Efficient index lookup for De-duplication backup system*

Youjip Won, Jongmyeong Ban, Jaehong Min
Department of Electronics and Computer Engineering
Hanyang University, Seoul, Korea

Jungpil Hur, Sangkyu Oh, Jangsun Lee
MacroImpact, Seoul, Korea

Abstract

We minimizes fingerprint management overhead(index lookup and index insert) via introducing main memory index lookup structure and workload-aware index partitioning of the index file in the storage. Backup server maintains three data structures for redundancy elimination: Header files, chunk files and fingerprint tables. These data structures altogether enables PRUNE to effectively eliminate redundancy and to perform efficient backup. 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%. With filter based in-memory index data structure and index partitioning, PRUNE eliminates 99.4% of disk accesses involved in fingerprint management.

1. Introduction

In this work, we develop state of art backup framework, PRUNE, which effectively exploits the inter-file and intra-file information redundancy. We adopt filter based main memory index lookup structure [1, 10]. To minimize the restructuring overhead of on-disk index and to improve the buffer cache miss rate, we develop an elaborate index partitioning method where we partition the large index file into a number of hash based index files. With this technique, PRUNE eliminate 99.4% of disk access involved in fingerprint management.

To reduce waste of storage, there have been a number of efforts to eliminate redundancy in file system, backup and archival system. Venti [8] and Single Instance Storage(SIS) [2] are archival system to eliminate redundancy data. SAN File System [5] is a file system to share the duplicate content among files. Venti and SAN file system use fixed size blocks in partitioning a file. SIS detect duplicate data on the file level. LBFS [7] can reduce both network traffic and wasting storage space. LBFS and Pastiche [3] divide files into variable sized chunk and hashes those to find. During the past few years, a number of de-duplication based software have been introduced. They are for email [9], web document [4] or generic file system [6, 10].

*This research is in part supported by KOSEF through National Research Lab (R0A - 2007 - 000 - 20114 - 0) at Hanyang University

2. Sharing Chunks

2.1. Structure of Fingerprint Table

In PRUNE, fingerprint table resides in both client and server. Server maintains fingerprint table for each client. The roles of fingerprint table in the client and the server are slightly different. The record structure of fingerprint for client and server are tailored to serve its own purpose. Fingerprint table in the client contains only ”fingerprint” values. Primary purpose of fingerprint table at the client is to detect redundancy. If a fingerprint value already exists in fingerprint table, the respective chunk data is consider as redundant and chunk header state is set to ”Redundant”. Server side fingerprint table carries more information. An entry in server side fingerprint table consists of fingerprint, location of the respective chunk data, and reference count. Server maintains a set of chunk data as a file. When new chunk data arrives from the client,

Table 1. Fingerprint Table in Client Side

fieldname	type	description
first	bigint(64)	First of fingerprint - 64bits
second	bigint(64)	Middle of fingerprint - 64bits
third	int(32)	Last of fingerprint - 32bits

Table 2. Fingerprint Table in Server Side

fieldname	type	description
first	bigint(64)	First of fingerprint - 64bits
second	bigint(64)	Middle of fingerprint - 64bits
third	int(32)	Last of fingerprint - 32bits
filename	char(30)	Stored filename
offset	int(32)	Chunk offset in stored filename

it is appended at the current chunk archive file. When current chunk archive reaches predefined threshold, server open a new chunk archive. Each chunk archive is currently implemented as a single file. Location of chunk data is $< file, offset >$ pairs, where file filed denotes path and file name of the chunk archive where the respective chunk data resides. Serve side fingerprint table is primarily used to locate the chunk data for a given fingerprint. We presently assume that fingerprints in the client and the fingerprint at the server are of the same.

2.2. Eliminating Redundancy

To eliminate redundancy, backup system needs to provide two features: redundancy checking mechanism and the mechanism to share chunks. We develop elaborate data structure to realize this feature. Client maintains fingerprint table. When new chunk is generated, backup system consults this table for redundancy check.

To effectively address this issue, we use filter based index lookup structure [10]. Also, to enforce the locality in fingerprint generation, we partition the large index file into a number of small index files so that fingerprints in the same index file is more likely from the same file contents. It increases the probability that newly generated sequence of index can be found in the page cache. We use a Bloom Filter [1] to determine whether fingerprint exists or not within short time. The false positives which cause unsuccessful search can be reduced by increasing size of bit array.

The recent research [10] shows that sequence of fingerprints in the file is always same until modifying the file. To use sequential locality, fingerprints must be managed in order of generation. We use index partitioning to exploit this phenomenon. Fingerprint table contains numbers of indexes which are much smaller than cache size and independent each other, instead of one large index. The indexes have a sequential order. The most recently generated index is the last. New fingerprint inserts into the last index. When the last index is full, fingerprint manager makes new index. Because small index can be handled in cache, inserting takes much less time than large index. Moreover, because the fingerprints of same file are stored in a few sequential indexes, search can be done within these indexes. This means that cache hit rate rises. As a result, amount of disk I/O is significantly reduced. PRUNE eliminate. Via maintaining a set of smaller index, we can use existing buffer cache management algorithm. We implement this approach using Berkeley DB. 99.4% of the disk access has been removed as a result.

3. Experiment

We develop prototype of PRUNE. PRUNE is developed as user-level application on Linux 2.6 platform. We implemented PRUNE on 2.66GHZ Intel processor with 2Gbytes of RAM with 320Gbytes SATA2 Hard-disk drive(7200RPM, Samsung).

3.1. 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. We compare the performance of incremental Modulo-K and Rabin's fingerprint algorithm. Fig. 1 illustrates the result. 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, PRUNE is able to reduce the overall latency significantly.

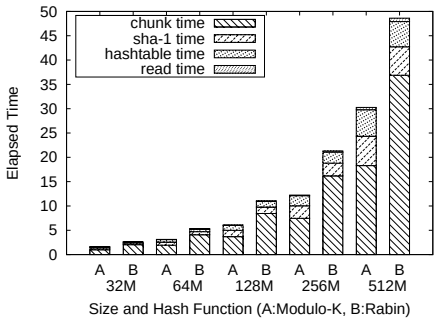

Figure 1. Performance of Modulo-K

3.2. Fingerprint Table Overhead

We evaluate the overall overhead of fingerprint table and compare each DBMS performance. We analyze each DBMS by using approximately 5GB, 10GB, 15GB multimedia files. Fig. 2 illustrates the result.

In this graph, most of the elapsed time is I/O overhead and it shows PRUNE using too many I/O access. Therefore, the most important part of our implementation is how to manage fingerprint table. Once we are implemented using B+Tree. But it is not enough, though B+Tree is the faster DB.

We trace frequency of write I/O by using 'blktrace'. We found its frequency is too much. Therefore, we try to other way. Finally, we use bloom filter and index partitioning. Fig. 3 shows performance of new method of managing hash table. PRUNE hash table has very low write I/O traffic(see Table III).

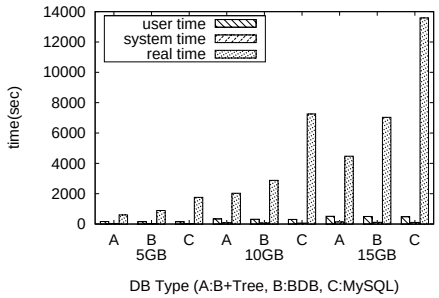

Figure 2. Overhead of Fingerprint Table

Table 3. I/O Trace of PRUNE and Other DB

DB Type	5GB	10GB	15GB
B+Tree	300,859	1,138,667	2,198,333
SkipList	28,251	283,410	1,189,667
PRUNE	2,872	6,160	14,895

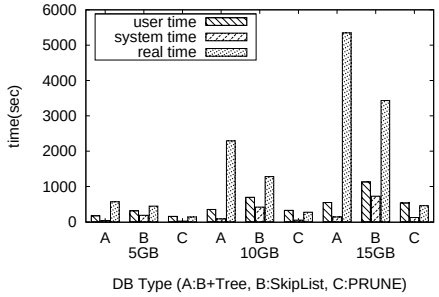

Figure 3. Performance of PRUNE hashtable

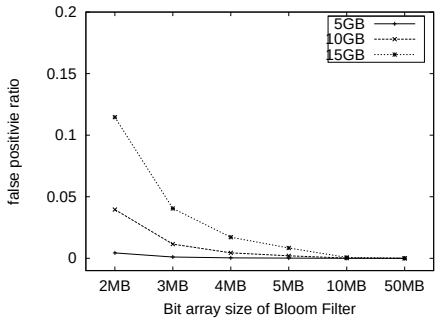

Figure 4. Comparison of bit array size and backup file size

3.3. Effect of Bloom filter size

Fig. 4 shows variation of false positive ratio at different bit array size. Performance depends on false positive ratio. If false positive ratio is greater than 0.02, elapsed time dramatically increases. Therefore, It is important to manage false positive ratio less than 0.02.

4. Conclusion

In this work, we developed state of art backup system, PRUNE, for massive scale data storage. We focus our effort on reducing the disk access overhead of fingerprint management. We adopt filter based in memory index lookup structure and index partitioning of on-disk index. We perform various performance experiments to examine the performance of PRUNE. PRUNE improve the signature generation overhead by 60% against Rabin's fingerprint algorithm. We were able to eliminate 99.4% of the disk access via in-memory lookup structure and index partitioning.

References

[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[2] 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*, pages 13–24, Seattle, WA, 2000.

[3] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: making backup cheap and easy. *SIGOPS Oper. Syst. Rev.*, 36(SI):285–298, 2002.

[4] D. Gomes, A. L. Santos, and M. J. Silva. Managing duplicates in a web archive. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 818–825, New York, NY, USA, 2006. ACM.

[5] B. Hong, D. D. Long, D. Plantenberg, and M. Sivan-Ziemt. Duplicate data elimination in a san file system.

[6] P. Kulkarni, F. Douglis, J. LaVoie, and J. M. Tracey. Redundancy elimination within large collections of files. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 5–5, Berkeley, CA, USA, 2004. USENIX Association.

[7] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. *SIGOPS Oper. Syst. Rev.*, 35(5):174–187, 2001.

[8] S. Quinlan and S. Dorward. Venti: A new approach to archival storage. In *FAST '02: Proceedings of the Conference on File and Storage Technologies*, pages 89–101, Berkeley, CA, USA, 2002. USENIX Association.

[9] A. Traeger, N. Joukov, J. Sipek, and E. Zadok. Using free web storage for data backup. In *StorageSS '06: Proceedings of the second ACM workshop on Storage security and survivability*, pages 73–78, New York, NY, USA, 2006. ACM.

[10] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the Conference on File and Storage Technologies*. USENIX Association, 2008.