An Efficient Data Fingerprint Query Algorithm Based on Two-Leveled Bloom Filter

Bin Zhou[1, 2], Rongbo Zhu [1,*], Ying Zhang [3], Linhui Cheng [1]
1. South-Central University for Nationalities/School of Computer Science and Technology, Wuhan, China
2. Huazhong University of Science and Technology/School of Computer Science and Technology, Wuhan, China
3. Huazhong University of Science and Technology/ School of Foreign Languages, Wuhan, China
* Corresponding author, Email: rongbozhu@gmail.com

Abstract—**The function of the comparing fingerprints algorithm was to judge whether a new partitioned data chunk was in a storage system a decade ago. At present, in the most de-duplication backup system the fingerprints of the big data chunks are huge and cannot be stored in the memory completely. The performance of the system is unavoidably retarded by data chunks accessing the storage system at the querying stage. Accordingly, a new query mechanism namely Two-stage Bloom Filter (TBF) mechanism is proposed. Firstly, as a representation of the entirety for the first grade bloom filter, each bit of the second grade bloom filter in the TBF represents the chunks having the identical fingerprints reducing the rate of false positives. Secondly, a two-dimensional list is built corresponding to the two grade bloom filter for the absolute addresses of the data chunks with the identical fingerprints. Finally, a new hash function class with the strong global random characteristic is set up according to the data fingerprints' random characteristics. To reduce the comparing data greatly, TBF decreases the number of accessing disks, improves the speed of detecting the redundant data chunks, and reduces the rate of false positives which helps the improvement of the overall performance of system.**

Index Terms—**data fingerprint, bloom filter, two-level bloom filter, de-duplication, hash**

I. Introduction and Motivations

By living in the information era, we encounter different information from different channels every day: call, television, message, and post, etc. At the same time, a great deal of information and data will generate every day, such as Micro-blog, Blog, and FRID, etc. Different kinds of information are everywhere. According to the *International Data Corporation* (IDC) statistics, the amount of data of the whole world was just 180EB in 2006, and this figure increased to 1800EB in 2011. It has increased by almost one order of magnitude in the past course of 5 years. All the data generated every year should be stored for the future reference. This number will continue to grow. Recent IDC reports predict that this figure will arrive to 8000EB (almost 8ZB) in 2015. That sounds good to store all the data. The fact is that IT budget becomes seriously critical. The growing rate of the annual input is just 3 percent [1]. A wide gap comes into being between the demand of the information storage and the affordability.

Research and experimental results also show that a large amount of data is duplicated in the growing data. For many users backup their important data periodically to prevent unexpected incidents. In fact, the 70% data in the storage system is duplicated, which has never been used in 90 days [1]. For the duplicate data, there is no need to store.

It is a hot research topic nowadays that how to use the existing storage capacity to store as many data as possible. Recently, data de-duplication, the hot emerging technology, has received a broad attention from both academia and industry, which called Intelligent Compression or Single-Instance Storage. Its basic thought is to divide the file data into different chunks and the same chunks in the saving system only save one copy, while the others, referenced to by a pointer are pointing to the location of the chunks. On the one hand, it eliminates the same chunks, which distribute in the saving system to optimize utilization of saving space and get the higher saving efficiency. On the other hand, it reduces the amount of data that delivers in the network and then lower energy depletion and network cost, and is data replication and instauration to save a great deal of network bandwidth, which is also an important aspect of green compute.

To the storage system for big data, the number of the chunks is very huge especially in the case of fine-grain. It is a key to improve the performance of the system that how to judge a new data chunk whether has already storied in the storage system as quickly as possible.

The information retrieval comes in many forms, such as dynamic array, database, RB/B/B+/B* tree and hash table. The hash retrieval is famous for its o (1) performance. It seems a good solution to adopt hash table to save the meta-data index information [2] [3] [4], because the overhead for querying the hash table is a

*Corresponding author, Rongbo Zhu, E-mail: rongbozhu@gmail.com
This research was supported by "the Fundamental Research Funds for the Central Universities", South-Central University for Nationalities (Grant Number: CZY12009). And this work also was supported by the National Natural Science Foundation of China (No. 60902053, 61272497).

constant in theoretically. But in fact, when the data scale becomes big, the index information will lead greatly ineluctably to not be complete saved in the memory and need to carry on an IO operation to access the disk in the query, which is a process that pretty much consuming. The key place how can avoid the IO operation to accelerate the query speed of data fingerprint to raises the system performance.

In this paper, we studied the questions above-mentioned and performed various experiments. The primary contribution of our work is as follows: First, we introduce the bloom filter that is a kind of high efficient memory data structure into *De-duplication Storage System* (DSS). According to the shortcoming of the false positive rate in the original bloom filter, we present a new mechanism named *Two-stage Bloom Filter* (TBF). In the new mechanism, each bit of the second grade bloom filter represents the entirety of the values of the hash functions for the identical chunk in the first bloom filter, that is to say that each bit of the second grade bloom filter represents the chunks who have the same fingerprints. Second, a two-dimensional list is created corresponding to the two grade bloom filter. The absolute addresses of the data chunks with the same fingerprints are together in the identical list that avoiding the saving of the fingerprints. This measure reduces the storage space greatly and improves the query mechanism of the fingerprints in the data de-duplication system. At last, since the fingerprints made by the MD5 algorithm have the characteristics of random, a new hash function class with the strong global random character comes out. Comparing with the existing algorithms of detecting the redundant data chunks, on the base of reducing the comparing data greatly, TBF decreases the number of accessing disks, improves the speed of detecting the redundant data chunks, and reduces the rate of false positives. All these measures improve the overall performance of system.

The following paper reads as follows. Section 2 introduces related works. Section 3 describes the organization of De-duplication Storage System and section 4 studies two-stage bloom filter mechanism in details. In section 5 various experiments are performed to measure the improvement of the rate of false positives based on the TBF. At last, section 6 draws the conclusion.

II. Related works

It is a serious concern that how to find out more redundancy data in the data de-duplication system, and how to find out the redundancy data more quickly is another research hot spot,

For the data quantitative is huge, we cannot compare the chunks directly, but make use of hash algorithm (MD5 or SHA-1) to compute a hash value (data fingerprint) of each chunk and deposit this fingerprint to a data structure. We will compute its data fingerprint firstly when the new chunk coming, then compare it with the already exist set of the data fingerprint. If this fingerprint has been in the set, we just store an index; otherwise, the new chunk and fingerprint all should be store in the system.

It is enough to meet the system performance by using the hash table to save the fingerprints that produced by the MD5 algorithms when the data quantity is not big. However, it will generate many other factors to effect the performance of the system thus while the data quantity become huge.

It is the most important factor affecting the system performance that the memory is not enough. By the time that should compare all the saved fingerprints to make sure that a chunk whether has been already existed, an I/O operation to query the fingerprints is already inevitable when the contents of hash table exceed far beyond the memory. This operation extremely consumes time that will affect the overall performance of system greatly, or even it is sometimes incapability bears with.

To tackle these problems, M. Lillibridge et al. [5] adopted the methods, such as sampling, sparse index and chunk locality, etc. Thwel et al. [6] introduced the B+ tree into the data de-duplication system, which stored the data fingerprints into the leaves of the B+ tree. Because of the characters of the B+ tree, when we need to know whether a certain chunk has been already existed, we just compared one part of nodes of the B+ tree that made the time complexities of the search descend from O (0) to O (logn). Still, this method involved the IO operations. Deepavali Bhagwa et al. [7] then proposed to divide the chunk indices as two levels. The upper level was termed primary index and kept in RAM, which use to identify the file; and the second level is termed bin and kept in the disk, which included the whole chunks of the data. In this way, we needed access the disk once to search a file, but it probably existed a great deal of redundancy data.

In some intuitive sense, the shorter the chunk, the higher the opportunity of discovering the redundancy data, as while as the whole metadata was also bigger. Bobbarjung et al. [4] proposed a new thinking of the stratify storage data fingerprint to find out more redundancy data and as far s possible to put down the overhead of the metadata processing. It tried to minimize the length of the chunk as 1 KB or so, as while as make a set of chunks as a unit to deal with. In this way, it had merits of both sides, raising the rate of discovering the redundancy and reducing the whole amount of the metadata. Nevertheless, in practical application the result was unsatisfactory when it operated the inserting and deleting in the files.

Kruus et al. [8] proposed a kind of two-stage chunking algorithm that re-chunks transitional and non-duplicated big CDC chunks into small CDC chunks. It could discover more redundancy chunks comparing with the basic CDC algorithm and obviously enlarged the amount of metadata. But [9] then proposed a new kind of algorithm that is according to the chunks appearing frequency to decide whether further divide the CDC chunks , it could declined the amount of metadata to a certain degree and improved the performance of de-duplication.

It is the focus of this paper that how to identify the redundancy data as well as reduce the metadata. Aiming at web service of wide area network, Li Fan et al. [10] presented a kind of improved flexible sharing cache protocol, namely summary cache, which made use of the bloom filter. And B. Zhu et al. [3] introduced the bloom filter into arrive document backup system, that also made use of the bloom filter to create a Summary Vector in the memory while backing up the data that could query the memory vector directly and knew whether the data chunks already had been in, while searching the data fingerprints. By studying and analyzing bloom filter [11] which is a kind of high efficiency memory data structure, we find that it can solve the problem that the memory can not store the excessive metadata to lead to the frequently IO operations during the retrieval process. In addition, the performance of overall system is improved.

However, some literature[12][13][14] also introduced the bloom filter the efficient memory data structure into the redundancy data detection and reduced the number of the IO operations to judge whether the chunk had been in the storage system, but all of them had seldom deep studied on the problem of the false positive. In addition, some other literature [15] thought that it was not a very important problem to minimize the false positive rate of the bloom filter. Such as in the query of web page the false positive rate sacrificed to obtain the better spatial efficiency, and could improve the performance of query of the data fingerprint by compressing the bit vector of the bloom filter.

For keeping a low rate of false positive, the availability of RAM space on the machine decides the length of the bloom filter. For the big datasets, the RAM space is not sufficient. Biplob Debnath et al. [16] advocate the flash memory to serve as suitable medium for storing bloom filters and Michael A. Bender et al. [17] supposed the Cascade Filter and Deke Guo et al.[18] proposed dynamic Bloom filters to overcome this shortcoming, respectively.

In our system, we present the solution of *the two-stage bloom filter mechanism* that make use of the second grade bloom filter on the base of the algorithm of the standard bloom filter. It reduces the number of IO operation directly to improve the performance of discovering the repeated chunks by as far as possible to decrease the false positive rate.

III. De-duplication Storage System Architecture

To provide the context of finding the redundancy data by the TBF mechanism, this section describes the architecture of the *Duplication Storage System* (DSS).

The DSS designed for distributed environment is divided into three parts: the front-end data input server, the metadata query server and the back-end storage server. The front-end server connects with the metadata query server and the back-end storage server by the network. The framework of the system is in Figure 1.

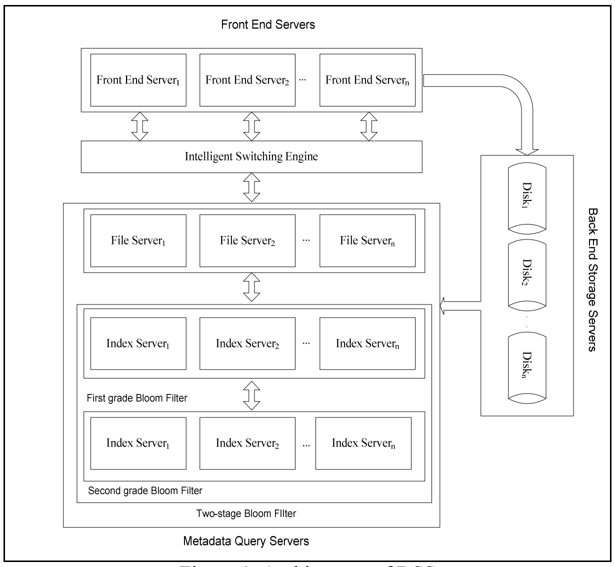

Figure 1. Architecture of DSS

The metadata query server divides into two components: the file server component and the index server component. The metadata of the file is stored in the file server and the TBF storing the chunk fingerprints is in the index server.

When a data file is inputted at the Front End Servers level, the fingerprint of the file will be built by DSS, and the metadata query server will search the relevant Meta information. It will notify Front End to delete the file if the file fingerprint has been in the file server; otherwise, the DSS will record the related information of this file, namely the table of logic file. Each record of the logical file includes the file name, file length, the modify time of the file, sum of chunk, size of chunk ID, and a unique set of data block number. The file can be restored by this logic file to get the entity data content.

The DSS will partitions the file into variable length chunks in a content dependent manner [19] and compute a fingerprint for each chunk when the file-level redundancy does not exist. The metadata query server will call the index server component to search whether the chunks have been in the storage system after receiving the query request of the array of chunk fingerprints, and return the chunk IDs.

The ID field value is 0 or 1. The value 0 means the corresponding chunk does not exist which needs to be transferred from the client to the back-end storage system, the value 1 means that the chunk already exists which does not need to be transferred.

TABLE I. LOGIC FILE

File name	File length	Modify time	Sum of chunks	size of chunk ID	ID_1	ID_2	…	ID_n

TABLE II. RETURN BY CHUNK FINGERPRINT QUERY

File name	File length	Modify time	Sum of chunks	ID1	ID2	…	IDn

In order to accelerate the query speed, decrease the memory occupied by the index, and reduce the false positive rate, the bloom filer goes to store the index in DSS. Moreover, the original bloom filter improves by a series second bloom filter, namely TBF to cut down the false positive rate, and reduce the frequent IO operations to bring up the system performance.

At last, the storage system receives the chunks from the front-ended node and stores them, and finally updates the logic files table in the file server.

VI. Chunk Fingerprint Query Based on the TBF

In the DSS, the large-scale data is the object. Under the assumption that we need to store the data about 2^n TB, each chunk is about 2^3 KB and it will have about $2^{n-3}*10^9$ unique chunks. If the index of each chunk is 16 Bytes, it needs about 2^{n+1} GB space to store them. The storage space increases exponentially when n increases. Not all these indices can be stored in the memory and the frequent disk IO operations are unavoidable. Supposed that the average time of one IO accessing is about 4 ms, it can retrieve 250 chunks per second, which means that the throughput of the system is about 2 MB/s. In addition, this is not acceptable.

Then it is the necessary conditions for avoiding to access the disks and keep the query time within the limits that how to simplify the index data to keep them in the memory fully.

A. Standard Bloom Filter (SBF) Algorithm

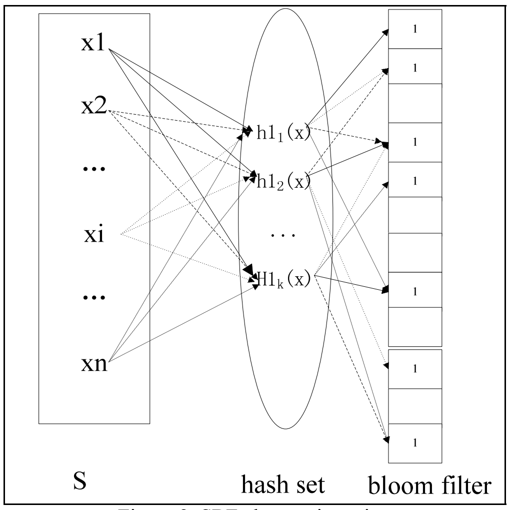

Figure 2. SBF element inserting

The original bloom filter composed of a large bit vector BV and k hash functions is a kind of efficient and simplified memory data structure suggested by Burton H. Bloom[11] in the 70's last century, which could realize the efficient query whether an element was in the set. Compared with the traditional tree query algorithm and hash query algorithm, the space that the bloom filter needs shows no correlation with the size of the element need by query, but it only shows a correlation with the numbers of the functions mapping the element to the bit vector, which will economize storage space greatly.

In the initial state, the bloom filter is a vector whose length is m, with the initial value of 0. The set S includes n elements(data fingerprint)$\{x_1,x_2,\ldots x_n\}$,there are k independent hash functions $h_i(x)$ in the Bloom Filter, that map the each fingerprint to $(0,\ldots,m-1)$,respectively. We will compute the value of $h_i(x_j)$ $(1\leq i\leq k)$ when we insert a data fingerprint x_j into the set S. If the hash value is s, the counterpoint of the bit vector BV should be set 1.

By applying k hash functions on the y data fingerprint respectively, it can be concluded that the fingerprint is not in the set if there is one or more 0s in the corresponding locations in the vector BV. However, it cannot be concluded that the data fingerprint belongs to the set if the values of the k locations are all 1, for one scenario of miscarriage of justice named false positive may appear at this time [3], which can be stated as the formula:

$$FP = \left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^k \approx (1-e^{-kn/m})^k \qquad (1)$$

In the formula, m is the length of the vector BV, namely the length of bloom filter; while n is the number of the set elements, that is the number of chunks and k is the number of hash functions.

Then it is known from the above analysis that for the given n and m, they need more hash functions, that is to say, increase the value of k, to assure enough low probability of the false positive, which will directly lead to low performance of adding and querying operations for the data fingerprints.

B.The Principle of the TBF Algorithm

It cannot decrease the rate of false positive by increase the value of k continuously, for k, the FP has a minimum value as FP_{min}=$0.6185^{m/n}$ [20]. In order to minis the value of FP. We enlarge the value of m/n at this time. It sets up larger bloom filter for the given element amount n, at the price of enlarging the memory occupation.

Inspired by the multidimensional bloom filter suggested by Guo et al. [21], we consider adopting TBF mechanism by two bloom filters in series to reduce the rate of false positive.

Because each hash function of the SBF is independent and has no contact with each other the values of the hash function for the different elements produce collision easily. While SBF takes no measure to deal with the above-mentioned situation. For example, when a same value of the different hash function for the other element generated after the location of the bit vector had been set, it just did nothing but simply queried.

The principle of the TBF mechanism is that all values of the hash functions for per element incorporates to one value, namely the whole characteristic of the element. It represents by another bloom filter. Therefore, the integrated information leads to the falling rate of the collision produced by the single hash function value that reducing the rate of the false positive for the overall system.

Given the set of the data fingerprints is S={$x_1, x_2, \ldots, x_n$},the TBF is composed of two bloom filters in series. The length of the first filter is *m*, which represents the values of the fingerprints generated by the set of the *k* hash functions:

$\text{element}_i\text{\_addr}_j = h1_j(x_i) \in \{0,1,\ldots,m-1\} (1 \le i \le n, 1 \le j \le k)$

$\text{element}_i\text{\_addr}_j$ represents the jth hash value for the ith element.

The length of the second filter is *n'*, which represents the integrated information of *k* hash functions for each element; the hash function shows as below:

$$h2_i(x) = \text{element}_i\text{\_addr}_1 \oplus \text{element}_i\text{\_addr}_2 \oplus \text{element}_i\text{\_addr}_3 \oplus \ldots \oplus \text{element}_i\text{\_addr}_k; (1 \le i \le n')$$

The second bloom filter address comprises of the value of XOR operation, which represents the integrity of the hash values of each element of the first bloom filter.

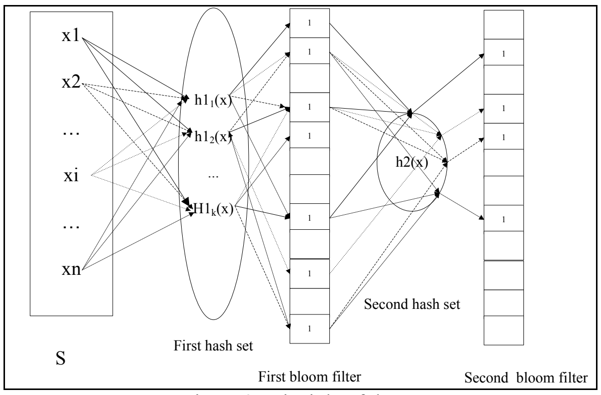

Figure 3. principle of the TBF

C. The implementing of the TBF Algorithm

It needs to build a two-dimension link list to save absolute address of the data chunks that shows as the fig.4. The length of the list is as same as that of the second bloom filter, namely *m'* and the initial values are *null*. Each location of the second grade bloom filter corresponds to a series of absolute addresses. The system will record the absolute address of a new data chunk when the data chunk is inserted as well as the value of the corresponding location of the second grade bloom filter is set to 1.

It is described at the below about the query algorithm of the TBF and the insertion algorithm of the TBF, respectively.

1. The query algorithm of the TBF

It needs to query the two bit-vectors respectively while needing to judge whether a data chunk has been in the system. As shown in algorithm *1*, first step, it applies *k* hash functions on the fingerprint of the data chunk, and query the first grade SBF according to the results of the calculation. It indicates that the data chunk has not been stored if one or more locations in the first SBF have been set to 0, and return the result. Otherwise, if all the *k* locations are set to 1, it goes into the next step to apply the XOR operation on the *k* hash values gaining by the first step, and then query the second grade bloom filter by the result. It indicates that the data fingerprint has not been stored if the location has not been set and otherwise it cannot judge whether the data fingerprint has been stored. That is to say the false positive occurs. The absolute addresses in the Absolute address_index list corresponding to the location of the second grade bloom filter traverses, and the data in the absolute address compare with the new chunk byte by byte. It indicates that the chunk has been stored in the system while it discovers that some chunk in the storage system has the same content as that of the new chunk's, otherwise, it indicates that is a new data chunk.

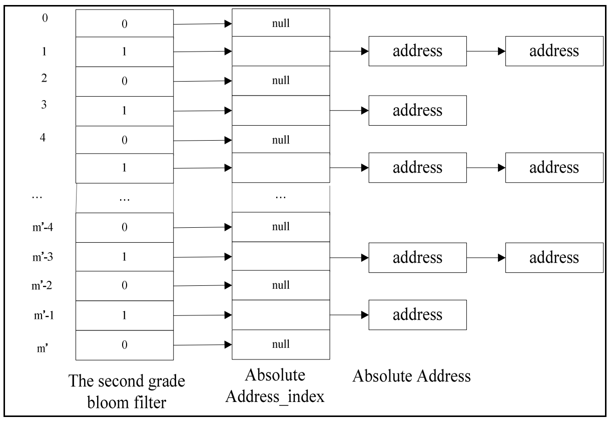

Figure 4. Absolute Addresses List for Initial Data Chunks

Algorithm 1: the query algorithm of the TBF
Input:a chunk data fingerprint
Output:bool //return true, it indicates that the chunk has been in; return false, it indicates that the chunk has been in

```
bool QueryFingerprint(Object fingprint)
{ int Faddr[k];// the k hash values of the first grade
 int sSddr;  // the hash value of the second grade
 Vector BV1, BV2;//the two bloom filters
 For(i=0;i<k;i++){
  Fadd[i]=hashi(fingerprint);
// to get the i hash values of the first grade
   If(getBV1(Fadd[i])==0)
     return true;

//to judge whether the locations of the first gradebloom
 filter have been set
   }
 sAddr=Faddr[0] ⊕ Faddr[1]… ⊕ Faddr[k];
// XOR operation, to get the hash value of the second
 grade
  if(getBV2(saddr)==0)
// to judge whether the location of the second grade
 bloom filter has been set
```

```
        return false;
      else
        return             taaverse_absolute_address(sAddr);
  //traverse the absolute address list, return true if find
  the same chunk, otherwise return false
    }
```

2. The insertion algorithm of the TBF

As shown in algorithm *2* is the insert algorithm of the element of the data fingerprint. At first, whenever a new data fingerprint inserted into the set, it will call the Query Fingerprint function to judge whether the chunk has been in the system. It will save the pointer that save the absolute address of the data chunk in the storage system, and delete the new data chunk when it finds the new data chunk has been in the storage system. Otherwise, it will save the new data chunk, create a new absolute address block and link it the corresponding location of the address list.

```
Algorithm 2: the insertion algorithm of the TBF
input:a chunk data fingerprint
Output:void
void AddFingerprint(Object fingerprint)
{ int Faddr[k];// the k hash values of the first grade
  int  sAddr;   // the hash value of the second grade
  Vector BV1,BV2://the two bloom filters
  Long int  *Address_index[m'];
  If(!QueryFingerprint(Object fingprint))
   {
    for(i=0;i<k;i++){
      Fadd[i]=hashi(fingerprint);
   //to get the i hash values of the first grade
      setBV1(Fadd[i]);
  //to set the first grade bloom filter
      }
      sAddr=Faddr[0]  ⊕  Faddr[1]…  ⊕  Faddr[k];
  //XOR operation, to get the hash value of the second
  grade
      setBV2(sAddr);
  // set the second bloom filter
      insert_chunk();
  //insert the new data chunk
      setAddress_index(sAddr);
  //insert the absolute addresses list for new data chunks
    }
   else
      save_Absolute_Address_pointer(fingerprint);
  //save the pointer
   }
```

D. Performance analysis for TBF

Theorem The rate of false positive for TBF algorithm is less than that of the SBF algorithm.

In this paper, the author proves the following: Let the rate of false positive for standard bloom filter algorithm be FP_1, the length of bloom filter is m, the number of elements is n, and the number of hash functions is k. The rate of false positive for the second grade be FP_2, the length of the second bloom filter is m', the number of elements is n', the number of hash functions is k'. The rate of false positive for the TBF algorithm is FP.

So $$FP_1=\left(1-e^{-kn/m}\right)^k=\left(1-e^{-n/m}\right) \quad (2)$$

For just one hash function is set in the second grade bloom filter, namely $k'=1$, the rate of the false positive for the second grade is as follows:

$$FP_2=\left(1-e^{-k'n'/m'}\right)^{k'}=\left(1-e^{-n'/m'}\right) \quad (3)$$

The two bloom filters are in series, so the overall rate of false positive is:

$$FP=FP_1*FP_2=\left(1-e^{-kn/m}\right)^k*\left(1-e^{-n'/m'}\right) \quad (4)$$

And $n'<m'$, then $\left(1-e^{-n'/m'}\right)<1$, so $FP<FP_1$

The above-mentioned 2^n TB data, if each chunk is 2^3 KB and there are about $2^{n-3}*10^9$chunks. They are stored in the bit vector. While the value of m/n for the first grade bloom filter is set as $2k$ (k is the number of the hash functions), the size of the vector is $2^{n-2}*k$ Gb, that is to say $2^{n-5}*k$ GB. The value of m'/n' for the second grade bloom filter is set as $2k'$ ($k'=1$) and $m'=1/4m,$ then the length of the second bloom filter is set to $1/4*2^{n-5}k'$GB, namely $1/4*2^{n-5}$GB. The space occupied by the Absolute address index is ($m'+4n$) Bytes ($m'=nk/2$). The whole space occupied by the two bloom filters is $(k+1/4+2k+16)*2^{n-5}$GB. That is to say about $(3k+16.25)*2^{n-5}$GB.

When one fingerprint is 16 Bytes, $2^{n-3}*10^9$ chunks take the space of 2^{n+1}GB. Owing to the large number of the fingerprints, they cannot be stored in the memory and then frequent accessing of the back storage system is unavoidable in the query process. While the TBF saves nearly 44% of the space by comparison when the k is 6. By adopting the TBF mechanism, the back storage system is avoided being accessed and greatly reduced the operation expenses.

Furthermore, comparing with initial bloom filter, the TBF algorithm represents the multiple separate hash functions of the data fingerprints as an entirety to achieve smaller rate of false positive. And its mechanism has the following merits standing out obviously: FP is about 40% of FP_1, that's to say about 60% of disk IO accessing operations can be reduced when the value of m'/n' is 2.

The experiment proves that the additional spatial expenses do little effect on the performance of the whole system while it reduces the query time greatly.

V. Experimental Results

A prototype system is designed to test how the TBF mechanism introduced in our paper influences the overall system performance. A comprehensive analysis is performed on various aspects of de-duplication backup. We test the performance of incremental by adopting the TBF mechanism, which avoids the disk bottleneck, and analyze the impacts of different amount of data, different number of hash functions and different length of the bloom filters.

A. The Experiment Ssetup

Our experiment is performed on two nodes which are configured with 2-way quad-core Xeon E5405 2GHz CPUs and 8GB DDR RAM. Their capacity of cache for each core is 6144KB. The data files (chunks) are stored on RAID 0 with two disks (Seagate Barracuda 7200RPM, 1TB each). Each node has an Intel 80003ES2LAN gigabit network interface card (NIC) and is connected via switched gigabit Ethernet. One node is the server and the other is the mirror.

B. Hash Functions Class

The most crucial factor is to decrease the collisions of hash functions in the TBF, and then the hash functions should have the upstanding global distributing character.

In the hash functions of H_3 defined by Carter et al. [22], each H3 hash function is corresponding to a 0, 1 function matrix. It has the unusual global distribution, but consumes excessive CPU resource.

Considering the rate of collision for the data fingerprints is less by MD5 algorithm, new hash functions should be created on the base of fingerprints.

1. The first grade index hash

The length of fingerprint created by MD5 is 128 bits, and let length of the first grade bloom filter be 2^n bits, which is shown as the Fig.5. We get the first 7 bits of the fingerprint to gain *p1*, whose value range is from 0 to 127.Then *p1* point to the location of the fingerprint, and *n* bits should be got, this value of the *n* bits is the hash value, who will point to the absolute address of the first bloom filter.

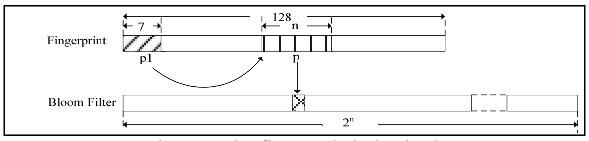

Figure 5. the first grade index hash

2. The second grade index hash

The second grade index hash is an improvement upon the first grade index hash. It gets the content of the first 7 bits *p1* as a pointer to point to the location of the fingerprint, and then gets the content of the next 7 bits *p2* as the second grade pointer to point to the next location, at last, the *n* bits should be got as the pointer *p*. In addition, the *p* is the value of the hash function and it will point to the location of the first grade bloom filter. It is shown as the Fig.6.

Furthermore, the first and the second grade index hash both have a shortcoming that when it gets the pointer *p1* or *p2*, namely the content of the 7 bits of the fingerprint, the state will follows: 128-*p1*<*n* or 128-*p2*<*n*. It should link the head and tail of the fingerprint to form into a loop.

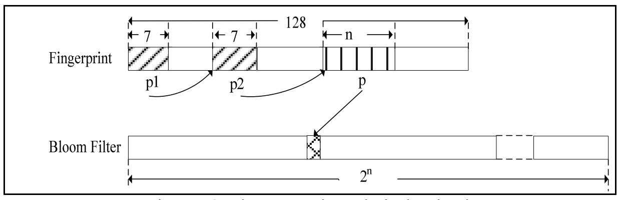

Figure 6. The second grade index hash

C. Performance of incremental for TBF

In these experiments, we mainly study the influence of the selection of the various number of hash functions to the throughput and the number of storage accessing of the false positive. In the experiments, the data is about 1TB, and the number of the data chunks, namely *n*, is 114,845,208.

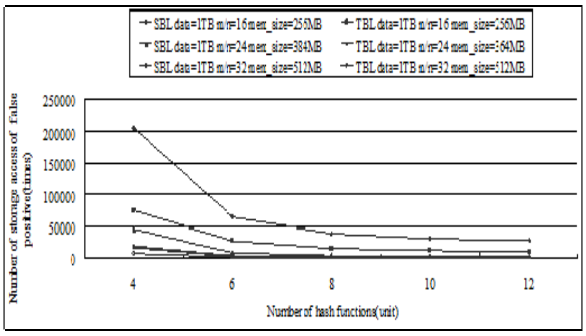

Figure 7. The number of storage accessing of the false positive

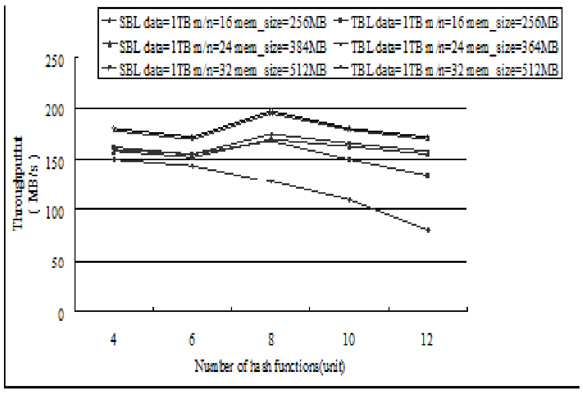

Figure 8. The throughput

For 1TB data, the total number of false positive storage accessing is no more than 250,000 after the filtering by the first grade Bloom Filter. While the memory space occupied by the second grade Bloom Filter is negligible. Fig.7 illustrates that TBF reduces about 30~40% storage accessing of false positive with the same length of the first grade Bloom Filter. In addition, it indicates that the performance-to-price is the optimal while the *k* is 8 and the length of the first Bloom Filter is 384MB. TBF has a limited capability to decrease the rate of the accessing of the false positive when increasing the numbers of the function or the length of the first grade Bloom Filter.

Fig.8 shows that the throughput achieves 195MB/s when *k* is 8 and the length of the first grade Bloom Filter is 384MB by adopting TBL. The throughput will decrease if *k* is increased. However when the length of first grade Bloom Filter increases, it is not obvious to observe the increase of the throughput.

VI. Conclusion

It is a key problem in the de-duplication backup system that how to judge whether a new partitioning data segment has been in the storage system as fast as possible. In this paper owing to sufficient investigations of related fields at domestic and abroad, a new mechanism TBF is proposed.

The false positive exists while introducing the SBF mechanism to the traditional data fingerprints comparing mechanism. The last result should be decided by accessing the disk through IO operations in most of the cases. However, introducing the TBF mechanism is not simply to add a new bloom filter on the SBF, the second grade bloom filter represents the entirety of the first bloom filter that each bit of the second grade bloom filter represents the chunks who have the same fingerprints. It also introduces a two-dimensional list, which gathers the absolute addresses of the data chunks having the same fingerprints in a list. By this characteristic, the fingerprints need not to be stored completely. It can judge whether the new input data chunk has been stored by traversing the list when meeting the same fingerprints. Contrary to it, the SBF must compare all the stored data chunks to get the result in the same case. A new hash function class based on the MD5 is also created to solve the problem of collisions for the fingerprints.

ACKNOWLEDGMENT

This research was supported by “the Fundamental Research Funds for the Central Universities”, South-Central University for Nationalities (Grant Number: CZY12009). And this work also was supported by the National Natural Science Foundation of China (No.60902053, 61272497).

REFERENCES

[1] (2011) The IDC website [Oneline]. Available: http://www.idc.com.
[2] Q. Sean and D. Sean, "Venti: A new approach to archival storage, " in proceedings of the Conference on File and Storage Technologies: USENIX Association, January 2002, pp.89-101.
[3] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system,” in Proceedings of the 6th USENIX Conference on File and Storage Technologies. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–14.
[4] D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki, "Improving duplicate elimination in storage systems,” ACM Trans. Storage, vol. 2, no. 4, 2006, pp. 424–448.
[5] M. Lillibridge et aI., "Sparse Indexing, Large Scale, Inline Deduplication Using Sampling and Locality," in proceedings of 7th USENIX Conference on File and Storage Technologies, USENIX Association, San Francisco, California, 2009, pp. 111-123.
[6] T. T. Tin, and T. L. Ni, "An efficient indexing mechanism for data deduplication," in proceedings of 2009 International Conference on the Current Trends in Information Technology (CTIT), December 2009, pp. 1-5.
[7] B. Deepavali, E. Kave, L. Darrell, and L. Mark, "Extreme Binning: Scalable, parallel deduplication for chunk-based file backup, " in Proceedings of the 17th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, London, UK, September 2009, pp. 1-9.
[8] K. Erik, U. Cristian, and D. Cezary, “Bimodal Content Defined Chunking for Backup Streams,” in Proceedings of 8th USENIX Conference on File and Storage Technologies, Feb, 2010, pp.18-18.
[9] L. Guanlin, J. Yu, and H.C. David, "Frequency Based Chunking for Data De-Duplication," in proceedings of 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2010 , pp.287-296.
[10] F. Li, C. Pei, A. Jussara, and Z. B. Andrie, "Summary Cache: A scalable wide-area web cache sharing protocol, " in proceedings of ACM SIGCOMM'98, Vancouver, Canada, October 1998, pp. 254-265.
[11] B. H. Burton, "Space/time trade-offs in hash coding with allowable rrors," Communications of the ACM, July 1970, pp. 422-426.
[12] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data, " in proceedings of OSDI ’06, USENIX Association, Berkeley, CA, USA, 2006, pp.205–218.
[13] N. Jain, M. Dahlin, and R. Tewari, "TAPER: Tiered Approach for Eliminating Redundancy in Replica Synchronization, " In proceedings of USENIX File And Storage Systems, USENIX Association Berkeley, CA, 2005, pp.435-447.
[14] B. Souvik, N. Ankur, and K. G. Vikas, "High throughput data redundancy removal algorithm with scalable performance," in proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers, Jan. Heraklion, Greece, 2011, pp.87-96.
[15] M. Mitzenmacher, "Compressed bloom filters," IEEE/ACM Trans. on Networking, vol. 10, no. 5, October 2002, pp. 604–612.
[16] D. Biplob, S. Sudipta, L. Jin, J. L. David, and H. C. D. David, "BloomFlash : Bloom Filter on Flash-Based Storage," in proceedings of the 31st International Conference on Distributed Computing Systems, Jun. 2011, pp. 635-644.
[17] A. Michael, F. Martin, J. Rob, C. Bradley. Kuszmaul, Dzejla, Medjedovic, M. Pablo, S. Pradeep, P. S. Richard., and Z. Erez, "Don't thrash: how to cache your hash on flash," in proceedings of the 3rd USENIX conference on Hot topics in storage and file systems, Jun. 2011, Portland, OR, pp.1-1.
[18] G. Deke, W. Jie, C. Honghui, Y. Ye and L. Xueshan,"The Dynamic Bloom Filters," IEEE Transactions on Knowledge and Data Engineering, IEEE Educational Activities Department, vol. 22, jan 2010, pp. 120-133.
[19] S. Brin, J. Davis, H. Carcia-Molina, Copy Detection Mechanisms for Digital Documents (weblink), 1994.
[20] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," Internet Mathematics, vol. 1, no. 4, 2003, pp. 485–509.
[21] G. EKE, C. HONGHUI, and W. JIE, “Theory and network application of dynamic bloom filters," in proceeding of IEEE Infocom Barcelona, Spain, 2006, pp. 1-12.
[22] L. CARTER, M. WEGMAN, “Universal classes of hash functions”, Computer and System Sciences, 1979, vol. 18, no. 2, pp.143-154.

Bin Zhou received the B.S. and M.S. degrees in School of Computer Science and Technology at National University of Defense Technology (NUDT), China, in 1994 and 2002, respectively. He is a PH.D Candidate in the College of Computer at Huazhong University of Science and Technology (HUST), China. He is currently an Associate Professor in

College of Computer Science of South-Central University for Nationalities. His main research interest is in massive data management.

Rongbo Zhu received the B.S. and M.S. degrees in Electronic and Information Engineering from Wuhan University of Technology, China, in 2000 and 2003, respectively; and Ph. D degree in communication and information systems from Shanghai Jiao Tong University, China, in 2006. He is currently an Associate Professor in College of Computer Science of South-Central University for Nationalities.

He is the Editor-In-Chief of International Journal of Satellite Communications Policy and Management, Associate Editor of International Journal of Radio Frequency Identification Technology and Applications. He serves as a guest editor for several journals, such as, Future Generation Computer Systems, Telecommunication Systems, and as a reviewer for numerous referred journals such as IEEE System Journal etc. He has been actively involved in around 10 international conferences, serving as TPC Chair of GCN'11, General Co-chair of ICICA'10 and so on. Dr. Zhu is a member of the ACM and IEEE.

Ying Zhang received the B.S. and M.S. degrees in School of Foreign Languages at Central China Normal University (CCNU), China, in 1994 and 1997, respectively. She is currently a lecturer in School of Foreign Languages at HUST.

Linhui Cheng received the M.S. degree in College of Computer Science at South-Central University for Nationalities, China, in 2008. She is a Lecture in the College of Computer Science at South-Central University for Nationalities, China. Her main research interest is in intelligent computing and data mining.