A Network Differential Backup and Restore System based on a Novel Duplicate Data Detection algorithm

GUIPING WANG [1], SHUYU CHEN [2*], AND JUN LIU [1]
[1] College of Computer Science
Chongqing University
No. 174 Shazhengjie, Shapingba, Chongqing, 400044
CHINA
{w_guiping, liujun_314}@cqu.edu.cn
[2] College of Software Engineering
Chongqing University
No. 174 Shazhengjie, Shapingba, Chongqing, 400044
CHINA
*Corresponding author: netmobilab@cqu.edu.cn

Abstract: - The ever-growing volume and value of data has raised increasing pressure for long-term data protection in storage systems. Moreover, the redundancy in data further aggravates such pressure in these systems. It has become a serious problem to protect data while eliminating data redundancy, saving storage space and network bandwidth as well. Data deduplication techniques greatly optimize storage systems through eliminating or reducing redundant data in these systems. As an improved duplicate data detection algorithm, SBBS (a sliding blocking algorithm with backtracking sub-blocks) enhances duplicate detection precision through attempting to backtrack the left/right quarter and half sub-blocks of matching-failed segments. Based on the SBBS algorithm, this paper designs and implements a network differential backup and restore system. It designs the structures of full and differential backup images. In addition, in order to fulfill the communication requirements of backup/restore on the Internet, this paper designs a protocol in Application Layer, referred as NBR (Network Backup and Restore Protocol). The experimental results show that, for three typical files, the designed backup and restore system respectively saves 9.7%, 11%, and 4.5% storage space compared with a differential backup system based on the traditional sliding blocking (TSB) algorithm.

Key-Words: - Full backup; Differential backup; Duplicate data detection; Sliding blocking algorithm; Matching-failed segment; NBR

1 Introduction

With the development of information technology, data become increasingly important for enterprises in various domains. However, computer systems may be destroyed by natural disasters (e.g., earthquake, flood, etc.) or human factors (e.g., mal-operation, virus, etc.), which causes data loss in these systems.

Backup is an effective measure to protect data. Data can be restored using backed up copies in case of data loss. Full backup [1-2], incremental backup [1-4], and differential backup [5-9] are three common backup strategies.

However, with the rapid growth of data, the backup of these massive data brings great pressure to storage systems, as well as network bandwidth in case of remote backup. Moreover, redundancy in data further aggravates such pressure in storage systems. It has become a serious problem to protect data while eliminating data redundancy, saving storage space and network bandwidth as well.

In order to eliminate or reduce redundant data in storage systems, deduplication techniques [3, 10-17] emerge and work well. These techniques detect duplicate data and eliminate redundant copies of same data, thus optimizing storage systems through only storing unique data. Since redundant data in storage systems are usually caused by duplicate copies or regions of files, the terms “redundant data/chunk” and “duplicate data/chunk” can therefore be interchanged.

The traditional sliding blocking (TSB) algorithm [18-20] is a typical chunk level duplicate detection algorithm. It divides the files into chunks and introduces a block-sized sliding window to move along the detected file and to find redundant chunks.

In order to enhance the duplicate detection precision of the TSB algorithm, Wang et al. [21] proposes a novel improved sliding blocking algorithm, called SBBS. For matching-failed segments, SBBS continues to backtrack the left/right quarter and half sub-blocks. Based on

SBBS, this paper designs and implements a network differential backup and restore system. To evaluate the performance of the proposed system, this paper conducts experiments on three typical files. The experimental results show that, compared with a differential backup system based on the TSB algorithm, the designed backup and restore system saves 9.7%, 11%, and 4.5% storage space for these three files respectively.

The main contributions of this paper are summarized as follows: 1) It designs and implements a network differential backup and restore system based on the SBBS algorithm; 2) In order to effectively represent backup data, it presents the structures of full and differential backup images; 3) It proposes a protocol in Application Layer, NBR (Network Backup and Restore Protocol), for communication between the clients and the backup server.

The rest of the paper proceeds as follows. Section 2 summarizes related work. Section 3 introduces the rationale of the TSB algorithm and the SBBS algorithm. Section 4 designs and implements a network differential backup and restore system based on the SBBS algorithm. Section 5 presents experiments and analyses. Finally, section 6 gives conclusions and looks into future work.

2 Related work

This section summarizes research work related to backup and restore, deduplication and duplicate data detection.

2.1 Backup and Restore

The ever-growing volume and value of data has raised an increasing demand for long term data protection through backup and restore systems. According to the backup contents, time, and mode, backup techniques can be classified into three categories: *full backup*, *incremental backup*, and *differential backup*.

Full backup [1-2] is a backup in which every object (e.g., a file) in a defined set of data objects is copied, regardless of whether they have been modified since the last backup. Full backup can be implemented easily. But there exists much redundancy between backup copies, especially for the scenarios that the files are modified slightly. Therefore, full backup is seldom adopted solely, but it is the basis of the other two backup techniques.

Incremental backup [1-4] is a type of data backup that provides a backup of files that have been changed or are new since the last incremental backup. It only backs up the data that are changed since the last backup - be it a full backup (for the first incremental backup) or incremental backup, as shown in Fig. 1. This backup strategy takes less time to complete a backup, saving storage space at the same time. After several times of incremental backup, the backups at a given point in time include a full backup image and several incremental backup images. However, once one or more incremental backup images are damaged, the original objects cannot be restored. Moreover, the complex dependencies between incremental backup images increase the instability of backup and restore operations.

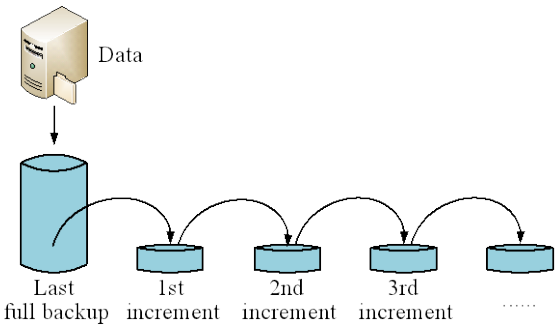


Fig. 1 Incremental backup

The overhead of an incremental backup increases in proportion to the total amount of updated files since last full backup. Therefore, it is necessary to determine when to make full backups. Nakamura et al. [1] propose a stochastic model with incremental and full backups to solve an optimal full backup interval. The optimal interval minimizes the costs incurred for these two backups.

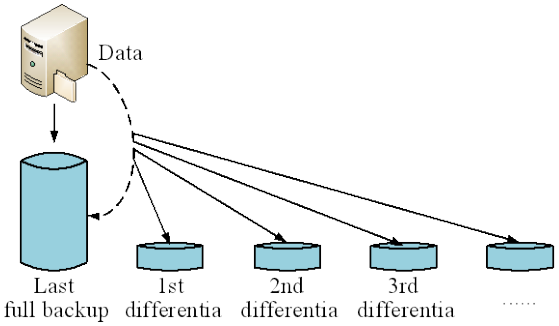


Fig. 2 Differential backup

Differential backup [5-9] is one which preserves data and saves only the difference in the data since the last full backup, as shown in Fig. 2. The backups at a given point in time include a full backup image and the last differential backup image. The underlying rationale of differential backup is that, since changes to data are generally few compared to

the entire amount of data in the data repository, the amount of time required to complete the backup will be quite smaller than that in full backup. Another advantage, at least as compared to incremental backup, is that when restoring a set of data objects, at most two backup media (i.e., a full backup and the last differential backup) are needed. This advantage simplifies data restores, at the same time increases the likelihood of shortening data restoration time.

2.2 Duplicate data detection

Redundant data in storage systems waste much storage space, thus aggravating pressure in storing massive data. Data deduplication techniques detect redundancies between data objects to reduce either storage needs or network traffic [3]. Therefore, deduplication techniques play an important role in improving the utilization of storage systems. Many studies in literature (e.g., [3], [10-24]) address this issue in various storage environments.

Duplicate data detection is the foremost step for deduplication techniques. Among various duplicate detection techniques, chunk level techniques [3, 18-24] are the most common ones. For the old version and the new version of a same set of data objects (usually, files), these techniques divide them into chunks, and further detect redundant chunks through computing and checking hash values of these chunks. Chunk level techniques make a well trade-off between performance and efficiency [21]. In addition, these techniques can be classified into three typical categories: fixed-sized partition, variable-sized partition, and sliding blocking.

Among these chunk level techniques, the traditional sliding blocking (TSB) algorithm [18-20] can be implemented without much effort, and its duplicate detection precision is relatively high. Therefore, it is widely adopted in deduplication systems. But it cannot effectively deal with resulting matching-failed segments, which will be analyzed in the next section.

Deduplication and duplicate data detection are always continuously concerned issues in literature, regardless that new storage techniques have been introduced (e.g., [16]) or new application environments are confronted (e.g., virtualization and cloud environments [17, 22]). For example, virtualization techniques implement multiple logically separate execution environments. Therefore, these techniques are widely adopted in servers. However, these techniques still consume large amounts of storage due to separate disk image (up to GB bytes) of each virtual machine (VM) instance. Jin and Miller [17] introduce deduplication techniques into virtualization. Their study shows that the combination of these two techniques not only reduces the total storage required for VM disk images, but increases the ability of VMs to share disk blocks.

Compared with the above researches, this paper mainly focuses on differential backup. Especially, it designs and implements a differential backup and restore system based on an improved sliding blocking algorithm, i.e., SBBS [21].

3 The rationale of the TSB and the SBBS algorithms

This section first summarizes the rationale of the TSB algorithm. Then it introduces the concept of *matching failed segment*. Finally, it summarizes the improved TSB algorithm, i.e., SBBS [21], which is the prior work of this paper.

3.1 The rationale of the TSB algorithm

The TSB algorithm [18-20] introduces a block-sized sliding window which moves along the detected file to check each block. It calculates the checksum and the hash value of every overlapping block-sized segment in the detected file to determine duplicate blocks.

Note that, a *chunk* is the detection granularity for chunk level techniques, while a *block* is a data segment covered by the sliding window. But in TSB and SBBS, "block" and "chunk" are identical concepts and therefore can be interchanged.

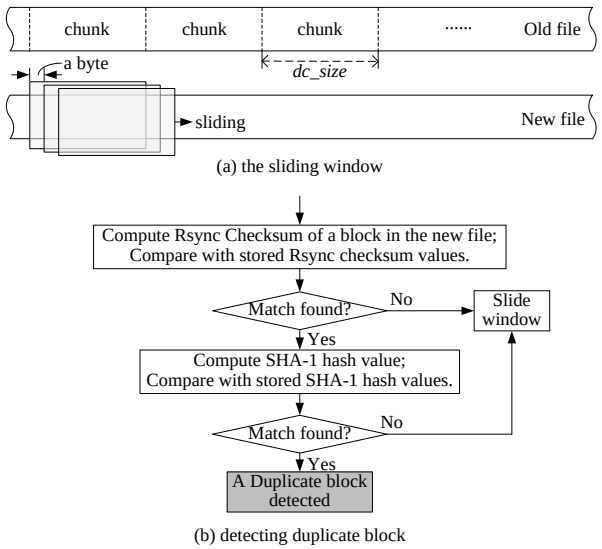


Fig. 3 The rationale of the TSB algorithm

The rationale of the TSB algorithm is concreted as that: the old file is divided into non-overlapping chunks with equal length, as illustrated in Fig. 3(a)

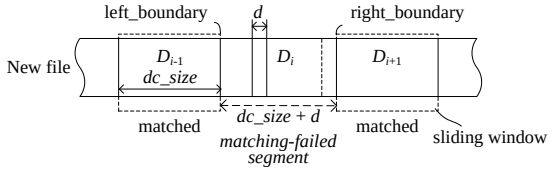
(*dc_size* is the size of a data chunk); then the checksum and the hash value of each chunk are computed and stored in a table; when the sliding window moves from the head of the detected file to the end byte by byte, the checksum and the hash value of each block covered by the sliding window are computed in turn and compared with stored values to detect a duplicate block, as detailed in Fig. 3(b).

3.2 The rationale of the SBBS algorithm

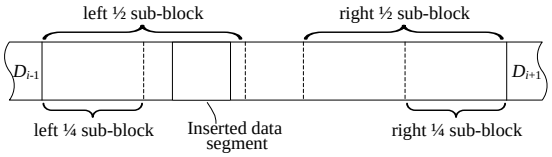
Although the precision of the TSB algorithm is relatively high, it cannot effectively deal with matching-failed segments caused by inserting, deleting, or modifying data in a chunk. Due to the space limitation, this subsection only analyzes the scenario of inserting a data segment into a chunk. As Fig. 4(a) shows, a data segment with length d is inserted into chunk D_i. Accordingly, the right boundaries of chunk D_i and all the subsequent chunks are moved backward distance of d.

For simplicity, assume that the chunks, D_{i-1} and D_{i+1}, have no any change. This simplicity does not affect the correctness of the analyses below and the SBBS algorithm [21]. Therefore, after the checking process of the TSB algorithm, the chunks, D_{i-1} and D_{i+1}, are identified and marked as duplicate chunks. While the chunk D_i, which is lengthened to dc_size+d, fails to be matched, as illustrated in Fig. 4(a). The lengthened chunk D_i is marked as a *matching-failed segment* [21].

In order to match unmodified data in matching-failed segments, SBBS attempts to backtrack sub-blocks. Each chunk contains 4 sub-blocks, as illustrated in Fig. 4(b). For clear illustration, the chunk D_i, including the inserted data segment of length d, is stretched four times in Fig. 4(b).



(a) matching-failed segment



(b) sub-block partition

Fig. 4 Matching-failed segment and sub-block partition

Although the chunk D_i fails to be matched, there are much redundant data in D_i. As illustrated in Fig. 4(b), the left quarter and right half sub-blocks are both redundant data segments. If these redundant data can be further detected, the duplicate detection precision will be undoubtedly enhanced. The rationale of SBBS is derived from this intuitive idea. The rationale can be concreted as that, for each matching-failed segment, SBBS continues to backtrack the left/right quarter and half sub-blocks.

SBBS combines weak hash check and strong hash check to determine redundant chunks. For chunk A and B, only when they satisfy weakhash(A) = weakhash(B) and stronghash(A) = stronghash(B), SBBS determines that these two data chunks are identical ones.

When matching the sliding block, SBBS adopts the Rsync rolling checksum for weak hash check. While when backtracking sub-blocks, SBBS uses the Adler-32 checksum as weak hash check. For both the sliding block and the sub-blocks, SBBS uses the MD5 hash algorithm as strong hash check [21].

4 The design and implementation of the SBBS-based network differential backup and restore system

This section first describes the architecture of the designed system. Then it introduces the structure of backup images. Lastly, it designs NBR protocol for communication on the Internet.

4.1 System architecture

The designed network backup and restore system, which is deployed as Fig. 5, is composed of a backup server and several clients. The server and the clients are connected through the Internet.

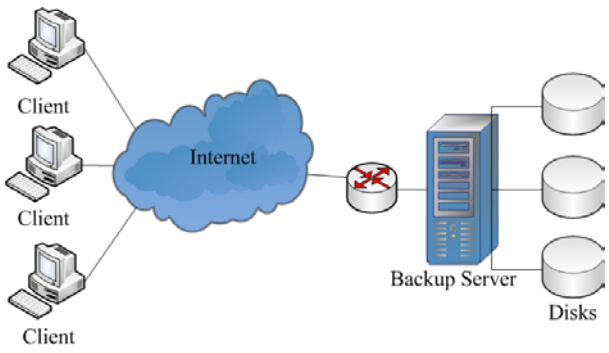


Fig. 5 The deployment of the designed network backup and restore system

The clients can login to the backup server via the Internet, and back up local files to the server. Before

backup, the software on the clients first detects differential data between local files and the backup image on the server. After deduplication, the software packs the differential data into a backup image, and transfers the image to the backup server. Alternatively, the clients can restore local files via backup images transferred from the server. The server responds to the clients' backup/restore request.

The software architecture of the designed backup and restore system is shown in Fig. 6.

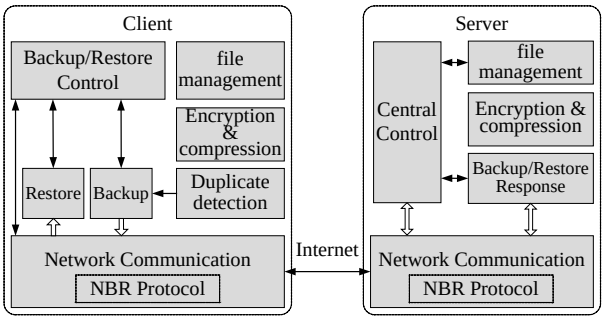


Fig. 6 The software architecture of the designed network backup and restore system

The most important modules are detailed below.

(1) The modules in the client

Backup/Restore Control module: it receives the instructions from users; completes backup/restore functions cooperating with the backup/restore modules; sends various requests to the backup server via network communication module; etc.

Network communication module: it constructs a NBR message; parses messages from the server and sends the received data to the corresponding modules for further processing; manages Socket connections; etc.

Backup module: it constructs the full backup images and the differential backup images; when constructing a differential backup image, it detects duplicate data in the backed up file, and only constructs the differential data segments into the image file.

*Duplicate detection modul*e: it includes the TSB and the SBBS duplicate detection algorithms, which are invoked by the backup module.

Restore module: it restores the local files by the backup image transferred from the server.

(2) The modules in the server

Central Control module: it is responsible for receiving, parsing, and dealing with all requests (including login/logout, backup/restore, etc.) from the clients.

Network Communication module: it receives the NBR messages from the clients, and sends these messages to the central control module; it packs the data from other modules; etc.

Backup/Restore Response module: it completes the concrete backup/restore functions; when responding to a backup request from a client, it receives the backup image from the client and stores in the server; when responding to a restore request from a client, it reads the specified image file and sends to the client; etc.

4.2 The structures of backup images

In order to effectively represent backup data, this paper designs the structures of backup images. The backup images include the full backup image and the differential backup image.

(1) The full backup image

The structure of the full backup image, shown in Fig. 7(a), includes the header of image, and image data. The former includes backup parameters, and backup files list; the backup parameters describe such information as backup type, backup time, etc; the backup files list includes the list of files in the full backup image, as well as their offsets in the image data part. The latter includes effective data of the files in full backup image.

(2) The differential backup image

The structure of the differential backup image, shown in Fig. 7(b), also includes the header of image, and image data. Unlike the backup files list in full backup image, the one in differential backup image designates the list of differential backups; for each differential backup, it also designates its differential index's offset in the header. In virtue of this offset, the differential index of a file determines the differential images' position in the image data.

4.3 NBR protocol

In order to satisfy the need of backup and restore functions, this paper designs a protocol in Application Layer, NBR (Network Backup and Restore Protocol), for communication between the clients and the backup server on the Internet. The format of NBR protocol message is shown in Fig. 8.

A NBR message includes the header of message and the body of message. The former occupies 8 bytes. The length of the latter depends on the type of message.

Several most important fields in a NBR message are described in detail below.

(1) Length of message body

This field occupies 4 bytes. It designates the length of message body. Therefore, the length can be up to 2^{32} bytes (4GB) theoretically.

(2) The type of message

According to the initiator of a NBR message, it can be classified into two categories, i.e., *client message* and *server message*. The client message is initiated in the client end and is processed in the server end. Its type code is between 1 and 127. While the server message is initiated in the server end and is processed in the client end. Its type code is between 128 and 255.

Each type of message has a unique code. The types of client messages and server messages are listed in Table 1 and 2, respectively.

(3) The format of message body

The content of message body depends on the types of message. If the content contains several fields, these fields are separated by a colon, ":". The formats of message body of several types of message are listed below.

a) Login – userid : password.

b) Logout – userid.

c) Backup Request – the length list of fields in the message body : the backed up files list : the expected size of image file.

d) Restore Request – the restored files list.

e) File Transfer – the length list of fields in the message body : identifiers : data of files.

f) User-Defined Message – the contents of the user-defined message.

g) Hash Table Request – the files list of the requested hash table.

h) Backup Request Refuse – the reason of refusing backup request.

i) Restore Request Refuse – the reason of refusing restore request.

j) Backup Fails – the cause of backup failure.

k) Restore Fails – the cause of restore failure.

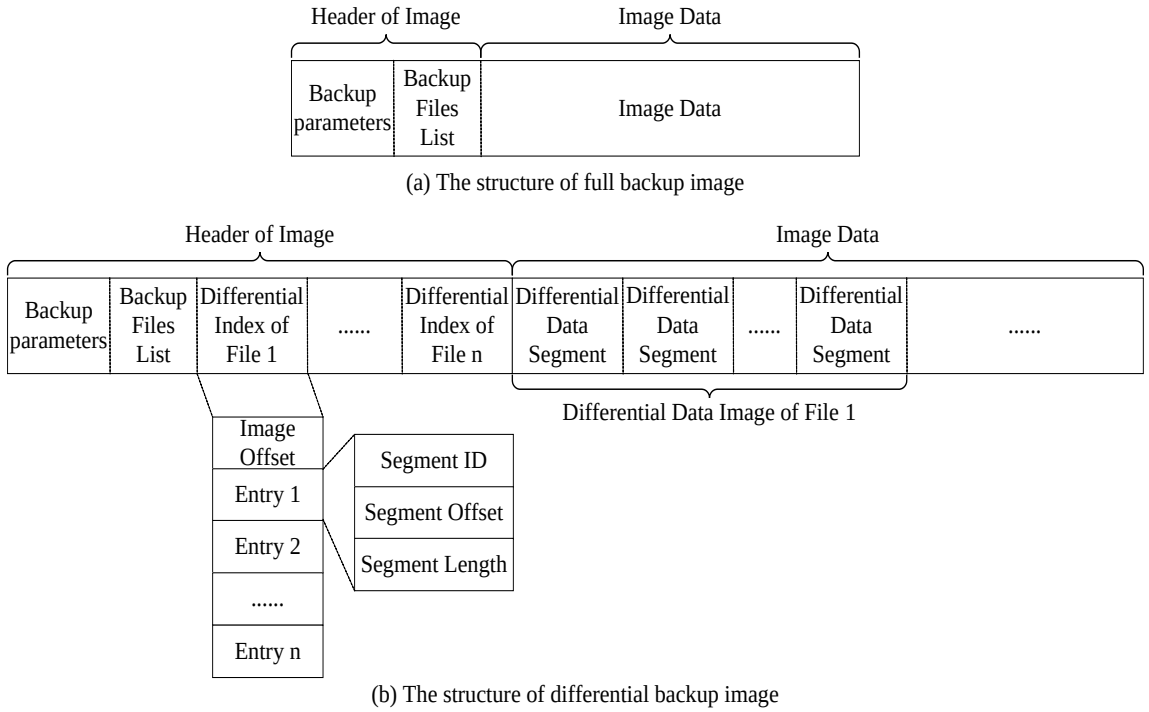


Fig. 7 The structures of the full and differential backup images

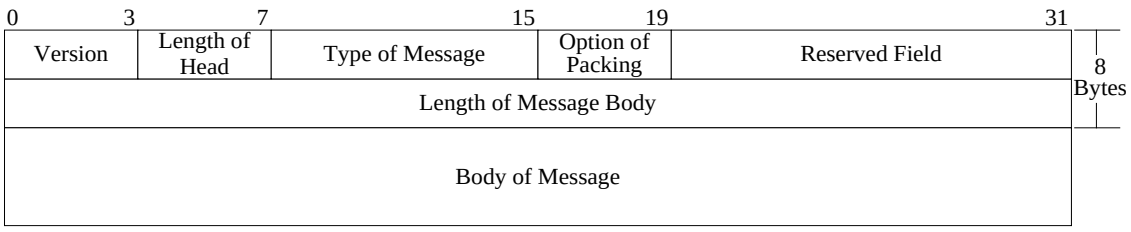


Fig. 8 The format of NBR protocol message

Table 1 The types of Client Messages

Type of Message	Macro definition	Code	Description
Login	C_LOGIN	1	A user logins to backup server.
Logout	C_LOGOUT	2	A user logouts.
Backup Request	C _BACKUP_REQUEST	3	
Restore Request	C _RESTORE_REQUEST	4	
File Transfer	C_FILE_TRANSFER	5	Transferring backup image, checklist, etc.
Cancel Transfer	C_CANCEL_FILETRANS	6	
User-Defined Message	C_MESSAGE	7	Sending out user-defined message.
Hash Table Request	C_HASHTABLE_REQUEST	8	Requesting the server to access the hash table of a specified file.
Restore Successes	C_RESTORE_SUCCSSES	9	Notifying the server that restore finishes.
Backup Fails	C_BAKUP_FAILS	10	Backup fails.
Restore Fails	C_RESTORE_FAILS	11	Restore fails.
Keep Alive	C_KEEPALIVE	12	Keeping heartbeat message alive.

Table 2 The types of Server Messages

Type of Message	Macro definition	Code	Description
Login Confirms	S_LOGIN_REPLY	128	Replying to a user's login.
Logout Confirms	S_LOGOUT_REPLY	129	Replying to a user's logout.
Backup Request Response	S_BAKUP_RESPONSE	130	Responding to a user's backup request.
Backup Request Refuse	S_BAKUP_REFUSE	131	Refusing a user's backup request.
Restore Request Response	S_RESTORE_RESPONSE	132	Responding to a user's restore request.
Restore Request Refuse	S_ RESTORE_REFUSE	133	Refusing a user's restore request.
Backup Fails	S_BAKUP_FAILS	134	Backup fails.
Restore Fails	S_RESTORE_FAILS	135	Restore fails.
Backup Success	S_BAKUP_SUCCSSES	136	Notifying the client that backup finishes.
User-Defined Message	S_MESSAGE	137	

5 Experiments and analysis

In order to evaluate the performance of the proposed SBBS-based backup and restore technique, this section compares it with the full backup technique and the TSB-based differential backup techniques.

5.1 Experimental environment and test files

The experimental environment is composed of one backup server and five backup clients. The backup server runs the server program of the backup/restore system, while the backup clients run the client program. Both these programs are implemented by C language.

The configuration of the server and the clients is listed as follows.

CPU: Intel i5 2400, 3.1 GHz.

Memory: 4G DDR3 1600MHz.

Disk: 500G SATA.

Network Interface Card: Intel 9301 CT, PCI-E, 1000Mbps.

OS: the server, Fedora 17 (Linux 3.3.4); the clients, Ubuntu 12.04 (Linux 3.2.12).

In order to evaluate the performance of these backup and restore techniques, this paper chooses three typical files as test files: the log file, *syslog*; the database file of mysql, *mysql.myd*; and source code of linux-2.6.11, *linux-src*.

For the original versions of these test files, this paper modifies the contents randomly to obtain the new files. The differential data of the original and new files are calculated by *diff*. The description of these three test files is listed in Table 3, where duplicate data ratio is defined as the ratio between the size of actual duplicate data and the size of the original file. Therefore, the size of each new file equals the size of duplicate data plus the size of differential data.

Table 3 Three test files

Test file	Size of the original file	Differential data	Duplicate data ratio
syslog	29,748 KB	7,482 KB	75%
mysql.myd	62,464 KB	4,456 KB	93%
linux-src	189,440 KB	71,721 KB	65%

5.2 Performance testing and analysis

This subsection evaluates the proposed SBBS-based differential backup technique from three aspects below.

(1) Duplicate detection rate

For the above three test files, the SBBS and TSB algorithms are used to detect duplicate data. The results are shown in Fig. 9.

This figure shows that the SBBS algorithm outperforms the TSB algorithm. For these three test files, SBBS detects 1450 KB (6.5%), 6390 KB (11%), 8619 KB (7%) more duplicate data than the TSB algorithm respectively. Comparatively speaking, for both algorithms, the duplicate detection rate in mysql.myd is highest, while that in linux-src is lowest. The higher the duplicate rate in a test file, the better the detection algorithms perform; moreover, the more promotion the SBBS algorithm brings.

The detection rate in linux-src is relatively low. The underlying reason is that there exist so many small files in Linux source code. Nevertheless, the SBBS algorithm improves detection rate in this file through sub-block backtracking.

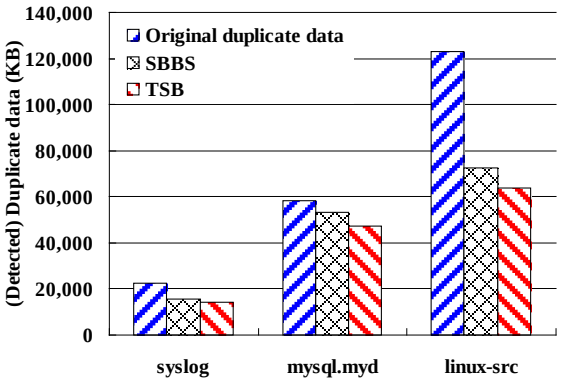


Fig. 9 The detected duplicate data of three test files for two duplicate detection algorithms (SBBS and TSB)

(2) Storage overhead and bandwidth ratio

For full backup, the SBBS-based and the TSB-based differential backup techniques, the image sizes of the three test files are illustrated in Fig. 10.

The results in this figure show that, for three typical files, the designed SBBS-based backup technique respectively saves 9.7%, 11%, and 4.5% storage space compared with TSB-based technique; and respectively saves 38%, 75%, and 34% storage space compared with full backup technique. The higher the duplicate rate in a test file, the more storage space these techniques save.

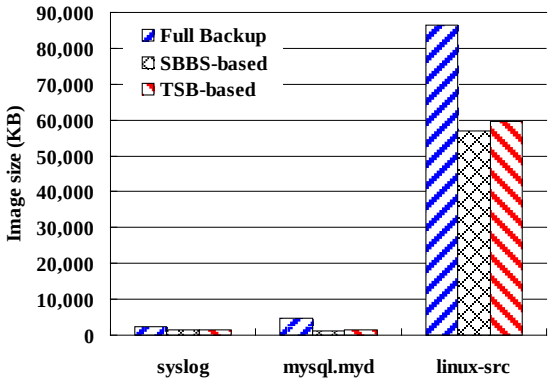


Fig. 10 The image sizes of three test files for three backup techniques

In the aspect of bandwidth saving, this paper introduces the term of *bandwidth ratio*, which is defined as the next equation.

$$\text{bandwidth ratio} = \frac{\text{the amount of transferred data in differential backup}}{\text{the amount of transferred data in full backup}} \quad (1)$$

For the three test files, the bandwidth ratios of SBBS-based and TSB-based backup techniques are illustrated in Fig. 11. As this figure shows, the bandwidth ratios of these two backup techniques are also proportional to the detection rate of test files.

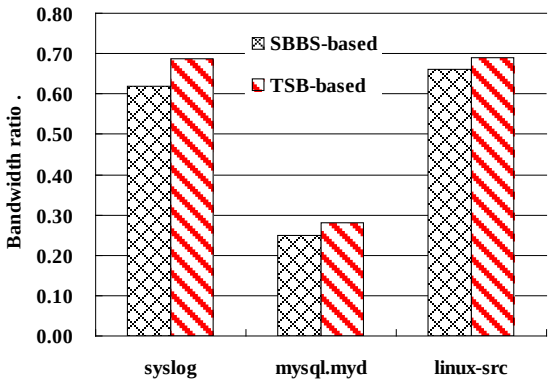


Fig. 11: The bandwidth ratios of three test files for two backup techniques (SBBS-based and TSB-based)

(3) Backup time

The backup time includes the time for detecting duplicate data, reading differential data, and packing differential data into an image, excluding the time for transferring network data.

SBBS backtracks sub-blocks based on TSB. For sub-blocks, SBBS needs to further calculate and compare hash values, which causes a slight increase in the number of computations compared with TSB. In addition, compared with TSB, SBBS itself needs more extra memory storage to store the hash values of sub-blocks. In order to improve detection precision, as well as control additional storage overhead, SBBS chooses a larger block size, 8 kB, for files larger than 1024 kB.

For three backup techniques, the backup time of three test files are shown in Fig. 12.

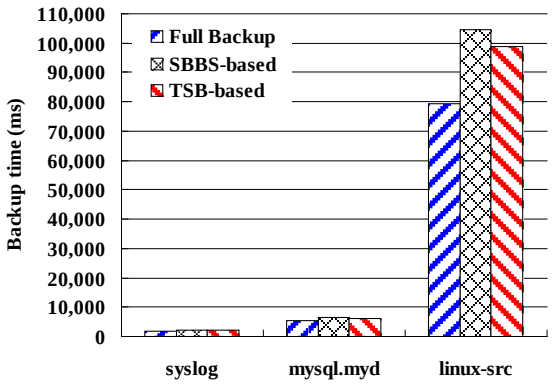


Fig. 12 The backup time of three test files for three backup techniques

The results in this figure show that, for syslog, the proposed SBBS-based technique spends 8.5% more time compared with the TSB-based technique, but the former saves 9.7% more storage space; for mysql.myd, the proposed SBBS-based technique spends 5.9% more time compared with the TSB-based technique, but the former saves 11% more storage space. More importantly, in the proposed SBBS-based technique, the calculations for backup are mainly fulfilled on the client end. In sum, the proposed SBBS-based technique saves storage space and alleviates storage pressure for the backup server through introducing little more calculations in the client end.

Based on above experiments, some implications are concluded as follows.

1) The SBBS algorithm [21] improves the duplicate detection precision by 6.5% to 11% compared with the TSB algorithm.

2) The proposed SBBS-based differential backup technique decreases the image size, thus reducing network bandwidth occupation and saving storage space of the backup server. Compared with TSB-based technique, the proposed differential backup technique saves 4.5% to 11% storage space.

6 Conclusion and future work

For duplicate detection in differential backup, the SBBS algorithm enhances duplicate detection precision through backtracking the left/right quarter and half sub-blocks of matching-failed segments. Based on the SBBS algorithm, this paper designs and implements a differential backup and restore system. Compared with TSB-based technique, the proposed SBBS-based differential backup technique saves up to 11% storage space.

However, for the scenario where the set of data objects includes so many small files, the proposed SBBS-based backup technique behaves not as good as expected, which is a direction of the future efforts of this paper. In addition, the deduplication and backup techniques under virtualization and cloud environments are still challenging currently, which is another direction of the future efforts of this paper.

Acknowledgments

The authors would like to thank both the editor and anonymous reviewers for their valuable feedback comments.

The work of this paper is mainly supported by National Natural Science Foundation of China (Grant No. 61272399), and Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20110191110038).

References:

[1] S. Nakamura, C. Qian, S. Fukumoto, and T. Nakagawa, Optimal backup policy for a database system with incremental and full backups, *Mathematical and Computer Modelling*, vol. 38, no. 11-13, pp. 1373-1379, 2003.

[2] C. H. Qian, Y. Y. Huang, X, F. Zhao, and T. Nakagawa, Optimal backup interval for a database system with full and periodic incremental backup, *Journal of Computers*, vol. 5, no. 4, pp. 557-564, 2010.

[3] D. Meister, A. Brinkmann, Multi-level comparison of data deduplication in a backup scenario, In: *Proceedings of the Israeli Experimental Systems Conference*, Article no. 8, 2009.

[4] C. H. Qian, S. Nakamura, and T. Nakagawa, Optimal backup policies for a database system with incremental backup, *Electronics and Communications in Japan Part III - Fundamental Electronic Science*, vol. 85, no. 4, pp. 1-9, 2002.

[5] J. Nie, Design the desktop backup system based on cloud computing, In: *Proceedings of the 8th International Conference on Computational Intelligence and Security*, pp. 183-185, 2012.

[6] R. J. T. Morris and B. J. Truskowski, The evolution of storage systems, *IBM Systems Journal*, vol. 42, no. 2, pp. 205-217, 2003.

[7] G. Pluta, L. Brumbaugh, W. Yurcik, and J. Tucek, Who Moved My Data? A Backup Tracking System for Dynamic Workstation Environments, In: *Proceedings of 18th USENIX Large Installation System Administration Conference* (LISA), pp. 177-186, 2004.

[8] W. Xu, M. Wang, X. He, and Z. J. Liu, BM-CVI: A backup method based on a cross-version integration mechanism, In: *Proceedings of International Conference on Convergence Information Technology*, pp. 781-788, 2007.

[9] J. Velvizhi, C. G. Balaji, Fingerprint lookup-an effective and efficient backup using deduplication technique, *Journal of Theoretical and Applied Information Technology*, vol. 38, no. 1, pp. 49-54, 2012.

[10] D. N. Simha, M. H. Lu, T. C. Chiueh, A scalable deduplication and garbage collection engine for incremental backup, In: *Proceedings of the 6th International Systems and Storage Conference*, Article no. 16, 2013.

[11] D. T. Meyer and W. J. Bolosky, A study of practical deduplication, *ACM Transactions on Storage*, vol. 7, no. 4, Article 14, 2012.

[12] Y. Zhang, Y. W. Wu, and G. W. Yang, Droplet: A Distributed Solution of Data Deduplication, In: *Proceedings of the 13th ACM/IEEE International Conference on Grid Computing*, pp. 114-121, 2012.

[13] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti, iDedup: latency-aware, inline data deduplication for primary storage, In: *Proceedings of the 10th USENIX conference on File and Storage Technologies*, 2012.

[14] M. Kaczmarczyk, M. Barczynski, W. Kilian, and C. Dubnicki, Reducing impact of data fragmentation caused by in-line deduplication, In: *Proceedings of the 5th Annual International Systems and Storage Conference*, Article no. 15, 2012.

[15] C. Kim, K. W. Park, and K. H. Park, GHOST: GPGPU-offloaded high performance storage I/O deduplication for primary storage system, In: *Proceedings of International Workshop on Programming Models and Applications for Multicores and Manycores*, pp. 17-26, 2012.

[16] B. Debnath, S. Sengupta, and J. Li, ChunkStash: speeding up inline storage deduplication using flash memory, In: *Proceeding of USENIX annual technical conference*, pp. 1-16, 2010.

[17] K. Jin, E. L. Miller, The effectiveness of deduplication on virtual machine disk images, In: *Proceedings of the Israeli Experimental Systems Conference*, Article no. 7, 2009.

[18] T. E. Denehy and W. W. Hsu, Duplicate management for reference data, *IBM Research Report*, RJ 10305 (A0310-017). 2003

[19] C. Policroniades and I. Pratt, Alternatives for Detecting Redundancy in Storage Systems Data, In: *Proceedings of the USENIX Annual Technical Conference*, pp. 73-86, 2004.

[20] J. T. Ma, A Deduplication-based Data Archiving System, In: *Proceedings of International Conference on Image, Vision and Computing*, 2012.

[21] G. P. Wang, S. Y. Chen, M. W. Lin and X. W. Liu, SBBS: A sliding blocking algorithm with backtracking sub-blocks for duplicate data detection, *Expert Systems with Applications*, Vol. 41, Issue 5, pp. 2415-2423, 2014.

[22] C. H. Ng, M. C. Ma, T. Y. Wong, P. P. C. Lee, and J. C. S. Lui, Live deduplication storage of virtual machine images in an open-source cloud, In: *Proceedings of the 12th International Middleware Conference*, pp. 80-99, 2011.

[23] D. Meister, J. Kaiser, and A. Brinkmann, Block locality caching for data deduplication, In: *Proceedings of the 6th International Systems and Storage Conference*, Article no. 15, 2013.

[24] Y. W. Ko, H. M. Jung, W. Y. Lee, M. J. Kim, and C. Yoo, Stride Static Chunking Algorithm for Deduplication System, *IEICE Transactions on Information and Systems*, vol. E96D, no. 7, pp. 1544-1547, 2013.