PRUN : Eliminating Information Redundancy for Large Scale Data Backup System

Youjip Won[1] Rakie Kim[1] Jongmyeong Ban[1] Jungpil Hur[2] Sangkyu Oh[2] Jangsun Lee[2]

[1]*Department of Electronics and Computer Engineering*
Hanyang University, Seoul, Korea

[2]*Macro Impact, Seoul, Korea*

Abstract

In this work, we develop novel backup system, PRUN, for massive scale data storage. PRUN aims at improving the backup latency and storage overhead of backup via effectively eliminating information redundancy in the files. PRUN eliminates intra-file and inter-file information redundancy. PRUN consists of client module and server module. PRUN consists of three key technical ingredients: redundancy detection, fingerprint manager, and chunk manager. File chunking for redundancy detection is the most time consuming task in backup. For efficient file chunking, we develop incremental modulo-K algorithm which enables us to improve the file chunking time significantly. We perform various experiment to measure the overhead of each tasks in backup operation and to examine the efficiency of redundancy elimination. Incremental modulo-K reduces the file chunking latency by approximately 60%. Redundancy elimination scheme can reduce the storage requirement of backup by 80% when we backup different minor versions of Linux 2.6 kernel source.

1. Introduction

1.1. Motivation

Due to the advancement of storage technology, input/out device technology and computer technology, larger fraction of data is now being maintained in digitized form. The amount of digitized data increases by factor of 10 in every three years. The type of data ranges from business data, e.g. medical record, financial record, and legal data to personal one(home-made video, phone books and etc.). Most of these data cannot afford to be got lost due to its financial, legal or sentimental value. Recent rapid advancement in video/image compression technology, network technology and computer technology enables the public deployment of internet based TV broadcasting service. It includes the service through wired(IPTV[1]) as well as wireless(DMB[2]) network. Also, recent wave of web 2.0 technology makes the content creation extremely easy and further accelerates the content creation speed. Taking a video with mobile phone and to upload it to UCC web site is now a part of our everyday life in all age spectrum. What all lies in this mega trend are that larger and larger fraction of information is being maintained in the digitized form.

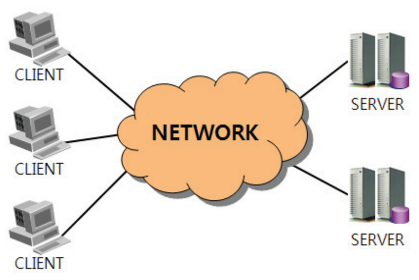
In this work, we develop state of art backup framework which effectively exploits the inter-file and intra-file information redundancy. We aim at reducing the time and space requirement of backup operation via effectively eliminating the information redundancy. The contribution of this work is three folds. First, we develop efficient file chunking algorithm and signature generation algorithm. Second, we develop efficient chunk synchronization mechanism between backup server and backup client. Via maintaining fingerprint information in both client and server we minimize the amount of backup data transferred between the backup client and the backup server. Third, we develop data efficient structure for backup and restore process.

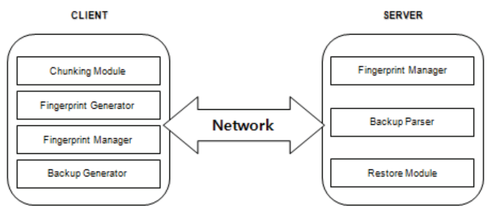
1.2. Related Works

A variety of researches to improve functions have been conducted as the importance of backup is increasing[3]. The previous research has focused on improving the function of the equipment[4, 5] to perform backup and developing a new Storage Management Technique. However, a new technique to reduce waste of Network Bandwidth and Storage has become necessary according to increasing backup data. The research to reduce Network Bandwidth has been performed in the field of Archival Storage[6] and Network File System like AFS[7], Leases[8] and NFS[9] from the past.

There have been a number of efforts to eliminate redundancy in file system, backup system and archival system. Venti[10] and Single Instance Store[11] are archival system to eliminate redundancy data. They use fixed size chunk in partitioning a file. Elephant File System[12] manages version of each file through redundancy check. SAN File System[13] is a file system to share the duplicate content among files. LBFS[14] and Pastiche[15] adopt Variable Sized Chunking in partitioning the file into chunks. Generating a signature for a file is important topics in file system content management[16], similarity detection and copy protection[17-19].

Rabin's fingerprint algorithm[20] is the most widely used algorithm for this purpose. Objective of these files systems are to reduce the storage overhead. A number of distributed file system attempts to reduce the remote I/O overhead via synchronizing cache contents between the server and the client[21]. During the past few years, a number of de-duplication based software has been introduced. They are for email[22], web document[23] or generic file system[24].





2. PRUN System Organization

Two main ingredients of PRUN are the components for generating backup and the components for restoring original file. PRUN is designed for distributed environment where backups are located at remote site. We use the term client and server of backup to denote the location of original data to be backed up and the location of backup files, respectively. Fig. 1 illustrates the topology of client and server.

There are four modules in the client side: chunking module, fingerprint generator, fingerprint manager, backup generator. Chunking module partitions a file into a number of chunks. We use sliding window approach in chunking the file. Different from Basic Sliding Windows algorithm, we develop more efficient signature generation algorithm called incremental modulo-K.

Fingerprint generator generates fingerprint for each chunk. We can use any hashing function, e.g. MD5(120bit), SHA-1(160bit) and SHA-256(256bit). With larger size fingerprint, collision probability decreases, but it requires more time to generate fingerprint. Currently, we are using SHA-1 algorithm for generating fingerprint.

Fingerprint manager manages fingerprints. Chunk size ranges from 2Kbytes to 64Kbytes. For 30Tbytes file system, total size of SHA-1 fingerprint can be a few hundred Gbytes. Fingerprint manager is responsible for "insert", "delete" and "search" fingerprints value. We can use simple index structure, e.g. B+Tree, or DBMS to manage fingerprints. Fingerprint manager of PRUN is designed to be compatible with a number of commodity DBMS. MySQL , BerkeleyDB, SQLite and B+Tree. Fingerprint database reside both at the client and the server.

In client, there exists Backup generator. Backup generator work in two mode: local backup and remote backup. In case of local backup, backup generator create a backup file on local storage. In remote backup mode, backup file generator transfers backup in streaming fashion.

Backup parser resides in the backup server. It performs a critical role of receiving incoming stream of backup file, parsing it, and storing it at proper data structure. Backup file consists of a number of different type data: backup header, file header, chunk header, chunk data and etc. Backup parser at the server receives incoming byte stream, assembles them into original data structure, and passes them to appropriate management module.

3. De-duplication

3.1. Chunking : Partitioning a File

Process of eliminating redundancy is also called "De-duplication". The objective of de-duplication is to reduce space requirement and time requirement for generating backup. De-duplication consists of three components: (i)chunking, (ii)fingerprint generation and (iii)detection of redundancy. Chunking is a process of partition a file into a number of chunks so that we can examine if a chunk appears multiple times in a file or across the file. There are two types of chunking approach: fixed size chunking and variable size chunking. In fixed size chunking, file is partitioned into fixed size units, e.g. 8Kbyte block. Fixed size chunking is conceptually simple and is fast. However, fixed size chunking has an important drawback. When small amount of data is inserted into a file or deleted from a file, file chunking yields totally different results after insertion(or deletion) even though most of the file contents remain unchanged.

If small data is inserted at the beginning of the file. Since fixed size chunking partitions a file into fixed size blocks, file chunking generates entirely different set of chunks in this case. Variable size chunking which is also known as content based chunking has been proposed to effectively address this problem. In variable size chunking, chunking boundary is determined based upon the content of a chunk, not upon the offset from the beginning of the file. Therefore, after inserting new data into a file, most of the chunks remain unaffected.

We use basic sliding window algorithm for chunking. Basic sliding window algorithm has three key ingredients: windows size, signature size, and signature generation algorithm. In basic sliding window protocol, it slides the window from the beginning of a file one byte at a time. When the content in the window satisfies a certain constraints, chunk boundary is set at the end of the window. Window size is tens of byte. In our experiment window size is 48Bytes as in LBFS. It is practically infeasible to perform 48 Bytes comparison for each window. Remind that window is shifted one byte at a time. For comparison efficiency, we generate signature on the window. If the signature satisfies predefined bit pattern, the ending position of a window is set as the end of a chunk. Otherwise, window is shifted by one byte and signature is generated again. There are a number of algorithms for generating signature for a given window. Most of the existing works use Rabin's fingerprint algorithm. Rabin's fingerprint algorithm is widely used in similarity search, content fingerprinting, peer-to-peer file system and etc. In this work, we develop novel signature generation algorithm, incremental Modulo-K.

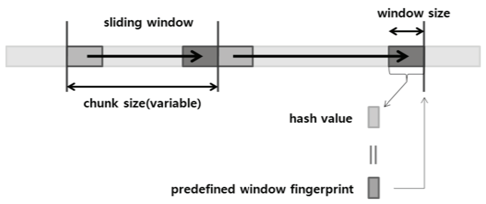


Figure 3. Basic Sliding Window Algorithm

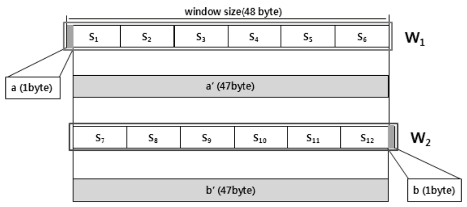


Figure 4. Modulo-K algorithm

In this work, we develop more novel signature generation algorithm, incremental Modulo-K to accelerate the signature generation algorithm. 48Bytes window consists of six one-byte value. In Fig. 5, $S_1S_2 \ldots S_6$ and $S_7S_8 \ldots S_{12}$ denotes each of these bytes. Original 48Bytes bit pattern W_1 can be represented as $S_1S_2 \ldots S_6$ and $W_2(S_7S_8 \ldots S_{12})$ means 1byte shifted from W_1.

We use shift operation to minimize the computational overhead. Applying modulo arithmetic for every 40 byte is not necessary. We perform modulo arithmetic in incremental fashion. Let % be the operator for modulo arithmetic. We can represent $W_1 \% K$ for a certain key value K as in Eq. 1. To minimize key collision, it is very important to use right value for K. Usually, prime number is used. In this work we use $K=2^{31}-1$.

$$\begin{aligned} W_1 \,\%\, K &= (S_1S_2S_3S_4S_5S_6) \,\%\, K \\ &= (2^{10} \times S_1 \,\%\, K + 2^8 \times S_2 \,\%\, K + 2^6 \times S_3 \,\%\, K \\ &\quad + 2^4 \times S_4 \,\%\, K + 2^2 \times S_5 \,\%\, K + S_6 \,\%\, K) \,\%\, K \end{aligned} \tag{1}$$

After shifting a byte, modulo arithmetic for new 48 byte can be presented as in Eq. 2 using the pre-computed value in Eq. 1.

$$\begin{aligned} W_2 \,\%\, K &= (S_7S_8S_9S_{10}S_{11}S_{12}) \,\%\, K \\ &= (b' \,\%\, K + b \,\%\, K) \,\%\, K \\ &= ((a' \times 2^8) \,\%\, K + b \,\%\, K) \,\%\, K \\ &= (((K+ (W_1\%K) - a\times 2^4) \times 2^8) \,\%\, K + b\%K)\% K \end{aligned} \tag{2}$$

When the result of modulo-K values are uniformly distributed, the probability that a given window satisfies predefined pattern corresponds to 1/K.

Therefore, to obtain average 8Kbyte chunk size, we need 13 bit signature size. It turns out that reality is different from theory. To make average chunk size 8KBytes, we needed 20bit signature.

3.2. Issue in incremental Modulo-K

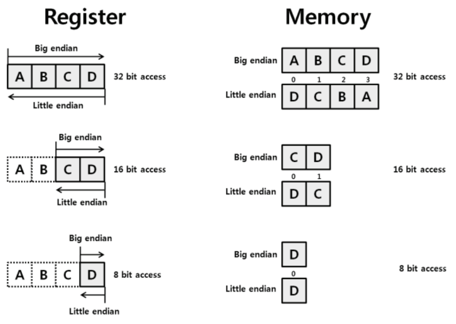


Figure 5. Big-endian and Little-endian

Existing processor has a two type of endian. Well known processor architectures that use the little-endian format include x86, 6502, Z80, and, largely, PDP-11, Motorola processors as the 6800 and 68000 have generally used big-endian.

Fig. 5 shows difference of between big-endian and little-endian. Modulo-K has a problem when using little-endian processors because basic algorithm of Modulo-K is based on big-endian. Therefore, we use *'htonll'* function that generally uses network application to arrange byte order from little-endian to big-endian. But using this function makes some overhead.

We only arrange data when first operation(Eq.1). First operation is called less frequently than second operation(Eq.2) and averagely, this ratio is 1:6000(average chunk size is 8Kbytes). Though Modulo-K has some overhead when use little-endian processor, Modulo-K still simple and fast algorithm than Rabin fingerprint. If using Modulo-K Algorithm, it is possible to chunk mass files at a much greater pace than Rabin fingerprint. The difference of an efficiency of Rabin and Modulo-K is explained in Section 5.1.

4. Sharing chunks

PRUN maintains three key data structure: 'Header File', 'Fingerprint Table', 'Chunk Files'.

Header file is basically a backup file. The only difference between client generated backup file and Header file in the server is the existence of chunk data. PRUN backup server removes chunk data from the backup file transmitted by the client. Backup server

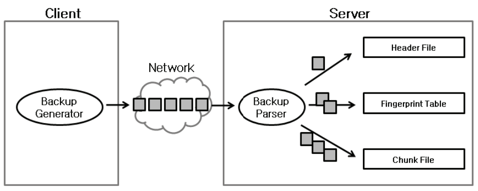


Figure 6. Process of Backup Parser

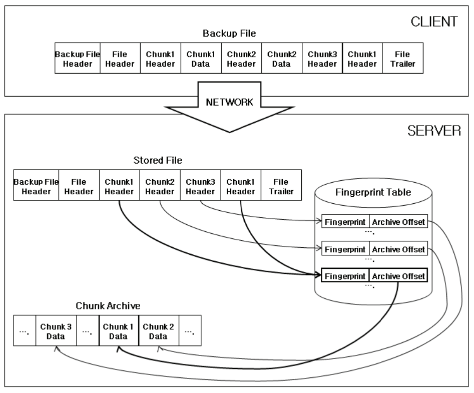


Figure 7. Layout of PRUN Server

stores chunk data in separate structure, which is to be explained in detail shortly. This Header File is later to be used to restore original information. Server also maintains fingerprint table. Fingerprint table is a collection of fingerprint of the chunks stored at the server. The objective of server side fingerprint table is slightly different from that of client's. Client side fingerprint table is primarily used to detect redundancy, and therefore it contains only fingerprints. Server side fingerprint table is used to restore the original information. It contains the fingerprint and the location of the respective chunk data.

Location of chunk file is specified by <chunk file id, offset>. PRUN maintains chunk data in the chunk file. When new chunk data arrives, it is appended at the end of the current chunk file. When current chunk file size reaches predefined limit, it is closed and the new chunk file is created. Any subsequent chunks will be appended at the new chunk file. Currently, maximum chunk file size is 2Gbyte.

In the backup server, there exist parser which assembles the incoming stream of backup data into Header file, Fingerprint table, and Chunk file component.

Restoring a backup is quite straight forward. Client issues a request for restoration. Server locates the respective Header File. Header file contains the fingerprint of the chunks. Backup server consults the fingerprint table for location of the respective chunk data, and restores original backup file, subsequently.

5. Experiment

We develop prototype of PRUN and perform a number of experiment. PRUN is developed as user-level application on Linux 2.6 platform. We implemented PRUN on 2.66GHZ Intel processor with 2GBytes of RAM with 320Gbytes SATA-2 Hard-disk drive(7200 RAM, Samsung).

5.1. Overhead of Detecting Redundancy

Detecting redundancy consists of four steps: file reading, chunking, Fingerprint generation and fingerprint table operation(search and insert). We examine the overhead of individual step. In this work, we develop efficient signature generation algorithm called incremental Modulo-K. We compare the performance of incremental Modulo-K and Rabin's fingerprint algorithm. We use SHA-1 to generate fingerprint for each chunk. We examine the time to detect redundancy in different size files: 32, 64, 128, 256 and 512Mbytes.

Fig. 8 illustrates the result. Hash table time denotes the time spent on fingerprint table operation. In all experiment, "chunk time" takes up dominant fraction of time (70~80% of total elapsed time). Incremental Modulo-K algorithm is approximately 40% faster than Rabin's fingerprint algorithm. Via using novel signature generation algorithm, PRUN is able to reduce the overall latency significantly.

5.2. The Effectiveness of Redundancy Check

We examine the effectiveness of redundancy check capability of PRUN. The ratio that Redundancy detection happens is proportion to data stored at Fingerprint Table. In other words, the rate of redundancy check of the specific backup data is determined by only chunk content that are backed up previously. Therefore, in this experiment, the test is proceeded after involving random redundancy data to each backup. Mass multimedia files are used as subjects and backup has been performed 10 times. For each dataset, we accumulate files.

In Fig. 9, when backing up firstly, PRUN makes a little bigger backup file than Tar Archive. Tar includes only 1 file header every files in order to express additional information but the capacity of PRUN is a little bigger than Tar because of a needed chunk header for redundancy check every chunks. However, from the second backup, backup file of PRUN is smaller as

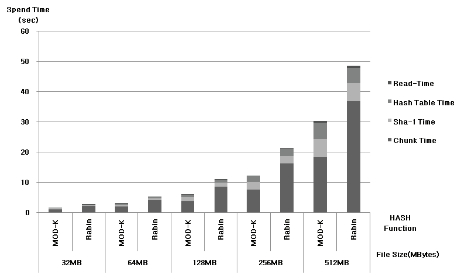


Figure 8. The performance of Modulo-K

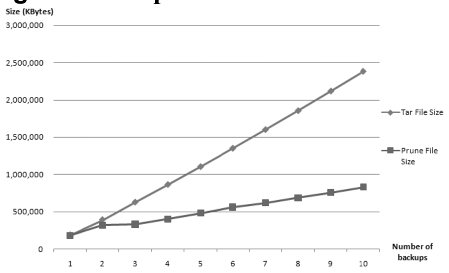


Figure 9. Backup capacity of multimedia files

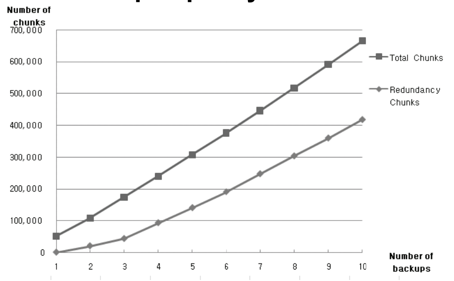


Figure 10. Capacity variation in backing up Linux kernel source code

redundancy data occur. At the last 10 backup, the capacity of backup file makes a great difference. The difference of the capacity as Tar backs up redundancy data continuously but PRUN backs up only new data. This tells that PRUN performs redundancy check normally. It is no wonder that PRUN can check redundancy check though they are not mass files.

Fig. 10 is a graph of capacity variation that backs up an updated each version of Linux Kernel 2.6 source code. When backup is performed, searching for only variation part from the second backup. Therefore, the capacity can be lessened. These results suggest that PRUN is very efficient of backing up files by the version.

6. Conclusion

In this work, we have successfully developed state of art backup system, PRUN, for massive scale data storage. Modern storage system harbors significant amount of data redundancy. Email attachments, copy and paste of UCC contents, verbatim copy of blog data are typical source of information redundancy. Same contents appear multiple times in a file or across different files. We focus our effort on eliminating information

We develop efficient signature generating algorithm and significantly improve file chunking overhead. Via maintaining fingerprint table in both client and server of backup, we reduce the network traffic.

PRUN is designed for scalability. It can use in-house B+ tree based fingerprint table. Also, it can use a number of commodity DBMS: SQLite, BerkeleyDB, MySQL as its storage manager. Subject to the size of information and user requirement, PRUN can flexibly choose appropriate storage management scheme. We perform various performance experiments. PRUN improve the signature generation overhead by 60% against Rabin's fingerprint algorithm. Via effectively eliminating redundancy, PRUN are able to significantly reduce the volume of backup.

8. References

[1] Y. WANG, W. SHU, and M. HER, "Internet Protocol Television," 2006.

[2] "Digital Multimedia Broadcasting," Telecommunications Technology Association, 2003.

[3] A. L. Chervenak, V. Vellanki, and Z. Kurmas, "Protecting file systems: A survey of backup techniques," in *Proceedings of the Joint NASA and IEEE Mass Storage Conference*, 1998.

[4] E. Melski, "Burt: the backup and recovery tool," in *Proceedings of LISA'99*, 1999.

[5] W. C. Preston, "Using Gigabit Ethernet to backup six Terabytes," in *LISA '98: Proceedings of the 12th Conference on Systems Administration,* 1998.

[6] A. Crespo and H. Garcia-Molina., "Archival storage for digital libraries. ," in *Arturo Crespo and Hector Garcia-Molina*, 1998.

[7] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West., "Scale and performance in a distributed file system.," *ACM Transactions on Computer Systems,* 1988.

[8] C. G. Gray and D. R. Cheriton, "Leases: An efficient fault-tolerant mechanism for distributed file cache consistency," in *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, 1989.

[9] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck., "NFS version 4 protocol," in *RFC 3010, Network Working Group*, 2000.

[10] Q. Sean and D. Sean, "Venti: A New Approach to Archival Storage," in *Proceedings of the Conference on File and Storage Technologies*: USENIX Association, 2002.

[11] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur., "Single instance storage in Windows 2000.," in *Proceedings of the 4th USENIX Windows Systems Symposium*, Seattle, WA, 2000, pp. 13–24.

[12] D. S. Santry, M. J. Feeley, N. C. Hutchinson, A. C. Veitch, R. W. Carton, and J. Ofir., "Deciding when to forget in the Elephant file system," in *Proceedings of the 17th Symposium on Operating Systems Principles*, 1999.

[13] B. Hong, D. Plantenberg, D. D. E. Long, and Sivan-Zimet, "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.

[14] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in *Proceedings of the eighteenth ACM symposium on Operating systems principles* Banff, Alberta, Canada: ACM Press, 2001.

[15] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: Making backup cheap and easy," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, 2002.

[16] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, T. Bressoud, and A. Perrig, "Opportunistic Use of Content Addressable Storage for Distributed File Systems," in *USENIX 2003 Annual Technical Conference*, 2003.

[17] S. Brin, J. Davis, and H. Garcia-Molina, "Copy detection mechanisms for digital documents," in *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, 1995, pp. 398–409.

[18] F. George, E. Kave, and C. Stephane, "Finding similar files in large document repositories," in *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining* Chicago, Illinois, USA: ACM Press, 2005.

[19] A. Z. Broder, "Identifying and Filtering Near-Duplicate Documents," in *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, 2000,

[20] M. O. Rabin, "Fingerprinting by Random Polynomials," Tech. Rep. TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[21] R. B. Deepak, J. Suresh, and D. Cezary, "Improving duplicate elimination in storage systems," *ACM Transactions on Storage* vol. 2, pp. 424-448, 2006.

[22] T. Avishay, J. Nikolai, S. Josef, and Z. Erez, "Using free web storage for data backup," in *Proceedings of the second ACM workshop on Storage security and survivability* Alexandria, Virginia, USA: ACM, 2006.

[23] D. Gomes, A. L. Santos, and M. J. Silva, "Managing duplicates in a web archive," in *Proceedings of the 2006 ACM symposium on Applied computing* Dijon, France: ACM, 2006, pp. 818-825.

[24] F. Douglis, J. LaVoie, and J. M. Tracey, "Redundancy Elimination within Large Collections of Files," in *USENIX Annual Technical Conference*, 2004.