1-1-2013

A novel approach to data deduplication over the engineering-oriented cloud systems

Zhe Sun
University of Wollongong, zs789@uow.edu.au

Jun Shen
University of Wollongong, jshen@uow.edu.au

Jianming Young
University of Southern Queensland

A novel approach to data deduplication over the engineering-oriented cloud systems

Abstract

This paper presents a duplication-less storage system over the engineering-oriented cloud computing platforms. Our deduplication storage system, which manages data and duplication over the cloud system, consists of two major components, a front-end deduplication application and a mass storage system as back-end. Hadoop distributed file system (HDFS) is a common distribution file system on the cloud, which is used with Hadoop database (HBase). We use HDFS to build up a mass storage system and employ HBase to build up a fast indexing system. With a deduplication application, a scalable and parallel deduplicated cloud storage system can be effectively built up. We further use VMware to generate a simulated cloud environment. The simulation results demonstrate that our deduplication storage system is sufficiently accurate and efficient for distributed and cooperative data intensive engineering applications

Keywords

approach, data, engineering, novel, oriented, systems, cloud, over, deduplication

Disciplines

Physical Sciences and Mathematics

Publication Details

Sun, Z., Shen, J. & Yong, J. (2013). A novel approach to data deduplication over the engineering-oriented cloud systems. Integrated Computer Aided Engineering, 20 (1), 45-57.

A novel approach to data deduplication over the engineering-oriented cloud systems

Zhe SUN [a, b], Jun SHEN[a, *], Jianming YONG[c]
[a] *School of Information Systems and Technology, University of Wollongong, Wollongong, NSW, Australia*
[b] *Information Management Center, Huaneng Shandong Shidao Bay Nuclear Power Co., Ltd, Shandong, China*
[c] *School of Information Systems, University of Southern Queensland, Toowoomba, QLD, Australia*

Abstract. This paper presents a duplication-less storage system over the engineering-oriented cloud computing platforms. Our deduplication storage system, which manages data and duplication over the cloud system, consists of two major components, a front-end deduplication application and a mass storage system as back-end. Hadoop distributed file system (HDFS) is a common distribution file system on the cloud, which is used with Hadoop database (HBase). We use HDFS to build up a mass storage system and employ HBase to build up a fast indexing system. With a deduplication application, a scalable and parallel deduplicated cloud storage system can be effectively built up. We further use VMware to generate a simulated cloud environment. The simulation results demonstrate that our deduplication storage system is sufficiently accurate and efficient for distributed and cooperative data intensive engineering applications.

Keywords: Cloud storage, data deduplication, Hadoop distributed file system, Hadoop database

1. Introduction

Modern society is a digital universe. Almost no information or industry applications can survive without this digital universe. The size of this digital universe in 2007 is 281 exabytes and in 2011 [10], it becomes 10 times larger than it was in 2007. The most critical issue is that nearly half the digital universe cannot be stored properly in time. This is caused by several reasons: firstly, it is hard to find such a big data container; secondly, even if a big container can be found, it is still impossible to manage such a vast dataset; and finally, for economic reasons, building and maintaining such a huge storage system will cost a lot of money. This is particularly challenging for non-IT sectors, for example, engineering and bio-chemistry industries. According to our experiences, a typical information management center at a city-level nuclear power generation factory needs to process hundreds of gigabytes of new data each day. Such data should also be easily accessible and used for different purposes by other information centers located in other cities in the power grid, as well as government authorities at different levels. In the area of computer aided engineering (CAE), some efforts are made to tackle challenges in the management of large quantity distributed data and knowledge [20, 28]. But the issue of scalability remains.

Fortunately, with the rocket-like development of cloud computing, the advantages of cloud storage have amplified significantly, and the concept of cloud storage has become vastly accepted by the community.

Cloud computing consists of both applications and hardware delivered to users as a service via the Internet [13, 18]. With the rapid development of cloud computing, more and more cloud services have emerged, such as SaaS (software as a service), PaaS (platform as a service) and IaaS (infrastructure as a service).

The concept of cloud storage is derived from cloud computing. It refers to a storage device accessed over the Internet via Web service application program interfaces (API). HDFS (namely Hadoop Distributed File System, hadoop.apache.org) is a distributed file system that runs on commodity hardware; it was de-

[*] Corresponding author: Jun Shen, E-mail: jshen@uow.edu.au, tel: +61-2-42213873, fax: +61-2-42214045

veloped by Apache for managing massive data. The advantage of HDFS is that it can be used in a high throughput and large dataset environment. HBase is Hadoop database, which is an open-source, distributed, versioned, column-oriented database [5]. It is good at real time queries. HDFS has been used in numerous large scale engineering applications. Based on these features, in our work, we use HDFS as a storage system. We use HBase as an indexing system.

Currently cloud computing is applied more in data-intensive areas such as e-commerce or scientific computations. There is little research on engineering-oriented cloud system. There especially lack direct applications or experiments for cooperative work in design, where data sharing and duplication management have always been challenges.

This paper presents a deduplication cloud storage system, named "DeDu", which runs on commodity hardware. Deduplication means that the number of the replicas of data that were traditionally duplicated on the cloud should be managed and controlled to decrease the real storage space requested for such duplications. At the front end, DeDu has a deduplication application. At the back end, there are two main components, HDFS and HBase, used respectively as a mass storage system and a fast indexing system. Promising results were obtained from our simulations using VMware to simulate a cloud environment and execute the application on the cloud environment.

Regarding contributions of this paper, there are two issues to be addressed. Firstly, how does the system identify the data duplications? Secondly, how does the system manage and manipulate the data to reduce the duplications, in other words, to deduplicate them?

For the first issue, we use both the MD5 and SHA-1 algorithm to make a unique fingerprint for each file or data block, and set up a fast fingerprint index to identify the duplications. For the second problem, we set up a distribution file system to store data and develop 'link files' to manage files in a distributed file system.

The details of the novel solutions will be presented in the rest of the paper. Section 2 introduces related work. Section 3 introduces our approaches. Section 4 discusses the system design. Section 5 presents our simulation and experiments. Section 6 contains an overview of the performance results and discussion of the evaluations. Finally, Section 7 is the conclusion and future work.

2. Related work

There are many distributed file systems that have been proposed for large scale information systems, which can be distributed over the Internet and this includes mutable and non-trusted peers. All these systems have to tolerate frequent configuration changes. For example, Ceph [26], RADOS [21], Petal [9], GFS [12], Ursa Minor [19], Panasas [6], Farsite [1], FAB [27], and P2CP [23] are all systems designed for a high performance cluster or data centered environments, which are not necessarily engineering oriented. Our DeDu is intended not only for large size industrial or enterprise-level data centers, but also for common users' data storage.

In CAE area, to deal with semi-structured data in database, a variant Hierarchical Agglomerative Clustering (HAC) algorithm called k-Neighbors-HAC is developed in [11] to use the similarities between data format (HTML tags) and data content (text string values) to group similar text tokens into clusters. This approach avoids the pattern induction process by using clustering techniques on unlabelled pages.

In [14], the authors describe a technique that supports the querying of a relational database (RDB) using a standard search engine. The technique involves expressing database queries through URLs. The technique also includes the development of a special wrapper that can process the URL-query and generate web pages that contain the answer to the query, as well as links to additional data. By following these specialized links, a standard web crawler can index the RDB along with all the URL-queries. Once the content and their corresponding URL-queries have been indexed, a user may submit keyword queries through a standard search engine and receive the most current information in the database.

To handle scalable deduplication, two famous approaches have been proposed, namely sparse indexing [17] and Bloom filters [29] with caching. Sparse indexing is a technique used to solve the chunk lookup bottleneck caused by disk access, by using sampling and exploiting the inherent locality within backup streams. It picks a small portion of the chunks in the stream as samples; then, the sparse index maps these samples to the existing segments in which they occur. The incoming streams are broken up into relatively large segments, and each segment is deduplicated against only some of the most similar previous segments. The Bloom filter exploits Summary Vector, which is a compact in-memory data structure, for identifying new segments; and Stream-Informed

Segment Layout, which is a data layout method to improve on-disk locality, for sequentially accessed segments; and Locality Preserved Caching with cache fragments, which maintains the locality of the fingerprints of duplicated segments, to achieve high cache hit ratios.

So far, several deduplication storage systems have been previously designed, including Venti [22], DeDe [2], HYDRAstor [7], Extreme Binnig [3], MAD2 [25] and DDE[17].

Venti [22] is a network storage system. It uses unique hash values to identify block contents so that it reduces the data occupation of storage space. Venti builds blocks for mass storage applications and enforces a write-once policy to avoid destruction of data. This network storage system emerged in the early stages of network storage, so it is not suitable to deal with mass data, and the system is not scalable.

DeDe [2] is a block-level deduplication cluster file system without centralized coordination. In the DeDe system, each host creates content summaries then the hosts exchange content summaries in order to share the index and reclaim duplications periodically and independently. These deduplication activities do not occur at the file level, and the results of deduplication are not accurate.

HYDRAstor [7] is a scalable, secondary storage solution, which includes a back-end consisting of a grid of storage nodes with a decentralized hash index, and a traditional file system interface as a front-end. The back-end of HYDRAstor is based on Directed Acyclic Graph, which is able to organize large-scale, variable-size, content addressed, immutable, and highly-resilient data blocks. HYDRAstor detects duplications according to the hash table. The ultimate target of this approach is to form a backup system. It does not consider the situation when multiple users need to share files.

Extreme Binning [3] is a scalable, paralleled deduplication approach aiming at a non-traditional backup workload which is composed of low-locality individual files. Extreme Binning exploits file similarity instead of locality, and allows only one disk access for chunk look-up per file. Extreme Binning organizes similar files into bins and deletes replicated chunks inside each bin. Replicates exist among different bins. Extreme Binning only keeps the primary index in memory in order to reduce RAM occupation. This approach is not a strict deduplication method, because duplications will exist among bins.

MAD2 [25] is an exact deduplication network backup service which works at both the file level and the chunk level. It uses four techniques: a hash bucket matrix, a Bloom filters array, dual cache, and Distributed Hash Table based load balancing, to achieve high performance. This approach is designed for backup service not for a pure storage system.

Duplicate Data Elimination (DDE) [16] employs a combination of content hashing, copy-on-write, and lazy updates to achieve the functions of identifying and coalescing identical data blocks in a storage area network file system. It always processes in the background.

However, what sets DeDu decisively apart from all these approaches is that DeDu exactly deduplicates and calculates hash values at the client side right before data transmission, all at file level.

3. Approaches

In the early 1990s, the ‘once write multi read’ storage concept was set up and the optical disk was widely used as storage media. The challenges posed by this storage concept were the great obstacles encountered when sharing data via the Internet and the enormous wastage of storage space to keep replications. In the novel cloud computing era, we propose a new network storage system, “DeDu”, to store data that is easy to share, and at the same time, to save storage space, even for duplicated replicas.

3.1 System Mechanism

In DeDu, when a user uploads a file for the first time, the system records this file as source data, and the user will receive a link file for this user himself, and the same for other potential users, to access the source data. When the source data has been stored in the system, if the same data is uploaded by other users, the system will not accept the same data as new, but rather, the new user, who is uploading data, will receive a link file to the original source data. Users are allowed to read the source data but not to write. Once the initial user changes the original data, the system will set the changed data as a new one, and a new link file will be given to the user. The other users who connect to the original file will not be impacted [24]. Under these conditions, the more users share the same data, the more storage space will be saved.

3.2 Identifying the duplications

To delete the duplications, the first step is to identify the duplications. The two normal ways to identify

duplications are: comparing data blocks or files bit by bit and comparing data blocks or files by hash values. To compare blocks or files bit by bit would guarantee accuracy, at the cost of additional time consumption. To compare blocks or files by hash value would be more efficient, but the chances of accidental collisions would be increased. The chance of accidental collision depends on the hash algorithm. However, the chances are minute. Thus, using a combination hash value to identify the duplications will greatly reduce the collision probability. Therefore, it is acceptable to use a hash function to identify duplications [4, 15].

The existing approaches for identifying duplications are always carried out on two different levels. One is at the file level; the other is at the chunk level. On the file level, the hash function will be executed for each file, and all hash values will be kept in the index. The advantage of this approach is that it decreases the quantity of hash values significantly. The drawback is that, the system will be experiencing some lag when dealing with a large file. On the chunk level, data streams are divided into chunks, each chunk will be hashed, and all these hash values will be kept in the index. The advantage of this approach is that it is convenient for a distributed file system to store chunks, but the drawback is having an increasing quantity of hash values. It means that hash values will occupy more RAM usage and increases lookup time.

In this paper, our deduplication method is at the file level and is based on comparing hash values. There are several hash algorithms, such as MD5 and SHA-1. We use both the SHA-1and MD5 algorithms to identify duplications. Because both MD5 and SHA-1 are mature products, and furthermore they have a series of hash algorithms, we can quickly find substitutes to update them if needed.

The reason for choosing file level deduplication is that we want to keep the index as small as possible in order to achieve high lookup efficiency. On the other hand, although the probability of accidental collision is extremely low, we still combine MD5 and SHA-1 together. We merge the MD5 hash value and the SHA-1 hash value as the primary value in order to avoid any possible accidental collisions. If the MD5 algorithm and the SHA-1 algorithm are not suitable for our system scale, it can be changed at any time.

3.3 Storage System

We need two mechanisms to set up our storage system. One is used to store mass data, and the other is used to keep the sparse index. On the one hand, there are several secondary storage systems, like Ceph [24], Petal [9], being used as mass data storage systems. On the other hand, there are several database systems such as SQL, Oracle, HBase, and BigTable [8] that can be used as index systems. All these systems have their own features, but we need two systems combined together to achieve our data access requirements. With regard to our requirements, in order to store masses of information, the file system must be stable, scalable, and fault-tolerant; for the sparse index, the system must perform nicely on real time queries.

Considering these requirements, we employ HDFS and HBase to structure our storage mechanisms. The advantage of HDFS is that it can be used under high throughput and large dataset conditions, and it is stable, scalable, and fault-tolerant. HBase is advantageous in queries. Both HDFS and HBase were developed by Apache for storing mass data which was modeled by Google File System and BigTable. Based on these features, in our system, we use HDFS as a storage system and use HBase as an index system. We will introduce how HDFS and HBase collaborate in the Section 4.

4. System design

4.1 Storage platform

Our storage platform consists of HDFS and HBase. Fig. 1 shows the architecture of the DeDu deduplication cloud storage system. It consists of one DFS master, one HBase master and several data nodes. DFS master is the master node of HDFS. The advantage of using HDFS and HBase is to have data awareness between the jobtracker and tasktracker. It does not store any file by itself, whereas it only keeps the location information of original data files which are stored on data nodes. HBase master does not store index either, it keeps the location information of sparse index which is stored on data nodes. Data nodes could be dynamic, scalable and load balanced. If a DFS master or HBase master is halted, any data node will be able to replace the master node. In this way, storage platform will keep running without downtime.

In the system, control flow passes the managing information of the storage system while data flow transfers data. In addition, control flow and data flow are separated in our system. Clients connect to master nodes with control flow; while master nodes also communicate with data nodes through control flow. On the other hand, original data are only transmitted between clients and data nodes, thus the system avoids the bottleneck which could be caused by the master node performance.

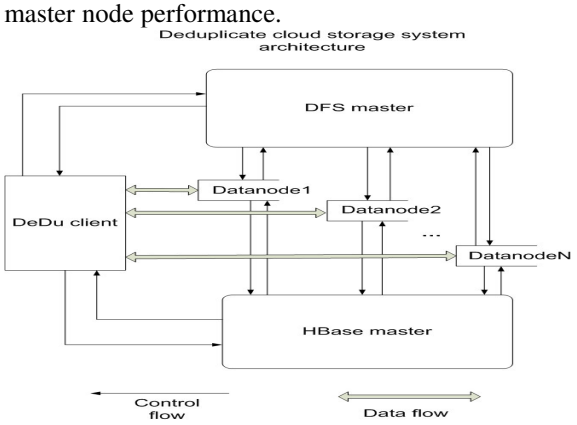

Fig. 1: Architecture of deduplication cloud storage system.

4.2 Data organization

In this system, HDFS and HBase must collaborate to guarantee that the system is working well. There are two types of files saved in HDFS, one is source file, and the other one is link file. We separate source files and link files into different folders (see Fig. 2).

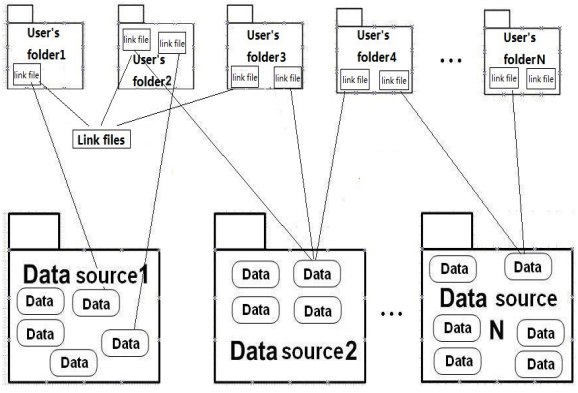

Fig. 2: Data organisation.

In the DeDu system, each source file is named after the combination hash value and saved in a folder which is named by date. When the source file is over 64MB, it will be divided into 64MB chunks and saved in the system, but these chunks will be distributed in different data nodes. As for the link file, the filename is in the form "***.lnk", where "***" is the original name/extension of the source file. Every link file records one hash value for each source file and the logical path to the source file, and it uses approximately 320 bytes to store the essential information. Both link file and the folder created by the user are saved in the distribution file system.

HBase records all the hash values for each file, the number of links, and the logical path to the source file. There is only one table in HBase, which is named "dedu". There are three columns in the table, which have the headings: hash_value, count, and path. Hash_value is the primary key. Count is used to calculate the number of links for each source file. Path is used for recording the logical path to the source file.

4.3 Procedures to store the files

In DeDu, there are three main steps to save a file. Firstly, it is necessary to make a hash value at the client; secondly, the system identifies any duplication; thirdly, the system saves the file. Fig. 3 shows the procedures for storing a file.

Firstly, users select the files or folders which are going to be uploaded and stored by using a DeDu application. The application uses the MD5 and SHA-1 hash functions to calculate the file's hash value, and then pass the value to HBase.

Secondly, the table 'dedu' in HBase keeps all file hash values. HBase is operated in the HDFS environment. It will compare the new hash value with the existing values. If it does not exist, a new hash value will be recorded in the table, and then HDFS will ask clients to upload the files and record the logical path; if it does exist, HDFS will check the number of links, and if the number is not zero, the counter will be incremented by one. In this case, HDFS will tell the clients that the file has been saved previously. If the number is zero, HDFS will ask the client to upload the file and update the logical path.

Thirdly, HDFS will store source files, which are uploaded by users, and corresponding link files, which are automatically generated by DeDu. Link files record the source file's hash value and the logical path of the source file.

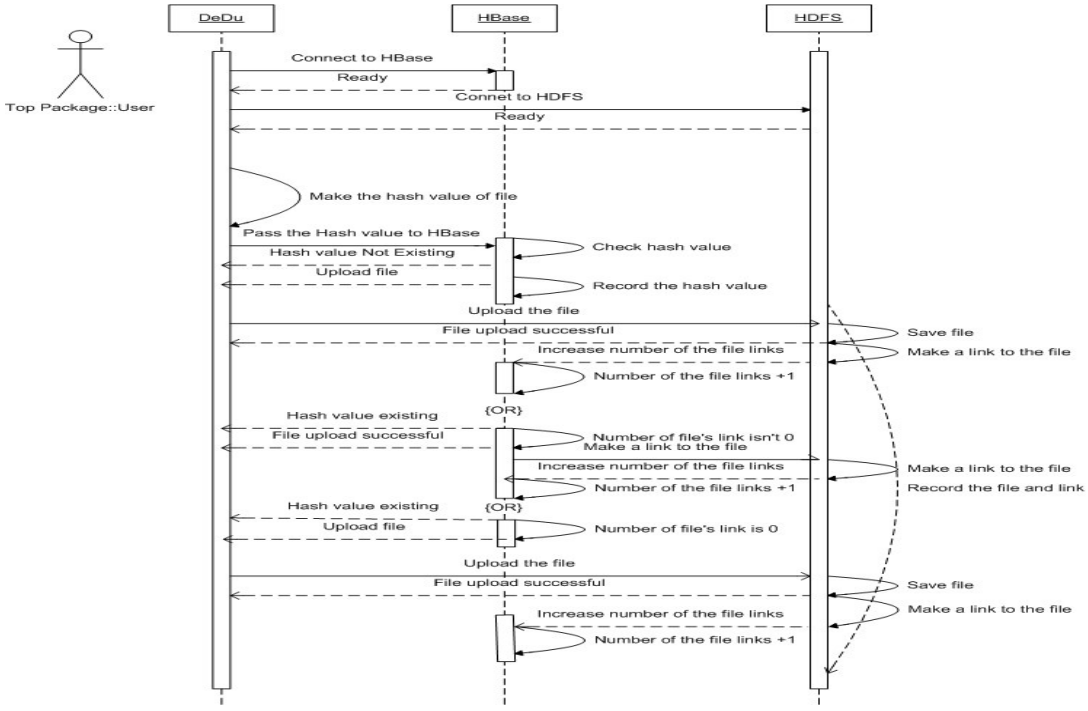

Fig. 3: Procedures for storing a file

4.4 Access to the files

In our system, we use a special approach to access a file, which is the link file. Each link file records two types of information: the hash value and the logical path to the source file. When clients access the file, they first access the link file, and the link file will pass the logical path of the source file to HDFS. HDFS will then enquire the master node for the block locations. When the clients get the block locations, they can retrieve the source file from the data nodes. Fig. 4 shows the procedures to access a file.

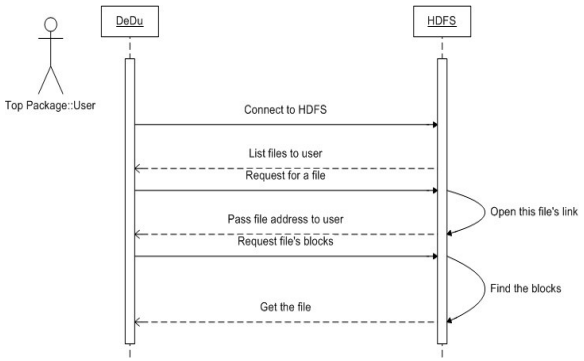

Fig. 4: Procedures to access a file.

4.5 Deletion of files

In our system, there are two types of approaches for deletion: in one case, the file is pseudo-deleted, and in the other case, it is fully-deleted. This is because different users may have the same authority to access and control the same file. We don't allow one user to delete a source file which is shared by other users, so we use pseudo-deletion and fully-deletion to solve this problem. When a user deletes a file, the system will delete the link file which is owned by the user, and the number of links will be decremented by one. This means that this particular user loses the right to access the file, but the source files are still stored in the HDFS. The file is pseudo-deleted. A source file may have many link files pointing to it, so while the user may delete one link file, this has no impact on the source file. When the last link file has been deleted, however, the source file will be deleted; so the file is now fully-deleted. Fig. 5 shows the procedures for deleting a file.

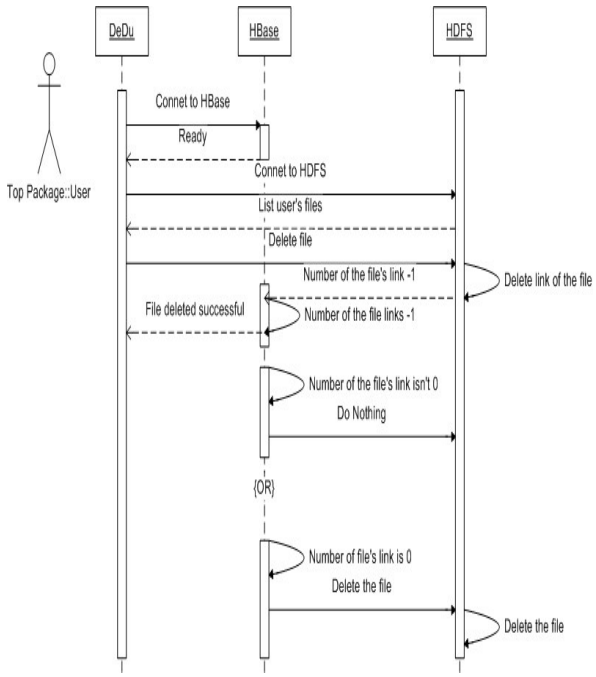

Fig. 5: Procedures to delete a file.

5. Simulations and experiments

In DeDu, we developed a graphic user interface to help clients to use it. Clients can upload, download, and delete files. The graphic interface is very easy to use and just involves dragging and dropping the file to the local system or HDFS. We designed 4 different experiments for performance evaluations on DeDu:

Experiment 1:

In our experiment 1, our cloud storage platform was set up on a VMware 7.10 workstation. The configuration of the host machine is that, the CPU is 3.32 GHz; RAM is 4 GB; Hard disk is 320GB; Operation system is Windows XP Professional 5.1.2600, Service Pack 3. Five virtual machines exist in the cloud storage platform, and each virtual machine has the same configuration. The configuration of each virtual machine is that the CPU is 3.32 GHz; RAM is 512 MB; Hard disk is 20 GB. The net adaptor is bridged for each machine. The operating system is Linux mint. The version of HDFS is Hadoop 0.20.2, and the version of HBase is 0.20.6.

Experiment 2:

In our experiment 2, the cloud storage platform was also set up on a VMware 7.10 workstation. But the configuration of the host machine is that the CPU is Intel Xeon E7420 2.13 GHz; RAM is 8 GB; Hard disk is 1T = 320GB * 4; Operation system is Windows Server 2003 Enterprise Edition, Service Pack 2. The configuration of virtual machine is that the master node's RAM being increased to 2GB, and the CPU is 8 cores. The data nodes' RAM is 512MB, and the CPU is 8 cores.

Experiment 3:

In our experiment 3, we change the parameter of the data nodes. The master node keeps the same configuration. All the data nodes' RAM is still 512MB, but the CPU is 2 cores.

Experiment 4:

In our experiment 4, we repeat the experiment 3, but with the disk compression. The detailed information is listed in Table 1.

Table 1

Configuration of virtual machines.

	Physical Host	Virtual Hosts	CPU	RAM	Disk compression
Ex 1.	CPU core 3.32 GHz, RAM 2GB, Disk 320GB	Master nodes	2 Core	512MB	None
		Data nodes	2 Core	512MB	None
Ex 2	CPU Xeon E7420 2.13GHz RAM 8GB Disk 4* 320GB	Master nodes	8 Core	2 GB	None
		Data nodes	8 Core	512MB	None
Ex 3		Master nodes	8 Core	2 GB	None
		Data nodes	2 Core	512MB	None
Ex 4		Master nodes	8 Core	2 GB	Yes
		Data nodes	2 Core	512MB	Yes

All the experiment results are showed in section 6.

6. Performance evaluations

6.1 Deduplication Efficiency

In our experiment, we uploaded 110,000 files, amounting to 475.2 GB, into DeDu. In a traditional storage system, they should occupy 475.2 GB as shown by the non-deduplication line in Fig. 6, and if they are stored in a traditional distribution file system, both the physical storage space and the number of files should be three times larger than 475.2 and 110,000, that is, 1425.6 GB and 330,000 files, because in the traditional distributed file system, data has three copies conventionally. We did not show it in the figure, because the scale of the figure would become too large. In a perfect deduplication distribution file system, where there is no duplication, the resulted system should take up 37 GB, and the number of files should be 3,200; but in DeDu, we achieved 38.1 GB, just 1.1 GB larger than the perfect situation. The extra 1.1 GB of data are occupied by link files and the 'dedu' table, which is saved in HBase. The following figure (Fig. 6) shows the deduplication system efficiency.

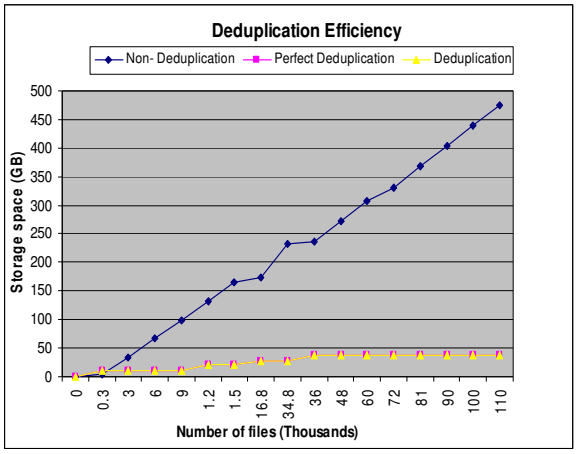

Fig. 6: Deduplication efficiency

By using the distribution hashing index, an exact deduplication result is achieved. In our storage system, each file could only be kept in 3 copies at different data nodes as backup, in case some data nodes are dead. This means that if a file is saved into this system for less than three times, the efficiency of deduplication is low. When a file is put into the system more than three times, the efficiency of deduplication will increase. Thus, the real deduplication efficiency depends on both the original data duplication ratio and how many times the original data has been saved. The higher the duplication ratio the original data has, the greater the deduplication efficiency will be achieved. It is also true that the more the number of times that the original data is saved, the greater the deduplication efficiency will be achieved.

6.2 Data Distribution

As we mentioned previously, in the DeDu system, each source file is named after the combination hash value and saved in a folder which is named by the DeDu system's manipulation date on it.

Fig. 7 shows that the source file is divided into 4 chunks. Each chunk has three replicates, but these chunks are distributed in different data nodes. For example, chunks may be distributed into data node 1, data node 2, and data node 3 and data node 5 (IP addresses for all nodes are ranging from 192.168.58.140 to 192.168.58.145). In this way, even if some data nodes may halt, the data availability will not be impacted. Our test result of load balance will be showed in the section 6.3.

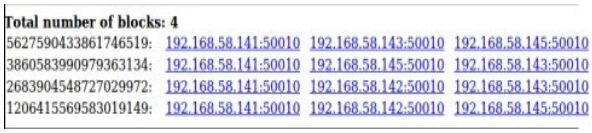

Fig. 7: Distribution of blocks.

6.3 Load Balance

Because each data node keeps different numbers of blocks, and the client will directly download the data from the data nodes. In this case we have to keep an eye on load balance, in case some data nodes are overloaded, while others are idle.

6.3.1 Static Load Balance

Fig. 8 shows the balanced situation in 4 data nodes. In the situation of no deduplication, DN1 (Linux mint2) stores 116 gigabytes of data; DN3 (Linux mint4) stores 114 gigabytes of data; both DN2 (Linux mint3) and DN4 (Linux mint5) each store 115 gigabytes of data. With deduplication, DN2 stores 6.95 gigabytes of data; DN3 stores 6.79 gigabytes of data; and DN1 and DN3 each store 7 gigabytes of data. The perfectly deduplicated situation, where there is no duplication in the system, should occur when each node stores 6.9 gigabytes of data.

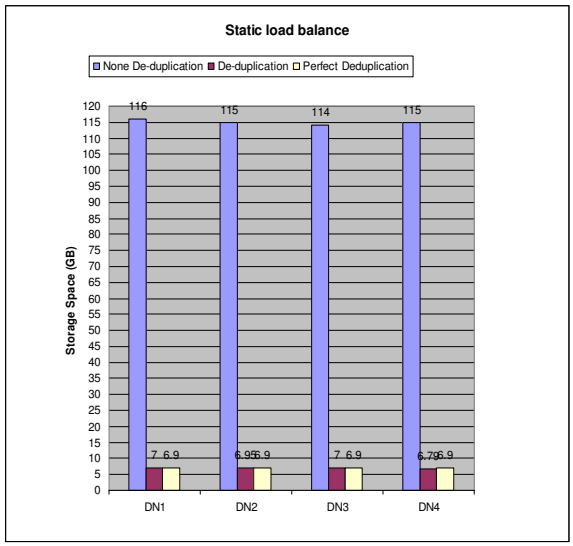

Fig. 8: Static load balance.

We can see that each data node stores a different amount of data, no matter whether it is at the hundred gigabytes level or at the dozens of gigabytes level. The usage of storage space in each node is slightly different, but the differences between space occupations in the same situation with respect to the numbers of blocks will not be more than 10%.

6.3.2 *Dynamic Load Balance*

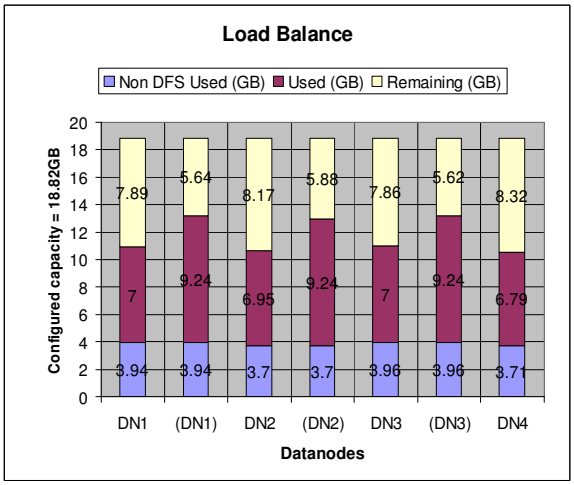

Fig. 9: Dynamic load balance.

When we delete a node from, or add a new node into, the system, DeDu will achieve balance automatically. Note the default communication bandwidth is 1 MB/s, thus the balance efficiency is a bit low, unless the balance command is entered manually. Fig. 9 shows that the deduplication load is balanced in a 3 data nodes environment, as indicated by brackets, and a 4 data nodes environment. We can see that when it is in the 3 data node environment, each data node stores 9.24 GB of data. After one more data node (DN4) is added into the system, DN2 stores 6.95 gigabytes of data; DN3 stores 6.79 gigabytes of data; and DN1 and DN3 each store 7 gigabytes of data.

6.4 *Reading Efficiency*

For file reading efficiency tests, we tested the system with two data streams. One data stream is 295 items amounting to 3.3GB and another data stream is 22 items, amounting to 9.2 GB, in other words, we tested a large number of small files as well as a small number of large files. The details of reading (downloading) efficiency are given in Table 2.

Table 2

Reading efficiency

Reading efficiency				
Two nodes	**Time (Seconds)**	**Items**	**Size (GB)**	**Speed (MB/s)**
Experiment 1 Testing1	356	295	3.3	9.49
Experiment 1 Testing2	510	22	9.2	18.47
Four nodes	**Time (Seconds)**	**Items**	**Size (GB)**	**Speed (MB/s)**
Experiment 1 Testing1	345	295	3.3	9.79
Experiment 1 Testing2	475	22	9.2	19.83
Experiment 2 Testing1	418	295	3.3	8.08
Experiment 2 Testing2	453	22	9.2	20.83
Experiment 3 Testing1	477	295	3.3	7.08
Experiment 3 Testing2	622	22	9.2	15.12
Experiment 4 Testing1	807	295	3.3	4.18
Experiment 4 Testing2	1084	22	9.2	8.71

6.5 *Writing Efficiency*

In this section, we will consider the system's writing (uploading) efficiency with two and four data nodes. Furthermore, in the real world, deduplication

happens randomly, thus we just consider the writing efficiency with complete deduplication and the writing efficiency without deduplication in this paper. We tested the system with two data streams as same as testing reading efficiency. Similarly, one data stream is 295 items amounting to 3.3GB and another data stream is 22 items, amounting to 9.2 GB. The details of writing efficiency without deduplication are listed in Table 3. The details of writing efficiency with complete deduplication are listed in Table 4.

Table 3

Writing efficiency without deduplication.

Without deduplication				
Write into Two nodes	**Time (Seconds)**	**Items**	**Size (GB)**	**Speed (MB/s)**
Experiment1 Testing1	771	295	3.3	4.38
Experiment1 Testing2	2628	22	9.2	3.58
Write into Four nodes	**Time (Seconds)**	**Items**	**Size (GB)**	**Speed (MB/s)**
Experiment1 Testing1	813	295	3.3	4.15
Experiment1 Testing2	2644	22	9.2	3.56
Experiment 2 Testing1	2421	295	3.3	1.40
Experiment 2 Testing2	2511	22	9.2	3.82
Experiment 3 Testing1	2477	295	3.3	1.36
Experiment 3 Testing2	1967	22	9.2	4.73
Experiment 4 Testing1	3776	295	3.3	0.90
Experiment 4 Testing2	3804	22	9.2	2.56

Table 4

Writing efficiency with complete deduplication

Complete deduplication				
Write into Two nodes.	**Time (Seconds)**	**Items**	**Size (GB)**	**Speed (MB/s)**
Experiment1 Testing1	401	295	3.3	8.43
Experiment1 Testing2	462	22	9.2	20.39
Write into Four nodes.	**Time (Seconds)**	**Items**	**Size (GB)**	**Speed (MB/s)**
Experiment1 Testing1	356	295	3.3	9.49
Experiment1 Testing2	475	22	9.2	19.83
Experiment 2 Testing1	482	295	3.3	7.01
Experiment 2 Testing2	522	22	9.2	18.04
Experiment 3 Testing1	491	295	3.3	6.88
Experiment 3 Testing2	457	22	9.2	20.61
Experiment 4 Testing1	672	295	3.3	5.02
Experiment 4 Testing2	820	22	9.2	11.49

All of these transmission speeds are calculated by total cost of time on transmitting files. When we monitor the net adaptor, the peak writing speed is 32MB/s; the peak reading speed is 71MB/s. All the test results shown in the tables are average of the multiple runs to minimize noise.

6.6 Discussion on testing results

Deduplication efficiency has two aspects: one is efficiency in identification of duplication, the other is efficiency in deletion of duplication.

We can get valuable findings from Table 2 to Table 4. First, we compare the reading efficiency results of experiment 1 (both testing 1 and testing 2) in two data nodes model and four data nodes model, respectively. In the two data node situation, the download speeds for two testing are 9.49 MB/s and 18.47 MB/s respectively; the upload speeds without duplications are 4.38 MB/s and 3.58 MB/s; the upload speeds with complete duplication are 8.43 MB/s and 20.39 MB/s. In the four data node situation, the download speeds are 9.79 MB/s and 19.83 MB/s. The upload speeds without duplication are 4.15 MB/s and 3.56 MB/s; the upload speeds with complete duplication are 9.49 MB/s and 19.83 MB/s.

From these results, we find that, generally, the fewer the data nodes the real DeDu system has, the higher the writing efficiency it will get, however it comes at the cost of lowered reading efficiency. The more data nodes there are, the lower the writing efficiency it will get, but the higher the reading efficiency.

Secondly, we compare the results of experiment 1 and experiment 2, as well as experiment 1 and experiment 3, in four data nodes model. We got a surprising result that neither reading nor writing efficiency was enhanced even if the system was equipped with better host machine's configuration. This is caused by the host machine’s operation system. Experiment 2 and experiment 3 were running upon the Windows Server 2003 enterprise edition. This version system is 32 bit, so it only supports 4 GB RAM, rather than 8 GB RAM. But the total RAM

of virtual machines is 4 GB and it does not include host operation system's RAM. Thus the required RAM is beyond the host machine's configuration. This leads both reading and writing efficiency to decrease no matter how we changed the machines' capability.

Third, when we compare the results of experiment 1's testing 1 and testing 2, as well as the results of experiment 2's testing 1 and testing 2 in four data nodes model, with complete deduplication and without deduplication, we can find that when a single file is large, the time to calculate hash values becomes higher, but the time of whole transmission cost is relatively low. When a single file is small, the time to calculate hash values becomes lower, but the whole transmission cost is relatively high. This is because the speed of calculating hash value is much faster than data transmission.

Fourth, when we compare the results of experiment 2 and experiment 3, we can get that the performance of DeDu is not only decided by the configuration of master node, but also the configuration of data nodes. The configuration of virtual machine in experiment 2 is that the master node's RAM is 2GB, and the CPU is 8 cores. The data nodes' RAM is 512MB, and the CPU is 8 cores. The configuration of virtual machine in experiments 3 is that, data nodes' CPU is changed to 2 cores. From the testing results, we can see that the performance of data nodes with 8 cores is much better than 2 cores in general, particularly for reading (downloading from data nodes), except that in testing 2 of both experiments 2 and 3, where the system shows an opposite result. This is because of the performance bottleneck associated with the master node. In our experiments, we calculated hash values on the master nodes, and each of the files needed to be hashed in testing 2 was quite large. This had caused the overall performance to deteriorate. In a real environment, calculation of the hash value can be deployed to clients.

On the other hand, this comparison shows that, in cloud storage systems, the configuration of a single node will probably not impact the whole system to a large extent, depending on what computation and data transmission task has been deployed to which nodes. To enhance the cloud storage system performance, we must improve or modify a group of nodes' configurations to balance the load and minimize the number of bottleneck nodes.

Fifth, comparing the results of experiment 3 and experiment 4, we can find that data compression is a nightmare for data transmission. Both reading efficiency and writing efficiency will be decreased significantly by using data compression. Data compression could save storage spaces, but it sacrifices data transmission efficiency. In our experiment, during the data transmission process, more than 30% time is spent on dealing with data compression.

7. Conclusions and future work

In conclusion, we have introduced a novel approach to data deduplication over the engineering-oriented cloud systems, which we have named as DeDu. DeDu is not only useful for IT enterprises or engineering industry to backup data, but also for common users who want to store data. Our approach exploits a file's hash value as an index saved in HBase to attain high lookup performance, and it exploits 'link files' to manage mass data in a Hadoop distributed file system.

In DeDu, the hash value is calculated at the client side prior to data transmission, and the lookup function is executed in HBase. When duplication is found, real data transmission will not occur. Based on our testing, the features of DeDu are as follows: The fewer the data nodes that the system maintains, the higher the writing efficiency; but the lower the reading efficiency, will be achieved. The more data nodes there are, the lower the writing efficiency, but the higher the reading efficiency DeDu will get. When a single file is large, the time to calculate hash values becomes higher, but the time of transmission cost is relatively lower; when a single file is small, the time to calculate hash values becomes lower, but the overall transmission overhead in such a deduplication system is relatively higher.

In DeDu, the higher configuration of a single node will not impact the performance of the whole system too much. To get better performance from such a cloud-type system, we need to consider how to tune up critical nodes' configurations. Furthermore, data compression is another potential hurdle in DeDu. It sacrifices too much overall data transmission efficiency to save the storage space in a deduplication application. We will look into these issues in future work, to circumvent the design objective of DeDu.

References

[1] A. Atul, J.B. William, C. Miguel, C. Gerald, C. Ronnie, R.D. John, H. Jon, R.L. Jacob, T. Marvin and P.W. Roger, Farsite: Federated, available, and reliable storage for an incompletely trusted environment, SIGOPS Oper. Syst. Rev., 2002, 36, pp. 1-14

[2] T.C. Austin, A. Irfan, V. Murali and L. Jinyuan, Decentralized deduplication in SAN cluster file systems, in Proceedings of the 2009 Conference on USENIX Annual technical conference, San Diego, California, 2009, pp. 101-114.

[3] D. Bhagwat, K. Eshghi, D.D.E. Long and M. Lillibridge, Extreme Binning: Scalable, Parallel Deduplication for Chunk-based File Backup, in 2009 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems Mascots, 2009, pp. 237-245.

[4] J. Black. Compare-by-hash: A reasoned analysis, in USENIX Association Proceedings of the 2006 USENIX Annual Technical Conference, 2006, pp. 85-90.

[5] D. Borthakur, The Hadoop Distributed File System: Architecture and Design, 2007. URL: http://hadoop.apache.org/hdfs/docs/current/hdfs_design.pdf, accessed in Oct 2011.

[6] W. Brent, U. Marc, A. Zainul, G. Garth, M. Brian, S. Jason, Z. Jim and Z. Bin, Scalable performance of the Panasas parallel file system, in Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST 2008), San Jose, California, 2008, pp. 17-33.

[7] D. Cezary, G. Leszek, H. Lukasz, K. Michal, K. Wojciech, S. Przemyslaw, S. Jerzy, U. Cristian and W. Michal, HYDRAstor: a Scalable Secondary Storage, in Proccedings of the 7th conference on File and Storage Technologies, San Francisco, California, 2009, pp. 197-210.

[8] 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 7th Symposium on Operating Systems Design and Implementation (OSDI 2006) Seattle, 2006, pp. 205–218

[9] K.L. Edward and A.T. Chandramohan, Petal: distributed virtual disks, in Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, Massachusetts, US, 1996, pp. 84-92.

[10] J.F. Gantz, C. Chute, A. Manfrediz, S. Minton, D. Reinsel, W. Schlichting and A. Toncheva, The Diverse and Exploding Digital Universe, March 2008. URL: http://www.emc.com/collateral/analystreports/diverseexplodin g-digital-universe.pdf, accessed in Oct 2011.

[11] X. Gao. L.P.B. Vuong and M. Zhang, Detecting Data Records in Semi-Structured Web Sites Based on Text Token Clustering, Integrated Computer-Aided Engineering, 2008, 15:4, pp. 297-311.

[12] S. Ghemawat, H. Gobioff and S.-T. Leung, The Google file system, in Proceedings of the nineteenth ACM symposium on Operating systems principles, ed. Bolton Landing, NY, USA: ACM, 2003, pp. 29-43.

[13] J.O. Gutierrez-Garcia and K.M. Sim, Agent-based cloud workflow execution, Integrated Computer-Aided Engineering, 2012, 19:1, pp. 39-56.

[14] B. Harrington, R. Brazile and K. Swigger, A Practical Method for Browsing a Relational Database using a Standard Search Engine, Integrated Computer-Aided Engineering, 2009, 16:3, pp. 211-223.

[15] V. Henson, An analysis of compare-by-hash, in Proceedings of the 9th conference on Hot Topics in Operating Systems Lihue, Hawaii, 2003, pp. 13-18.

[16] B. Hong and D.D.E. Long, Duplicate data elimination in a san file system, In Proceedings of the 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST), 2004, pp. 301-314.

[17] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise and P. Camble, Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality, in 7th USENIX Conference on File and Storage Technologies, San Francisco, California 2009, pp. 111-123

[18] A. Michael, F. Armando, G. Rean, D.J. Anthony, K. Randy, K. Andy, L. Gunho, P. David, R. Ariel, S. Ion and Z. Matei, Above the Clouds: A Berkeley View of Cloud Computing, 2009..URL:www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf, accessed in Oct 2011.

[19] A.E.M. Michael, V. William, I. Courtright, C. Chuck, R.G. Gregory, H. James, J.K. Andrew, M. Michael, P. Manish, S. Brandon, R.S. Raja, S. Shafeeq, D.S. John, T. Eno, W. Matthew. and J.W. Jay, Ursa minor: versatile cluster-based storage, in Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies, USENIX Association, 2005, 4, 59-72

[20] U. Reuter, A Fuzzy Approach for Modeling Non-stochastic Heterogeneous Data in Engineering Based on Cluster Analysis, Integrated Computer-Aided Engineering, 2011, 18:3, pp. 281-289.

[21] A.W Sage, W.L. Andrew, A.B. Scott and M. Carlos, RADOS: a scalable, reliable storage service for petabyte-scale storage clusters, in Proceedings of the 2nd International Workshop on Petascale Data Storage: held in conjunction with Supercomputing, Reno, Nevada, 2007, pp. 35-44.

[22] Q. Sean and D. Sean, Venti: A New Approach to Archival Data Storage, in Proceedings of the 1st USENIX Conference on File and Storage Technologies, ed. Monterey, CA: USENIX Association, 2002, pp. 89-101.

[23] Z. Sun, J. Shen and G. Beydoun, P2CP: A New Cloud Storage Model to Enhance Performance of Cloud Services, in Proceedings of International Conference on Information Resources Management, Conf-IRM, 2011, on CD-ROM

[24] Z. Sun, J. Shen and J. Yong, DeDu: Building a Deduplication Storage System over Cloud Computing. 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2011). pp 348 - 355

[25] J. Wei, H. Jiang, K. Zhou and D. Feng, MAD2: A scalable high-throughput exact deduplication approach for network backup services, in Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on Incline Village, NV, USA 2010 pp. 1-14

[26] S.A. Weil, S. A. Brandt, E.L. Miller, D.E.E Long and C. Maltzahn, Ceph: a scalable, high-performance distributed file system, in Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI), Seattle, Washington, 2006, pp. 307-320.

[27] S. Yasushi, F. Svend, V. Alistair, M. Arif and S. Susan, FAB: building distributed enterprise disk arrays from commodity components, in Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, MA, USA, 2004, pp. 48-58.

[28] F. Zhang, Z.M. Ma and L. Yan, Construction of Ontology from Object-oriented Database Model, Integrated Computer-Aided Engineering, 2011, 18:4, in press.

[29] 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 (Fast '08), 2008, pp. 269-282.