Improving the Performance of Backup Candidate File Selection using Inode Bitmap

Sosuke Matsui, Tsuyoshi Miyamura, Noriko Tanemura, Terue Watanabe, Norie Iwasaki
Tokyo Software & Systems Development Laboratory
IBM Japan
e34975@jp.ibm.com

***Abstract*—An incremental backup copies data and metadata of created, updated, and deleted files since the previous backup. The performance of finding candidate files for an incremental backup becomes a limiting factor on the number of files in a file system. We propose a method for finding backup candidate files using inode bitmaps as intermediate files to improve the performance of incremental backups. Our method reduces the size of intermediate files used by the backup candidate file selection and efficiently finds deleted files by calculating the bitwise operation of inode bitmaps. We implemented our method with a file system backup function supported by IBM Spectrum Scale (formerly known as the General Parallel File System, GPFS) 4.2, and showed that our method improves the performance of backup candidate file selection up to 44.5%.**

I. Introduction

Running periodical backups of a file system is important to prevent loss of business data. In many business environments, periodical incremental backups are scheduled to reduce the amount of time for a full backup. An incremental backup copies created or updated data and metadata since the previous backup. Deleted data and metadata since the previous backup are marked inactive so that they will not be copied from backup media to disk in case of restoration.

There are two steps in an incremental backup process of a file system: the first step is to traverse the directory structure and find created, updated, and deleted files and the second step is to copy data and metadata of the files to backup media such as tapes. According to previous work, traversing the directory structure to find candidate files for a backup takes a long time when the number of files in a file system is large [1], [2]. Therefore, improving the performance of finding the candidate files is essential, and much work has been done [1]–[3].

One reason it takes a long time to find backup candidate files is the size of intermediate files. Previous work such as [2], [3] creates a list of all files in a file system to find deleted files since the previous backup. The list needs to include the full path of files and the inode number [3], [4]. For example, IBM Spectrum Scale (formerly known as the General Parallel File System, GPFS) supports more than a billion of files per file system, and the size of the list of all files can become a few hundred GB. As a result, finding backup candidate files takes a long time because of reading such a large list.

We propose a method for reducing the size of the intermediate file using an inode bitmap, which improves the performance of the step for finding the backup candidate files. An inode is an object that stores metadata of a file such as owner, last access time and file size. An inode bitmap manages the usage of inodes using a bit array. Our method finds deleted files by calculating the bitwise operations of inode bitmaps. We implemented our method with IBM Spectrum Scale 4.2 and verified that the performance of backup candidate file selection improved up to 44.5%.

The rest of this paper is organized as follows: in the next section, we describe related work. In Section 3, we report a problem with previous backup methods. In Section 4, we explain our method. We evaluate our method in Section 5 and finally present our conclusion in Section 6.

II. Related Work

A backup strategy can be categorized as either a logical or physical backup [5]. On one hand, a physical backup is a block-based strategy and works efficiently compared with a logical backup. On the other hand, the weakness of a physical backup is that a restored storage system needs to have the same configuration as the original storage system [5]. A logical backup is a file-based strategy and has more flexibility compared to a physical backup. For example, a restored storage system can have a different configuration from the original system. In many enterprise IT environments, a restored storage system located at a backup site usually has a smaller configuration compared with the original system at a production site. Therefore, our interest is in the logical backup, and this paper focuses on the file-based strategy.

Patterson et al. proposed a method for efficiently determining updated data by comparing two snapshots of the NetApp WAFL file system [1]. However, this method can only be applied to a physical backup.

Kaplan and Bisson proposed methods for improving the performance of traversing a directory structure and reading inodes in a file system [2], [3]. Kaplan's method is used by a backup function of IBM Spectrum Scale [4], which is a logical backup. These methods have a certain effect on the performance of the step for finding created, updated, and deleted files. However, they still suffer from a problem in which the size of intermediate files gets larger as the number of files in a file system increases. Our method reduces the size of intermediate files and can be used with their methods.

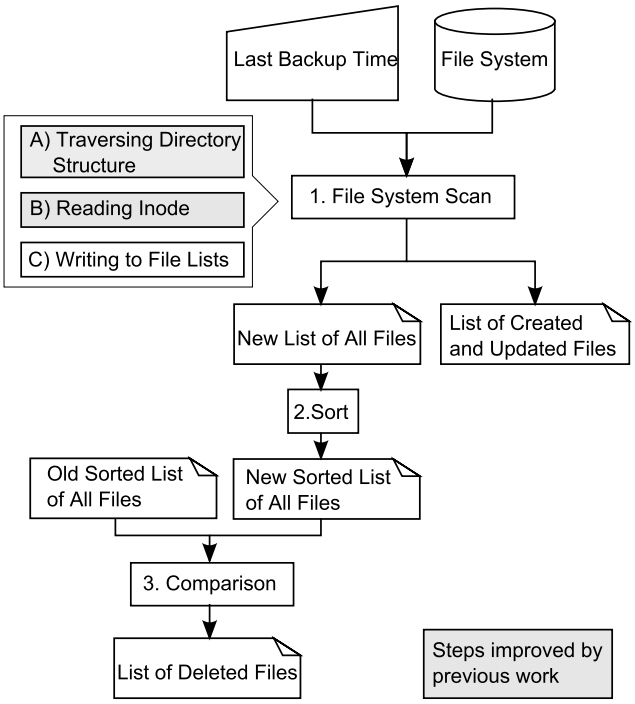

Fig. 1. Flowchart for finding created, updated, and deleted files

III. Overview of the Previous Work and Its Problem

Figure 1 shows the algorithm of the previous work [2] for finding backup candidate files. There are three steps for finding the candidate files:

1) File system scan for finding created and updated files. This step is composed of the following three parts:
   a) Traversing directory structure
   b) Reading inodes to find created and updated files
   c) Writing full path of created and updated files to file lists
2) Sorting file list
3) Comparing file lists to find deleted files

Previous work such as [2], [3] improves the performance of steps a) and b) for finding created, updated, and deleted files. However, the previous work still suffers from a problem in which the performance for finding backup candidate files degrades as the number of files in the file system increases because of the size of the intermediate file. In the next section, we explain the algorithm of the previous work and its problems.

A. Overview of the Previous Work

First, a file system is scanned using a previous backup time as input to find files that have been created or updated since the last backup. Specifically, the directory structure is traversed, and the ctime and mtime of files are read from inodes. The ctime and mtime are compared with the previous backup timestamp, and if the ctime or mtime is newer than the timestamp, the inode number and full path of the file are written to a list of created and updated files. In addition, regardless of the result of the timestamp comparison with the ctime and mtime, the inode number and full path of a file is written to another file list that is a list of all files in the file system.

Then, the list of all files is sorted by inode number. The sorted list of all files will be used by the further processing and also by the next incremental backup.

Last, the sorted list of all files created by this backup and the previous one is compared by inode numbers. If a file that existed at the previous backup time but that does not exist now is found, the inode number and full path of the file is written to a list of deleted files.

All the created, updated, and deleted files since the previous backup will be found by the three steps described above. Once these steps are complete, the data of created and updated files are copied to backup media. Deleted files are marked as inactive so that they are not copied from backup media to disks when a restoration is performed. For example, IBM Spectrum Protect (formally known as Tivoli Storage Manager) reads a list of deleted files and stores them in a database so that they will not be restored [4].

B. Problem with the Previous Work

The previous work generates lists of all files created by the last and the current backup to find deleted files. As the number of files in a file system becomes larger, the size of the lists gets bigger, and backup candidate file selection takes a long time.

Spectrum Scale supports file system scanning with 10 billion files in a file system [6]. If the average length of a file path is 64 bytes, the size of the list of all files will become 640 GB. In such an environment, the list of all files will not likely reside in cache memory. As a result, file system scanning and list sorting require access to disks to read and write the 640 GB list.

Reducing the size of intermediate files is important for solving this problem. However, using compression techniques like zip cannot solve this problem because the size of the compressed list still varies depending on the number of files in the file system. To completely solve this problem, formatting the intermediate files independently from the number of files in a file system is preferable.

IV. Proposed Method

A bit array is frequently used to represent a set of numbers in a specific range [7]. Well known file systems such as ext4 assign a unique ID called an inode number to a file which is in a certain range. Usage of inode numbers are managed by a data structure called the inode bitmap. Our method writes the content of an inode bitmap to a file and uses it as an intermediate file to find backup candidate files. Therefore, our method solves the problem in which finding backup candidate files takes a long time because of the size of the intermediate file. However, finding deleted files just by using inode bitmaps

Format of intermediate file	Size of intermediate file	Time to create intermediate file
File List	640 GB	21629.0 seconds
Inode Bitmap	1.25 GB	42.4 seconds

TABLE I
COMPARISON OF FILE LIST AND INODE BITMAP

is impossible. We propose a method for calculating the bitwise operation of inode bitmaps to find deleted files.

In this section, we explain our method in detail and its expected result.

A. Core Idea of the Proposed Method

As described in the previous section, an inode bitmap is a data structure that manages the usage of inode numbers. When a file system assigns an inode number to an inode, the corresponding bit is set.

If a file system supports up to 10 billion files, the usage of an inode number can be managed by 10 billion bits. Therefore, the size of the inode bitmap of the file system will be 1.25 GB. Compared to the previous work, the size of the intermediate file becomes smaller by storing the content of the inode bitmap in the intermediate file. As a result, the time for reading and writing the intermediate files can be reduced. In addition, the size of the inode bitmap depends only on the maximum number of files supported by a file system. Usually, the maximum number of files in a file system is determined on creation of a file system. That is, the size of the inode bitmap is constant and does not vary depending on the number of files in a file system.

To investigate the effect of using the inode bitmap as an intermediate file, we compared the time to write a list of file paths and the inode bitmap to a file on a Linux server with 16 GB of memory installed. We installed Redhat Enterprise Linux 6.5 and Spectrum Scale 4.2 in the test environment.

We measured the time to create a list of 10 billion files and the time to write 10 billion bits of the inode bitmap to a file. During our test, the average file path length of test files was set to 64 bytes. Table I shows the result. By writing the content of the inode bitmap to a file instead of writing a list of file paths, the time to create the intermediate file was reduced by about 5 hours and 59 minutes.

B. Overview of the Proposed Method

Figure 2 shows a flowchart for finding backup candidate files by our method.

First, the file system scan creates a list of files that have been created or updated since the previous backup. We use the same technique as the previous work [2] to traverse the direrctory structure and read inodes. The file system scan writes the inode bitmap (hereinafter referred to as new inode bitmap) to a file and uses it as an intermediate file for finding candidate files. As a result, the size of the intermediate file gets reduced.

Then, we read the inode bitmap of the previous backup (hereinafter referred to as old inode bitmap) from a file and calculate the inode bitmap of deleted files by comparing the

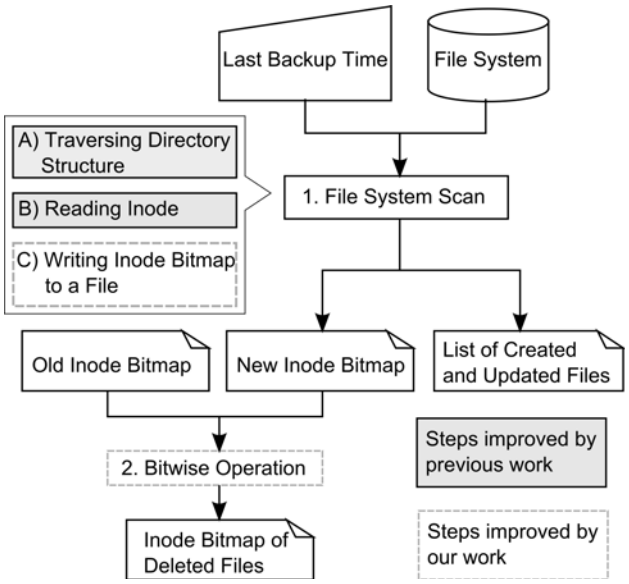

Fig. 2. Flowchart for finding created, updated, and deleted files by our method

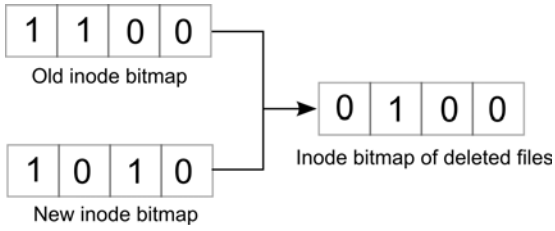

Fig. 3. Calculating an inode bitmap of deleted files

new and the old inode bitmaps. As shown in Figure 3, there are four conditions depending on values set to a particular bit in the new and the old inode bitmaps.

”0” is set to both the old and the new inode bitmaps
: A file with this inode number did not exist at the previous backup time and does not exist now. We set ”0” to the inode bitmap of deleted files.

”0” is set to the old and ”1” is set to the new inode bitmap
: A file with this inode number did not exist at the previous backup time but exists now. Therefore, the file was newly created. We set ”0” to the inode bitmap of deleted files.

”1” is set to the old and ”0” is set to the new inode bitmap
: A file with this inode number existed at the previous backup time but does not exist now. Therefore, the file was deleted. We set ”1” to the inode bitmap of deleted files.

”1” is set to the old and the new inode bitmaps
: A file with this inode number existed at the previous backup time and still exists now. There are three possible situations in this case: a file may not have been updated since the previous backup time, the file may have been updated, or the file may have been deleted and created with the same inode number. If

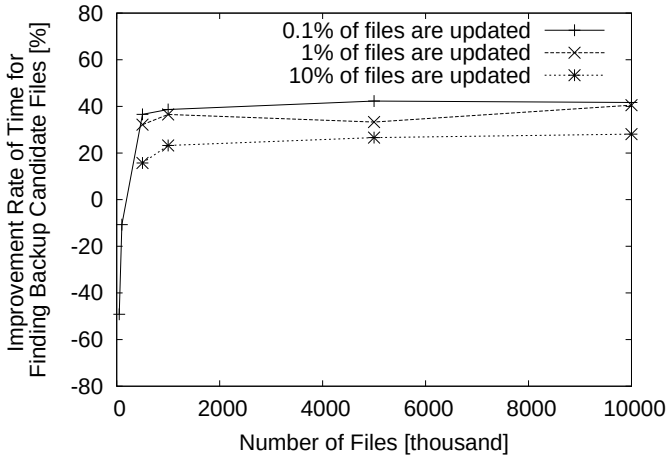

Fig. 4. Improvement rate of time for finding backup candidate files when average file path length is 64 bytes

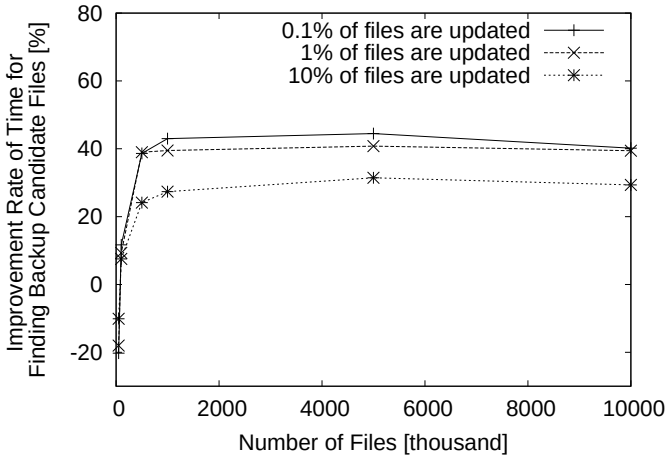

Fig. 5. Improvement rate of time for finding backup candidate files when average file path length is 128 bytes

the file has not been updated or has been updated, that means the file has not been deleted. If the file has been deleted and created with the same inode number, the file will be included in the list of created and updated files. Therefore, the file data will be copied to backup media, and this file does not need to be deleted. We set "0" to the inode bitmap of deleted files.

For example, in Figure 3, the third bit from the right is "1" in the old inode bitmap and "0" in the new inode bitmap. Therefore, we assume this inode has been deleted since the previous backup and does not exist now. As a result, "1" is set to this inode in the inode bitmap of deleted files.

The inode bitmap of deleted files D can be calculated by

$$D = O \wedge (\neg N) \tag{1}$$

where O is the old and N is the new inode bitmap. We reference the inode bitmap of deleted files and mark those files as deleted so that they will not be restored. For example, if we use IBM Spectrum Protect as backup and restore software, we look up a file from an inode number of deleted files and mark them as deleted. Created and updated files are copied to backup media by referencing the list of created and updated files.

V. Evaluation

We implemented our method with the Scalable Backup and Restore (SOBAR) function supported by Spectrum Scale 4.2 and evaluated the performance of the previous work [2] and our method in the test environment described in Section IV-A. SOBAR is a backup function of a file system with Hierarchical Storage Management (HSM) enabled [8]. HSM continuously copies files stored in disks to tapes. When the SOBAR is used, it assumes file data are already backed up by HSM and creates a backup of inodes and the directory structure of the file system. By taking a backup of inodes and directory structures without copying file data, SOBAR works efficiently compared to existing backup methods.

Since SOBAR skips copying file data to backup media, the step for finding backup candidate files takes up the most amount of time during an incremental backup. Therefore, improving the performance of backup candidate file selection is important. For example, during our tests, the maximum size of inodes and directory structures was 3.6 GB. The amount of time for taking a backup of inodes and directory structures can be estimated at 10 seconds using the IBM TS1150 tape drive whose maximum data transfer rate is 360 MB/sec [9]. Since the maximum amount of time for finding backup candidate files was 529 seconds, the step for backup candidate file selection took up 98.8% of the time during an incremental backup.

Because of the limitation of our test environment, we created up to ten million files in a file system and measured the performance of a backup by the previous work and our methods. In a high performance computing environment, 70% of files had a 64 to 128 byte file path length [10]. Therefore, we created files with 64 and 128 byte file path lengths and used them as test data. Then, we measured the time to find backup candidate files while updating 0.1%, 1%, and 10% of files in the file system. Figure 4 and 5 show our test results. With a 128 byte file path length, when 0.1% of files are updated, the performance of finding backup candidate files improved up to 44.5%.

Finally, when the number of files in a file system is small, we found that the size of the intermediate file is smaller when using the previous work compared to the size of the file using our method. As a result, the previous work showed better performance. For example, with a 64 byte file path length, when there are 50,000 files in a file system and 1% of files are updated, the previous work is faster by 70.8% compared to our method. However, with 50,000 files in a file system, our method took just a few seconds longer time than the previous work because a backup completes in a few seconds.

VI. Conclusion and Future Work

We focused on the fact that the size of intermediate files gets larger as the number of files in a file system grows and proposed a method to write an inode bitmap to a file and use it as an intermediate file of a backup to improve the performance of incremental backups. We use a bitwise operation of two inode bitmaps to find deleted files since the previous backup. Our method is implemented with a backup function called SOBAR, and we verified that the performance of backup candidate file selection improved up to 44.5% in our test environment.

Next, we will test our method with 10 billion files and compare its performance with the previous work.

References

[1] R. H. Patterson, S. Manley, M. Federwisch, D. Hitz, S. Kleiman, and S. Owara, "Snapmirror: File-system-based asynchronous mirroring for disaster recovery," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, ser. FAST '02. USENIX Association, 2002.

[2] M. A. Kaplan and W. A. Sawdon, "Scalable file management for a shared file system," 2012, US Patent 8,892,531.

[3] T. Bisson, Y. Patel, and S. Pasupathy, "Designing a fast file system crawler with incremental differencing," *SIGOPS Oper. Syst. Rev.*, vol. 46, no. 3, pp. 11–19, Dec. 2012.

[4] IBM Knowledge Center, "Tuning backups with the mmbackup command," http://www-01.ibm.com/support/knowledgecenter/SSFKCN_4.1.0/com.ibm.cluster.gpfs.v4r1.gpfs100.doc/bl1adm_backuptuning.htm.

[5] N. C. Hutchinson, S. Manley, M. Federwisch, G. Harris, D. Hitz, S. Kleiman, and S. O'Malley, "Logical vs. physical file system backup," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. USENIX Association, 1999, pp. 239–249.

[6] R. F. Freitas, J. Slember, W. Sawdon, and L. Chiu, "GPFS scans 10 billion files in 43 minutes," IBM Research, Tech. Rep., 2011.

[7] J. Bentley, *Programming Pearls (2nd Edition)*. Addison-Wesley Professional, 1999.

[8] IBM Knowledge Center, "Scale out backup and restore (SOBAR)," http://www-01.ibm.com/support/knowledgecenter/SSFKCN_4.1.0/com.ibm.cluster.gpfs.v4r1.gpfs100.doc/bl1adm_sobar.htm.

[9] IBM Knowledge Center, "3592 tape drives," https://www.ibm.com/support/knowledgecenter/en/STQRQ9/com.ibm.storage.ts4500.doc/ts4500_ipg_drives_3592.html.

[10] Y. Wang, "A statistical study for file system meta data on high performance computing sites," 2012.