Research Article

Dynamic Deduplication Decision in a Hadoop Distributed File System

Ruay-Shiung Chang, Chih-Shan Liao, Kuo-Zheng Fan, and Chia-Ming Wu

Department of Computer Science and Information Engineering, National Dong Hwa University, No. 1, Section 2, Da Hsueh Road, Shou-Feng, Hualien 974, Taiwan

Correspondence should be addressed to Chih-Shan Liao; toptom@mail.ndhu.edu.tw

Received 5 December 2013; Revised 11 March 2014; Accepted 16 March 2014; Published 28 April 2014

Academic Editor: Yue Shan Chang

Copyright © 2014 Ruay-Shiung Chang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Data are generated and updated tremendously fast by users through any devices in anytime and anywhere in big data. Coping with these multiform data in real time is a heavy challenge. Hadoop distributed file system (HDFS) is designed to deal with data for building a distributed data center. HDFS uses the data duplicates to increase data reliability. However, data duplicates need a lot of extra storage space and funding in infrastructure. Using the deduplication technique can improve utilization of the storage space effectively. In this paper, we propose a dynamic deduplication decision to improve the storage utilization of a data center which uses HDFS as its file system. Our proposed system can formulate a proper deduplication strategy to sufficiently utilize the storage space under the limited storage devices. Our deduplication strategy deletes useless duplicates to increase the storage space. The experimental results show that our method can efficiently improve the storage utilization of a data center using the HDFS system.

1. Introduction

With the development of science and technology, the ubiquitous networks and the trend of smart devices have led some people to a more fulfilling life. People can take a picture and upload the picture to social platforms (such as Facebook, Twitter, and LINE) anywhere and anytime. Data and information come from user generated contents (UGC) all around the world. Hence, this is the reason that the term “big data” [1] has become so popular after the advent of cloud computing.

Over the past few years, cloud computing has become one of the top industries in information technology (IT). It has also been applied to many commercial products and companies [2–5]. Cloud computing can combine physical and virtual resources to cope with a great deal of computing services including infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS).

Facing the problems of big data, this paper focus discusses the concept of storage space. The motivation will be expounded with the topic related to the Hadoop distributed file system (HDFS) [6], which plays an important role in cloud computing. However, users of cloud computing are not only companies, but also a person with interests or an educational environment, with the latter often not as rich as the former. Therefore, without purchasing infrastructure, the latter will have to consider other solutions and, last but not least important, the prerequisite of this paper is this condition.

Nowadays, data are generated and updated per second. In daily life, no matter whether a company or a person, we usually exploit multiple replicates to prevent disappearance of data, especially sensitive data. An artificial or natural disaster may occur in the far distant future. We cannot conceive of such long term perceptions of time accurately with our current science and technology. If sensitive data disappeared, it may cause damage which would be an additional expense, uncalculated or unable to be recovered forever. Therefore, data backup is important, even if there are good techniques to improve data reliability.

With the amount of data growing explosively, the backup mechanism of HDFS can ensure data reliability, but it has brought the burden of storage space. To ensure reliability, the triple duplicates are generated when a file is newly written or...
an old file is updated, even if only a few blocks are updated in HDFS. Therefore, there will be too many similar and identical blocks in the system, and it also has to play ducks and drakes with the usage of storage space. To state this issue further, if we want to store a file, then four times the storage space is needed, and the cost of the storage system is also increased due to the threefold amount of added data. For persons with interests or an educational environment, they usually are not able to afford an excess budget, and for companies or movements there is no great trade in this quadruple investment of a file, especially in a petabyte scale or more, because the more the amount of data, the more the duplicate data. Hence, to win the maximal benefit, many researches and products have been devoted to eliminating the redundant data with a deduplication or erasure code.

However, those solutions are still not optimal. Although the block-level deduplication removes the redundant data as far as possible, the data reliability cannot guarantee the remnants. The duplicate data is still needed for the erasure code, but the number of copies can be only one, and, compared to deduplication, saving a file needs more storage space. Nevertheless, for persons with interests or an educational environment, they will want to save as much data in their system as possible, although, unfortunately, they cannot afford the storage space if they do not have enough funding. For this reason, under the limitation of storage space, using deduplication can improve the usage of the storage system effectively, and, according to the ability of clusters and the capacity of the data in the current situation, it makes different decisions that will improve system operation. Hence, a dynamic storage and management mechanism is needed, so we proposed a dynamic deduplication decision maker to improve memory space. Besides, the data reliability issue is also still important; therefore, in addition to the main storage system, we need the offsite backup which is also applied in most enterprises, even if their storage system has a great recovery mechanism for providing data reliability.

The rest of this paper is organized as follows. Section 2 describes the background and related works; it contains HDFS and deduplication. Section 3 mentions the overview of our proposed system. The evaluation of our implementation is presented in Section 4. Finally, conclusion and future work will be discussed in Section 5.

2. Background and Related Works

2.1. Background. Currently, distributed computing technology is one of the important roles. Consumers can enjoy this technology from Amazon [7] or Hadoop [3]. In fact, there are several subprojects in Hadoop which are listed in [3], such as HBase [8] (a distributed NoSQL database) and Hive [9] (a tool of data warehouse), and the main parts of those subprojects, including the Hadoop Common, Hadoop distributed file system (HDFS), and Hadoop MapReduce which are inspired from the Google file system (GFS) [2] and MapReduce of Google [10]. As shown in Figure 1, there are two Racks and many nodes for six clients. When the original block is finished, the first copy will appear on the node which is different than the original block's but in the same client. Then, the second copy will be found in the node which is on a different Rack. Finally, the third copy stored on the other node is distinct from the client of the second copy but in the same Rack.

Data compression technology is applied widely in many spaces to improve the data storage space and data transmission on the networks [11–15]. The bits of data are reduced by decreasing the repeat symbol of data; then, as a result of the bits of data being smaller than the original, the data transmission on the networks will operate faster and consequently the storage space needed to save data can be a lighter size. In addition, data compression sometimes can be applied in security. Therefore, the advantages of data compression are saving storage space, reducing the cost of transmission, cutting down the transmission time, and increasing security when necessary. As shown in Figure 2, first, there are several files, which are individually compressed with data compression or deduplication. Next, compared to data compression, the number of files A, B, C, and D is reduced to one after deduplication, but the total size of a single file is larger than the same file using data compression. Therefore, reducing the bits of data is the characteristic of data compression; it needs to eliminate the repeating data while in deduplication.

However, according to the distinct hash algorithms, the precision of the fingerprint will also be different. The common hash algorithms in deduplication are the Message-Digest Algorithm 5 (MD-5) [16] and the Secure Hash Algorithm 1 (SHA-1) [17]. Any amount of data in MD-5 can be calculated to a 128-bit fingerprint with four rounds of 16 operations, and a 160-bit fingerprint is transformed with four rounds of 20 operations whose max message size is $2^{64}$.-1 bits in SHA-1. Therefore, the former is faster, but the latter is more accurate. By the way, in terms of the block-level
deduplication, there are also different chunk algorithms, Fix-Size Chunking (FSC) [18] and Content-Defined Chunking (CDC) [19] algorithms. Although the CDC-based deduplication is more precise than the FSC-based (i.e., CDC-based deduplication will enable more storage space to be saved), FSC-based is applied in HDFS. Therefore, we will not discuss CDC in detail.

2.2. Related Works. Recently, with the generation of “big data” [1], the popular cloud computing industry is facing several challenges, including the ability of computation, storage, and management. To solve those problems, Hadoop has been applied in many researches and products because it is open source. In addition, the other reasons are not only its distributed computing ability, but also its reliability. In fact, the important core qualities of Hadoop are the Hadoop distributed file system (HDFS) and MapReduce [3]. However, the main disadvantage is that we need to incur high costs to save those duplicates. After a file has been stored in the storage system, it also needs triple memory space to save those copies. As shown in Figure 3, a file which is divided into ten blocks by HDFS generates triple duplicate. It increases the additional outcomes, because it is unnecessary to keep all of files in triplicate. Notwithstanding those copies, the reading performance can be improved, while the writing performance will be decreased. Nevertheless, with a large scale amount of data (achieving PB or more), it seems unreasonable to keep the triple copies because it will require an unforeseen payment for the storage space.

However, to improve this issue, HDFS-RAID has been developed which is a project by Facebook [20]. Different from triplication, HDFS-RAID divides a file into stripes consisting of several blocks and generates the corresponding parity blocks, which can not only save storage space more significantly, but also provide reliability for protection against data corruption. Although one duplicate is still needed, the parity blocks are smaller than the copies blocks. Besides, the parity blocks are placed in different DataNodes as triplication in HDFS. The primary technique uses erasure code such as XOR and Reed-Solomon code [21], with the latter used in HDFS-RAID. The XOR algorithm can only tolerate one error (only one failed block) in a stripe by running exclusive, or with two parity blocks between the original and duplicate stripe. However, the Reed-Solomon code can tolerate any four errors for a stripe. The system allows the user to determine the parity length, which results in the number of parity blocks. For example, if a file is divided into ten blocks in a stripe, there will be four parity blocks generated when the parity length is equal to four as shown in Figure 4. Actually, a file is perhaps divided into several stripes that rely on the stripe length. Compared with triplication in HDFS, HDFS-RAID only needs about 1.4 times the additional storage space for a stripe.

Although the Reed-Solomon code can improve the storage space and provide the reliability effectively, it still requires a high cost for maintenance. To solve this issue, M. Sathiamoorthy et al. [5] proposed the XORing Elephants in 2013 which are called Xorbas and which were developed from HDFS-RAID. However, as shown in Figure 5, HDFS-RAID runs the (10,4) Reed-Solomon, which represents that there are ten data blocks and four parity blocks in each stripe, while the LRCs use additional parity blocks which are generated from several coefficients of data blocks by a randomized and deterministic algorithm. When a block is lost, it can be recovered by accessing the other local blocks. For example, if block 4 is failure, then the block can be retrieved by $X_4 = c_1^{-1}(S_1 - c_1 X_1 - c_2 X_2 - c_4 X_4)$, where $X_4$ represents block 4 and $S_1$ represents local parity block 1. Besides, the system does not save the $S_1$ where $S_1 + S_2 + S_3 = 0$. However, in comparison with HDFS-RAID, although Xorbas needs more than 14% storage space to save the local parity blocks, disk I/O and repair network traffic can be reduced approximately by half.

However, the other interesting backup mechanism is the snapshot technique [22, 23]. Snapshot has been applied in many products, file systems especially. The benefits of snapshot are fast recovery and reduced storage capacity. Images are captured periodically by snapshot which records the current system status. When a disaster occurs, although it cannot recover all the data, the system can be repaired to the last status which was healthy by snapshot and logs. Suppose that the operation of HDFS is combined with snapshot; there will more and more similar images in HDFS, despite only different data being captured as an image periodically in each DataNode. Contrarily, snapshot can be applied to NameNode and JobTrack which are responsible for recording and assigning jobs, and although it is helpless in terms of the read performance, it can help the system to regularly...
Figure 4: The example of a file which is divided into a stripe with ten blocks in HDFS-RAID.

Figure 5: The example of a file which is divided into a stripe with ten blocks in Xorbas.

Figure 6: The example of a file which is divided into ten blocks with full deduplication.

Figure 7: The example of a file which is divided into ten blocks with deduplication which keeps only one copy for each block.

3. HDFS Architecture Using Dynamic Deduplication Decision Making

In the real world, the amount of data is growing exponentially; small and medium enterprises or educational organization will suffer from insufficient space problem. Therefore, in this paper, to face this challenge for storage systems, we propose a dynamic deduplication decision maker to improve the utility of storage space in HDFS, which is currently one of the best solutions for big data. Then, the proposed system architecture will be presented as the following. In this section, we will make a description of the system framework and the details of the components, algorithms, and flow charts separately.
3.1. The Overview of the System Framework. Facing the challenge of big data, where data is generated in a short time, the storage space is perhaps also filled with this data. To stock more and more data and make the storage space more effective, we propose a dynamic deduplication decision algorithm which makes a suitable deduplication decision to improve the storage space. However, our proposed system is built on HDFS, which is divided into two tiers as shown in Figure 8, and the two tiers present the timing of running deduplication with our dynamic deduplication decision algorithm. In fact, the outcome of our system is similar to the traditional HDFS in the initial period. As for the utility of memory space changes, this will bring different consequences, perhaps still maintaining triple replicates per block or no replicates at all.

In Figure 8, the users want to upload the data into the storage system from their personal devices or other servers, and then the storage system stores this data via two-tier deduplication, which runs on two different components, which are prefilter and postfilter, respectively. In fact, they also run different levels of deduplication. The prefilter is responsible for the deduplication in the file level. The reason is that when a file is already copied with triple duplication in HDFS to provide data reliability and to further save the storage space, storing redundant data repeatedly in the storage system is unnecessary and we can also gain the great benefit of time saving, because the duplicate file will not be processed in detail. However, the different files may include the same blocks (after chunking with HDFS), so we have also built the deduplication on a block level in our system, which runs on postfilter and which is responsible for eliminating the redundant blocks. Due to the triple duplicates in HDFS keeping per block of a file in quadruplicate (every block with one original and triple copies), even though the files are homoplastic (only a few blocks are different), HDFS still writes triple copies per block of every file. Therefore, the storage space can be saved by removing the same blocks of similar files. By doing so, the storage space can be improved more efficiently. Next, only the data from the data center to the other additional storage spaces which supports the offsite backup should be backed. Last but not least, all the data should be considered as useful for users, because we believe that if the data is ineffectual, then it will not be saved in the system (i.e., data should be used off-the-shelf actively by users). Without a doubt, actually, the users also do not want to upload useless data to the storage system which is on the cloud, because one characteristic of cloud is “pay-per-use.” Therefore, we can draw this inference from this fact.

3.2. The Situation of Operation in the Prefilter. To avoid storing the redundant files in the storage space, the main task in prefilter is to percolate the same file which has been saved in the storage system. In fact, as shown in Figure 8, we divided the HDFS into two tiers, with the prefilter as the first level which combines the chunk component of HDFS. Therefore, during the operation of the prefilter, all the data is forwarded in the file type. However, the remaining components in the prefilter are file filter and metadata. The file filter is responsible for percolating the input data which comes from users, and the metadata component records the information on the data in a table for identifying whether the file is redundant or not. Then, before the file is forwarded to the data center, the file will be divided into several blocks by the chunk component.

The flow chart of operation in the prefilter can be drawn as in Figure 9. When users want to save the data in the storage system by inputting the data, then, in the prefilter, the file filter will first check the storage to see whether saving this data will cause an overflow. If there is not enough memory to save this data, then the file filter will return the message to deny the request from users. Otherwise, the file filter will examine the data to see whether there is a duplicate with the attributes of a file from the metadata table. If it is redundant, then the data will be dropped from the file filter. Otherwise, the data will be divided into several blocks, and, at the same time, the attributes of the file will be updated in the metadata table. Finally, the data is forwarded to the data center in blocks.

However, the deduplication algorithm of the file level in the prefilter is shown in Algorithm 1, which is designed with pseudocode and with the time complexity of $O(n)$.

As in the above design, our deduplication of file level is according to the attributes of the data, whether any collisions will take place or not, rather than relying on the fingerprint of the file. The reason for this is to forward the new data without any delay. Comparing the fingerprint with the attribute, for the fingerprint, the system needs to first calculate the fingerprint, so the data has to wait until the computation has finished. On the other hand, for the attribute, no matter how large the data is, we only need to fetch the properties such as file name, user, built time, modified time, and size. Therefore, the computing time for the attribute is smaller than the time for the fingerprint. However, users will not create the same file with the same name at the same time theoretically. Thus, the deduplication of file level using identification of the properties of a file is feasible. By doing so, a new file will be written in our storage system only once.
3.3. The Function of the Postfilter. As mentioned in the last section, we can save some computing time before the redundant files are written into the system and thus make the usage of storage space more effective. However, only the file-level deduplication is still insufficient, because there is a high probability of redundant data that takes place between different but similar files; the intention is especially for updating the data. Traditionally, for the acceptable data, the storage system will follow the copy rule of HDFS, where the system can adopt the original data only for 25% of the total storage system with no extra storage space, because quadruple storage space is needed for saving a file in HDFS. The triple duplicate method in HDFS is designed for providing data reliability, but it also influences the usage of the storage system. Therefore, we need the further block-level deduplication.

As shown in Figure 8, the components after the prefilter are data center, postfilter, and metadata server. The data center is responsible for storing data and all the metadata are collected in the metadata server. To avoid the redundant blocks being kept in the storage space, the principal duty of the postfilter is to make the same block to be wiped out which has been stocked in the storage space. However, the postfilter is the second level of the divided HDFS structure which is responsible for the management of blocks. Besides, we examine the duplicate block with the fingerprint which is calculated by hash function.

As shown in Figure 10, when the data received from the prefilter has been divided into several blocks, the fingerprint of the block in the postfilter will be calculated with the hash
function. Then, we seek out all of the fingerprints in the metadata server to confirm whether they are duplicate blocks or not. If there is any block with the same fingerprint, then the duplicate block is dropped. Otherwise, it is forwarded to the data center and the blocks are saved in the DataNodes. Next, to offer data reliability, the duplicate blocks will be created as HDFS. Finally, according to the current situation of the storage space, there will be distinct deduplication decisions.

As in the above flow chart, our deduplication of block level is according to the fingerprint of the data to see whether any collisions will take place or not. If there are any collisions, then the block will be dropped out by the postfilter. Otherwise, until the blocks are stored in the data center, the system will check the utility of storage space to run the dynamic deduplication with different decisions. Before we describe the dynamic deduplication decision-making algorithm, we first provide some clear definitions of parameters which will be applied in our deduplication algorithm with the definitions as follows:

\[ U_T = \sum U_{M_H} + \sum (U_{M_C} \times N_{M_C}), \] (1)

where the meaning of \( U_{M_H} \) is the utility of memory space per block of the original, while \( U_{M_C} \) represents the utility of memory space per block of copy. Then, the parameter, \( N_{M_C} \), represents the total number of copies per original. Finally, the sum of the summation of \( U_{M_H} \) and the product of \( U_{M_C} \) and \( N_{M_C} \) is \( U_T \), that is to say the total utility of memory space in the primary storage system. Therefore, (1) indicates the current total utility of memory space in the system which will be referred to as the condition that determines the different deduplication decisions.

Besides, we also provide other clear definitions of the parameters, which are the hot and cold data, respectively; the hot data means that data is accessed frequently. In other words, it is very important data in the current system with a strong probability, whereas, compared to hot data, cold data is accessed infrequently. Thus, the value of cold data does not seem as important as the hot data. Consequently, the definitions are as follows:

\[
\begin{align*}
\text{Compare} & (PT, LT) \\
\text{block} &= \text{cold data, if } PT < LT, \\
\text{block} &= \text{hot data, otherwise,}
\end{align*}
\] (2)

where the intent of \( PT \) is to check the situation for accessing the data in a period of time, and we assume \( PT \) as one month. Then, the parameter, \( LT \), represents the time of last accessed data. Therefore, in order to compare the interval between \( PT \) and \( LT \), if the result is positive (i.e., \( PT \) is greater than \( LT \)), it describes that the data will be accessed with high probability in the future, and thus the data is called hot data. Otherwise, the data is called cold data, because in general we do not place the kind of data which will be accessed after a long time with pay-per-use in cloud computing. By doing so, the data will be determined as still so important by our system and it can also run the next dynamic deduplication decision with the relevant consequences. Therefore, as mentioned above, the equation can be revised from formulae (1) and (2) as follows:

\[
U_T = \sum (U_{M_{CH}} + U_{M_{CC}}) + \sum (UM_{CH} \times N_{M_{CH}} + UM_{CC} \times N_{M_{CC}}),
\] (3)

where the \( H \) and \( C \) behind the original parameters mean the parameters of hot data and cold data individually. For example, the meaning of \( N_{M_{CC}} \) is the amount of cold duplicate data. In fact, this modification of the original formulas is just the completed definitions.

However, in our proposed dynamic deduplication decisions, the deduplication algorithm of the block level in the postfilter is shown in Algorithm 2, which is designed with the following pseudocode, and the time complexity is also \( O(n) \).

Under the permission of system space, to provide high reliability and decrease the workload of data when read, at first, the system will follow the copy rule of HDFS, but with the operation of deduplication of block and file level. That is to say, our system allows a file to be saved into the storage space only once, and its blocks will be copied in triplicate (one original and triplicate duplicates), but if there are any collisions of the fingerprint place between the block which is writing and the other block which is already in the system, then the block which is writing will be dropped out. The reason is that the redundant blocks will deteriorate the utility of the storage space. Therefore, in the block level deduplication algorithm, there are two added variables, \( \text{threshold}_1 \) and \( \text{threshold}_2 \), respectively, which are the norm for different deduplication decisions, and the details of those variables are evaluated and illustrated in Figures 11, 12, and 13. When \( U_{M_T} \) is smaller than \( \text{threshold}_1 \), the decision is to discard all the redundant data, except for the copies from HDFS, and the number of copied blocks is three. Thus, in general, under this condition, the utility of storage space can be written as (3); that is,

\[
U_{M_T} = \sum (UM_{OH} + UM_{OC}),
\] (4)

where both the \( N_{M_{CH}} \) and \( N_{M_{CC}} \) are three. However, to ensure that the file can still be read normally after eliminating the redundant data, the original index which points out the redundant data will be modified to point out the same data which is still alive in the system.

Then, when the \( UM_T \) is greater or equal to \( \text{threshold}_1 \), if there are any number of copies of cold data which are greater or equal to one, then all of the copies of the per cold data will be removed. The opportunity is to provide more space to save more data, so we will release the storage space from those copies of cold data, because the cold data maybe will be accessed after a long time, as we do not care about the reliability too much in the short term and when a failure happens to this cold data, the system can still rely on the recovery of the offsite backup. Therefore, when \( UM_T \) achieves
**Algorithm 2**

Block level De-duplication Algorithm

// Input: data in the type of block from pre-filter
// Output: several blocks or no output to datacenter
// all the fingerprint of stored blocks in the metadata server
While receive the data from pre-filter do

1. Compute the fingerprint of data with hash function;
2. For seek out the index table
3. If there are any collision between the fingerprint and index table
   Then drop out the request of save data;
   Else forward the block to a DataNode;
4. If there are any successful for saving the block
   Then fetch the utility of storage space;
5. Switch (the utility of storage space)
   Case of the utility storage space is greater or equal to threshold₁
       If the number of copies of any cold data is greater or equal to one
           Then remove all of the copies of per cold data
           break;
   Case of the utility storage space is greater or equal to threshold₂
       If the number of copies of any hot data is greater or equal to one
           Then eliminate one of the copies of those hot data
           break;
   Default
       Following the copy rule of HDFS
       but run the De-duplication of file and block level;
   break;
End While

\[ UM_T = \sum (UM_{OH} + UM_{OC}) + \sum (UM_{CH} \times \text{Num}_{CH}) \]  \hspace{1cm} (5)

Next, when the \( UM_T \) is greater than or equal to \( threshold_2 \) and all of the cold data is without any copies, if there are any number of copies of hot data that are greater or equal to one, then one of the copies of that hot data is eliminated (i.e., \( \text{Num}_{CH} \) subtracts one from every copy of hot data). The motive is to supply more space to store more data, so we have no choice but to release the storage space from those copies of hot data. However, the hot data is accessed frequently, so although the reliability is not like before, the workload of reading can still be improved until there are no copies of hot data.
data. Therefore, when $UM_T$ achieves $\text{threshold}_2$ or more, the decision is to eliminate one of the copies of hot data gradually. Unfortunately, if there are no copies, the $UM_T$ is equal to the real size of all data; that is, $UM_T = \sum(UM_{off} + UM_{cc})$. This is the worst circumstance where all of the data cannot be repaired from local data, because there was no same data. For this reason, the reliability will only rely on the offsite backup.

3.4. Offsite Backup. For many companies or a person, it is common to use offsite backup technology, even if the primary system can ensure high reliability. Without a doubt, we never know when a calamity will arrive, and if we only trust the recovery ability of the primary system, then unfortunately that primary system could become the victim in a calamity. Then important data could be lost forever and could cause damage which is impossible to estimate. Therefore, despite the high reliability of the primary system, an offsite backup is needed.

Besides, the entire offsite backup, which we have discussed, is an independent workshop, so it does not involve the inner operations of the major system but offers storage space to save the backup away from the major system. In other words, the offsite backup is an extra data center rather than a secondary DataNode. However, recovery with the offsite backup is involved with the transmission issue of the networks. There are too many problems in the networks, such as networks unable to be connected between the source and destination, packets lost or failed during communication, transmission delay, and propagation delay. Therefore, to provide a great offsite backup, backups in both the local area network (LAN) and the wide area network (WAN) are our choice.

Finally, due to the deduplication methods in our system, in general, the same file does not allow writing into the main storage system. Therefore, to recover lost data, when only one block is extinct, the principal system will permit the repair of the target file from the offsite backup, and then after the target file has been mended, the permission to write the same file from the offsite backup will return to default. For instance, if the specific recovery is from LAN and WAN offsite backups, and the repaired job has been finished by the system from LAN, then the allowance of special writing will occur from active to native; hence, the request from the system from WAN will be denied.

4. Simulation

In terms of the system memory, we will test and verify the efficiency of our proposed method in this section and compare it with some related research. However, it consists of two parts in which our experimental environment will be revealed in the first part, while the second part will exhibit the outcome of the experiment.

4.1. Experimental Environment. It was fortuitous that we could rent a virtual machine from Chunghwa Telecom (i.e., hicloud [30]) according to the terms of an industry-academia cooperation agreement, and, for building our system architecture, we used the listed infrastructure in Table 1. We built the cluster using numbers 1, 2, 3, and 4 in Table 1, with the HDFS running on this; in other words, this was also our main storage system. To simulate the circumstances in the real world, our system was constructed on different machines which were connected to the network instead of using VMs on the same machine. First, we hired two kinds of infrastructure resources which have different sized RAM, disks, and distinct processing time. In addition, we still applied another infrastructure resource which plays the role of the Offsite Backup system in LAN. Finally, we still made different infrastructures as the Offsite Backup system in WAN which are numbers 6 and 7 in Table 1, both of which are physical machines, rather than the virtual machines as previously mentioned. In fact, these might also have different domain names, because it is possible that, in the real world, an enterprise would place their data in different areas to prevent the disappearance.

However, as mentioned earlier, we set up the HDFS on the cluster which is composed of the hosts from number 1 to 4. The initial summary of the cluster is shown as in Table 2, which also illustrates that the total capacity of our system is only 117.65 GB. Therefore, to meet the prerequisites of this paper, we will not expend any of the capacity of our system by using extra infrastructure resources. Nevertheless, it is ironic that we cannot elastically build our system with hicloud, because the provider does not enable this authority. If we want to increase the total capacity in hicloud, the only solution is to rescind the current contract and make a new request when we need hicloud again.

4.2. The Outcome of the Experiment. First, as remarked in Section 3, we displayed the framework of our dynamic deduplication decisions and also provided the values of $\text{threshold}_1$ and $\text{threshold}_2$. In our thinking for $\text{threshold}_1$, if we set the value too low, the HDFS fails to get the best out from the distributed computing, because all the copies of the cold data will first be removed when the utility of the storage system has achieved $\text{threshold}_1$. Therefore, we only consider the value of $\text{threshold}_1$, which should be greater than half of the utility of the storage space, but less than or equal to $\text{threshold}_2$. However, $\text{threshold}_2$ also cannot take a low value, and the reason is that the reliability of the hot data needs to rely on the duplicates. Nevertheless, in our experiment, we regularly input the data to the system and the amount of data is of a fixed size every time which occupies about 5% of the total storage space. Due to the new data in the system following the copy rule of HDFS in the default, the system needs 20% of the storage space every time, whereas, we want to retain storage space so that the data can be written into the system next time, so the maximal value of $\text{threshold}_2$ can be set as 80%.

Therefore, we sampled the similar variables to run the experiments for determining the ($\text{threshold}_1$, $\text{threshold}_2$) with the results shown as in Figures 11 and 12. In Figure 11, we sample the (50, 80) as the first base model of ($\text{threshold}_1$, $\text{threshold}_2$) and compare them; we also take the case of
with 5% of the data which we inputted at the same time, our system will produce the same outcome when the $\text{threshold}_1$ is set between 50 and 60 or 70 and 80 with the equivalent $\text{threshold}_2$. Moreover, dynamic deduplication takes place in the noteworthy variation of curves, and then, according to our statistics, the least amount of operation appears in $(70, 80)$ and $(80, 80)$, while the most occurs in $(50, 70)$. In addition, it is worth mentioning that the second worst case is in $(70, 70)$. We infer the reason for this is that we assume that the minimum $\text{threshold}_1$ should be greater than half of the storage space at first, so the replicas of the cold data will not exist in the system when the utility of the storage system has reached $\text{threshold}_1$. Besides, as time goes by, the utility of the storage system will be maintained between 60% and 80% accordingly by our deduplication method, which is conducive to the maximum amount of times of execution taking place when the $\text{threshold}_2$ is equal to 70.

Nevertheless, the more times the deduplication occurs, the more the performance is influenced, and, to confirm the final value, we vary the data size when inputting from 4% to 6% with the $(70, 80)$ and $(80, 80)$ because they had the best performance on the last experiment. The new result is shown in Figure 13. Therefore, we can draw the conclusion that $(\text{threshold}_1, \text{threshold}_2)$ is set as $(80, 80)$ in our system because the rounds of deduplication in $(70, 80)$ are more than the number in $(80, 80)$. A reasonable reason for explaining this consequence is that the total utility of storage has sometimes reached 70% or more but less than 80% when input different sizes of data are inputted, so the timing of deduplication of $(80, 80)$ is delayed.

After $(\text{threshold}_1, \text{threshold}_2)$ has been settled, in terms of storage space, we attempt to compare the evaluation of our system with related works by using three kinds of experiments. Owing to our proposed system being built on HDFS, we simulate the traditional HDFS and Xorbas as a reference. First we increase the utility of new data to about 5% in a time period while all the data is different from the reference. First we increase the utility of new data to about 5% in a time period while all the data is different from the others with the result of this evaluation shown as in Figure 14. Unfortunately, no matter which system, it is impossible to stock all the data, but our system can save most of the data from the results of the analysis. The traditional HDFS can

| No. | OS          | CPU                  | RAM | Disk   | Intention          |
|-----|-------------|----------------------|-----|--------|--------------------|
| 1   | CentOS6.2   | 4 Virtual Core (4.0 GHZ) | 8 GB | 100 GB | Cluster (Main System) |
| 2   | CentOS6.2   | 4 Virtual Core (4.0 GHZ) | 8 GB | 100 GB | Cluster (Main System) |
| 3   | CentOS6.2   | 2 Virtual Core (2.0 GHZ) | 4 GB | 100 GB | Cluster (Main System) |
| 4   | CentOS6.2   | 2 Virtual Core (2.0 GHZ) | 4 GB | 100 GB | Cluster (Main System) |
| 5   | Ubuntu 10.0 | 4 Virtual Core (4.0 GHZ) | 8 GB | 100 GB | Offsite Backup     |
| 6   | Windows 7   | Intel Xeon CPU × 2 E3110 3.00 GHZ | 2 GB | 465 GB | Offsite Backup     |
| 7   | Ubuntu 10.0 | AMD Athlon II × 4 635 Processor 2.90 GHz | 6 GB | 600 GB | Offsite Backup     |

Table 2: The initial cluster summary.

(a) Cluster summary: 40 files and directories, 8 blocks = 48 total. Heap size is 28.83 MB/1.39 GB (2%)

| Configured capacity | 117.65 GB |
|---------------------|-----------|
| DFS used            | 344 KB    |
| Non-DFS used        | 0 KB      |
| DFS remaining       | 117.65 GB |
| DFS used%           | 0%        |
| DFS remaining%      | 100%      |
| Live nodes          | 3         |
| Dead nodes          | 0         |
| Decommissioning nodes | 0    |
| Number of under-replicated blocks | 0 |

(b) NameNode storage

| Storage directory /opt/hadoop/hdfs/dfs/name | IMAGE_AND_EDITS | Active |
|--------------------------------------------|------------------|--------|

(50, 70) and (80, 80) where we have the relationship to the (50, 80) by varying $\text{threshold}_1$ or $\text{threshold}_2$. In addition, we also sample the (70, 80) as the second base model of ($\text{threshold}_1$, $\text{threshold}_2$), and to compare them, the cases of (70, 70) and (60, 80) are taken for comparison by varying $\text{threshold}_1$ or $\text{threshold}_2$ from the (70, 80) in Figure 12. To explain and analyze this conveniently, we call (50, 80), (50, 70), and (80, 80) group A, while group B is composed of (70, 80), (70, 70), and (60, 80). The task in group A is to observe whether the outcome will vary or not when only one of the thresholds is changed, while group B is for testing whether the transition will take place or not when $\text{threshold}_1$ is different from group A. Furthermore, it is worth noting that, no matter what $\text{threshold}_1$ or $\text{threshold}_2$ is, the system will produce the same result initially and at the end, because the operation of the system is as we have designed.

However, it is an interesting observation that there are some of the same outcomes which makes their curves redundancy such as (50, 80) and (60, 80). As shown in Figures 11 and 12, due to 20% of the data being generated by HDFS

Table 1: The host environment.

| No. | OS          | CPU                  | RAM | Disk   | Intention          |
|-----|-------------|----------------------|-----|--------|--------------------|
| 1   | CentOS6.2   | 4 Virtual Core (4.0 GHZ) | 8 GB | 100 GB | Cluster (Main System) |
| 2   | CentOS6.2   | 4 Virtual Core (4.0 GHZ) | 8 GB | 100 GB | Cluster (Main System) |
| 3   | CentOS6.2   | 2 Virtual Core (2.0 GHZ) | 4 GB | 100 GB | Cluster (Main System) |
| 4   | CentOS6.2   | 2 Virtual Core (2.0 GHZ) | 4 GB | 100 GB | Cluster (Main System) |
| 5   | Ubuntu 10.0 | 4 Virtual Core (4.0 GHZ) | 8 GB | 100 GB | Offsite Backup     |
| 6   | Windows 7   | Intel Xeon CPU × 2 E3110 3.00 GHZ | 2 GB | 465 GB | Offsite Backup     |
| 7   | Ubuntu 10.0 | AMD Athlon II × 4 635 Processor 2.90 GHz | 6 GB | 600 GB | Offsite Backup     |
only accept about 25% of the data, because of the limiting factors of the copy rule of HDFS. However, Xorbas can sustain 60% of the data or so, which can only support one copy per the original data and the generated parity blocks from the system, while our system can still support more than 25% of the data or thereabouts by discarding the replicates at various times.

Next, we perform the second test in which we not only upload the new data at about 5% of the total system capacity in a period of time but also submit the last added data since the second time. In the other words, except for the 5% of data inputted during the first time, there is 10% of the data that will be forwarded to the system with half of that data as duplicates. The tentative consequence is depicted in Figure 15. An observation we can make is that it is a failure to store all the data again, but both the traditional HDFS and Xorbas declare the information that they will withdraw from the competition of this evaluation ahead of time, because they cannot remove the redundant data as ours can. That is to say, although we extend the total amount of data for inputting, the duplicate data captures 95% or so, and our proposed system has the competence of deduplication on the file- and block-level which saves more storage space just for new data, which makes our system able to support 85% of distinct data at the end of this case.

Finally, the third experimental result is presented as Figure 16 which is repeated varying the data size between 4% and 6% when inputting the data into the systems. The traditional systems have the same outcome as their performance in the first test while our proposed system still outperforms both of them. We ascribe this consequence to the data size of the input, because the utility of our storage system after inputting new data will not have reached the thresholds yet where the deduplication will not be enforced. Additionally, there is
still enough memory for saving the eighteenth data, so the amount of writing is been raised 6% more than previously.

Unfortunately, although we have improved the usage of the storage space, our system is still not at an optimum level. In general, the data reliability can be guaranteed by multiple replications; that is to say, we can still acquire the data from its replications even if the target data is broken. However, in our system, the promise of reliability will be broken piecemeal when the utility of the storage space has reached the threshold $\theta_1$ and threshold $\theta_2$. To describe the drawback of our system, we adopt two variables as shown in Table 3, which are Delay time and Repair time, respectively. As mentioned above, accessing data is still successful before the target data and all the replications disappear, but when it is unable to read the target data, then we will spend some time searching for substitutes with the interval between them called Delay time. In fact, this change will not take too much time, but note that, with the appearance of destroyed data, a new and important repair action will start immediately, and the time of the status of data changing from destroyed to active is called Repair time. In other words, if the Repair time increases, there will be destroyed data. Nevertheless, it is possible that the Delay time is tantamount to being the same as Repair time because the success in accessing data takes place after the data has been recovered. Moreover, it also means that data loss occurs during this Delay time.

At the end of this paper, we perform the evaluations of mean Delay time and Repair time, and the results are presented in Figures 17 and 18, respectively. Destroyed data is under 50% in all systems, except for Xorbas which can rapidly recover data when the parity blocks are still alive, and no matter if the replications are from other Racks or the Offsite-backup, the remainders only can rely on the real network bandwidth. Therefore, although the performance of our system is equal to HDFS, it falls short when compared to Xorbas.

Actually, as mentioned above, the comparison between our method and Xorbas can be summarized in Table 4.

| Number of data block | Delay time | Repair time |
|----------------------|------------|-------------|
| 4                    | 0          | 0           |
| 3                    | Scarcely   | Increasing  |
| 2                    | Scarcely   | Increasing  |
| 1                    | Scarcely   | Increasing  |
| 0                    | Increasing | Increasing  |

5. Conclusion and Future Works

In the past few years, the cloud computing technique has brought some great benefits. However, with the arrival of big data, some difficult issues have come to light. In this paper, we discussed one of the important problems which is about storage space in which HDFS plays a principal role in cloud computing, but which is also a cause for concern because the copy rule of HDFS needs quadruple storage space just for saving a file, and the added replicates will occupy most of the storage space. Actually, this is an expensive cost, especially in a petabyte or greater scale, because the retained data; the probability of redundant data becomes higher. Therefore, we proposed a dynamic deduplication decision maker to improve the usage of storage which was compared with some related works in the paper. Our proposed method, according to the current situation of systems, forms a suitable solution. Our system is aimed at small enterprises and organizations, especially people with interests or educational environments, that can more effectively use storage space by removing duplicates without investing any additional funding in infrastructure. However, the evaluation of our proposed system shows that our method can save more data than the others.

For all that, our method is not optimal. Although the system can thus save more data, the data reliability will not be guaranteed gradually by the system when the utility of storage space has reached $\theta_1$ and $\theta_2$. Due to the contradiction between data reliability and storage
space, perhaps the best solution is to increase the capacity of system, while the prerequisite is that we must have enough money. Therefore, in the future, we need a suitable method to ensure data reliability without the loss of storage space. Otherwise, we can only rely on the Offsite Backup, and we must overcome the transmission issue on the networks. Although our simulations only run on Hicloud, we will find new experimental environment to run our simulations on a large scale cluster system in the future work.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This research is supported in part by ROC NSC under contract no. NSC 101-2221-E-259-005 MY2.

References

[1] D. Laney, 3D Data Management: Controlling Data Volume, Velocity and Variety. Application Delivery Strategies, META Group, 2001, http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.

[2] S. Ghemawat, H. Gobioff, and S. Leung, “The google file system,” in Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP ’03), pp. 29–43, Bolton Landing, NY, USA, October 2003.

[3] Hadoop, http://hadoop.apache.org/.

[4] Sun Microsystems, Take Your Business to a Higher Level: Sun Cloud-Computing Technology Scales Your Infrastructure to Take Advantage of New Business Opportunities, 2009, http://www.aeiseguridad.es/descargas/categoria6/4612546.pdf.

[5] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos et al., “XORing elephants: novel erasure codes for big data,” Proceedings of the VLDB Endowment, vol. 6, no. 5, pp. 325–336, 2013.

[6] D. Borthakur, The Hadoop Distributed File System: Architecture and Design, 2007, http://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf.

[7] Amazon, http://www.amazon.com/.

[8] HBase, http://hbase.apache.org/.

[9] Hive, http://hive.apache.org/.

[10] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” in Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation (OSDI ’04), pp. 137–149, San Francisco, Calif, USA, 2004.

[11] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” IEEE Transactions on Information Theory, vol. 23, no. 3, pp. 337–343, 1977.

[12] J. Ziv and A. Lempel, “Compression of individual sequences via variable-rate coding,” IEEE Transactions on Information Theory, vol. 24, no. 5, pp. 530–536, 1978.

[13] T. A. Welch, “A technique for high-performance data compression,” Computer, vol. 17, no. 6, pp. 8–19, 1984.

[14] P. Ning and L. Hesselink, “Vector quantization for volume rendering,” in Proceedings of the 1992 Workshop on Volume Visualization (VVS ’92), pp. 69–74, Boston, Mass, USA, 1992.

[15] P. Ning and L. Hesselink, “Fast volume rendering of compressed data,” in Proceedings of the 4th Conference on Visualization (VIS ’93), pp. 11–18, San Jose, Calif, USA, 1993.

[16] MD-5-Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/MD5.

[17] SHA-1-Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/SHA-1.

[18] S. Quinlan and S. Dorward, “Venti: a new approach to archival storage,” in Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST ’02), pp. 89–102, Monterey, Calif, USA, 2002.

[19] A. Muthitacharoen, B. Chen, and D. Mazieres, “A low-bandwidth network file system,” in Proceedings of the 18th ACM symposium on Operating Systems Principles (SOSP ’01), pp. 174–187, Banff, Canada, 2001.

[20] HDFS-RAID wiki, http://wiki.apache.org/hadoop/HDFS-RAID.

[21] I. Reed and G. Solomon, “Polynomial codes over certain finite,” Journal of the Society for Industrial and Applied Mathematics, vol. 8, no. 2, pp. 300–304, 1960.

[22] O. Rodeh and A. Teperman, “zFS-a scalable distributed file system using object disks,” in Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST ’03), pp. 207–218, San Diego, Calif, USA, April 2003.

[23] S. Marathe, P. Massiglia, N. Pendharkar, and P. Vajgel, Using Local Copy Services, 2006, http://eval.symantec.com/mktginfo/enterprise/yellowbooks/using_local_copy_services_03_2006.pdf.

[24] N. Kuromatsu, M. Okita, and K. Hagihara, “Evolving fault-tolerance in Hadoop with robust auto-recovering JobTracker,”

### Table 4: The comparison between our method and Xorbas.

| System name       | Dynamic deduplication decision making                                                                 | Xorbas                                                                 |
|-------------------|--------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------|
| Operation manner  | (1) Provides the data reliability using triplicate at first and (2) eliminates some copies when the storage space achieves a high load | (1) Divides a file into a fixed size stripe and (2) maintains one copy with few parity blocks per stripe |
| Advantage          | (1) Avoids redundant data with block and file level deduplication and (2) releases more storage spaces for new data | (1) Provides the data reliability with few copies and parity blocks and (2) has a faster repair capability |
| Disadvantage       | (1) Data reliability decreases when the elimination has been beginning and (2) the distributed computing capability turns into unavailable without copies | The more the amount of same or similar files, the more the redundant data |
Bulletin of Networking, Computing, Systems, and Software, vol. 2, no. 1, pp. 4–11, 2013.

[25] C.-I. Fan, S.-Y. Huang, and W.-C. Hsu, “Hybrid data deduplication in cloud environment,” in Proceedings of the International Conference on Information Security and Intelligence Control (ISIC ’12), pp. 174–177, Yunlin, Taiwan, 2012.

[26] W. Xia, H. Jiang, D. Feng, L. Tian, M. Fu, and Z. Wang, “P-Dedupe: exploiting parallelism in data deduplication system,” in Proceedings of the IEEE 7th International Conference on Networking, Architecture and Storage (NAS ’12), pp. 338–347, Fujian, China, 2012.

[27] B. Mao, H. Jiang, S. Wu, Y. Fu, and L. Tian, “SAR: SSD assisted restore optimization for deduplication-based storage systems in the cloud,” in Proceedings of the IEEE 7th International Conference on Networking, Architecture and Storage (NAS ’12), pp. 328–337, Fujian, China, 2012.

[28] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson, “DiskReduce: RAID for data-intensive scalable computing,” in Proceedings of the 4th Annual Petascale Data Storage Workshop (PDSW ’09), pp. 6–10, Portland, Ore, USA, November 2009.

[29] Y. Zhang, Y. Wu, and G. Yang, “Droplet: a distributed solution of data deduplication,” in Proceedings of the ACM/IEEE 13th International Conference on Grid Computing (GRID ’12), pp. 114–121, Beijing, China, 2012.

[30] Chunghwa Telecom hicloud, http://hicloud.hinet.net/.