relative importance	Definition
1	Equal importance
3	Weak importance
5	Strong importance
7	Demonstrated importance
9	Absolute importance
2,4,6,8	Intermediate value between

For different file types, the importance of the capacities of the node is distinct. So each DataNode has three distinct values evaluated by AHP in our work. For instance, for a computing-type file, the DataNodes with low CPU utilization rate are reasonable candidates. More specifically, we have three distinct evaluating criteria for each DataNode by exploiting AHP. The three criteria which are considered are: *CPU idle rate; I/O idle rate; Storage idle rate.*

By using AHP, we form a pairwise comparison matrix shown in Table. 3. The value in the i row and j column evaluates the relative importance of the factor i compared with factor j.

Table. 3 Pair-wise Comparison Matrix

Criteria	CIR	IIR	SIR
CIR	$\frac{\omega_{CIR}}{\omega_{CIR}}$	$\frac{\omega_{CIR}}{\omega_{IIR}}$	$\frac{\omega_{CIR}}{\omega_{SIR}}$
IIR	$\frac{\omega_{IIR}}{\omega_{CIR}}$	$\frac{\omega_{IIR}}{\omega_{IIR}}$	$\frac{\omega_{IIR}}{\omega_{SIR}}$
SIR	$\frac{\omega_{SIR}}{\omega_{CIR}}$	$\frac{\omega_{SIR}}{\omega_{IIR}}$	$\frac{\omega_{SIR}}{\omega_{SIR}}$

For example, when we need to choose the DataNodes for placing the computing-type files, we might arrive at the following matrix:

$$C=\begin{bmatrix} 1 & 3 & 5 \\ \frac{1}{3} & 1 & 3 \\ \frac{1}{5} & \frac{1}{3} & 1 \end{bmatrix}=\begin{bmatrix} 1.000 & 3.000 & 5.000 \\ 0.333 & 1.000 & 3.000 \\ 0.200 & 0.333 & 1.000 \end{bmatrix}$$

For example, the value $C_{12} = 3$ in this matrix indicates that we think the *CPU idle rate* is more important than the *I/O idle rate* in this scenario.

In Table. 4, we synthesize the pairwise comparison matrix C by dividing each element in the matrix C by the amount value of its column. For instance, the value *0.652* in the first row and first column is gained by $\frac{1}{1+1/3+1/5}=0.652$; the value can be found in the matrix C.

Table. 4 Synthesized Matrix *C*

Criteria	CIR	IIR	SIR	Weight
CIR	0.652	0.692	0.555	0.633
IIR	0.217	0.231	0.333	0.260
SIR	0.130	0.077	0.111	0.106

CIR is the CPU idle rate; IIR is the I/O idle rate; SIR is the Storage idle rate.

The weight of the *CIR*, *IIR* and *SIR* in Table. 4 can be calculated by averaging the value in each row, respectively. For example, the weight of *CIR* is calculated by dividing the sum of the first row *(0.652+0.692+0.555)* by *3*. In this example, the weight of *CIR*, *IIR* and *SIR* are $w = [0.633 \quad 0.260 \quad 0.106]$, respectively. Let us use an example to demonstrate how to use *AHP* to evaluate the DataNodes in HDFS.

Assume all the information of the DataNodes in Table. 5 is collected by *Ganglia Monitoring System*.

Table. 5 Illustration of the Evaluation of the DataNodes

	CIR	IIR	SIR	Value
DN_1	0.70	0.63	0.51	0.682
DN_2	0.63	0.52	0.43	0.598
DN_3	0.35	0.34	0.84	0.381
DN_4	0.91	0.80	0.13	0.798

For example, in Table. 5, the value of DN_4 is calculated as:

$$V_{DN1} = CIR_{DN1} * \omega_{CIR} + IIR_{DN1} * \omega_{IIR} + SIR_{DN1} * \omega_{SIR}$$

$$= 0.70*0.633 + 0.63*0.260 + 0.51*0.106 = 0.798$$

The value of DN_2, DN_3, DN_4 can be calculated in the same way, respectively.

3.4.2. Roulette Wheel Selection (RWS) Method

As I mentioned, it is unpractical for the NameNode to update the information of all DataNodes and execute the AHP evaluation process for each writing request. This period can be set to one day or more. However, if we always select the DataNode with the greatest value in this period, this DataNode may suffer I/O throughput, CPU or network congestion. Intending to solve this problem, I adopt a selection approach which has been used in Genetic Algorithm called *Roulette Wheel Selection (RWS).* By using AHP, DataNode DN_i is evaluated a value which is represented by V_{DNi}. Moreover, for further improving the efficiency of the *roulette wheel selection*, we do not put all of the DataNodes in the process. There is a threshold δ and if $V_{DNi} < \delta$, this DataNode will not be a candidate in *roulette wheel selection.*

In *roulette wheel selection*, the probability that a DataNode DN_i is selected $P_{\{DNi\ is\ selected\}}$ is calculated as follows:

$$P_{(DN_i\ is\ selected)} = def \frac{V_{DNi}}{\sum_{j=1}^{n} V_{DNj}} \tag{3.3.10}$$

As this formula shown us, the DataNode with higher value has more probability to be selected by the NameNode. Hence, the files are prone to be distributed on the DataNodes with relatively high idle resources in a long period.

3.5 Numerical Results

In this section, I evaluate the performance of our file type aware replica placement strategy. Given *100* DataNodes, there are *10000* new files will be uploaded to the HDFS system. For most of the DataNodes, the range of *CIR*, *SIR* and *IIR* is from *30%* to *70%*.

It is worth noting that, in this simulation, for *10%* DataNodes, the values of *CIR*, *SIR* and *IIR* are all *100%*, which means they are all new machines. In order to evaluate our new strategy, we assume that all files' type can been identified by the NameNode. Particularly, files consume resources of the DataNodes; if the file is *computing-type* file, the consumption rate of the CPU is assumed to be *0.1% to 0.3%*; and the consumption rate of I/O throughput and disk space could be *0.01%-0.03%* and *0.01%-0.05%*, respectively. If the file is *I/O throughput-type* file, these three consumption rate could be *0.01%-0.03%*, *0.1%-0.3%* and *0.01%-0.05%*, respectively. Similarly, when the file is identified as *storing-type file*, these three consumption rate could be *0.01%-0.03%*, *0.01%-0.03%* and *0.1%-0.5%*, respectively.

In this section, I compared the performance of our *File type Aware Replica Placement (FARP)* strategy with the *Current Replica Placement (CPR) strategy* in HDFS by using the *Minimum Mean-Square Error (MMSE)* of *CIR*, *IIR* and *SIR*. For example, the value of *MMSE* of *CIR* indicates the difference of the CPU utilization rate between the DataNodes. The value of *MMSE* reducing implies that the CPU utilization rate of the DataNodes prone to balance.

Fig. 5 shows us the *MMSE* of *CIR* of the DataNodes that after uploading the files to the system by executing our *FARP* strategy and by executing *CPR* strategy. As we expected, by using *FARP*, when the number of files increase, the CPU utilization rates of these *100* DataNodes tend to achieve balance; because the values of *MMSE* is observed going down. By contrast, I observe that in Fig. 5, the values of *MMSE* of *CIR* rarely change by using *CPR* strategy. It indicates that the *CPR* strategy does not

contribute on resource utilization balancing. The similar situations in terms of *IIR* and *SIR* are observed in Fig. 6 and Fig. 7, respectively.

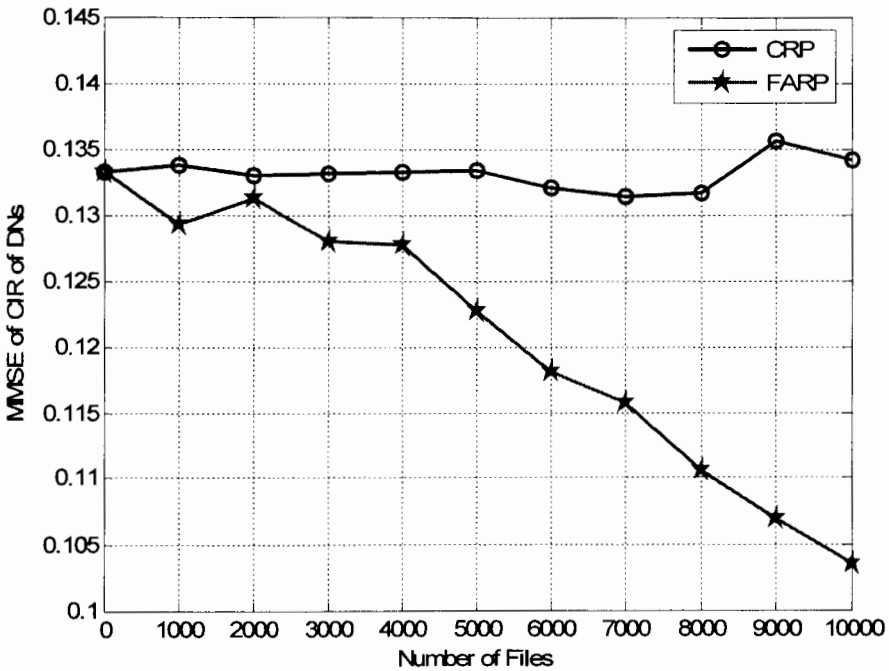


Fig. 5 MMSE of the CIR of DNs

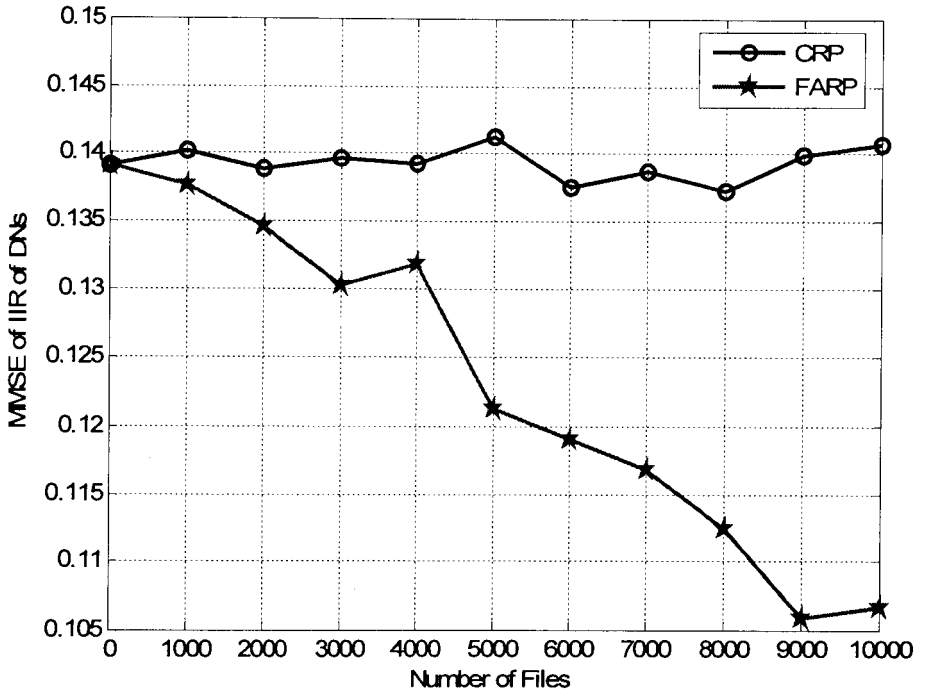


Fig. 6 MMSE of the IIR of DNs

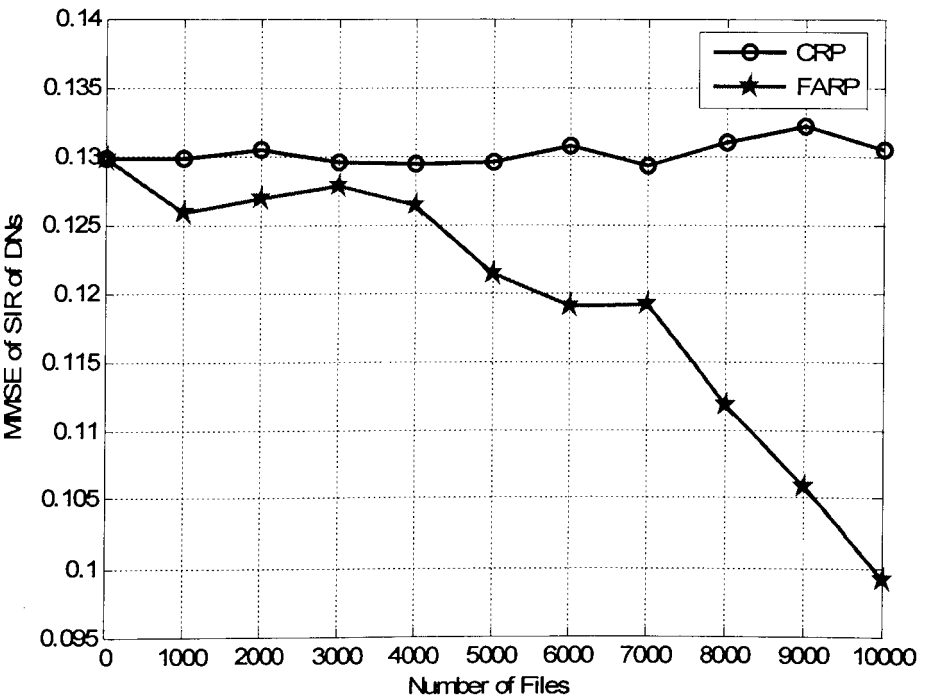


Fig. 7 MMSE of the SIR of DNs

CHAPTER 4. APPLICATION REPLICA STRATEGY ON CLOUD-FRONT SERVER

4.1 Introduction

Cloud computing opens a new area of supplement, consumption, and delivery framework for IT services, and it involves over-the-Internet provision of dynamically scalable and virtualized resources which is significant trends with the potential to increase agility and lower costs of IT [22]. Virtual infrastructure cloud services (e.g., [21], [25]) are virtual hardware provider, where customers can deploy virtual servers and run applications. The virtual server vendor which is an emerging cloud service is the motivation of this topic. Cloud customers can order Applications which can be delivered by the cloud providers on the cloud. Three characters are involved in this model: 1) cloud service vender, 2) application provider, and 3) application customer. Application providers develop the applications and put them on the original cloud servers which are provided by the cloud providers. The customers purchase the applications from the application providers. It is worth noting that customers always expect the applications ordered can be delivered as fast as possible. However, it is difficult for the original cloud server to provide all reservations in time due to the limitation of the distance and network bandwidth. As a consequence, the provisioning time of lots of reservations is much longer than expected.

To reduce the latency time, cloud service vender employs a original-front server strategy (e.g., [21]), shown in Fig. 8.

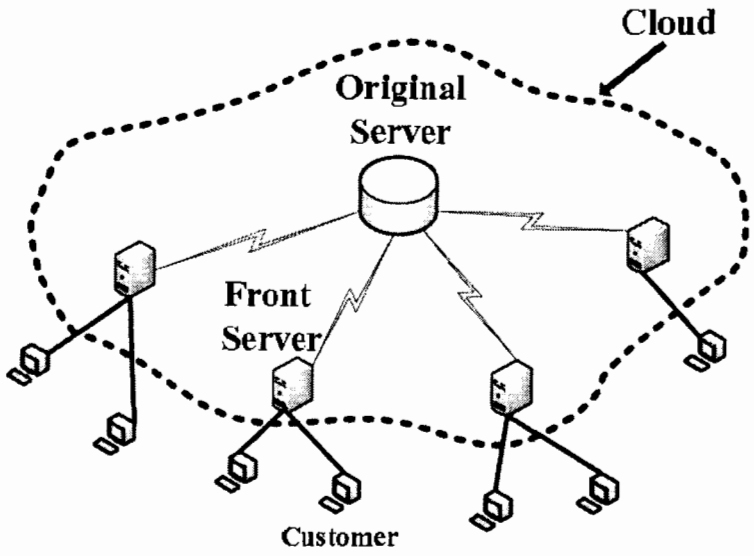


Fig. 8 Illustration for Original-Front Server Model

There are two types of server in this *OFS* model: the original server and the front server. All applications are stored in the original server initially, and when the information of reservations is gathered by the original server, it needs to execute the application replica placement strategy to distribute the replicas. We approximately calculate the latency time for delivering the applications with the distance between the front servers and customers. Because the front servers are located near to the customers geographically and globally, the provisioning time can be reduced. Obviously, if the applications are always routed to the nearest front servers for every customer who makes the reservation, contents will be delivered with the best possible performance. However, we notice that the application providers cannot achieve maximum profits because they have to pay the cloud vender to use this service. For example, the expense is e'_k when the application provider puts the application k on front server j. If the customers choose to use the front service to have low latency delivery, they have to pay

the application providers for using the advanced service. For example, when the customer i makes the reservation of the application k, the cost for using the front service is v^i_k. Hence, the profits of application providers are the income minus the payout. In this Chapter, I assume that the replica of the application distribution strategy can be decided by the application providers and be executed by the cloud vender. Obviously, if application providers try to maximize their profit Φ_{total}, they need to satisfy all their customers with minimum front servers. We assume that if the expected latency time for delivering the application k to the customer j is t^k_{exp}, there is at least one can deliver the content to customer j in time. Our objective is to maximize the profit Φ_{total} for application providers.

4.2 Related Work

How to provision the applications through cloud rapidly has been studied recently [26], [24]. In paper [24], the authors studied a fundamental storage staging problem and presented it as a scheduling problem with capacity constraints under two models: *continuous model* and *integral model*.

Similar to replication placement, content distribution has been studied in the context of web content through Content Distribution Networks [28], [29]. Some Content Distribution Networks implementations introduce related job scheduling problems. The scheduling problem for cache pre-filling is studied in [23]. Many content distribution systems adopted web caching techniques [27], where frequently accessed objects are stored near the customers. These techniques can reduce both access latency and network traffic.

In could computing, virtual infrastructure cloud services (e.g., [21], [25]) are both a virtual hardware provider and a virtual hosting premise, where customers can deploy virtual servers and run applications. Cloud providers provide some special web services for content delivery, such as Amazon CloudFront [21] which cooperates with other Amazon Web Services to serve developers and businesses an easy way to distribute content to end customer with high data transfer speeds, low latency, and no commitments. With a global network of edge locations, Amazon CloudFront can deliver your static and streaming content rapidly.

In this work, I target to maximize the profit Φ_{total} for application providers; and to the best of our knowledge, this problem is rarely studied in the previous works.

4.3 Problem Statement

In this work, the *Original-Front Server (OFS)* model is adopted. We are given a set of customers $C = \{c_1, c_2, \ldots, c_t\}$, a set of front servers $F = \{f_1, f_2, \ldots f_t\}$ and a set of applications $K = \{k_1, k_2, \ldots, k_n\}$. Multiple of applications can be ordered by a single customer. For customer i, h_i denotes the number of applications reserved by customer i; and v_k^i denotes the expense that customer i orders application k with front service. For the application providers, the expense is $e^j{}_k$ for putting the application k on front server j. The price strategies for both the cloud venders and application providers are out of the scope of this topic. Because the capacity and network bandwidth of each front server is limited, we have the following constraints: a front server can only serve maximum ε customers at the same time. When customer i reserves application k, the expected delivery time denotes by $t_{\exp}^{i,j}$. If the provisioning time $t_{pro}^{i,k}$ is greater than $t_{\exp}^{i,k}$, we

assume that customer i will cancel this reservation and the charge v_k^i is refunded. Moreover, the reservation which is delivered in time is a *satisfied reservation*; otherwise it is an *unsatisfied reservation.* When the customer i reserves multiple applications, the ratio of the *satisfied reservations* and the *unsatisfied reservations* should not less than the threshold a_i, $a_i \geq \frac{N^i_{satisfied\ reservation}}{N^i_{unsatisfied\ reservation}}$; otherwise, the customer will cancel all of the reservations.

Definition. 1 (Eligible Front Server): For customer i, if the front server j can deliver the reserved applications in time, then the front server j is called eligible front server to customer i. In other words, if the latency time is no more than the expected delivery time, $t^{i,k}_{pro} \leq t^{i,k}_{epx}$, *then the front server j is an eligible front server to customer i.*

Definition. 2 (Maximum prOfit of Application Replication(MOAR)): Given a set of customers $C = \{c_1, c_2, ..., c_t\}$, a set of applications $K = \{k_1, k_2, ...k_n\}$, and a set of front servers $F = \{f_1, f_2, ...f_m\}$, for the application provider, the income γ_{in} is calculated as:

$$\gamma_{in} = \sum_{i=1}^{t} \sum_{k=1}^{h_i} v_k^i s_k^i a_i \tag{4.3.1}$$

where s_k^i denotes that if the application provider provides reservation k to customer i with front service. s_k^i can be determined by the following formula:

$$x_i = Max[x_k^{i,1}, ..., x_k^{i,j}, ..., x_k^{i,m}] \tag{4.3.2}$$

where $x_k^{i,j} = [\frac{t^{i,k}_{exp}}{t^{i,j,k}_{pro}}] \times f(j,k)$, and $f(j,k)$ is the decision variable which denotes if there is a replication of application k exists on front server j. If $x_i \geq 1$, then $s_k^i = 1$; otherwise, $s_k^i = 0$.

For the application provider, the payout is calculated as:

$$\gamma_{out} = \sum_{j=1}^{m}\sum_{k=1}^{n} e_k^j f(j,k) \tag{4.3.3}$$

Our objective is to maximize the total profit of the application provider:

$$Maximize \sum_{i=1}^{t} a_i [\sum_{k=1}^{h_i} v_k^i s_k^i] - \sum_{j=1}^{m}\sum_{k=1}^{n} e_k^j f(j,k) \tag{4.3.4}$$

4.4 Proposed Solutions

In this section, I study two different scenarios of application replication placement and present two heuristic algorithms to solve the *Maximum prOfit of Application Replication (MOAR)* problem in these two scenarios.

4.4.1 OFS Model with Single Application

We start with a special case where there is only one application provided in the *OFS* model. Because there is only one application, the threshold $a = 1$ for all customers.

A tree network G is constructed in which the original server is the root of the tree. The set of intermediate nodes $V = \{n_1, n_2, ..., n_m\}$ denotes the front servers set, and the set of leaf nodes $l = \{l1, l2..., lj\}$ denotes the customers sets. (In the following, we use front server and intermediate node interchangeably, as well as customer and leaf node.) Let the set $Di = \{d^i_1, d^i_2, ... d^i_a\}$ represents the leaves on intermediate node i, which also means the customers who are routed to the front server f_i. According to the geographical location information, the eligible front servers of customer i who reserved application k is calculated and represented by $Wi = \{w^i_1, w^i_2, ... w^i_p\}$ where p is no more than the number of front servers m.

When there are x customers are served by the front server j, the income of the application provider in terms of server j is $\gamma^j_{in} = \sum_{i=1}^{x} v_k^{i,j}$. We use Ω_j to denote the ratio of

the income and the payment of front server j, and Ω_j is calculated as:

$$\Omega_j = \frac{\gamma_{in}^j}{\gamma_{out}^j} \tag{4.3.5}$$

Where $\gamma_{out}^j = e_k^j$.

In our solution, we assume the application replications have been deployed on all of the front servers initially. Our *H-MOAR* algorithm, shown in *Algorithm 1*, first chooses the node n_i with the minimum value Ω_{min} in network G. Then, if the leaf on node n_i has other eligible front nodes, it is transferred to one of these nodes which has the maximum value Ω_{max}. For example, leaf l_a has three eligible nodes n_1, n_2 and n_3, and l_a currently connected to node n_1. According to Algorithm 1 in Fig. 9, if $\Omega_{n2} \geq \Omega_{n3}$, l_a needs to be transferred to n_2.But if leaf l_a does not have any other eligible node, l_a is kept on the current node. After executing these steps, I recalculate the value Ω'_{ni} for node n_i. If $\Omega'_{ni} \leq 1$, node n_i and its present leaves are removed from network G; If $\Omega'_{ni} \geq 1$, node n_i is marked as *pruned*. Our algorithm continues to repeat this process on the *unpruned* nodes until none *unpruned* nodes left. Based on our algorithm, these steps are repeated iteratively until there is no leaf transfer occurs in G.

Algorithm 1 Single App H-MOAR(G)

1: Construct sets of intermediate nodes V, V', and V'';

2: $V \leftarrow \{n_1, n_2, \cdots, n_i\}$; $V' \leftarrow V$; $V'' \leftarrow \phi$;

3: **for** each *leaf i* in G **do**

4: Connect i to the node n_{nea} which is nearest node to i;

5: **end for**

6: **while** $V \neq V''$ **do**

7: Find the node n_i with the minimum value $\Omega_{min}^{n_i}$;

8: **for** each *leaf l* in the set D_{n_i} **do**

9: **if** $|W_l| > 1$ (E_l is the set of eligible noedes of l) **then**

10: Transfer l to the node n_{max} which is the node with

The maximum value in set W_l.

11: **end if**

12: **end for**

13: **if** $\Omega_n^{'} \geq 1$ **then**

14: Mark n as "*pruned*";

15: **else**

16: Delete n_i and leaves in $D_{n_i}^{'}$;

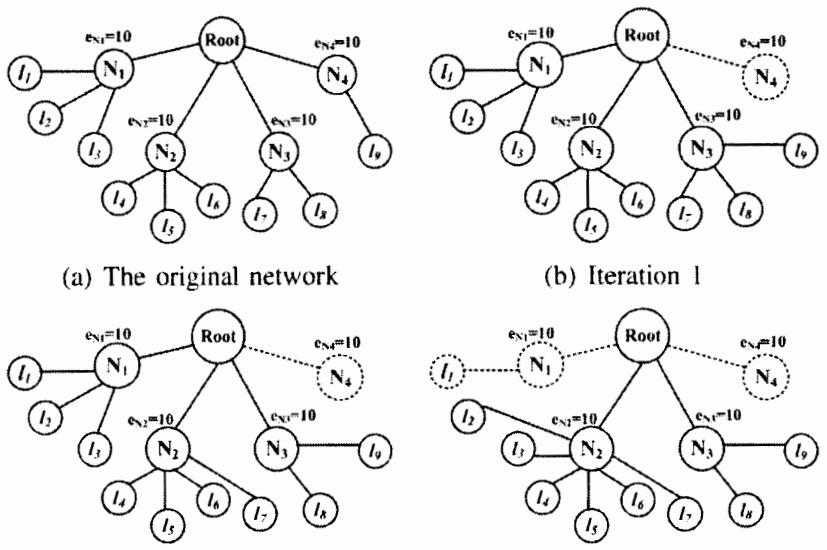
17: **end if**

18: Repeat step 6 to step 15; Move node n_i to set V''.

19: **end while**

Fig. 9 Algorithm 1 Single App H-MOAR

Let us use an example to illustrate Algorithm 1. In Fig. 10(a), I simply assume that the payout of the application k on each front server is the same which is $e_k = 10$, and the income from each customer who uses front service is $v_k = 5$. Initially, leaves are automatically routed to the nearest nodes, shown in Fig. 10(a). Following to Algorithm 1, in the first iteration, node N_4 is selected due to $\Omega^k_{N_4} = \frac{5}{10}$ which is the minimum value in network G.



(a) The original network (b) Iteration 1

(c) Iteration 2 (d) Iteration 3

Fig. 10 Illustration of Algorithm 1 Single App H-MOAR

From the set $W_{l9} = \{N_3, N_4\}$ which includes the eligible nodes of leaf l_9, we choose node N_3 as a target node to transfer leaf l_9, shown in Fig. 10(b). There is no leaf left on node N_4 after moving leaf l_9 to N_3. Hence, node N_4 is deleted from the network, which means the replication of application k will not be deployed on front server N_4. In the second iteration, because $\Omega^k_{N_2} = \Omega^k_{N_3} = \frac{15}{10}$, node N_3 is selected randomly. The sets of eligible front nodes of leaves l_7, l_8 and l_9 are $E^k_{l_7} = \{N_2, N_3\}$, $E^k_{l_8} = \{N_3\}$, $E^k_{l_9} = \{N_3\}$,

respectively. Only leaf l_7 is moved to node N_2 in this iteration, shown in Fig. 10(c). Because $\Omega'^{k}_{N_3} = \frac{10}{10} \geq 1$ which means N_3 can be kept in the network and marked as *pruned*. Fig. 10(d) illustrates the third iteration. The same situation happens on node N_1. Leaves l_2 and l_3 are transferred to N_2, and the value of node N_1 is $\Omega'^{k}_{N_3} = \frac{5}{10}$ which means N_1 should be deleted from the network. At this time, the network achieves stable. As a consequence, N_1 and N_3 are chosen as the front servers which are placed with application replications k.

4.4.2. OFS Model with Multi-Applications

Now we need to generalize our approach to the scenario refers to multiple applications. For the general network model, the number of applications is k. In this multiple applications scenario, I present a multi-layers strategy which could separate the *MOAR* problem into numbers of sub-problems. It is worth noting that in this multi-layers *OFS* model, I aim to maximize the value of the whole network, not only in the single layer. And I adopt a new variable φ and $\varphi = \frac{1}{\beta_i - \alpha_i}$. $\Omega_{\min}$ is calculated as:

$$\Omega_{\min} = \frac{\sum_{i=1}^{x.q} \frac{v_{ik}^{q}}{\varphi}}{\sum_{i=1}^{x.q} e_k^{j,q}} \tag{4.3.6}$$

A set of tree networks $G_{mul} = \{g_1, g_2, \ldots g_k\}$ is constructed to demonstrate the multi-layers model. Each tree network represents a layer, and only one application is considered in each layer. Fig. 11 illustrates our Multi-application heuristic algorithm.

Algorithm 2 Multi-Apps H-MOAR(G)

1: Construct sets of intermediate nodes V, V', and V'';

2: $V \leftarrow \left\{n_1^q, n_2^q, \cdots, n_i^q\right\}; V' \leftarrow V; V'' \leftarrow \phi$;

3: **for** each *leaf* i in G^q **do**

4: Connect i to the node n_{nea} which is nearest node to i;

5: **end for**

6: **while** $V \neq V''$ **do**

7: Find the node n with the minimum value

$$\Omega_{min} = \frac{\left(\sum_{i=1}^{x,q} \frac{v_{i_k}^q}{\beta_i - \alpha_i}\right)}{\sum_{i=1}^{x,q} e_k^{J,q}}$$ where β_i is the number of remained applications to leaf i in all of the layers;

8: **for** each *leaf* l in the set D_n^q, where D_n^q is node n's leaf set **do**

9: **if** $|W_l| > 1$ **then**

Transfer l to the node n_{max} which is the node with the maximum value in the set W_l;

11: **end if**

12: **end for**

13: **if** $\Omega_n^{'} \geq 1$ **then**

14: Move n to set V'';

15: **else**

16: Delete n and the leaves in $D_n^{'}$;

17: **end if**

18: Repeat step 6 to step 15; Move node n to set V''.

19: **end while**

Fig. 11 Algorithm 2 Multi-Apps H-MOAR

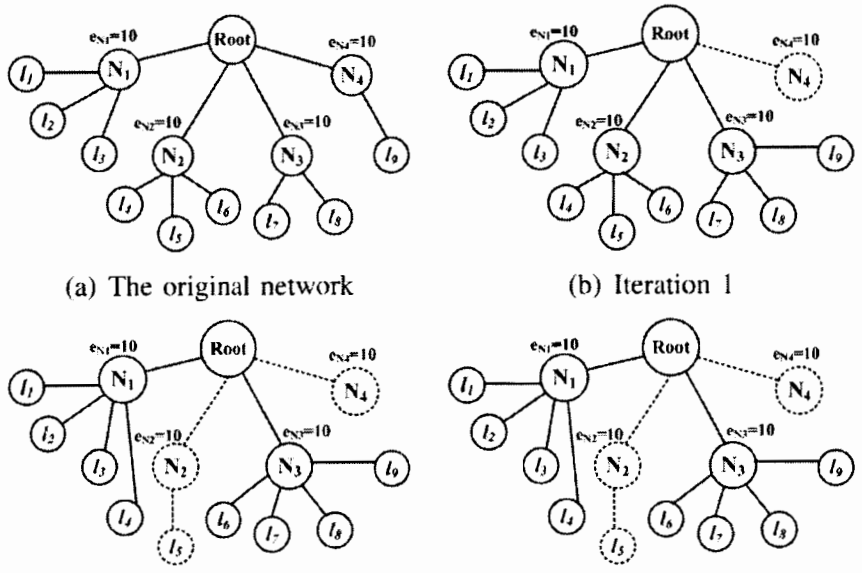
In the layer q, customer i has a set of eligible front server which represented by $W^{i,q} = \{w_1^{i,q}, w_2^{i,q}, ..., w_\alpha^{i,q}, \}$. For the front server j, the income is

$$\gamma_{in}^{j,q} = \sum_{i=1}^{x,q} v_{ik}^q a_i \tag{4.3.7}$$

where x is the number of the customers connected to the front server j. The profit for the application provider in layer q is

$$\Phi_j^q = \gamma_{in}^{j,q} - \gamma_{out}^{j,q} = \sum_{i=1}^{x,q} v_{ik}^q a_i - \sum_{i=1}^{x,q} e_k^{j,q} \tag{4.3.8}$$

We use the following example to illustrate algorithm 2, shown in Fig. 12.



(a) The original network (b) Iteration 1

(c) Iteration 2 (d) Iteration 3

Fig. 12 Illustration of Algorithm 2 Multi-Apps H-MOAR

The value of α, β and φ of each leaf can be found in Table. 6. In layer q, we simply assume that the payout for each front server is $e_k^q = 10$, and the income from each customer can be calculated by $\gamma_{in}^{j,q} = \sum_{i=1}^{x,q} \frac{v_{ik}^q}{\beta_i - \alpha_i}$, where $v_{ik}^q = 1$ in this example. Algorithm 2 is implemented on each layer with considering the variable α. First, we calculate the value Ω of each node, where $\Omega_{N_1} = \frac{12.5}{10}$, $\Omega_{N_2} = \frac{10.3}{10}$, $\Omega_{N_3} = \frac{14}{10}$,

$\Omega_{N_4} = \frac{5}{10}$. Hence, in the first iteration, node N_4 which has the minimum value in network G^q is selected. We choose node N_3 as a target node to which leaf l_9 can be transferred. Because N_3 is another eligible node of l_9. We delete N_4 after the transfer, because no leaf left on N_4, shown in Fig. 12(b). In iteration 2, we recalculate the value for each node where $\Omega_{N_1} = \frac{12.5}{10}$, $\Omega_{N_2} = \frac{10.3}{10}$, $\Omega_{N_3} = \frac{19}{10}$. Obviously, node N_2 should be selected. The sets of eligible nodes of leaves l_4, l_5 and l_6 are $E^k_{l4} = \{N_1, N_2\}$, $E^k_{l5} = \{N_2\}$, $E^k_{l6} = \{N_2, N_3\}$, respectively. According to 2, leaf l_4 is transferred to node N_1 and l_6 is transferred to node N_3 in this iteration, shown in Fig. 12(c).At this time value of node N_2 $\Omega_{N_1} = \frac{5}{10} \leq 1$ which means N_2 should be deleted from G^q. Fig. 12(c) illustrates the second iteration. In the third iteration, neither the leaves on node N_1 nor the leaves on node N_3 have other eligible nodes, which means network G^q achieve stable. As a result, N_1 and N_3 are chosen to be distributed with application replications in this case. The value of α, β and φ of each leaf can be found in Table. 6.

Table. 6 Example of Algorithm 2 Multi-Apps H-MOAR

Symbols	Value	Symbols	Value	Symbols	Value
α_{l1}	0.5	β_{l1}	0.9	φ_{l1}	2.5
α_{l2}	0.5	β_{l2}	0.8	φ_{l2}	3.3
α_{l3}	0.6	β_{l3}	0.75	φ_{l3}	6.7
α_{l4}	0.6	β_{l4}	0.9	φ_{l4}	3.3
α_{l5}	04	β_{l5}	0.6	φ_{l5}	5
α_{l6}	0.4	β_{l6}	0.9	φ_{l6}	2
α_{l7}	0.7	β_{l7}	0.8	φ_{l7}	10
α_{l8}	0.7	β_{l8}	0.95	φ_{l8}	4
α_{l9}	0.5	β_{l9}	0.7	φ_{l9}	5

4.5 Numerical Results

In this section, I presented numerical results to evaluate the performances of our solutions. I implemented our heuristic algorithm, which was denoted as H-MOAR in the figures. For comparison, I also implemented the scenario without optimization which aims to satisfy all the customers. This scenario was denoted as original distribution in the figures. All our simulation runs were performed on a 2.8 GHz Linux PC with 2G bytes of memory. I used different network topologies in a 100×100 sq. units playing field to evaluate our proposed solutions. All the front servers and customers were randomly distributed in the playing field.

In our simulation, the number of front servers was set to *20*. The cost e_k^j of deploying an application on a front server was set to *20*. The cost of customer by using a particular application was set to *3*. We also set the constraint ε that the number of customers connected to one server less than *50* in our simulations. In our simulation, I implemented the scenario of OFS model with single application. The scenario of OFS model with multiple applications will be further studied and implemented in our future work.

I tested the performances in terms of the profit of application providers, satisfaction ratio of customers, and number of deployed front server of our solution, which were shown in Fig. 12 and Fig. 13. Fig. 12 illustrated that H-MOAR always has a better performance of profit. Another observation is that as the number of customers increased, the profit also increased.

For the satisfaction ratio, both H-MOAR and Original Distribution have the similar

performance. The satisfaction ratio of Original Distribution is a little better than the one of H-MOAR, because the Original aims to satisfy all the requirements of the customers. Fig. 14 shows us that, comparing to the original distribution, our H-MOAR protocol can satisfy the near maximum number of customers with much less front servers.

To sum up, our simulations demonstrated that the H-MOAR protocol achieves similar satisfaction ratio as the optimal solution, while increasing the profit of application providers. Hence, the *H-MOAR* protocol is suitable for *Original-Front Server* framework.

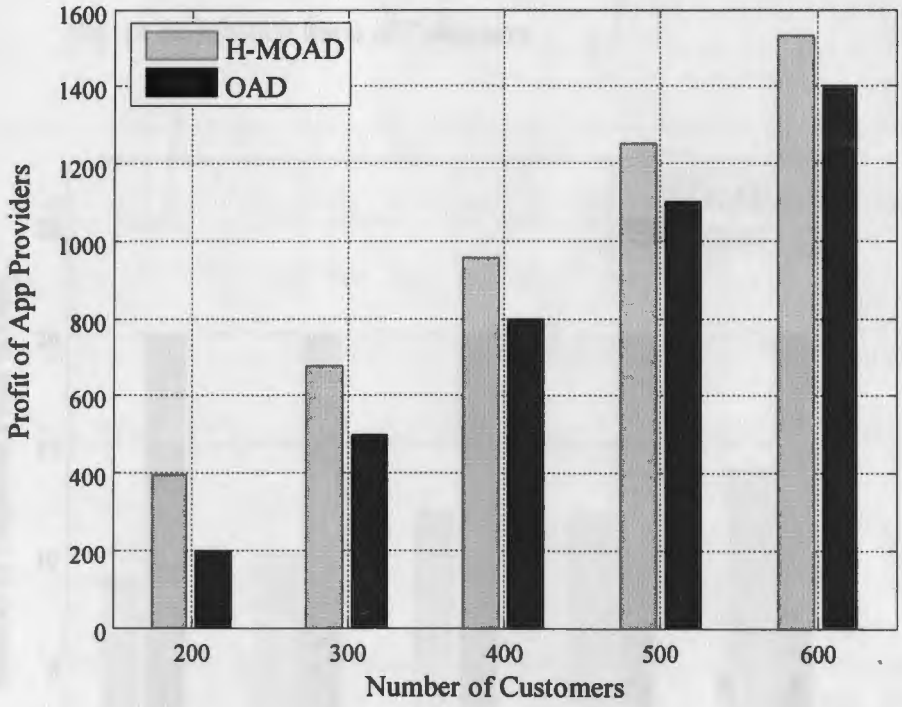


Fig. 13 Profit of Application Providers

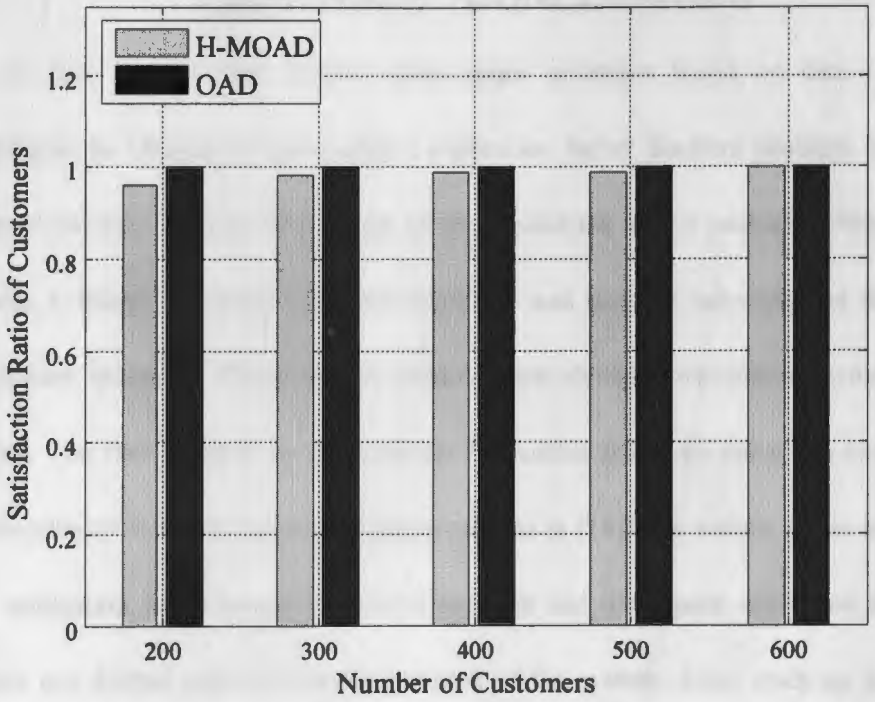


Fig. 14 Satisfaction Ratio of Customers

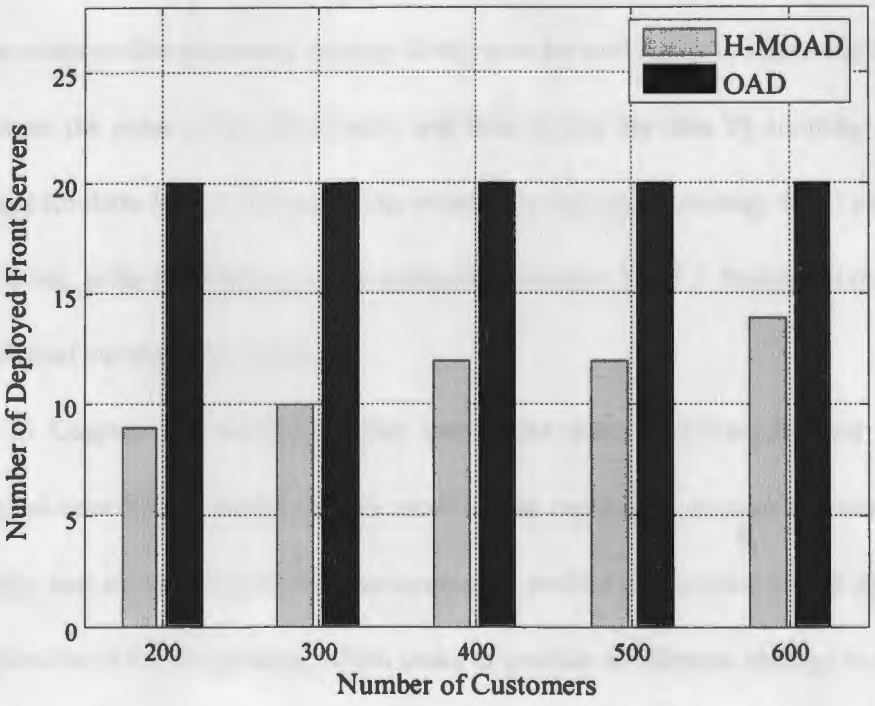


Fig. 15 Number of Deployed Front Servers

CHAPTER 5. CONCLUSIONS

In this work, I have studied three major problems based on data replication technique. In Chapter 2, I presented a replication factor decision problem in Hadoop Distributed File System. In contrast to the replication factor problem from previous works, I intend to satisfy both the reliability and achieve network and disk space utilization balancing. There are two phases in our dynamic replication factor decision policy. The first phase is to calculate the replication factor by using the concept that probability of data loss during rebuilding process in [18]. The second phase is to adjust the replication factor intend to achieve network and disk space utilization balancing, which can further improve the performance of the system. After studying how many replicas are necessary, I studied the replica placement problem for distributing the replicas in Chapter 3. Our objective is to achieve resource utilization balancing. Our file type aware replica placement strategy firstly uses the tool Analytic Hierarchy Process to evaluate the value of the DataNodes, and then deploy the data by adopting a method named Roulette Wheel Selection. The entire data replication strategy that I proposed in this work, is the combination of the schemes in Chapter 2 and 3. Numerical results have confirmed our theoretical analysis.

In Chapter 4, I studied another use of the data replication method based on original-front server model. In this model, data replication is used to reduce access latency and network bandwidth consumption. I studied a *Maximum prOfit Application Replication (MOAR)* problem, which seeks to provide an efficient strategy to maximize the profit of the application providers. I proposed two heuristic algorithms which are

called *H-MOAD* and *Multi-app H-MOAD* to solve the *MOAR* problem. Our simulation results show that the *H-MOAD* scheme can increase the profit of application providers.

REFERENCES

[1] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system," in *Proc. of ACM Symposium on Operating Systems Principles,* Lake George, NY, Oct 2003, pp 29–43.

[2] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proc. of the 6th Symposium on Operating Systems Design and Implementation,* San Francisco CA, Dec. 2004.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," *Proceedings of IEEE MSST 2010*, Incline Village, NV, USA, May 2010.

[4] K. V. Shvachko, "HDFS Scalability: The limits to growth," April 2010, pp. 6–16.

[5] "Apache Hadoop," http://hadoop.apache.org/hdfs/.

[6] M. K. McKusick, and S. Quinlan, "GFS: Evolution on Fast-forward," *ACM Queue,* vol. 7, no. 7, New York, NY. August 2009.

[7] V. Venkatesan, I. Iliadis, X. Hu, R. Haas, and C. Fragouli, "Effect of Replica Placement on the Reliability of Large-Scale Data Storage Systems," in *Proc. MASCOTS*, 2010, pp.79-88.

[8] J. R. Douceur and R. P. Wattenhofer, "Modeling Replica Placement in a Distributed File System: Narrowing the Gap between Competitive Analysis and Simulation," *ESA 2001*, Aug 2001.

[9] "Name-Node Memory Size Estimates and Optimization Proposal," August 6, 2007, https://issues.apache.org/jira/browse/HADOOP-1687.

[10] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn, "Ceph: A Scalable, High-Performance Distributed File System," *Proceedings of OSDI '06: 7th Conference on Operating Systems Design and Implementation (USENIX Association, 2006).*

[11] D. A. Patterson, G. Gibson, and R. H. Katz. "A case for redundant arrays of inexpensive disks (raid)," *ACMSIGMOD Rec.*, 17(3):109{116, 1988}.

[12] "Parallel Virtual File System," http://www.pvfs.org/.

[13] "The gLite File Transfer Service," http://egee-jra1-dm.web.cern.ch/egee-jra1-dm/FTS.

[14] B. Fan, W. Tantisiriroj, Lin Xiao, and Garth Gibson, "DiskReduce: RAID for Data-Intensive Scalable Computing," *PDSW 09 (4th Petscale Data Storage Workshop Supercomputing)*.

[15] "Amazon Elastic MapReduce," http://aws.amazon.com/elasticmapreduce/.

[16] "Amazon CloudFront," http://aws.amazon.com/cloudfront/.

[17] "Ganglia Monitoring System," http://ganglia.sourceforge.net/.

[18] V. Venkatesan, I. Iliadis, X. Hu, R. Haas, and C. Fragouli, "Effect of Replica Placement on the Reliability of Large-Scale Data Storage Systems", in *Proc. MASCOTS*, 2010, pp.79-88.

[19] "Module 2: The Hadoop Distributed File System," http://developer.yahoo.com/hadoop/tutorial/module2.html.

[20] Saaty TL, "The analytic hierarchy process," New York, McGraw Hill, 1980.

[21] "Amazon Elastic Compute Cloud (EC2)," http://aws.amazon.com/ec2.

[22] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A Berkeley view of cloud computing," University of California, Tech. Rep., 2009.

[23] R. Cohen, L. Katzir, and D. Raz, "Scheduling algorithms for a cache pre-filling content distribution network," in *INFOCOM*, vol. 2, 2002, pp. 940-949.

[24] A. Epstein, D. H. Lorenz, E. Silvera, and I. Shapira, "Virtual Appliance Content Distribution for a Global Infrastructure Cloud Service," in *INFOCOM Proceedings IEEE*, Mar 2010.

[25] "Google App Engine," http://code.google.com/appengine.

[26] E. Kotsovinos, T. Moreton, I. Pratt, R. Ross, K. Fraser, S. Hand, and T. Harris, "Global-scale service deployment in the Xeno Server platform," in *Proceedings of the First Workshop on Real, Large Distributed Systems (WORLDS'04)*, Dec. 2004.

[27] M. Rabinovich and O. Spatschek, "Web caching and replication," Boston, MA, USA, *Addison-Wesley Longman Publishing Co., Inc.*, 2002.

[28] A. Vakali and G. Pallis, "Content delivery networks: Status and trends," *IEEE Internet Computing*, vol. 7, no. 6, pp. 68-74, 2003.

[29] D. C. Verma, "Content Distribution Networks: An Engineering Approach," New York, NY, USA, *John Wiley & Sons, Inc.*, 2002.

[30] O. A. Jadaan, L. Rajamani, and C. R. Rao, "Improved selection operator for GA," *Journal of Theoretical and Applied Information Technology*, 4(4):269277, 2008.

[31] Y. Chen, R. H. Katz, and J. Kubiatowicz, "Dynamic Replica Placement for Scalable Content Delivery," in *IPTPS 01 Revised Papers from the First International Workshop on Peer to Peer Systems*, 2002.