International Journal of Smart Computing and Artificial Intelligence
International Institute of Applied Informatics
2017, Vol.1, No.1, P.1 - 19

Directory-Aware File System Backup to Object Storage for Fast On-Demand Restore

Jun Nemoto [*], Atsushi Sutoh [*] ,
Masaaki Iwasaki [*]

Abstract

With emerging "pay-as-you-go object storage" services, file system backup to object storage is an attractive option in terms of total cost of ownership. However, if a failure or a disaster occurs, a conventional backup system with object storage requires a long time to recover the necessary data since it must restore the entire directory structure of the file system from object storage before accessing the data. In this paper, "directory-aware backup" (DAB), which achieves on-demand restore as part of file system backup to object storage, is proposed. On-demand restore is a function that restores only a certain directory or file according to the request from an end user. The proposed method backs up relationships between a file and an object on a directory basis in order to efficiently handle the relationships in the restore. It is experimentally shown that the proposed method reduces the response time of file access involving restore by over an order of magnitude.

Keywords: File system backup, On-demand restore, Object storage

1 Introduction

File servers and network-attached storage (NAS) are widely used for sharing files among enterprise systems. In the case of such systems, data protection is a necessary functionality for enterprises in order to minimize the impact on their business when a system failure occurs.

Remote backup technologies for protecting data in file storage have been well-studied [1]. Using data backed up to a remote file server by means of such technologies, a business can recover from a failure quickly.

With emerging "pay-as-you-go object storage" services, file system backup to object storage is attracting more and more attention [2]. Compared to other backup systems, total cost of ownership can be reduced by using object storage services such as Amazon S3[3] as a backup storage[i].

[*] Hitachi, Ltd., Tokyo, Japan

However, it is hard to resume a business quickly by using a conventional backup system with object storage since it requires a long time to restore the directory structures necessary for accessing the file system. This is because a conventional backup system must restore the entire directory structure of the file system from object storage before accessing the desired data [2].

In light of the above-described issues, a new backup method—which enables a function called "on-demand restore" for resuming a business quickly—is proposed in this study. On-demand restore restores files and directories specified by end-user requests prior to less important ones. Utilizing on-demand restore, end users can restart their business without waiting for complete restoration of all directory structures in the file system.

The proposed method backs up each directory structure as a single object while keeping references between files and directories. It thus reduces the amount of restore data needed to access files requested by end-users.

In this study, the proposed method was implemented on a proto-type system and experimentally demonstrated on-demand restore. In addition, it was found to reduce the time to backup and restore the entire file system. As the experimental environment, file storage and object storage in a local area network was used.

The reminder of this paper is organized as follows. Section 2 discusses related works. Sections 3 and 4 present the design and implementation of the proposed method. Section 5 evaluates the effect of the proposed method, and Section 6 provides conclusions and future works.

2 Related Works

Storing file system data in object storage has been well-studied. For instance, Venti [4] and HydraFS [5] store files and directories of file systems in "content-addressable storage" (CAS), which is an object storage system for storing data that can be retrieved by calculating its address from its content. These systems provide a full restore using a snapshot mechanism, but they do not consider on-demand restore.

Cumulus [2], which is a backup system for file systems, can use object storage services such as Amazon S3 [3] to store backup data. To reduce network communication for storing a large number of files in remote object storage, Cumulus aggregates multiple files into a single segment (which is a unit for backup). It also groups together the directory structure and file attributes. However, on-demand restore cannot be achieved with the backup format in Cumulus because it cannot identify an arbitrary file until the directory structures of a whole file system are restored. The backup format used in Cumulus are described in detail in Section 3.

A "cost-aware cloud backup system" (CAB) has also been proposed [6]. CAB is similar to Cumulus in that it reduces the amount of network communication by aggregating multiple files into segments for efficient backup. It adopts a segment structure suitable for update operations, so it can provide file access through multiple devices. However, CAB cannot provide on-demand restore for the same reason as for Cumulus.

UniDrive also adopts a similar approach to Cumulus in backup formats [7]. UniDrive uses a single metadata file to eliminate the maintenance of massive tiny metadata files and reduce the metadata overhead in the multi-cloud and multi-device synchronization. As UniDrive evaluates only a few hundred to a thousand of files in the experiments, it mainly focuses on personal use. For enterprise use, there may be hundreds of millions of files per file system. Therefore, in such applications, a single metadata file is inefficient in differential backup and it cannot also realize on-demand restore.

“Fast and secure laptop backup” and MetaSync are the most similar approaches to the proposed method in backup formats [8][9]. Their approaches are same as the proposed approach in that a directory's structure is stored as an object. However, their approaches suffer a performance issue; that is, they need to back up all ancestor directories of a modified file because their backup formats use a hash value based on the content of the files to identify objects due to security reasons. These security problems are beyond the scope of this paper. However, we believe that the proposed approach provides a level of security equivalent to their approaches by encrypting the backed-up object in the object storage and storing the hash value of the object in the local file system (See Section 4.2). The hash value can also be used for avoiding redundancy of the objects. The proposed approach does not provide the hash-based de-duplication and it has room for improvement. Note that both [8] and [9] do not mention the concept of on-demand restore, introduced here.

Remote backup from file storage to other file storage has also been well-studied [1][10][11][12]. SnapMirror [1] provides an efficient asynchronous remote backup using a differential backup mechanism that identifies and transfers only modified files or data blocks when the file system updated. If a failure occurs in the local file server, services can be restarted by using the remote file server that holds the replicated files. However, all the data needs to be restored from the remote file server in order to restart the service at local sites. Rsync is a tool that provides an asynchronous remote backup like SnapMirror but it uses the different mechanism in detecting the update portion of the file. Rsnapshot and rdiff-backup are backup tools of point-in-time snapshots using rsync or the same algorithm of rsync. Using these tools, any point-in-time snapshots can be manually restored but the restore is not an on-demand fashion.

Although Panache [13] is not a backup system, it provides a function that is similar to on-demand restore. In the case of Panache, cluster file systems are deployed at both local and remote sites, and the local cluster file systems can be used as a cache of the remote cluster file system. Panache is similar to the on-demand restore proposed in the present study in that it acquires a file in an on-demand fashion when an end user requests a file that does not exist in the local cache. However, Panache uses the file storage for both local and remote sites; thus, considering the backup formats is not necessary in Panache, though it is an important issue for file system backup to object storage.

As a summary, there is no research that achieves both on-demand restore and efficient file system backup to object storage.

3 Design

The design of a backup format that achieves on-demand restore is explained as follows.

3.1 System architecture

The architecture of the proposal backup system is shown schematically in Figure 1. The backup system consists of file storage and object storage.

The file storage provides file sharing with standard protocols, such as NFS [14] and CIFS [15]. It stores files and directories in the local file system when it receives requests from the clients. These files and directories can be shared among end users. Note that files and directories stored in the file storage are simply called “files” if not specifically mentioned.

The object storage provides a function that stores, finds and deletes objects by the PUT/GET/DELETE method based on HTTP protocol [16].

Newly-stored or updated files in the file storage are backed up to the object storage according to a predetermined schedule or cycle (e.g., once a day at 2am) specified by system administrators. Backed-up files are then used for restoring if a failure or a disaster occurs.

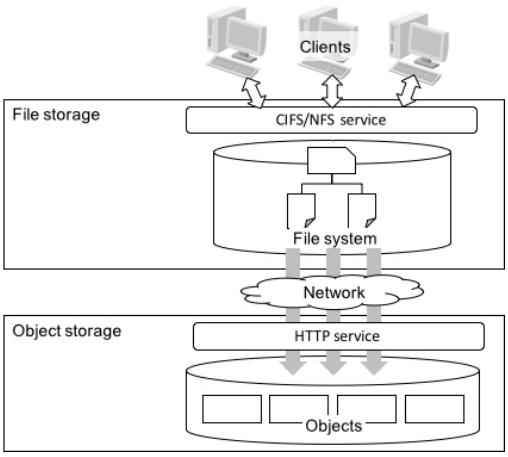

Figure 1: Overview of proposal system

3.2 Backup target

The proposed backup system backs up the following three items in the target file system.

(1) Namespace of the file system: directory structure of the file system.

(2) File data: data body of each file.

(3) File attributes: metadata of each file, namely, file size, timestamp (atime, mtime, and ctime), owner, group, UNIX permission, and ACL.

3.3 Prior art and challenges

Cumulus [2] is a backup system with the architecture shown in Figure 1. The backup format of Cumulus is shown in Figure 2.

Cumulus backs up the namespace of a file system and file attributes as "metadata log". The metadata log is a list of files stored in the file system. Each entry in the metadata log consists of the absolute path of a file, the object ID for the data body, and the file attributes.

Cumulus aggregates multiple entries into a fixed-size object (e.g., 4MB). Specifically, it packs entries into an object in the order of directory scan, and creates the next object when the first object reaches full capacity. Therefore, the number of metadata log objects and the number of entries in an object depends on the size of the object, the length of the file path, the file attributes and so on. In the case of Figure 2, two metadata log objects are created for all files in the file system as an example.

Cumulus does not manage the relationship between these metadata log objects and the namespace of the file system. Thus, when restoring a specified file, Cumulus cannot identify which metadata log object has the ID of the file. As a result, it needs to get all the metadata log objects, restore the directory structure of the entire file system, and finally restore the specified file. For the same reason, Cumulus must recreate all the metadata log objects for each backup since it cannot determine which metadata log object the update of namespace corresponds to.

Cumulus is used mainly for full restore of an entire file system [2]. It has been reported that an efficient partial restore can be achieved by managing the metadata log in a database; however, a practical management method has not been proposed. CAB has the same issue because it must create a hash map before starting the partial restore based on the file chunk and the segment chunk map that correspond to the metadata log [6]. Therefore, to design a backup format in consideration of the relationship between the namespace of a file system and the metadata log objects is a key challenge for achieving on-demand restore.

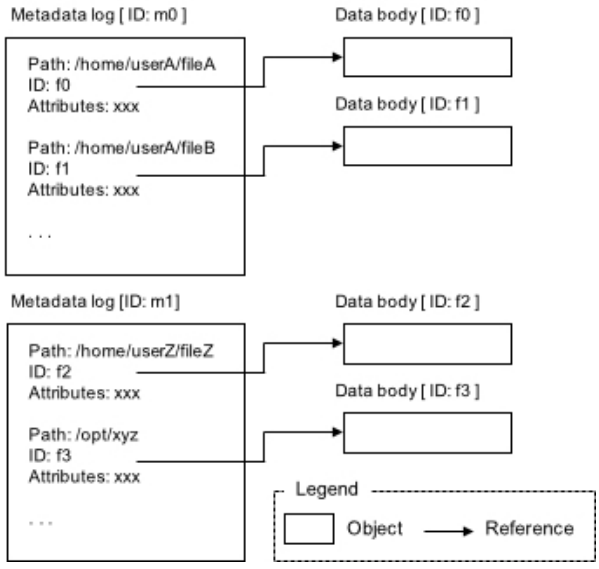

Figure 2: Backup format of Cumulus

3.4 Directory-aware backup (DAB)

To easily manage the relationship between the namespace of a file system and the metadata log objects, a concept called “per-directory backup of a metadata log” is proposed here. Hereafter, this backup and its format are referred to as “directory-aware backup” (DAB) and “DAB format,” respectively.

DAB format is overviewed in Figure 3. Each directory in the file system corresponds to a single metadata log object in DAB format. Each metadata log object consists of multiple entries, and each entry includes the name of the file that exists in the directory, the ID of the object that contains the data body of the file (or the metadata log in case of the directory), and file attributes. Unique and immutable IDs are used as object IDs because it can avoid re-backup of all ancestor directories unlike the content-based IDs as in [8][9] when a file is modified. Table 1 shows the data which is necessary to back up when various file operations are performed.

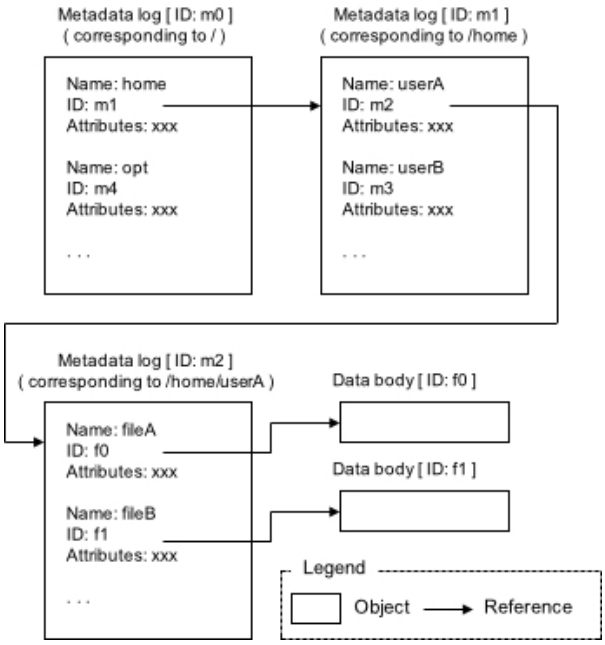

Figure 3: DAB format

Table 1: Backup data for each file operation

Operation	Cumulus	DAB
Create	File data and metadata log in which the entry of the file is included	File data and metadata log of the directory in which the file is created
Delete	Metadata log in which the entry of the file is included	Metadata log of the directory in which the file is deleted
Move	Metadata log in which the entry of the file is included	Metadata log of the source and target directory
Update file data	File data and metadata log in which the entry of the file is included	File data and metadata log of the parent directory of the updated file
Update file attributes	Metadata log in which the entry of the file is included	Metadata log of the parent directory of the updated file

DAB format can reduce the number of metadata logs that are required for restoring a certain file and achieve on-demand restore. For instance, in Figure 3, if /home/userA/file0 is accessed, only the directories that are necessary for restoring the specified file must be restored first by acquiring the metadata log objects m0, m1, and m2 in order.

However, compared to the backup format of Cumulus, DAB format increases the number of metadata log objects (depending on the directory structure of the file system). As a result, the number of HTTP requests also increases during full backup and restore. On the other hand, DAB can execute the backup and restore processes of the metadata logs much more in parallel than Cumulus. Cumulus does not consider carefully a range of the file system namespace in each metadata log object, thus does not provide the parallel execution. For example, if Cumulus divided the file system namespace into each child directory (and its descendent directories and files) of the root directory, it could process the metadata log objects in parallel. However, the number of threads would be limited in the number of the child directories and a detailed design would be necessary in order to increase the degree of parallelism in Cumulus.

4 Implementation

An implementation of DAB and on-demand restore is explained below.

4.1 Software architecture

A software architecture of the proposed backup system is overviewed in Figure 4. Each component is described as follows. File storage consists of an NFS/CIFS server, a data manager, backup/restore commands, and a file system (XFS [17]). These components run on Linux 2.6[ii].

The NFS/CIFS server provides file sharing services for clients. An ordinary NFS/CIFS server running on Linux is used without modification.

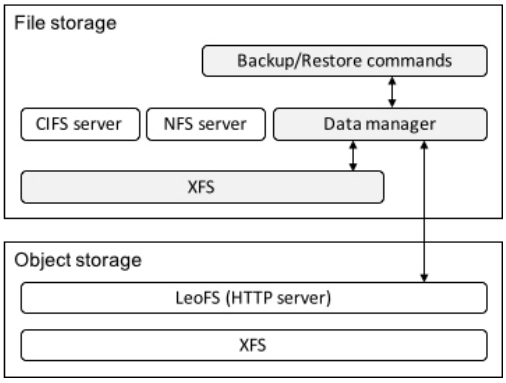

Figure 4: Software architecture

The data manager communicates with the object storage and performs backup and restore. Backup/restore commands are tools for starting the backup/restore processing.

XFS is used as a local file system of file storage, which is a modified version of a Linux 2.6 kernel. (The modification is described in detail later.) Note that it is assumed that the backup system is provided as part of a file storage appliance, and the kernel is modified to increase performance even though the backup system can be implemented as a user-space application using FUSE [18].

The object storage is provided by using an HTTP server, which have Amazon S3-compatible APIs. In this study, LeoFS [19] was used. Note that Linux 3.5 was used as the OS of the object storage and vanilla XFS in Linux 3.5 was used as a local file system.

4.2 Backup sequence

A backup sequence of the backup system is shown in Figure 5. First, the data manager traverses the file system and creates a list of new files and modified files when users execute backup commands. To create the list, XFS manages whether the file is modified or not and whether the file is backed up to the object storage in the inode that is a data structure used to represent a filesystem object.

Second, the data manager creates IDs for the files, which are used to identify objects in the object storage. UUID is used in the proposed backup system. After creation of IDs, the data manager issues a HEAD request to the object storage, and it confirms whether the ID exists or not. If it does not exist, the data manager stores the ID in the extended attribute of the file. Note that a special ID is used for the root directory for the restore process (which is described later).

Then, the data manager backs up files according to the file list. Each file is transferred as a single object by an HTTP PUT request in which the ID of the file is included in the URI. If the object storage returns a success, the data manager calculates the hash value of the file and compares it

with the hash value included in the response. If they match, the data manager stores the hash value in the extended attribute and changes the state of the file to "backup completed". Note that the hash value can be used for detecting alteration of the object in the object storage.

After all files have been backed up, the data manager backs up directories in the directory list. In the proposed backup system, an extended tar command is used to create the metadata log that is formed as an archive including name, ID, and attributes of all the files in the directory. The metadata log is created in a temporary area and then transferred as a single object by an HTTP PUT request in the same manner as the files. Note that ID creation and confirmation phase, the file backup phase, and the directory backup phase can be processed in parallel in each phase.

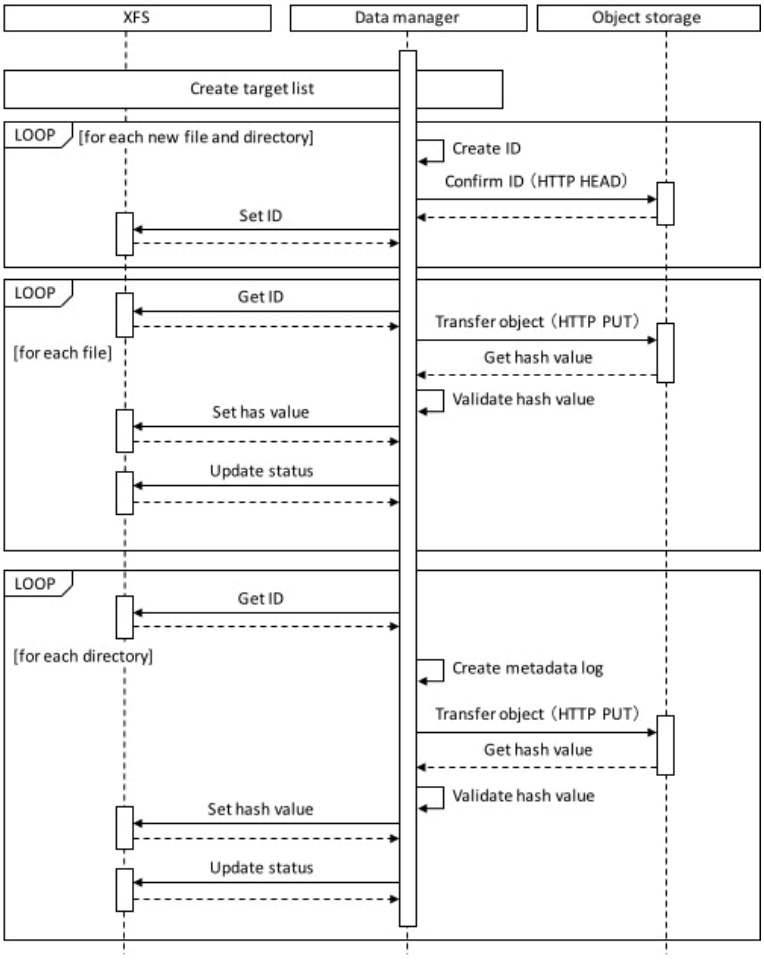

Figure 5: Backup sequence diagram

4.3 Restore sequence

A restore sequence for backup data with DAB format in the case of an empty file system is described as follows. When users execute the restore command, only the root directory is restored. After that, other files are restored on-demand according to client requests. Unaccessed files are also restored in the background for subsequent accesses.

A sequence of the directory restore in the whole restore process is shown in Figure 6. When the restore command is executed, the data manager requests the object including the metadata log by an HTTP GET request in which the special ID of the root directory is specified in the URI. The obtained metadata log is stored in a temporary area.

The data manager then extracts the metadata log in the target directory using the extended tar command in the same manner as the backup sequence. The extended tar creates empty files or directories based on the contents of the metadata log, restores only the attributes, and then sets the inode to "stub" state, where "stub" means a state in which the data body of the file is not yet restored. In the case the directory is in stub state, the files in the directory are not restored yet.

When all the files in the metadata log are processed, the restore of the root directory is completed. Then, the file access from the clients is enabled and the background restore is initiated.

If the clients access the directory in stub state, XFS issues a directory restore request to the data manager and accepts the access when the directory restore is completed. Unlike the root directory, the data manager gets the ID of the directory and the metadata log from the object storage using the ID. In case of an access to the file in stub state, the data manager gets the data body of the file instead.

Since only the root directory is restored first, and other files are then restored in an on-demand fashion, file services can be restarted in a short time.

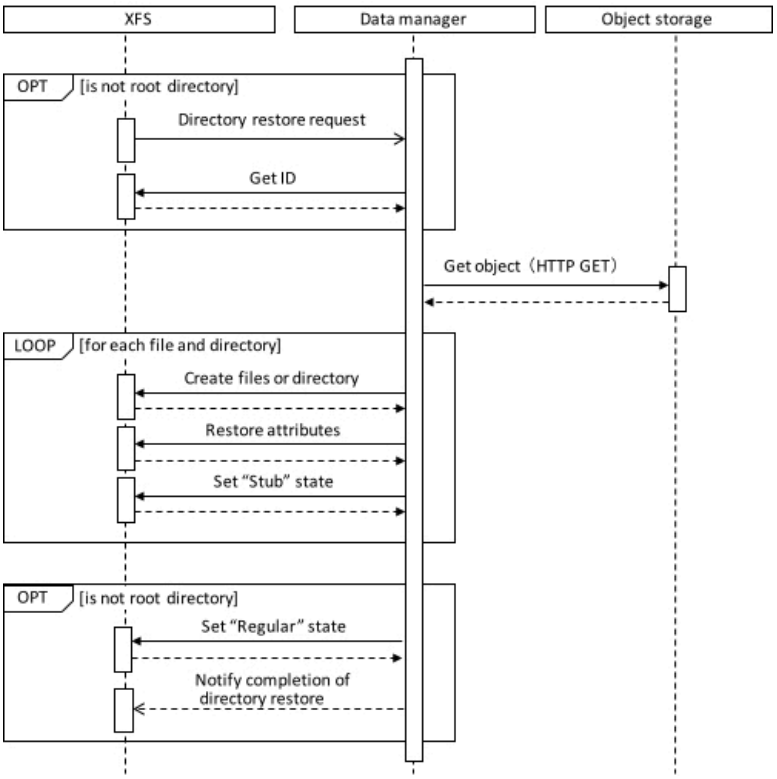

Figure 6: Restore sequence diagram

5 Experiments

Performance of backup/restore using the implementation described in Section 4 was experimentally evaluated in comparison with Cumulus [2].

5.1 Experimental setup

The following experimental environments were used: a file storage with a 2.0GHz quad-core Intel® Xeon® processor and 6GB RAM and an object storage with two 2.0GHz quad-core Intel® Xeon® processors and 10GB RAM[iii]. The file storage and object storage were connected by 1Gb Ethernet. As for storage volume, a RAID1 volume configured with two embedded 147GB SAS HDDs was used for the file storage and a RAID5 volume configured with four embedded 147GB SAS HDDs was used for the object storage.

5.2 Restore performance

5.2.1 On-demand restore

To evaluate the performance benefit of on-demand restore when a failure occurs, we first compared response times of file access and directory access. Response time of a “cat” command executed for a file in a directory in stub state is plotted in Figure 7. Since Cumulus cannot restore the files during transferring the objects and must get all the objects before executing the restore command, and it does not provide on-demand restore, a total response time (composed of the transfer time for all the metadata logs, partial restore time of the file, and execution time of the command) was used. The backed-up file system has 1000 directories under the root directory and each directory has 32KB files. The numbers of files in the directory are 10, 100, and 1000. As described in Section 3.3, Cumulus packs the entries of the files into a fixed size object in the order of directory scan. 4MB is used as the size of the object. The x-axis shows the number of files in each directory, and the y-axis shows response time(s) on a log scale. Note that both commands are executed while background restore of other files is performed.

As shown in Figure 7, Cumulus has a longer response time than the proposed system because it must restore all the metadata logs of the entire file system. It is also shown that in the case of Cumulus, response time increases with increasing number of files in the file system. On the contrary, in the case of the proposed system, response time increases with increasing number of the files in the directory.

Response time for a “ls -l” command executed for a directory in stub state is shown in Figure 8. As for Cumulus, a total response time, composed of the transfer time for all the metadata logs, partial restore time of the directory (except for restore time of the file body), and execution time of the command, was used.

The backed-up file system has 100 first-tier directories under the root directory, and each directory has 100 second-tier directories. Each second-tier directory has 100 files. The x-axis shows the depth of the directory, and other conditions are same as the case of the "cat" command.

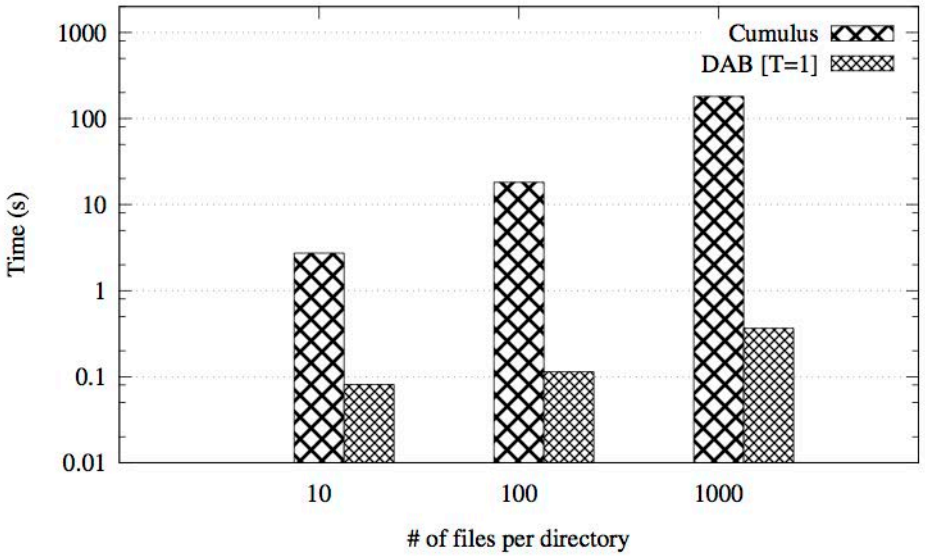

Figure 7: Command response time with "cat"

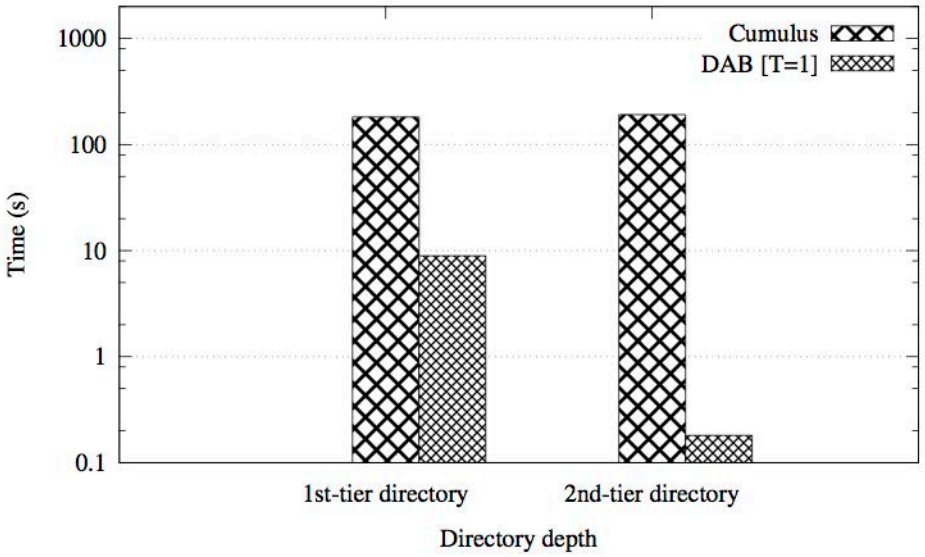

Figure 8: Command response time with "ls -l"

As shown in Figure 8, the proposed system has better performance than Cumulus in both cases. As for the proposed system, the second-tier case and the first-tier case differ because only the specified first-tier directory and second-tier directory must be restored in the former case, although the 100 directories in the specified first-tier directory must be restored in the latter case.

Although only 32KB files are used for both the “cat” command and the “ls -l” command, the difference between the proposed system and Cumulus would be smaller if the file size were larger than 32KB in the case of the “cat” command because the restore time of the data body would become dominant. On the other hand, even if a different file size were used for the “ls -l” command, the result would be the same as shown in Figure 8 because the data body of the file is not required.

5.2.2 Full restore

Next, to evaluate how the number of metadata log objects and HTTP requests affects the full restore, the full restore time (namely, the time to complete the restore of the entire file system) was measured (see Figure 9). The backed-up file system has 1000 directories under the root directory, and each directory has 32KB files. Number of the files is varied. And number of threads (T) in the proposed system is varied from 1 to 200, though the full restore with Cumulus is performed with only one thread because Cumulus does not provide multi-thread execution.

As shown in Figure 9, except for the case of 1000 files per directory, Cumulus is superior to the proposed system for single-thread configuration because Cumulus can get and restore multiple files in a single request. However, the difference in the number of communication packets is quite small because the amount of transferred data is almost the same as that mentioned in Section 5.4. Thus, the overhead for the number of HTTP requests is canceled with multi-threads, and the proposed system with multi-threads outperforms Cumulus.

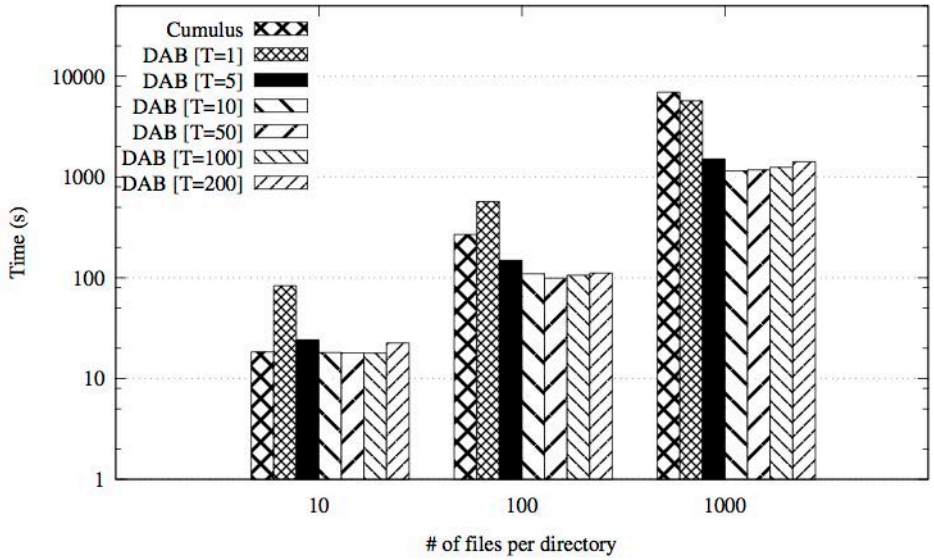

Figure 9: Full restore time

5.3 Backup performance

To evaluate the effects of number of metadata log objects and HTTP requests on full backup, the full backup time was also measured.

5.3.1 Full backup

Backup times to back up all the new files and directory with Cumulus and the proposed system are plotted in Figure 10. The configuration of the file system is the same as that used in the full restore experiment previously described. Number of threads (T) was varied from 1 to 200 in the same manner as full restore. Note that target files with 1000 threads were created to avoid being allocated disproportionately.

As shown in Figure 10, full backup shows the same tendency as full restore. Cumulus shows better performance in the case of single-thread configuration. However, the proposed system outperforms Cumulus in the case of multi-thread backup.

5.3.2 Differential Backup

Differential backup time, while the "ratio of updated files" was varied from 10% to 100% with the same file system configuration as for full backup, was measured next. The ratio of updated files means the proportion of files that are updated in each directory. The number of files in each directory is 100, and the number of threads in the case of the proposed system is 1 or 200.

The times for differential backup are plotted in Figure 11. In the case of a single thread, Cumulus is superior to the proposed system, regardless of the ratio of updated files. The difference between the lower percentages is far larger due to the disk I/O overheads. It is necessary to back up the attributes of non-updated files to create the metadata log with the DAB format; therefore, disk IO overheads appeared in the case of lower update rates. However, these overheads would be canceled in the case of multi-threads configurations with higher update rates.

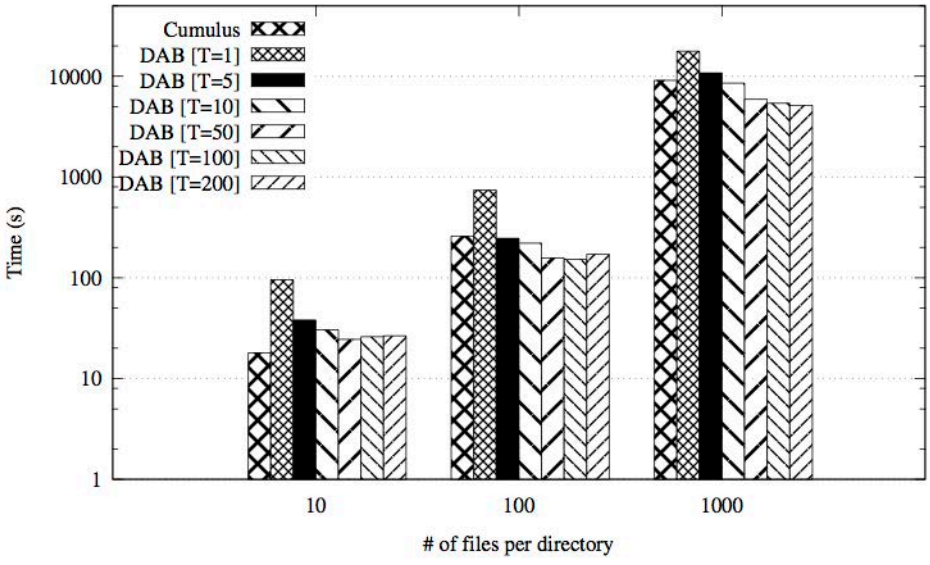

Figure 10: Full backup time

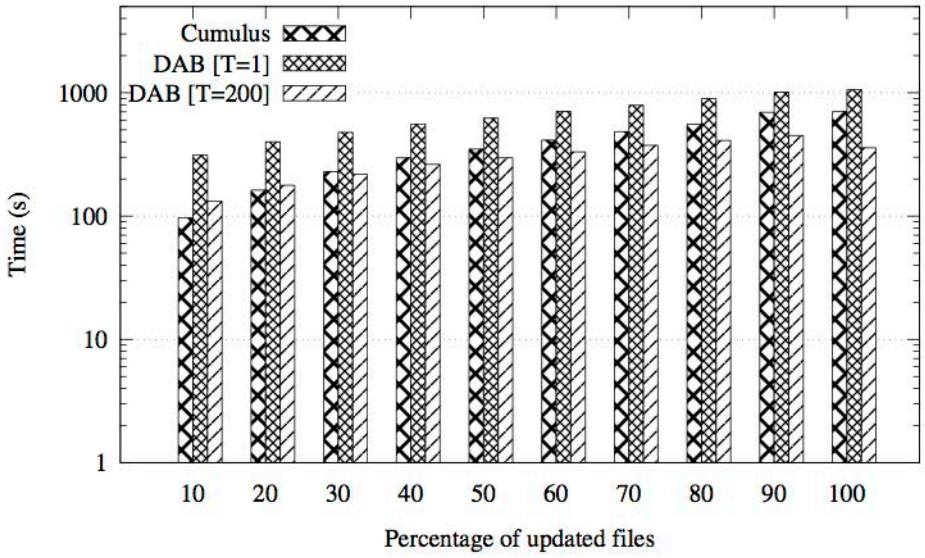

Figure 11: Differential backup time

Table 2: Number of HTTP requests and transferred data

	Number of files	Cumulus	DAB
HTTP requests in full backup	10	82	22,001
	100	792	202,001
	1000	7,939	2,002,001
HTTP requests in full restore	10	81	11,001
	100	791	101,001
	1000	7,938	1,001,001
Transferred data (GB)	10	0.32	0.37
	100	3.11	3.11
	1000	31.25	32.47

5.4 Number of HTTP requests and transferred data

Number of HTTP requests and amount of transferred data in the cases of full backup and restore (described in the previous sections) are listed in Table 2. The methods used for counting HTTP requests were GET, PUT, and HEAD.

As shown in Table 2, Cumulus keeps a lower number of HTTP requests because it can aggregate multiple files in a single segment. Thus, Cumulus has an advantage regarding monetary costs if a

pay-as-you-go service like Amazon S3 is used for the backend object storage. However, the amount of transferred data is almost the same in the case of both systems. As shown in the evaluation of full backup and restore, the time of backup and restore does not increase as much as the number of HTTP requests. It is thus concluded that the number of HTTP requests does not affect the performance very much.

6 Discussion

Related works that are similar to the presented work are classified into the following two categories depending on whether it considers the namespace of a file system (ie, the directory structures) in creating the metadata log.As for future work, since file storage and object storage in a local area network were only focused on in this study, it is necessary to evaluate the effect of network latencies on the performance of DAB in the case of a wide area network and make further improvements in such an environment.

(1) Non-DAB: non-directory-aware backup that creates the metadata log regardless of the namespace like Cumulus [2] and UniDrive [7].

(2) DAB with content-based addressing: directory-aware backup that creates the metadata log on a per-directory basis like MetaSync [9] and "Fast and secure laptop backup" [8]. Note that the scheme of assigning object IDs in these works is different from the proposed approach. These works use content-based addressing scheme but the proposed approach uses UUID independent of the content of an object.

Details of the difference from the proposed approach are explained as follows.

6.1 Non-DAB

Although Non-DAB formats cannot realize on-demand restore, full backup and restore are faster than the proposed approach since the metadata log can be transmitted and received at once (or few times). However, since there is almost no difference in the amount of data to be transferred, the difference can be reduced by parallel processing as described in Section 5.4. Note that the data transfer amount in Section 5.4 is the total capacity of the transferred objects, and the actual transfer amount would be larger by checking IDs. However, we think that it will not disturb the other services by pressing the network bandwidth since it is at most a few hundred bytes and is sufficiently smaller than the object size.

As for the differential backup, Non-DAB would increase the transfer amount of the metadata log, so the proposed method would be better. Furthermore, since Non-DAB obtains the attributes of all files, small random reads will increase, which may degrade the file service performance if the backup is executed while the file service is running. However, when the total number of directories in the file system is small and the total number of files is large, the difference between both DAB and non-DAB would be small. This is because it is likely to happen that the proposed approach also access almost all the files. Of course, it also depends on the update part.

As for on-demand restore, in the case where there are many files in the file system, the proposed approach is better regardless of the number of directory tiers. Since there is no relationship between the metadata log and the target object of the on-demand restore, Non-DAB approach cannot acquire only the metadata log of the specific tier and it must obtain and restore all the metadata log even if the number of tiers is small. As shown in Figure 7, if the total number of files is small, the difference between DAB and Non-DAB becomes small.

6.2 DAB with content-based addressing

The time of full backup, full restore and on-demand restore is the same between DAB and DAB with content-based addressing. Namespace backup time in differential backup will increase as backup of all the ancestor directories is necessary even if minor updates are occurred in the subordinate directories. In order to access the ancestor directories and attributes of the files, small random reads will increase, which may affect the normal file service. However, if the number of tiers and files in the ancestor directories is small, the difference would be noticeable.

The advantage of content-based addressing is that it is relatively easy to verify the integrity and to avoid redundancy of the objects. Because the ID is the hash value of the object, the integrity is simply verified by accessing it. Also, it is possible to avoid backing up the same object by checking whether the same ID exists in the object storage. On the other hand, the proposed approach, for example, must manage the hash values in the file server to achieve the same functionality.

7 Conclusion

A new backup method, called “directory-aware backup” (DAB), for achieving on-demand restore in file system backup using object storage, was proposed, and its performance benefits were demonstrated. It was experimentally shown that DAB reduces the response time of file access involving restore by over an order of magnitude.

As for future work, since file storage and object storage in a local area network were only focused on in this study, it is necessary to evaluate the effect of network latencies on the performance of DAB in the case of a wide area network and make further improvements in such an environment.

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 FAST, 2002, pp. 117–130.

[2] M. Vrable, S. Savage, and G. M. Voelker, “Cumulus: Filesystem Backup to the Cloud,” in FAST, 2009, pp. 225–238.

[3] Amazon S3; http://aws.amazon.com/s3/

[4] S. Quinlan and S. Dorward, “Venti: A New Approach to Archival Storage,” in FAST, 2002, pp. 89–101.

[5] C. Ungureanu, B. Atkin, A. Aranya, S. Gokhale, S. Rago, G. Całkowski, C. Dubnicki, and A. Bohra, “HydraFS: a High-Throughput File System for the HYDRAstor Content-Addressable Storage System,” in FAST, 2010, pp. 225–238.

[6] Y. Zhu and J. Masui, “Backing up your data to the cloud: Want to pay less?,” in ICPP, 2013, pp. 409–418.

[7] H. Tang, F. Liu, G. Shen, Y. Jin, and C. Guo, “UniDrive: Synergize Multiple Consumer Cloud Storage Services,” in Middleware, 2015, pp. 137–148.

[8] P. Anderson, “Fast and Secure Laptop Backups with Encrypted De-duplication,” in LISA, 2009.

[9] S. Han, H. Shen, T. Kim, A. Krishnamurthy, T. Anderson, and D. Wetherall, “MetaSync: File Synchronization Across Multiple Untrusted Storage Services,” in ATC, 2015, pp. 83–95.

[10] rsnapshot; http://www.rsnapshot.org/

[11] rdiffbackup; http://www.nongnu.org/rdiff-backup/

[12] A. Tridgell, “Efficient Algorithms for Sorting and Synchronization,” 1999.

[13] M. Eshel, R. Haskin, and D. Hildebrand, “Panache: A Parallel File System Cache for Global File Access.,” in FAST, 2010, pp. 155–168.

[14] B. Callaghan, B. Pawlowski, and P. Staubach, “RFC 1813: NFS version 3 Protocol Specification,” 1995.

[15] P. J. Leach and D. Naik, “A common Internet file system (CIFS/1.0) protocol,” Internet-Draft, IETF, 1997.

[16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee, “RFC 2068: Hypertext Transfer Protocol -- HTTP/1.1,” 1997.

[17] XFS; http://oss.sgi.com/projects/xfs/

[18] FUSE; https://github.com/libfuse/libfuse/

[19] LeoFS; http://leo-project.net/

---

[i] Amazon S3 is a trademark of Amazon.com, Inc. or its affiliates in the United States and/or other countries.
[ii] Linux is a registered trademark of Linus Torvalds in the United States and other countries.
[iii] Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.