Research Article

A Bit String Content Aware Chunking Strategy for Reduced CPU Energy on Cloud Storage

Bin Zhou,¹ ShuDao Zhang,² Ying Zhang,³ and JiaHao Tan¹

¹College of Computer Science, South-Central University for Nationalities, Wuhan, Hubei 430074, China

²School of Information Engineering, Wuhan Technology and Business University, Wuhan, Hubei 430065, China

³School of Foreign Languages, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China

Correspondence should be addressed to Bin Zhou; binzhou@mail.scuec.edu.cn

Received 11 May 2015; Accepted 30 July 2015

Academic Editor: Lu Liu

Copyright © 2015 Bin Zhou 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.

In order to achieve energy saving and reduce the total cost of ownership, green storage has become the first priority for data center. Detecting and deleting the redundant data are the key factors to the reduction of the energy consumption of CPU, while high performance stable chunking strategy provides the groundwork for detecting redundant data. The existing chunking algorithm greatly reduces the system performance when confronted with big data and it wastes a lot of energy. Factors affecting the chunking performance are analyzed and discussed in the paper and a new fingerprint signature calculation is implemented. Furthermore, a Bit String Content Aware Chunking Strategy (BCCS) is put forward. This strategy reduces the cost of signature computation in chunking process to improve the system performance and cuts down the energy consumption of the cloud storage data center. On the basis of relevant test scenarios and test data of this paper, the advantages of the chunking strategy are verified.

1. Introduction

Along with the development of the next generation of network computing technology, such as the application of Internet and cloud computing, the scale of the data center is showing the explosive growth in the past 10 years. The total amount of global information is double every 2 years. It was 1.8 ZB in 2011 and will reach 8 ZB in 2015. In the next 5 years (2020), the data will be 50 times higher than that of today [1].

According to the 2005 annual report of the international well-known consulting company Gartner [2], the total power consumption was 45,000,000,000 kwh of electricity for the whole data centers in USA in 2005. In its latest report about the data center efficiency the Natural Resources Defense Council (NRDC) [3] suggested that it was 91,000,000,000 kwh of electricity in 2013, which was more than double the total power for homes of New York. The annual electricity consumption will reach 140,000,000,000 kwh by 2020. Data center has developed into one of the great energy consumption industries rapidly.

How was electrical energy wasted in the data center? References [4–6] carried out experimental analysis and found

out that the biggest power consumption was from first refrigeration system and the server, second IT infrastructure, such as storage and network air conditioning, and last air conditioning and lighting, and so forth. Among them, the storage system consumes about 14% electrical energy [7]. Construction of energy-efficient green data center is the future research and development tendency for the industry. Most businesses including IBM, Microsoft, HP, SUN, Cisco, Google, and other companies have launched a research on green data center.

Green storage technology refers to reduction of the data storage power consumption, electronic carbon compounds, the construction and operation cost, and improvement of the storage equipment performance in terms of the environment protection and energy saving. The study shows that there is a large amount of redundant data in the growing data. And the proportion of redundancy will be higher as time goes on, leading to a sharp decline in the utilization of storage space. The low utilization of storage space leads to a waste of both storage resources and lots of energy.

In order to improve the utilization of storage space, a large number of intelligent data management technologies, such

as virtualization, thin provisioning, data deduplication, and hierarchical storage, have been widely applied in the cloud data center. Deduplication, now a hot research topic, meets the ends of energy conservation, efficiency of customers' investment and reduction of emission, and operation costs by saving lots of storage space, improving the performance of data read and write, and lessening the bandwidth consumption effectively.

Research shows that a large amount of data is duplication in the growing data. Therefore, the key for reducing the data in the system is to find and delete the redundant data. The basis of detecting it is the high-performance and stable chunking strategy.

It takes advantage of signature calculation to partition the data objects in the most existing literatures [8–10] on the variable length data chunking algorithms. The research on the optimization measure of time cost for data chunking algorithm is relatively few.

It is CPU intensive for detecting the chunk boundaries through signature calculation. For a 100 MB file, if the expected average length of chunk is 8 KB, the minimum length of chunk is 2 KB, and the file is expected to be divided into 12800 chunks. For each chunk boundary, 6144 times signature calculations and comparisons are required. $12800 * 9144$ times signature calculations and comparisons are required for a 100 MB file. In the heavy CPU operation, the ratio of the number of chunk to the total number of signature calculation is only 1:6144, which is a difference of nearly three orders of magnitude. A lot of valuable CPU resources are consumed by invalid signature calculations and comparisons, which leads to more waste of energy.

The key problem of improving chunking performance lies in cutting down the consumption of CPU resources by reducing the number of invalid signature calculations and comparisons.

In this paper, we build the BCCS to a prototype of storage system. The primary contribution of our work is as follows. Firstly, as the deficiencies of total signature computations of Rabin fingerprinting algorithm are very huge, it is very CPU-demanded to divide the chunks that decrease the system performance. A new strategy, BCCS, is presented, reducing the overhead of generating fingerprints, as well as converting the question of files data partitioning stably to matching two binary bit strings. It samples a special bit from a text byte to constitute its data fingerprint, which converts the signature computation to binary string comparison. Only 1/8 of the text reads originally, and the bit operation is taking place of the traditional comparison. Secondly, by replacing the comparison operation, BCCS uses bitwise operation to optimize each matching process and exclude the unmatching positions as much as possible, getting the maximum jumping distance to quicken the matching process of binary string. It also reduces calculation and comparison costs by taking advantage of the bit feature information brought by failure matching every time. This measure reduces the cost of signature computation in chunking process and brings down CPU resource consumption to improve the system performance.

BCCS also divides the chunking process into two steps: the preprocessing step and the partitioning step. Then,

BCCS adopts the parallel processing mode dealing with the preprocessing at a preprocessing node that synchronizes with partitioning at the front-end node. It saves the time of the preprocessing when the system is working. All of the measures minimize the total time consumption for data partitioning.

The rest of this paper provides the following. Section 2 introduces related works of traditional chunking algorithms. Section 3 studies BCCS algorithm in detail. In Section 4, we perform various experiments to measure the overhead of the data chunking, and Section 5 draws the conclusion.

2. Related Works

In order to solve the stability problem of chunking, Muthitacharoen et al. [11] adopted a variable length chunking algorithm named CDC based on content in the Low Bandwidth Network File System. It predetermined a piece of data as the boundary of the chunk in the pending data objects (called Marker in this paper) and determined the boundary of each chunk by finding the Marker. In actual practice, Rabin algorithm [12] is used to calculate the fingerprint of the data in the window to find the boundary of each chunk in the data objects. But the CDC algorithm remains inadequate. Firstly, for the chunk of different sizes, in some cases, since volatility is relatively large, it increases the difficulty of storage management. Secondly, the granularity settings will directly affect the precision and effect of duplicate. The higher the precision of deduplication, the smaller the granularity, but the chunk metadata overhead will increase to affect the overall performance of the system. Conversely, if the granularity is too large, reducing the metadata overhead, it is possible that the duplicate precision cannot reach the requirements. Thirdly, it is possible that the boundaries of chunks cannot be found, and hard chunking will be employed.

Eshghi and Tang [13] proposed double threshold algorithm TTTD (Two Thresholds Two Divisors) to get the stability for chunking, which regulated the size of chunk only in the upper and lower bounds ($\min_C < \text{length}_C < \max_C$, length_C is the length of chunk). To a certain extent, TTTD alleviated the contradiction between chunking granularity and metadata overhead. However it did not completely solve the problem of hard chunking.

Bobbarjung et al. [14] put forward another chunking algorithm named FingerDiff based on variable length chunking. The main idea of this algorithm was merging multiple chunks that were not changed into a single SuperChunk as far as possible to reduce the amount of chunk metadata. To a certain extent, it could reduce the storage space and it accelerated the speed of query.

Research works on how to avoid long chunks, how to choose the optimal chunk size to achieve the optimal deduplication effect, and so on have been done a lot; nevertheless it is seldom mentioned in the literature how to reduce the overhead of fingerprint calculation for the data in the sliding window in the chunking process. When the amount of data becomes large, the calculation of the CPU overhead will be greatly increased; thus chunking algorithm will face enormous pressure on the performance.

3. The Bit String Content Aware Chunking Strategy

Most existing chunking strategies [15–17] had used Rabin algorithm in signature calculation in which the total amount of chunk and the total number of signature calculations were nearly 3 orders of magnitude. There still existed a lot of invalid signature calculations. And each signature calculation called for modulo operation which consumed a large amount of CPU resources.

Two key problems should be solved in order to reduce the signature calculation and save CPU resources. The first is to reduce the calculations of fingerprints as far as possible; the second is to minimize the number of signature computations.

According to binary string, BCCS borrows the incremental calculation method of fingerprint from the Rabin fingerprint algorithm to convert the problem of chunking file stably to the matching process of two binary strings, reducing the overhead of chunk fingerprints generation. BCCS also learns the leap-forward match from the BM [18, 19] to get the maximum length of jump through each match, reducing the middle overhead of calculation and comparison. All above bring down the cost of signature in the process of chunking to improve the efficiency of matching.

3.1. Fast Fingerprint Generation Based on Bitwise Operation. The Rabin algorithm is based on the principle that a pattern string whose length is m is mapped into a binary bit string (fingerprint) and each text string whose length is m is also mapped into binary bit string. Then the algorithm determines whether the two strings are equal by comparison.

The Rabin algorithm was more efficient, but it consumed a large amount of CPU resource for modulo operation. Therefore, the key to reduce CPU overhead is improving algorithm to minimize modulo operation.

The basic comparison unit is byte in the Rabin algorithm. The new algorithm can be considered to select on binary bit to represent the basic comparison unit to accelerate the algorithm speed. The process can be completed by bitwise operation instruction without modulo operation and greatly reduces the CPU overhead of generated fingerprints.

Therefore, the new algorithm selects the low order bit of each basic comparison unit to form its fingerprint. If the fingerprint $f(T_s)$ of string T_s ($T_s = [b_s, b_{s+1}, \dots, b_{s+m-1}]$) is known, the $f(T_{s+1})$ can be got by the following bitwise operations.

Let us assume that $BX = f(T_s)$ and $AL = T_{s+1}$ (the next new substring):

SHR AL, 1;

SHL BX, 1.

Then, the data stored in the BX is the fingerprint of T_{s+1} , namely, $f(T_{s+1})$.

3.2. Implementation of BCCS. Assume that the length of the original pattern string is m basic units; the m bits pattern string fingerprint P is consisting of the low order bit of each basic unit. The BCCS prefetches n low order bit of the text

FIGURE 1: Matching the substring of text from pattern.

FIGURE 2: The perfect matching of text bit string.

basic units to storage in the bit array BT and then computes the text string fingerprints $f(T_s)$ by the fast fingerprint generation algorithm.

It considers the following two ways for pattern matching in this paper.

(I) *BCCS Based on Pattern (BCCS-P)*. Similar to the BM algorithm, BCCS-P uses the scanning mode from left to right to pattern match. As shown in Figure 1, in order to reduce the matching computation, the pattern and text should be left-justified and then a substring at the tail of text in alignment with the pattern whose length is $\text{len}(x)$ should be selected; the BCCS-P will search the match substring from the right of pattern, but not matching the whole pattern at first.

As bit string comparison, BCCS-P is slightly different from BM, each time comparing a substring through a bitwise operation. Two cases are considered separately in this paper.

(i) *The Perfect Matching of Text Bit String*

Definition 1. As shown in Figure 2, the perfect matching of text bit string is that that the length of the bit substring of the text will increase by 1 if the match is successful. And then it matches the corresponding bit substring at the tail in the pattern until the length increases to m whose matching direction is from right to left.

The good suffix match of text bit string should be working if there is a failure of matching.

As shown in Figure 3, it suggests that the perfect matching of text bit string is successful if the string matches with substring of text in the pattern successfully $(m - x) + 1$ times. The further accurate comparison is done at this time that is the actual string comparison. If the string does not match, then BCCS-P continues to match the substring x ; otherwise the chunk boundary should be marked, and pattern skips

FIGURE 3: Success of perfect matching of text bit string.

FIGURE 4: The good suffix matching of text bit string.

2 KB; namely, $d = d + 2048$. Then BCCS-P goes to the next round of comparison (the minimum chunk length is 2 KB).

(ii) The Good Suffix Matching of Text Bit String

Definition 2. As shown in Figure 4, the good suffix matching of text bit string happens when the bit substring of text whose length is $\text{len}(x)$ matches with a corresponding bit substring of pattern. If the match is not successful, the pattern slides to the left one bit and reselects another bit substring of pattern to match until it finds the matching substring in the pattern in the match process.

The jumping distance to right for the pattern is d when the good suffix matching of text bit string is successful. It should obtain maximum jumping distance in order to reduce the amount of matching computation in the matching process. As shown in Figure 5, the biggest jumping distance is the length of pattern that the pattern jumps to right for m bits if no substring matched in pattern.

(II) BCCS Based on Text, BBCS-T. As shown in Figure 6, similar to (I), BBCS-T uses the scanning mode from right to left to pattern match and pattern and text should be left-justified at first in order to reduce the matching computation; different from (I), BBCS-T selects the bit substring whose length is $\text{len}(x)$ at the right-most position in pattern and searches the matching bit substring in text from $m - \text{len}(x)$ position.

The following two cases are attached importance in this paper.

(i) The Perfect Matching of Pattern Bit String

Definition 3. As shown in Figure 7, the perfect matching of pattern bit string is that the length of the bit substring of the pattern increases by 1 if the match is successful. And then it

FIGURE 5: d -jumping distance from right ($d \leq m$).

FIGURE 6: Matching the substring of pattern from text.

matches the corresponding bit substring at the tail in the text until the length increases to m .

The good suffix match of pattern bit string is working once the matching fails.

It suggests that the perfect matching of pattern bit string is successful if the string matches with substring of pattern in the text successfully $(m - x) + 1$ times.

Similar to Figure 3, the further accurate comparison is done at this time that is the actual string comparison. If the string does not match, then BCCS-T continues to match the substring x ; otherwise the chunk boundary is marked, and pattern skips 2 KB; namely, $d = d + 2048$. Thus BCCS-T goes to the next round of comparison (the minimum chunk length is 2 KB).

(ii) The Good Suffix Matching of Pattern Bit String

Definition 4. As shown in Figure 8, the good suffix matching of pattern bit string when the length of bit substring is $\text{len}(x)$ matches with a corresponding bit substring of text; the text slides to right one bit and reselects another bit substring of text to match until finding the matching substring in the text in the matching process.

The jumping distance for pattern to right is still d when the good suffix matching is successful. But as shown in Figure 9, the jumping distance d can be far greater than the length of pattern and the maximum reaches $n - \text{len}(x)$ due to the scanning from left to right.

3.3. Performance Analysis. Through the above analysis we know that, in order to reduce the comparison cost, BCCS obtains the maximum jumping distance d and accelerates the speed of matching. In actual operation, being aimed at

FIGURE 7: The perfect matching of pattern bit string.

FIGURE 8: The good suffix matching of pattern bit string.

the bit string comparison, BCCS obtains the matching results quickly through the bitwise operation XOR which greatly reduces the CPU overhead compared to modulo operation in the Rabin algorithm.

The following items are on the range of d in detail.

3.3.1. BCCS-P. BCCS-P selects a bit substring whose length is $\text{len}(x)$ in the text and searches the matching bit substring in the pattern from right to left. In this case, the biggest jumping distance is m by the good suffix matching of text bit string; that is, $1 \leq d \leq m$, for the length of pattern is m .

3.3.2. BCCS-T. Compared with BCCS-P, BCCS-T selects the bit substring whose length is $\text{len}(x)$ in the pattern and searches the matching substring in text by changing the scanning mode as from left to right, which changes the main matching body in this way. The biggest jumping distance d of BCCS-T is far greater than m ; the length of pattern, for the biggest length of text, is n ($n \gg m$). It saves the intermediate data comparisons overhead to the most degree.

The detailed discussion of the values range of the jumping distance d in BCCS-T is as follows.

Assuming that $\bar{d}$ is the average jumping distance for matching,

$$\bar{d} = \sum_{i=0}^{n-\text{len}(x)} i \times \text{prob}. \quad (1)$$

The $\text{len}(x)$ is the length of the bit substring pattern containing x binary bit; prob is the matching probability of failure for pattern whose length is $\text{len}(x)$ and text matching i times, so the prob is $\text{prob} = q^i * r$; r is the matching probability of success: $r = (1/2)^{\text{len}(x)}$. q is the matching probability of failure: $q = 1 - r = 1 - (1/2)^{\text{len}(x)}$.

FIGURE 9: d -jumping distance from right ($1 < d < (n - x)$).

From [20],

$$\sum_{i=0}^n i \times q^i = q \times (1 - q)^{-2} - (n + 1) \times q^{n+1} \times (1 - q)^{-1} - q^{n+2} \times (1 - q)^{-2}. \quad (2)$$

Here, $(1/2) \leq q < 1$, n is a great number, and (2) can be simplified to

$$\sum_{i=1}^n i \times q^i \approx q (1 - q)^{-2}. \quad (3)$$

Equation (1) can be

$$\bar{d} \approx \frac{q}{r} = 2^{\text{len}(x)} - 1. \quad (4)$$

Compared with the traditional pattern matching algorithm, the speedup of f is written as

$$f = \frac{\bar{d}}{m} = \frac{2^{\text{len}(x)} - 1}{m}. \quad (5)$$

Through the above analysis, the bigger $\bar{d}$, the lower hit rate for the x in pattern matching in text, while the setting of the length of bit substring x , that is, $\text{len}(x)$, is the key problem. When $\text{len}(x)$ is smaller, the hit probability is higher and $\bar{d}$ is relatively smaller, which does not jump enough distance and leads to smaller speedup. On the contrary, if the $\text{len}(x)$ is bigger, and $\bar{d}$ and f are bigger, it also affects the efficiency of the matching because of the increasing text units.

4. Performance Evaluation

This paper establishes a prototype system for big data storage based on deduplication technology. It does the performance evaluation of the Bit String Content Aware Chunking Strategy, and the effects of the different length of the target bit string on chunking speed, chunk size, and chunk compression ratio are analyzed.

4.1. The Experimental Environment Settings. The prototype system consists of two server nodes. The CPU of each server node is 2-way quad-core Xeon E5420 with 8 GB DDR RAM, whose frequency is 2 GHz. And the cache of each core is 6114 KB. The chunks are stored in RAID 0 with two disks

TABLE 1: Chunking test data set.

Data set	File type	Set size (MB)	File number	Average length of files (B)	Minimum length of files (B)	Maximum length of files (B)
Modified	Linux source codes 2.6.32, 2.6.34, 2.5.36	999	76432	13,083	6	145,793
Unmodified	rmvb	883	2	464,507,195	439,263,182	489,751,207

(Seagate Barracuda 7200 RPM, 2 TB capacity of each one). While each node is equipped with an Intel 80003ES2LAN Gigabit card connecting to the Gigabit Ethernet, one of the nodes is the main server and another is the mirror server.

4.2. Performance Test. Four different chunking algorithms, Rabin algorithm, FSP algorithm, BCCS-P, and BCCS-T, are compared in the experiment.

Two different test data sets are set up to test the influence of chunking speed of different file types by different chunking algorithms. As shown in Table 1, the first test data set contains 3 different versions of the Linux kernel source code files, with a total of 76432 documents, 999M bytes. This test data set is modified data set for the files in the set are modified frequently. The second test data set is unmodified data set. This one contains two RMVB files. These two files are different versions of the same movie, shorter RMVB files from the RMVB file by RealMedia long and Editor Clip. The overall capacity of the second data set is 883 MB. The files in the second data set are modified rarely.

An adaptive control scheme on chunking for the two different test data sets is proposed. The prototype system automatically chooses the optimal algorithm according to different test set types.

The prototype system determines the type of the file by the suffix of file name. The multimedia files and compressed files are classified as unmodified file type, and system establishes a static text file, containing suffix name of unmodified file. Other types of documents are classified as modified file. When the files belong to the unmodified the system analyses the suffix of a file name at first and then chooses FSP algorithm or BCCS algorithm to chunk according to the different conditions whether the given parameters need faster duplicate speed or higher duplicate rate. Otherwise the system uses BCCS algorithm directly when the files belong to the modified data set.

The chunking is divided into two steps by BCCS: the first is preprocessing and the second step is a binary string matching, namely, chunking.

The preprocessing is that the system processes the input text data, extracting one bit of each text byte to generate fingerprints, providing support for the following chunking before calling the chunking module.

The experiment first compares the time of the preprocessing time and chunking time by BCCS-T and BCCS-P for the modified data set. The time overhead ratio is as shown in Figures 10 and 11.

X-axis and Y-axis in the figures indicate the length of the target pattern and the percentage of the processing time

FIGURE 10: Ratio of chunking time and preprocessing time for modified data set of BCCS-T.

FIGURE 11: Ratio of chunking time and preprocessing time for modified data set of BCCS-P.

accounting for the total processing time, respectively. In fact, for the same test data set, preprocessing time of BCCS-T and BCCS-P is the same, but the chunking times are different. It is shown as in Figures 10 and 11 that the preprocessing time of BCCS-T accounts for the total processing time between 45% and 60% and that of BCCS-P accounts for about 42% and 45%, according to the different lengths of the pattern. It is a great time overhead. Therefore, the preprocessing operations are processed in a processing node and chunking operations are processed in the front-end server node to achieve similarly the pipeline operation mode through the parallel operation by BCCS. It just needs a startup time of preprocessing for the big data and the actual data processing time is the time of chunking when the pipeline works normally, thus greatly reducing the actual time overhead.

The minimum chunk size is proposed as 2 KB by BCCS to avoid too many short chunks, which reduces the overhead of metadata and decreases the unnecessary computation.

TABLE 2: Comparison of BCCS-T and FSP algorithms for unmodified data set.

Chunking algorithm	Total time cost (ms)	Chunking time cost (ms)	Saving time (ms)	Deduplication rate
BCCS-T-11	16453	5218	10642	1.2061
BCCS-T-12	18303	5503	11588	1.2050
BCCS-T-13	19336	6485	11851	1.0952
BCCS-T-14	19627	5743	13942	1.0920
BCCS-T-15	20045	7956	13920	1.0920
FSP	16167	686	15456	0.9771

FIGURE 12: The comparison of chunking throughput of modified data set.

As shown in Figure 12, it is the chunking throughput for BCCS-T, BCCS-P, and Rabin to modified test set when the different lengths of pattern are set. The X-axis represents the length of pattern; the Y-axis represents the chunking throughput. Rabin algorithm reaches its maximum throughput rate of about 98 MB/s when its length of pattern is 11; BCCS-T and BCCS-P reach maximum chunking throughput, about 198 MB/s and 123 MB/s, respectively, when the length of pattern is 12; the three algorithms reach the minimum chunking throughput, about 122 MB/s, 105 MB/s, and 52.5 MB/s, respectively, when the length of pattern is 15. Different matching modes by BCCS-T, BCCS-P, and BCCS-T reduce more intermediate matching overhead through the good suffix matching of pattern bit string rule. Therefore, the chunking throughput of BCCS-T is almost 1.63 times as that of BCCS-P when the length of pattern is the same. For modified data set, with 12-bit target pattern, the BCCS-T chunking speed is raised to be close to 160% compared to that of the Rabin algorithm or so; with 15-bit target pattern, it is 80%. The length of pattern greatly affects the chunking throughput rate.

The unmodified test data sets are chunked by FSP and BCCS-T algorithms, respectively. It is shown in Table 2 that it only takes 686 ms to chunk the RMVB files with 883 MB by FSP algorithm, and the deduplication rate is only 0.9771 because it does not find any similar chunks and spends extra metadata cost. Furthermore, the storing time of chunks by FSP algorithm is far longer than that by BCCS-T algorithm. But by BCCS-T algorithm, with 11-bit pattern, the chunking time is 5218 ms, and the total processing time is 16453, just 1.8% over FSP with the deduplication rate 1.2061. Therefore, for the unmodified files, the prototype system provides

a flexible method to select the chunking algorithm in the light of actual conditions.

5. Conclusion

It is an essential prerequisite for finding redundant data and improving deduplication rate to rapidly and stably divide the data objects into suitable size chunks. Most existing chunking algorithms obtain the fingerprints in the sliding window by Rabin algorithm to determine the chunk boundary. It consumes a large amount of CPU computing resources on fingerprint computing. The CPU cost of fingerprints computing is greatly increased when the processed data becomes huge, and the chunking algorithm faces plenty of pressure on the performance.

This paper proposes a novel chunking strategy, namely, the Bit String Content Aware Chunking Strategy (BCCS), through researching and analyzing the advantages of Rabin algorithm and BM algorithm.

The primary contributions of our work are as follows: first, according to characteristics of the binary bit string, it simplifies the fingerprint calculation requiring a lot of CPU computing resources into the simple shift operation with reference to calculation methods of Rabin incremental fingerprint algorithm; secondly, it converts the problem of data objects chunking stability into the process of two binary bits' string matching.

According to the different matching subjects, this paper proposes two kinds of chunking strategies based on Bit String Content Aware Chunking Strategy, namely, BCCS-P and BCCS-T, with the bad character matching rule and the good suffix matching rule in BM. BCCS-P is pattern matching centered when the jumping distance for unmatching is limited every time. The maximum jumping distance does not exceed the length of pattern; thus the benefits are limited. BCCS-T is text-matching centered that rules out the unmatching positions and gets the maximum jumping distance to reduce intermediate calculation and comparison costs by making best use of the bit feature information brought by failure matching every time. This measure reduces the cost of signature computation in chunking process to improve the system performance.

To a certain extent in the large data center using the deduplication technology to store the data, it reduces the storage system reliability and also has some limitations in storage overhead and system performance with multiple files sharing data objects or chunks. It is also one of the problems that needs to be further studied in the future regarding how

to make up for this defect, ensure the safety and reliability of big data storage, and provide the QoS.

Conflict of Interests

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

Acknowledgment

This research was supported by Natural Science Foundation of Hubei Province (no. 2013CFB447).

References

- [1] J. Gantz and D. Reinsel, "Extracting value from chaos," IDC IVIEW, 2011, <http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf>.
- [2] Gartner, *Meeting the DC Power and Cooling Challenges*, 2008.
- [3] NRDC, "Report to congress on server and data center energy efficiency," 2013, <http://www.nrdc.org/>.
- [4] J. Sweeney, "Reducing data centers power and energy consumption: saving money and go green," 2008.
- [5] W. Tschudi, T. Xu, D. Sartor et al., *Energy Efficient Data Center*, California Energy Commission, Berkeley, Calif, USA, 2004.
- [6] J. Hamilton, "Cooperative expendable micro-slice servers (CEMS): low cost, low power servers for internet-scale services," in *Proceedings of the 4th Biennial Conference on Innovative Data Systems Research (CIDR '09)*, pp. 1–8, Asilomar, Calif, USA, 2009.
- [7] S. Y. Jin, *Research on some key issues for resource management in green virtualized data center [Ph.D. thesis]*, University of Electronic Science and Technology of China, Chengdu, China, 2012.
- [8] G. Lu, Y. Jin, and D. H. C. Du, "Frequency based chunking for data de-duplication," in *Proceedings of the 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '10)*, pp. 287–296, IEEE, August 2010.
- [9] H. M. Jung, S. Y. Park, J. G. Lee, and Y. W. Ko, "Efficient data deduplication system considering file modification pattern," *International Journal of Security and Its Applications*, vol. 6, no. 2, pp. 421–426, 2012.
- [10] Y. Wang, C. C. Tan, and N. Mi, "Using elasticity to improve inline data deduplication storage systems," in *Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD '14)*, pp. 785–792, IEEE, Anchorage, Alaska, USA, June–July 2014.
- [11] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network files system," in *Proceedings of the Symposium on Operating Systems Principles (SOSP '01)*, pp. 174–187, ACM Press, 2001.
- [12] M. O. Rabin, *Fingerprinting by Random Polynomials*, Center for Research in Computing Technology, Aiken Computation Laboratory, Harvard University, Cambridge, Mass, USA, 1981.
- [13] K. Eshghi and H. K. Tang, "A framework for analyzing and improving content based chunking algorithms," Tech. Rep. HPL-2005-30R1, Hewlett-Packard Labs, 2005.
- [14] D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki, "Improving duplicate elimination in storage systems," *ACM Transactions on Storage*, vol. 2, no. 4, pp. 424–448, 2006.
- [15] Y. Liu, X. Ge, D. H. C. Du, and X. X. Huang, "Par-BF: a parallel partitioned Bloom filter for dynamic data sets," in *Proceedings of the International Workshop on Data Intensive Scalable Computing Systems (DISCS '14)*, pp. 1–8, IEEE Press, 2014.
- [16] N. Jain, M. Dahlin, and R. Tewari, "Taper: tiered approach for eliminating redundancy in replica synchronization," in *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST '05)*, p. 21, USENIX Association Press, Berkeley, Calif, USA, 2005.
- [17] J. B. Wang, Z. G. Zhao, Z. G. Xu, H. Zhang, L. Li, and Y. Guo, "I-sieve: an inline high performance deduplication system used in cloud storage," *Tsinghua Science and Technology*, vol. 20, no. 1, Article ID 7040510, pp. 17–27, 2015.
- [18] S. Robert, J. Boyer, and S. Moore, "A fast string searching algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762–772, 1977.
- [19] Z. Galil and T. Aviv, "On improving the worst case running time of the Boyer-Moore string matching algorithm," in *Automata, Languages and Programming: Fifth Colloquium, Udine, Italy, July 17–21, 1978*, vol. 62 of *Lecture Notes in Computer Science*, pp. 241–250, Springer, Berlin, Germany, 1978.
- [20] A. N. M. E. Rafiq, M. W. El-Kharashi, and F. Gebali, "A fast string search algorithm for deep packet classification," *Computer Communications*, vol. 27, no. 15, pp. 1524–1538, 2004.