ADMAD: Application-Driven Metadata Aware De-duplication Archival Storage System

Chuanyi LIU [1, 2], Yingping Lu [2], Chunhui Shi [2], Guanlin Lu [2], David H.C. Du [2], Dong-Sheng WANG [1]
1(Department of Computer Science and Technology, Tsinghua University, Beijing, China)
2(Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA)
cy-liu04@mails.tsinghua.edu.cn; (lu, lv, du)@cs.umn.edu; wds@tsinghua.edu.cn

Abstract

There is a huge amount of duplicated or redundant data in current storage systems. So Data De-duplication, which uses lossless data compression schemes to minimize the duplicated data at the inter-file level, has been receiving broad attention in recent years. But there are still research challenges in current approaches and storage systems, such as: how to chunking the files more efficiently and better leverage potential similarity and identity among dedicated applications; how to store the chunks effectively and reliably into secondary storage devices. In this paper, we propose ADMAD: an Application-Driven Metadata Aware De-duplication Archival Storage System, which makes use of certain meta-data information of different levels in the I/O path to direct the file partitioning into more Meaningful data Chunks (MC) to maximally reduce the inter-file level duplications. However, the chunks may be with different lengths and variable sizes, storing them into storage devices may result in a lot of fragments and involve a high percentage of random disk accesses, which is very inefficient. Therefore, in ADMAD, chunks are further packaged into fixed sized Objects as the storage units to speed up the I/O performance as well as to ease the data management. Preliminary experiments have demonstrated that the proposed system can further reduce the required storage space when compared with current methods (from 20% to near 50% according to several datasets), and largely improves the writing performance (about 50%-70% in average).

1. Introduction and Motivations

It has been widely known that huge amount of duplicated or redundant data existing in current storage systems [1]. Not only data duplications exist among variants of the same file (e.g. backup files), it can also occur among different files. The huge amounts of data duplications result in extra storage spaces to be used and much more power consumptions, greatly lowering the storage utilization. They also impose extra burden on the data management.

Thus Data De-duplication has received a broad attention from both academia and industry. Data De-duplication refers to the approaches that use lossless data compression schemes to minimize the duplicated data at the inter-file level. This de-duplication can also help to reduce the amount of data sent over the network when backing up from application servers to storage servers.

To the extent of our best knowledge, almost all of the current de-duplication schemes work on the binary file level. That is, they consider the files to be archived as bit strings and use typical chunking methods, such as whole file chunking, fixed size dividing, or Hash functions (e.g. Rabin fingerprinting algorithm). Each file is firstly partitioned into non-overlapped chunks. Only one instance of the same chunks is actually stored in the secondary storage devices, and the others are referenced to by a pointer pointing to the location of this chunk.

There are many remaining research challenges for the current Data De-duplication storage systems, including:

(1) How to chunk the files more efficiently and better leverage potential similarity and identity from applications? For example, an HTML file consists of several semantic segments embraced by certain tags as <head> </head>, <title> </title>, <body> </body> and so on. When using current de-duplication methods, e.g. Rabin fingerprinting, to divide a file into chunks, a break point [3] may be within a tag, which will result in the loss of semantic information associated with the meaningful segments. What is worse, as most of the chunk boundaries are semantic meaningless, it will impose a further burden on the management and will reduce the efficiency of future file retrievals.

978-0-7695-3408-4/08 $25.00 © 2008 IEEE
DOI 10.1109/SNAPI.2008.11

(2) How to store the chunks more effectively and reliably into secondary storage devices? As the chunks may be of different lengths and variable sizes, storing them directly in storage devices will result in a lot of fragments and involve a high percentage of random disk accesses which is very inefficient [11]. Moreover, how to distribute these chunks, especially among large-scale distributed storage systems, is also a challenging problem. For example, if we just randomly distribute these chunks among storage devices, when we want to retrieve archived files by a query, the archival system should first index the relevant de-duplicated chunks which may spread randomly among the storage devices, reconstruct all the files one by one, then check if the content of the file matches the query. This process is expensive and usually degrades the performance of many upper layer applications [24].

In this paper, we propose ADMAD: an Application-Driven Metadata Aware De-duplication Archival Storage System. It exploits certain meta-data information from different levels in the I/O path to direct the file partitioning into more Meaningful data Chunks (MC for short in the following sections) that are variable-sized, self-identifying and self-describing logical units. These meta-data information of archived files may include: 1) **application metadata**, such as file type, file format, application software, etc., 2) **application or user tags**, such as the tags used to describe the characteristics of blogs, images or multimedia, and 3) **file system metadata**, such as directory entries, inode information of a file. Currently ADMAD uses the file type and file format as the meta-data information to direct the file partitioning besides the currently used cryptographic hash functions, such as MD5, SHA1, and chunking methods, such as Rabin fingerprinting.

While the main goal of ADMAD is to reduce the inter-file level duplications as much as possible, since the MCs may be of different lengths and variable sizes, as discussed above, storing them directly into storage devices is very inefficient. Thus, we have also designed an efficient way to store these MCs. We breakdown the MCs into fixed sized objects with suitable object size that better makes use of the performance characteristics of commodity secondary storage disk devices (The performance of a big sequential access is about one magnitude faster than the small access or random access requests). Note that an object may contain part of, a whole or several logical MCs. Preliminary experiments have demonstrated that the performance of ADMAD can be comparable with the current approaches.

The rest of this paper is organized as follows: Section 2 introduces related approaches used to de-duplicate the inter-file redundancies; the architecture of ADMAD as well as the chunking algorithm and the archival & restore protocol are presented in Section 3; Section 4 gives the evaluation results of compression ratio under some typical workloads and the performance of ADMAD, then compares them with current mainstream approaches; Section 5 draws the conclusion.

2. Related Work

Data de-duplication, which is also called Inter-file data compression, should be lossless compressions, i.e. no information is lost when a file is compressed and then uncompressed. Generally speaking, data de-duplication consists of two phases: the first phase is the file dividing, i.e. how to divide the files into chunks so as to reduce the inter-file duplication to the furthest possible extent; and the other part is data distribution and storing, i.e. how to store the chunks into storage devices.

So far there are two categories of file dividing approaches: delta encoding based and chunking based.

Chunking based approach partitions each file into a number of non-overlapping chunks and stores only unique chunks into storage devices. In terms of the chunk sizes and chunk boundaries, several algorithms have been used and proposed: the simplest one takes the whole file as a chunk, and calculates the hash of the whole file's content as the chunk identifier; CASPER [4] adopts a fixed-sized file dividing method, it divides each file into predefined fixed size chunks; LBFS [3] uses Rabin fingerprint algorithm to divide each file into variable sized chunks based on the statistic information, which can further reduce duplication.

Delta encoding based method generates the delta file given the source file (a.k.a reference file) and the target file. It is based on the resemblance detection between data objects and uses delta encoding to store only deltas instead of entire data objects [5]. The de-duplication is achieved since the size of delta file is usually much smaller than the target file. Delta encoding [6] is a widely used resemblance compression technique, which uses the first version of the file as the base, and stores the deltas from the base for subsequent versions of the same file. But this method requires explicit version relationships between files to determine which files are bases and which files are deltas; thus it is usually used in versioning file systems. Recently, there is another method named Fuzzy Block Matching [7] that originates from CASPER [4]. In this method,

every object has some features [14]. By comparing the features of different objects, the resemblance relationship can be determined. Then the method makes use of ECC (Error Correcting Code) to generate the difference of two objects, and stores the difference to reduce redundancy.

By now there are also some commercial systems based on de-duplication techniques, such as Data Domain [19], Centera [30] by EMC, PureDisk [31] by Symantec.

3. Overview of ADMAD

ADMAD is mainly designed for archival storage, where the archival processes can run in the background and be deployed on different application servers. In current mainstream disk-based archival systems, the response time should preferably be on-line or near-line when archived files are retrieved, which indicates that the distribution and organization of data in the storage devices is very critical. ADMAD is a distributed storage system, which fits for large-scale archival or disaster recovery deployments.

3.1. System Architecture Overview

The architecture of ADMAD is depicted in Figure 1. The system consists of four main components: Application Servers (AS), File Archival Servers (FAS), Metadata Servers (MDS) and Intelligent Storage Nodes (ISN). Application servers like email servers, multimedia servers are deployed and run to support associated applications. The main task of FAS is to archive the files into the storage servers, and it is the place where Application-Driven Metadata Aware De-duplication is implemented (FAS can also deployed onto ASes as background daemons). MDS takes charge of metadata management, metadata operations interactively with ASes, the deployment of security mechanisms, token management and the control of system-wide activities such as object allocation and migration among ISNs. ISNs are based on commodity elements such as off-the-shelf hardware, operating systems, file systems (we use EXT3 file system in our preliminary implementation). Distributed ISNs are interconnected with high speed storage area network. In our system, all the components are implemented in the user space in Linux operating system based on commodity hardware. In actual deployment, some of the components can be deployed in the same physical machine as long as resources and performance requirements permit.

Metadata Servers (MDS)

To avoid detracting the main aspect of this paper, we mainly focus on the typical functions in ADMAD: namespace management and metadata. There are three important data structures in MDS to support:

(1) *File_Attibutes_Table*, which is used to manage the file system namespace and to map from hierarchical pathnames of regular files used by AS to MC identifiers used by ISNs (from MC identifiers we can further get the mapping to the location of the object that holds the very MCs).

(2) *Object_Metadata_Table*, which holds the metadata of objects. The structure of an object stored on the ISN is shown in Figure 2.

(3) *Chunk_Object_Table*, which holds a Reverse Index [9] used for locating the object a MC belongs to.

In the current implementation, these tables are finally stored as MySQL [25] tables on disk. In order to improve performance, they are cached in memory as much as possible. We plan to use non-volatile RAM to support reliability in real system deployment.

In order to avoid the single point of failure, we use MySQL's replication mechanism in our primary system. MySQL replication is based on the master server keeping track of all changes to the databases (inserts, updates, deletes, etc.) in its binary logs [25]. Each slave server receives from the master the saved updates that the master has recorded in its binary log, so that the slave can execute the same updates on its copy of the data.

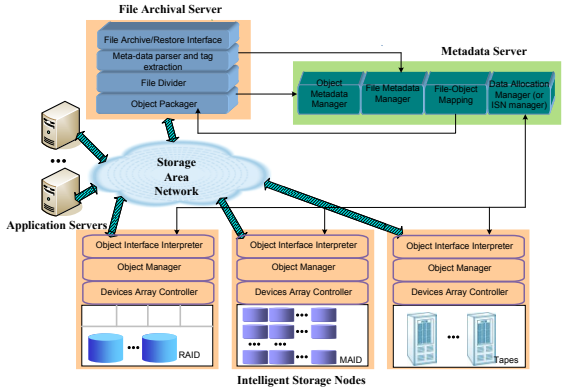

Figure 1. System Architecture of ADMAD

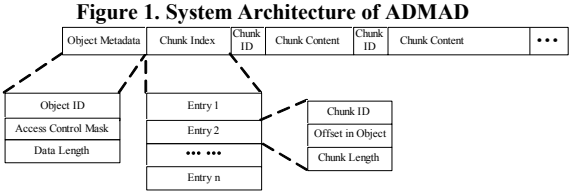

Figure 2. Structure of an object stored on the Intelligent Storage Nodes (ISN)

Intelligent Storage Nodes (ISN)

ISNs support a scalable, efficient and self-manageable object-based storage model, which is the prevailing system architecture for petabyte-scale storage systems nowadays [16], [17], [18].

ISN is compliant to OSD T-10[1] standard. It uses a flat namespace management. Every object is uniquely and globally identified by an object ID within the ISN. However, some prior research and practices [10] have demonstrated that using variable object size, especially separating the real data, metadata, and attributes of an object into three independent files of underlying file systems will severely hinder the performance, and dramatically increase the fragmentation within the storage devices, which further reduces the access bandwidth [11]. Considering the characteristics of archival data workload, we adopt a fixed size object scheme, and the object size can be configured based on different and some related research and practice [17]. The structure of an object is shown in Figure 2, in which chunk is the minimal access unit for application servers, specified by a chunk ID and the byte range in an object (Note that an object may contain part of, one or several logical MCs according to the size of the MCs).

As ISN is an autonomous system, it also can perform other functions such as access control, garbage collection, search optimization, etc., which are beyond the focus of this paper.

3.2. Chunking Library and File Archival API

As ADMAD uses metadata information of the archival files to direct the file dividing policy, "*Metadata parser and tag extraction*" module and "*File Divider*" module are dependent on the specific applications. We adopt application specific chunking libraries to implement chunking routines, and expose a unified chunking API for FAS and AS.

3.3. Chunking Routines

We choose several typical applications to demonstrate our system, including: (1) enterprise email system archival; (2) web documents archival and (3) multimedia data archival. In order to prove-of-concept, we simply use the file type and file format as the metadata information. Due to the space limitation, we only present the design for enterprise Email archival and Flash Video files archival.

3.3.1 Enterprise Email Archival

In the current implementation, Extmail [29], a widely used open source enterprise-level mail server, is used as the email application server. Every standard email file can be divided into 6 semantic parts (the MIME format email can also be divided into MCs using the same method), as: FROM address; TO address; SUBJECT; TEXT; ATTACHMENT FILENAME; and ATTACHMENT content. The chunking procedure is depicted in Figure 3.

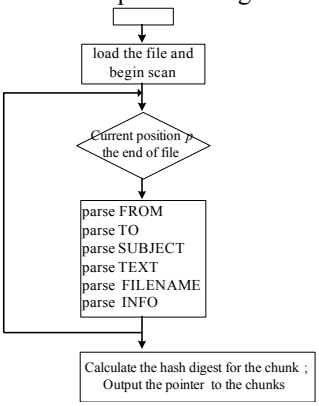

Figure 3. Chunking procedure for email files

3.3.2 Flash Video Archival

Flash Video (FLV) is a popular video format proposed by Adobe [32], and it is mostly used to deliver video over the Internet using Adobe Flash Player (formerly known as Macromedia Flash Player). As web 2.0 has increasing in a fast speed, archival of FLV videos becomes important for these web 2.0 websites, multimedia repositories and Internet proxies.

We currently use the FLV format [33] metadata. A typical FLV file consists of header part and body part, depicted in Figure 4. After the FLV header, the body part (the remainder of an FLV file) consists of alternating back-pointers and tags, which encode synchronized audio and video streams. Each tag type in an FLV file constitutes a single stream. There can be, at most, one audio and one video stream, synchronized together, in an FLV file. The detailed header and tag fields definition can be referred to in the FLV format specification [33].

Figure 4. The structure of a typical Flash Video file

[1] OSD T10 standard has been ratified by ANSI in January 2005 as SCSI Object-Based Storage Device command interface extension based on object representation of underlying storage. It provides an opportunity for the interested vendors/researchers to obtain hands-on experience of what OSD can provide, and can serve as a conformance point to test for interoperability when multiple OSD products arrive on the market.

Table 1. Comparison of compression ratio on Email archival system using different compress methods

Original Space (Megabytes)	Robin Fingerprint	Gzip	ADMAD
161	10.3	11.8	9

Based on the file format, we divide each FLV into several temporal MCs. The video segmentation procedure for a FLV file is described as follows:

(1) parse the file, and extract the time-indexed features of the frames

(2) sample the frames, and get the key frames for each temporal scene (here we mainly use the related segmentation algorithm and Shot boundary detection technique [34], [35])

(3) restore every key frame, and generate the image for each key frame

(4) generate a feature code for the image, and take this feature code as the identifier for the corresponding temporal scene

Suppose a FLV file 1, after the extraction of its 7 frames, sampling the key frames for the 3 scenes, and generating the feature codes for the 3 key frames, we divide file 1 into 3 MCs, and use the corresponding feature codes to identify the MCs. This procedure is illustrated in Figure 5.

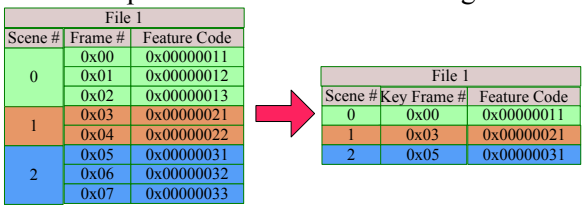

Figure 5. Illustration of dividing a FLV file into several scenes as the MCs

3.4. Archival Procedure

The archival procedure for ADMAD is as follows: every file to be archived is first divided into several MCs according to the algorithms in the application specific chunking library. While partitioning the file into MCs, these algorithms maintain the structures obtained from MDS. Then the MCs are packaged into objects. Note that objects are flushed from the buffer to storage devices periodically according to the flushing policy of the host Operating System. There can be an asynchronous completion message sent from the MDS to the archival server to indicate the completion of metadata operations, and there is also reply from ISN to indicate the actually storing of objects if configured in the synchronous mode.

4. Evaluation

4.1. Experimental Setup

In our implementation, iSCSI protocol [26] is used as IP-SAN transport protocol. The chunk IDs are generated using MD5 hash algorithm. The objects are internally stored as files on the underlying ext3 filesystem with the object size of 4MB. We take an experimental approach to compare it with the performance of two other typical traditional IP-network storage platforms: NFS [20] and iSCSI based regular file system [28]. We measure performance by a modified IOmeter [10]. The testbed is deployed to consist of one ISN and one archival server. A Sun Fire V40z server is used as the ISN equipped with 4 AMD dual-core processors at 2.4 GHz, 16 GB RAM, and 6 Ultra-320 SCSI disks (160 GB per disk). A SUN Workstation is used as the archival server equipped with 2 AMD dual-core processors at 1 GHz, 8 GB RAM, and a 200 GB Ultra-320 SCSI disk. The nodes are connected with 1 Gb/s independent Ethernet network.

4.2. Workload Characteristics

While by now it is difficult for us to get the real enterprise level dataset, we do collect some workloads which are general enough to represent the characteristics of the real massive data and to prove the concept. The workloads include: 5498.33MB of web documents; 6364MB of MP3 song files from Album Series Collections of several Chinese pop stars; 590MB of popular flash videos about the NBA games on a video website; 161MB of email files collected from personal archives of three colleagues in our research group

We use the Teleport™ software to download some set of news web pages from three web sites: http://www.sohu.com, http://www.sina.com.cn/, and http://www.tom.com/ with the same time period, and recursively download the pages linked from them, up to five levels. The MP3 files are collected from some ftp servers shared among some individuals internally who donate their MP3 collections to the ftp server. The FLV files are sets of NBA games collected from several web2.0 video websites within a month. Email files are collected from parts of personal archives of three colleagues in our research group, totaling to approximately 3000 email files.

4.3. Results and Discussions

The compression results of ADMAD compared with other methods (we use Gzip and Rabin fingerprint here) are described in Figure 6 and Table 1. Since the absolute compression ratios are highly dependent on the datasets used, e.g. the compression ratio on web documents may be one or two orders of magnitude larger due to the locality of encoded characters, while the traditional compressions on multimedia files are trivial or even worse, we calculate the relative compression ratios to represent the comparisons between different approaches at the same time to avoid the impact from different datasets. From Figure 6 we observe that, for MP3 file set, the compression ratio improves 61% compared with Gzip, and 43.9% compared with Rabin fingerprint. For flash video files, the compression ratio improves 46% compared with Gzip, and 27% compared with Rabin fingerprint. For web HTML files, the compression ratio improves 48% compared with Rabin fingerprint. And for the Email files, the compression ratio improvement is 31% and 14.3% respectively.

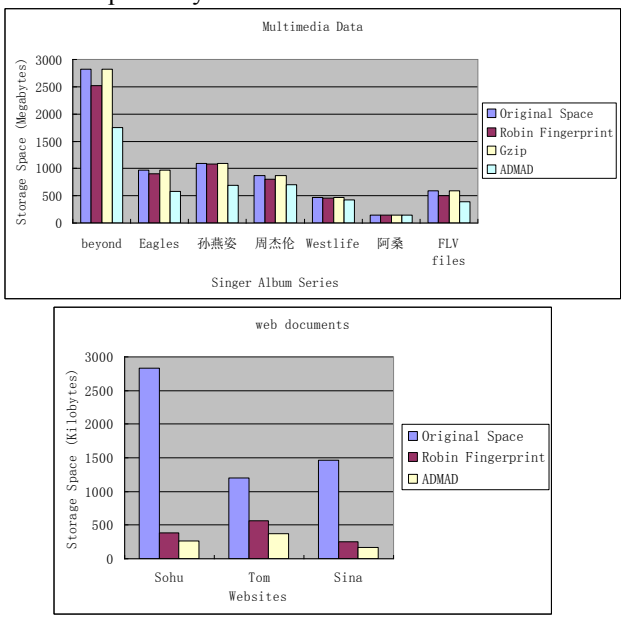

Figure 6. Comparison of compression ratio using different compress methods

Figure 7 shows the bandwidth of read/write operations for the three systems. It can be seen that the read bandwidth of ADMAD is relatively higher than in other systems when the data size is small (below about 400KB). This is because when deploying traditional file systems, the inode blocks are separate from the content blocks, which imposes random device access overhead. When the data size is above 400KB, the read bandwidth of ADMAD is comparable to others except for iSCSI. This is because in ADMAD, the objects of a file may be spread into different locations of a device, which imposes additional disk head seek and rotational time. As for the write bandwidth, we can see that ADMAD is much better than others, improving about 50%-70%. This is because the storage unit in ADMAD is object, with size of 4MB in our test configuration which is much bigger than the mainstream block size of 512B. Writing an object continuously is more efficient than writing many blocks with no guarantee of sequential distribution in device [11]. As ADMAD is on top of a file system, it can take the advantages of the underlying buffering and caching mechanisms of the file system transparently.

5. Conclusion

This paper introduces ADMAD: an Application-Driven Metadata Aware De-duplication Archival Storage System, which leverages the metadata information of different levels in the I/O path to guide the dividing of each file into more Meaningful data Chunks (MC). We also design the fixed size object based storage scheme to store the MCs into devices. In our current implementation, we only use file type and file format as the metadata information. Preliminary experiments on the collected representative workloads of Email, HTML, MP3, FLV files show that ADMAD has better compression ratio than some of the current mainstream methods (from 20% to near 50% according to different datasets), and the read bandwidth is comparable with similar systems while the write bandwidth is better than the other systems (about 50%-70% in average).

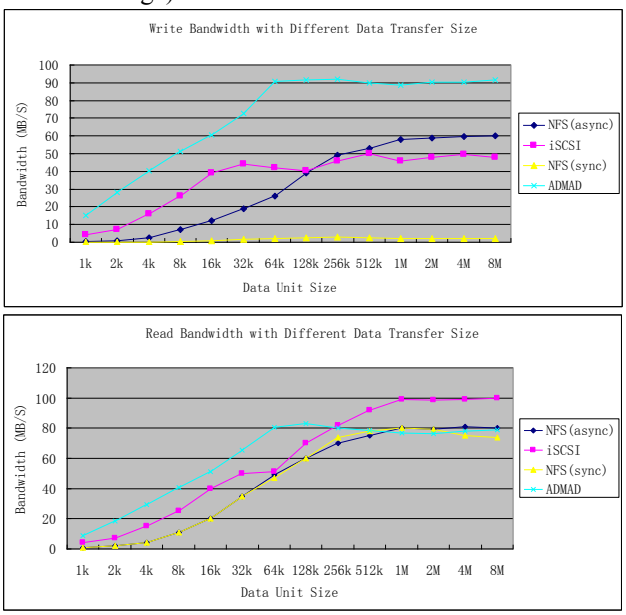

Figure 7. Throughput in each of three systems in 1 Gb/s IP-SAN network

6. References

[1] J F Gantz, et al. The Expanding Digital Universe: A Forecast of Worldwide Information Growth through 2010. IDC, March 2007.

[2] M. Mesnier, G. R. Ganger, and E. Riedel, Object-based storage, IEEE Communications Magazine, Aug. 2003.

[3] A. Muthitacharoen, B. Chen, D. Mazières, and A. Rawat. A low bandwidth network file system, SOSP 2001.

[4] N. Tolia, M. Kozuch, M. Satyanarayanan, et al. Opportunistic Use of Content Addressable Storage for Distributed File Systems, In Proc. of Usenix 2003 Annual Technical Conference, San Antonio, TX, USA

[5] L. L. You and C. Karamanolis, Evaluation of Efficient Archival Storage Techniques, 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies, April 2004.

[6] M. Ajtai, R. Burns, R. Fagin, D. D. E. Long, and L. Stockmeyer, Compactly encoding unstructured inputs with differential compression, Journal of the ACM, 49(3):318–367, May 2002.

[7] B. Han and P. Keleher, Implementation and Performance Evaluation of Fuzzy File Block Matching, 2007 USENIX Annual Technical Conference, Santa Clara, CA, June 2007

[8] B. V. Rompay, On the security of dedicated hash functions, In the 19th Symposium on Information Theory in the Benelux, 1998

[9] S. Brin and L. Page, The anatomy of a large-scale hypertextual web search engine, In WWW Conference, volume 7, 1998.

[10] David Du, Dingshan He, et. al. "Experiences in Building an Object-Based Storage System based on the OSD T-10 Standard," Submitted to 14th NASA Goddard — 23rd IEEE (MSST2006) Conference on Mass Storage Systems and Technologies May, 2006.

[11] M. Rosenblum and J. K. Ousterhout, The design and implementation of a log-structured file system, ACM Transactions on Computer Systems (TOCS), 10(1), Feb. 1992.

[12] M. McMinnXAM, Architecture Specification (Working Draft) of Storage Networking Industry Association. Aug. 2007, http://www.snia-dmf.org/xam/index.shtml

[13] M. O. Rabin, Fingerprinting by random polynomials, Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[14] A. Z. Broder, S. C. Glassman, M. S. Manasse, G. Zweig. Syntactic Clustering of the Web, Digital Equipment Corporation Systems Research Center (SRC) Technical Note 1997-015, July 1997

[15] P. Lyman, H R Varian, et al. How much information? http://www.sims.berkeley.edu/research/projects/how-much-info-2003/, Oct. 2003.

[16] Storage Networking Solutions – Europe. Object Storage Architecture: Defining a new generation of storage systems built on distributed, intelligent storage devices, Oct. 2004, http:// www.snseurope.com/featuresfull.php?id=2193.

[17] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," Proc. SOSP'03, 2003.

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

[19] Benjamin Zhu; Kai Li; Hugo Patterson. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System. Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST '08), February 2008, Pages (269–282)

[20] R Sandberg, D Goldberg, S Kleiman, et al. Design and Implementation of the Sun Network File System. In Proceedings of the summer 1985 USENIX conference, June 1985, Pages:(119-130).

[21] Intel Server Architecture Lab, IOmeter: The I/O Performance Analysis Tool for Servers. http://www.intel.com/design/servers/devtools/iometer/index.htm

[22] HTML 5: A vocabulary and associated APIs for HTML and XHTML. W3C Working Draft 22 January 2008. Accessed from http://www.w3.org/TR/html5/

[23] D. Bhagwat, K. Pollack, D. D. E. Long, T. Schwarz, and E. L. Miller, Providing High Reliability in a Minimum Redundancy Archival Storage System, Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '06), Sep 2006

[24] B. Zhu, K. Li, and H. Patterson, Avoiding the Disk Bottleneck in the Data Domain Deduplication File System, Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST '08), February 2008, Page(s): 269-282

[25] MySQL. http://www.mysql.com.

[26] Meth, K.Z.; Satran, J. Design of the iSCSI protocol, Mass Storage Systems and Technologies, 2003, April 2003, Page(s):116 – 122

[27] Intel's Open Storage Toolkit, Intel iSCSI reference implementation, Sourceforge.net.

[28] Y. Lu and D. Du, Performance Study of iSCSI-Based Storage Subsystems, IEEE Communications Magazine, Aug. 2003.

[29] Extmail website. http://www.extmail.org/

[30] EMC Centera. Content Addressed Storage, product description. http://www.emc.com/pdf/products/centera/centera guide.pdf. 2002

[31] Symantec Veritas NetBackup PureDisk Product Overview, http://www.symantec.com/business/products/overview.jsp?pcid=2244&pvid=1381_1

[32] Flash Video Introduction, From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Flash_video

[33] Adobe Player Licensing. Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification. Accessed in http://www.adobe.com/licensing/developer/

[34] Ramin Zabih; Justin Miller; Kevin Mai. A feature-based algorithm for detecting and classifying scene breaks. Proceedings of the third ACM international conference on Multimedia, San Francisco, California, United States, Pages: 189 - 200, 1995

[35] Silvio Jamil Ferzoli Guimares, Michel Couprie, Arnaldo de Albuquerque Araújo, Neucimar Jerffnimo Leite; Video segmentation based on 2D image analysis; Pattern Recognition Letters, v.24 n.7, p.947-957, April 2003