Versioned File Backup and Synchronization for Storage Clouds

Shuang Qiu*, Jingyu Zhou* and Tao Yang†
*Shanghai Jiao Tong University, China
†University of California at Santa Barbara, USA

***Abstract*—Cloud storage has become widely used for data backup and archiving. The current archiving systems typically support a specific cloud service and this vendor lock-in problem can cause a challenge in data migration and even data loss when a cloud service provider ceases to exist. We propose a system called *RosyCloud* for automatic backup and synchronization of client data on different clouds. RosyCloud uses a thin interface of storage cloud through HTTP requests, keeps all encrypted file versions on the cloud, and allows different devices to asynchronously synchronize with the cloud. Conflicts due to concurrent writes are detected and resolved using a DAG model. RosyCloud also supports secure file sharing among different users. We have implemented the RosyCloud prototype with three popular storage clouds. Experiments show that RosyCloud is effective for both backup and synchronization with low monetary cost.**

I. Introduction

Cloud storage services have attracted many organizations and individuals to put their data in the cloud [24, 5, 25]. Comparing to local drives, cloud storage offers higher availability, unlimited space, and lower cost.

Storing data on specific cloud storage is convenient, but is vulnerable too. Dependence on specific protocols and tools of a cloud vendor may make future migration costly and difficult. The shutdown of Megaupload [2] demonstrates that although service providers can guarantee five-nines of availability, there is a possibility that users may still lose their valuable digital assets if some of them evaporate.

Previous work [5] has addressed the vendor lock-in problem by distributing data on multiple clouds and the focus is mainly on data backups. However, typical users often not only need to back up data, but also want the devices' data synchronized with the cloud. Existing tools such as Dropbox [1] make it possible to synchronize data on different devices, but have the same vendor lock-in problem. Thus, it is desirable to have a tool that backs up data to multiple storage clouds and that supports synchronization for different devices.

Building such a tool is challenging. Storage clouds do not have the capability of executing users' code, and can only be accessed using their standard interfaces, which lack atomic operations. As a result, it is difficult to synchronize writes from multiple asynchronous devices, unless complex locking protocols are applied [5]. A problem with locking is that one device holding the lock prevents others from writing, which delays backup and synchronization operations. Considering devices may disconnect from the cloud from time to time, this problem can become worse.

Corresponding author: Jingyu Zhou (jingyuzhou@sjtu.edu.cn).

We propose *RosyCloud*, a tool that orchestrates versioned file backup and synchronization among different clouds and end devices. Specifically, RosyCloud hosts versioned file backups on multiple storage clouds, avoiding vendor lock-in and providing high availability for files. Files can be updated simultaneously and all the modifications are versioned, while synchronization between devices and clouds is periodically performed, with automatic conflicts detection and resolution based on a directed acyclic graph (DAG) model. To protect users' data from network attacks and misbehaved clouds, RosyCloud encrypts all data stored in the cloud and provides secure file sharing among different users. We have implemented a RosyCloud prototype for three popular storage cloud services. Experimental results show that with very low cost, RosyCloud can effectively backup and synchronize data for a typical office workload.

The contributions of this work include: 1) an automatic synchronization mechanism using storage clouds with a thin interface and no server computing requirements, and 2) a DAG-based snapshot dependence model for conflict resolution.

The remainder of this paper is organized as follows. Section II summarizes related work. Section III gives the design of RosyCloud and Section IV presents the implementation. Section V evaluates the performance of RosyCloud. Finally, Section VI concludes with future work.

II. Related Work

Backup is a widely accepted practice to prevent data loss [18, 8, 27]. The advance of cloud computing makes cloud storage an ideal medium for data backup and archiving. Cumulus [24] implements a backup system over Amazon S3 [3] cloud storage, with a thin-cloud assumption. To improve service availability, backing up with multiple clouds is studied in [6, 25]. RosyCloud not only performs file backup, but also synchronizes files on multiple devices, which is not a goal in all previous cloud backup systems.

DepSky [5] is a backup system with multiple general storage clouds as backends. Files in DepSky are stored with version numbers. A set of Byzantine quorum protocols is employed to implement a single-writer, multiple-reader data

model. To support multiple writers, a complex file locking protocol is used to serialize concurrent write operations. Compared to DepSky, synchronization in RosyCloud is more efficient because RosyCloud uses content hash for versioning instead of version numbers to avoid expensive write synchronization.

File synchronization has traditionally been achieved in a centralized manner. Most source control systems, e.g., SCCS [16], CVS [4], Subversion [17], Perforce [29], take this approach, where a centralized server ensures synchronized access to the file repository and uses version numbers to label different updates. If the server crashes or disconnects, the repository is unreachable for all clients.

To address the problem of a centralized repository, many distributed version control systems have been developed [14, 30], where a user keeps a local replica of the repository and can synchronize the local replica with other users or other servers. Our work is inspired by these distributed version control systems. Similar to Pastwatch [30], revision history in RosyCloud is organized as a tree with forks and *write-write* conflicts are detected and resolved. The difference is that in the RosyCloud model, the cloud storage servers are passive and they only provide basic file accesses without any extra computing requirements. This broadens the use of RosyCloud for different public cloud services in today's market. But without server computing capability, ensuring automatic updates becomes more difficult [5] and we have to rely on a client to perform common version control operations such as hashing, `diff` or comparison.

Commercial products such as Dropbox [1] provide file backup, sharing, and synchronization on multiple devices across diverse users. These products use dedicated cloud storage to store users' data and have dedicated servers [26]. In contrast, RosyCloud is designed to support different storage clouds using a thin interface and without imposing server computing requirement. This design avoids the problem of vendor lock-in. However, without dedicated servers deployed on cloud side, RosyCloud needs to perform extra computation and synchronization logic at devices.

Finally, distributed file systems [11, 19, 9, 22, 23, 21] provide a device-transparent, scalable storage with a file system interface. In this sense, RosyCloud differs from them by residing in a lower level of the storage stack (more close to block devices), and provides a limited set of file operations. As a result, RosyCloud can be implemented in a fully distributed paradigm without centralized metadata servers. Coda [21] supports disconnected file modifications and detects *write-write* conflicts, which are presented to the users for resolution. In contrast, RosyCloud uses a DAG-based scheme to resolve conflicts.

III. Design

In RosyCloud, a user can set up a directory in each device, such that RosyCloud backs up data in the directory to multiple clouds and synchronizes data across these devices. Clouds store all file versions, while a device only has a snapshot of a user's files.

RosyCloud assumes cloud storage services expose a minimum HTTP-based interface:

- **store:** creates a new object;
- **remove:** deletes an object;
- **retrieve:** reads the content of an object; and
- **list:** returns a list of object identifiers.

The rest of this section first discusses the data model of RosyCloud, and then presents backup and synchronization procedures. Finally, it discusses file sharing.

A. Data Objects in RosyCloud

All data stored on the cloud side in RosyCloud are objects, and can be categorized into metadata objects and file objects. Each object is indexed by a 32-byte string, hexadecimally encoded from 128-bit MD5 hash of the object content. The probability of hash collision is small enough to be ignored in practice.

There are three types of objects: metadata objects, i.e., `Dir`, and `Snapshot`, and file objects. They are discussed as follows.

- `Dir`. A `Dir` object metaphors a physical directory in a traditional file system, which is filled with a list of directory entries, called `DirEntry`. Each `DirEntry` in a `Dir` object represents a file or a directory and contains all metadata necessary to describe the file or directory.
  Table I summarizes all metadata fields in a `DirEntry` structure in RosyCloud. The `Mode` field indicates if an object is a file or a directory. If the entry points directly to a file, `Size` field records the size of the file. Otherwise, the entry refers to another `Dir` object with zero in the `Size` field. `Name` represents the real file name used in clouds, which is the hexadecimally encoded 128-bit MD5 hash of object contents. `File Name` is the file or directory name on end devices. The `Encryption Key` field is the key used to encrypt the corresponding object, which is a randomly generated 128-bit key for each object in order to increase security. Keys are stored inside `DirEntry` objects such that they are encrypted recursively by parental keys and can be retrieved during directory traversal. This design eliminates the requirement of a separate key hoard for storing all encryption keys, which may become sizable and costly to maintain for a large number of keys.
- `Snapshot`. A `Snapshot` object represents a file system state at a specific time, which can be later reverted to. Once created, a snapshot is immutable. Any changes to the file system result in a new snapshot, which refers to the previous snapshot as a parent.
  Figure 1 illustrates the contents of a `Snapshot` object. A snapshot object contains a `DirEntry` structure

Table I
DESCRIPTION OF FIELDS IN A `DirEntry` OBJECT.

Field	Description
`Mode`	Type of entry, a file or a directory.
`Size`	Size of the file, 0 for directory.
`Name`	The identifier of a content-holding storage object in cloud for the file/directory.
`File Name`	File or directory name.
`Encryption Key`	Key to encrypt object.

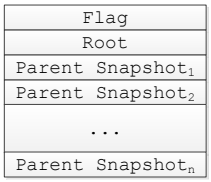

Figure 1. A `Snapshot` Object.

indicating the root directory. To detect and resolve conflicts, a snapshot should be able to position itself in the revision history of the file system. So references to parent snapshots, from whom current snapshot is derived, are also included in a snapshot object. Additionally, some flags can be associated with a snapshot. Evolution of snapshots forms a DAG as discussed in Section III-C) Conflicting updates may exist with multiple sink nodes in the graph and RosyCloud tries to resolve such a conflict.

- *File Object*. A file in RosyCloud is stored as a separate object in each cloud. Previous work [24, 25] organizes files into blocks and aggregates small files into bigger segments. However, maintaining blocks for a file is not efficient for backup restoration and file synchronization. This is because a file's blocks can be spread into many segments, which means that retrieving the file data may require fetching multiple segments, potentially wasting time and bandwidth for retrieval. As a result, RosyCloud chooses not to use internal data blocks as basic backup units and doesn't aggregate small files.

B. Backup

RosyCloud automatically backs up files to all storage clouds. Specifically, one storage cloud is designated by a user as the *primary*, which will receive all user backups and store all snapshots. Without the primary, the version DAG on a cloud may miss some snapshots, causing synchronization problems. Typically, the primary cloud is updated at a higher frequency than others. Users may choose to backup all snapshots on all storage clouds with the same frequency, which increases backup latency and cost.

RosyCloud doesn't require users to keep connected to cloud storage. Users are free to decide when to set the device offline and when to become online. During offline time, all modifications to the local file system are logged. Once connected to clouds at some later time, these changes will be synchronized with data in the clouds.

Backup is driven by the notification of file system changes [15, 13], which is available in most modern operating systems. When a notification is fired, RosyCloud filters write events and uploads the changed files to clouds. RosyCloud updates cloud objects by creating a new one with the stale objects unchanged. When the local file system notifies a modification, the entire object will be uploaded to the cloud instead of incrementally updating differences [16, 17, 24]. Although expensive in the sense that even one byte modification will lead to the whole file to be uploaded, this update strategy eases restoration and rollback. Given a version at a specific time, the object can be recovered without tracing snapshot histories and applying patches one-by-one.

Figure 2 illustrates the data dependence between snapshots, also directory and file objects of snapshots. Squares and circles represent directories and files, respectively. Snapshot 1 taken at a previous time is the parent of Snapshot 2. Snapshot 1 contains one $root$ directory and two files, $file_1$ and $file_2$, reside in dir_1, a subdirectory of $root$. At some later time, $file_2$ is updated, and a new version of the file is created. Thus, a new version of dir_1 directory object is also created. Snapshot 2 contains the updated root directory along with the new file objects and references to the unmodified files.

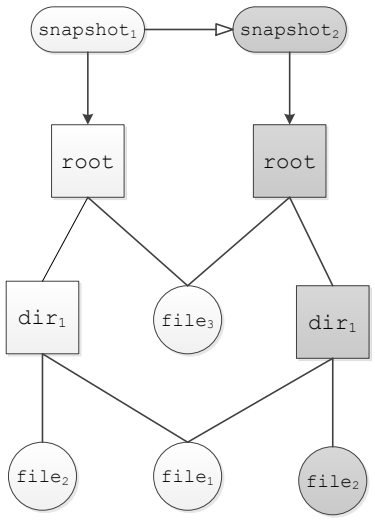

Figure 2. RosyCloud updates cloud side storage each time a modification occurs. Rounded rectangles, squares, and circles represent `Snapshot`, `Dir`, and `File` objects, respectively. The shaded ones denote newly created objects. The arrow between two snapshots indicates the derivation dependence relationship.

C. Synchronization

RosyCloud maintains a timer to synchronize the local file system with the primary storage cloud periodically. Because

backup and synchronization between devices and storage clouds are asynchronous, conflicts can happen and exhibit as forks in the snapshot DAG. We discuss how conflicts are detected and then describe how to automatically merge the conflicts.

1) Conflict Detection: Conflicts of files can easily occur when using multiple end devices, because a user editing a file on one device may not be aware of concurrent modifications to the same file from other devices.

When a device synchronizes with cloud storage, to detect whether a conflict exists, all snapshots already taken are traversed to construct a dependence DAG of snapshots. In this graph, each node represents a snapshot and an edge represents a data derivation relationship from a parent snapshot to its child snapshot. If there is only one sink node in this DAG, this node represents the latest version with a consistent image. If there exists more than one sink nodes, then these sink nodes represent conflicting snapshots and we discuss a resolution shortly.

RosyCloud builds the snapshot dependence DAG on the fly during synchronization. This is mainly because there is no cloud-side reference for the most recent snapshot. Even if there is such a file containing the reference, there are no atomic operations to update the file and complex locking protocols are required [5]. However, when using locks, a device holding the lock would prevent other devices from concurrently writing to cloud storage, which affects the performance. In contrast, our approach of immutable snapshots and late resolution of conflicts makes concurrent writes possible.

2) Conflict Resolution: For two conflicting snapshots n_x and n_y, the merge process creates a new node with two pseudo dependence edges to n_x and n_y respectively. The new node represents a snapshot after resolving conflicts between objects in n_x and n_y. To figure out if an object should stay in the new node representing merged results, we compute the lowest common ancestor (LCA) in the snapshot DAG with respect to n_x and n_y. The LCA is needed as a reference point to determine which object is updated from the LCA to n_x or to n_y, utilizing the file object relationship among snapshots. Then starting from two root directories of conflicting snapshots, merge is performed recursively down the directory hierarchy.

If a snapshot DAG has more than two sink nodes, the merge process for conflict resolution first chooses two with the smallest hash values, combines these two snapshots into a new node in the DAG, and continues until all conflicting snapshots are merged. Finally, when all conflicting snapshots are merged, the merged snapshot will be uploaded to clouds.

Figure 3 illustrates an example of the merging process. Initially, snapshot A contains file a and o. Snapshot B is produced by deleting a and snapshot C is generated after o is changed to o'. Clearly, B and C are conflicting snapshots. However, given only these two snapshots, we

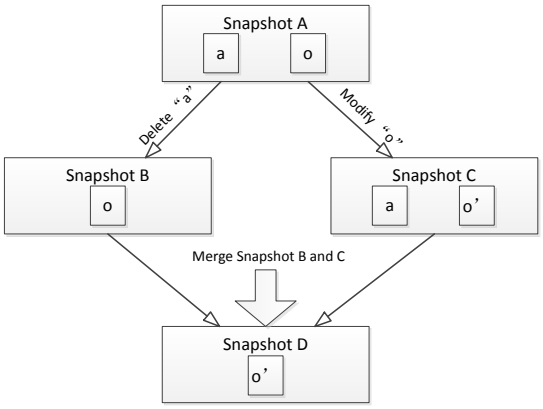

Figure 3. A merge example for conflict resolution.

cannot determine whether a is deleted in B or is created in C, and which one of o and o' is newer. Note we cannot rely on timestamps to determine which one is newer too, because the clocks of two devices may not be synchronized. These problems can be easily solved if we know A is the LCA of B and C. In this example, because a appears in both A and C, we know a is deleted in B. Similarly, we can determine o is modified in C. As a result, the merged snapshot only contains o'.

In the merging process, if the changed files in two snapshots are disjoint, then these two snapshots are silently merged using the changed file versions. For example, if two writes are performed on different files, then they can be safely merged. If two writes are applied to the same file, then user intervention is needed unless write changes are identical. In general, there are two cases during the merge requiring manual inspection:

- *Write-write conflict* happens when different changes are made to the same file.
- *Write-delete conflict* happens when one snapshot deletes a file while the other modifies the same file.

For the above two conflicting cases, two versions of files are kept and one version is renamed for users' further inspection. Specifically for *write-write* conflicts, one version of the file that has a smaller hash value is renamed by prefixing the file name with "MODIFY-CONFLICT-". For *write-delete* conflicts, the version removed from LCA is renamed by prefixing with "DELETE-CONFLICT-". Because multiple conflicting snapshots may exist, the prefix also includes the file hash to distinguish from each other.

Directory conflicts are different from file conflicts. Because merge starts from the root directory, a *write-write* conflict of a directory can be a *write-write* conflict of a file in the directory, or a sub-directory conflict that will be recursively merged, or a name conflict of a file and a sub-directory. In the last case, the conflicting file is renamed. For the *write-delete* type of directory conflicts, the deleted directory is renamed.

3) Device-Device Synchronization: In RosyCloud, devices do not directly synchronize files with each other. Instead, each device populates local changes to the primary and periodically synchronizes with the primary to obtain modified files from other devices. Thus a modification of the same file from one device can be propagated to another device through the primary cloud storage.

D. File Sharing

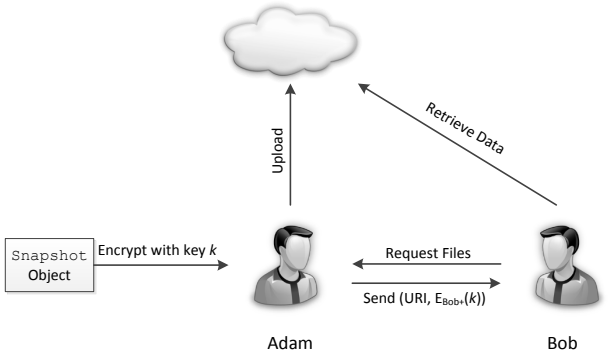

Figure 4. Sharing files between two users involves five steps. Arrows indicate snapshot derivation.

Snapshots facilitate file sharing in RosyCloud. Each object in a cloud is encrypted by a random 128-bit AES key. We assume that each user has a secure public and private key pair, and users have a secure way of verifying public keys. Figure 4 illustrates the five steps for file sharing.

1) Bob sends his public key to Adam when asking for files.
2) After Adam receives the request, he collects all files to be shared into a new snapshot and encrypts the snapshot with a random key k.
3) The encrypted snapshot is uploaded to the clouds.
4) Adam sends Bob the snapshot URI and key k encrypted with Bob's public key.
5) Bob decrypts key k and accesses the snapshot, decrypts the snapshot with k, and obtains access to shared files.

E. Discussions

All users' devices usually back up to and synchronize with the primary storage cloud. If the primary becomes unavailable (e.g., ceases to exist), RosyCloud may lose some of the snapshots. However, copies in other storage clouds exist and all up-to-date files are stored on end devices. After the user decides the new primary, all snapshots afterwards will be stored on the new primary and different devices will be synchronized again. Because the new primary may contain incomplete snapshots, the first time a device synchronizes with the new master may result in emerging of deleted files and some conflicts. However, no up-to-date files are lost at this time.

A possible problem is that when the synchronization process updates a local file, the user may be actively editing the file, resulting in a *write-write* conflict. For this type of conflict, RosyCloud postpones the update of the file until the next synchronization round, waiting for the file to be closed.

IV. Implementation

We have implemented a RosyCloud prototype on Linux platform in Python. Currently, RosyCloud supports three storage clouds, Alibaba Aliyun's OSS [1], Windows Azure [2], and Google Drive [3]. Because RosyCloud only uses the minimum storage interface, it is fairly easy to support these three clouds.

A. Overview

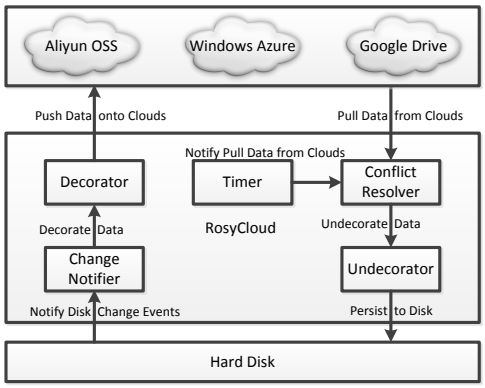

Figure 5. The architecture of RosyCloud.

As illustrated in Figure 5, data in RosyCloud flows in two asymmetric directions driven by two different events. The notification events of local file system changes initiate data backup to clouds. Periodically, timer events trigger synchronization between the local device and storage clouds. Modules in RosyCloud are:

- **Change Notifier** captures modification events in the file system of the local device, and then initiates the backup of modified files to different clouds. We use Linux inotify [15] API to watch file system changes. Because many inotify events do not change files, RosyCloud only responds to write events listed in Table II. The backup times are set right after each modification with a tunable time interval, where a small interval reduces the chance of conflicts but increases the volume of data to the cloud.
- **Decorator** performs data compression and encryption before uploading to clouds. Thus, cloud storage only saves decorated data, i.e., compressed and encrypted data. Currently, we use `bzip2` to compress data and

[1] http://www.aliyun.com/product/oss/
[2] https://www.windowsazure.com/en-us/
[3] https://drive.google.com/

Table II
INOTIFY EVENTS CAPTURED BY ROSYCLOUD.

Inotify Events	Description
IN_CREATE	Create a directory.
IN_DELETE	Delete objects.
IN_CLOSE_WRITE	Close updated files.
IN_MOVE_FROM	Move object out of the file system.
IN_MOVE_TO	Move object into the file system.

each file is encrypted using AES algorithm with a random 128-bit key.

- **Timer** triggers synchronization periodically. RosyCloud checks whether updates from other end devices are available on the clouds and downloads corresponding objects.
- **Conflict Resolver** detects and resolves update conflicts.
- **Undecorator** performs the reverse function of **Decorator** to compute the original data in clear-text form, and then saves the data on disks.

B. Utilities

Table III
UTILITIES PROVIDED BY ROSYCLOUD.

Name	Functionality
versions	List all available versions given a cloud and object name.
tag	Tag a snapshot with a memorable name.
xtract	Extract one version of a file.
grant	Grant or revoke file access privilege to another user.
share	Download other user's shared files to local disk.

RosyCloud provides a number of utilities facilitating the use, which are summarized in Table III. Utility `versions` lists all available versions of an object stored in a cloud. Utility `tag` associates a snapshot identifier with a memorable name, instead of a 32-byte hash value. Another utility, `xtract`, extracts a versioned file into the local file system. Finally, `grant` and `share` help share files among different users. `grant` is responsible to grant and revoke access privileges, and `share` downloads shared files from other users to a local disk.

C. Optimization

To reduce bandwidth usage, all metadata objects, i.e. `Dir` and `Snapshot` objects, are cached locally, because they will be intensively referenced when building the snapshot DAG and reconstructing the file system structure. Because synchronization is performed periodically and snapshot DAG is constructed each time, each device also caches the DAG structure and only updates the DAG with new snapshot objects.

To reduce network traffic, we only upload a file to clouds when it is closed rather than each time `write` is called. For temporary files, such as the swap files when editing documents, RosyCloud allows users to define a set of rules to ignore their changes.

V. EVALUATION

We evaluate the performance of RosyCloud to answer the following questions:

- What is the space overhead of RosyCloud?
- How much time does RosyCloud take to back up, to restore, and to synchronize devices?
- What's the monetary cost of RosyCloud when using multiple clouds?

Experiments were performed on PCs with Intel i5 2.54 GHz CPU, 2 GB memory, and two 5400 RPM disks. We mainly use Alibaba and Microsoft storage clouds during evaluation. This is because accessibility to Google's service has been unstable recently in China.

A. Benchmarks

Two benchmarks were used in our evaluation of RosyCloud, an office workload trace and a mail server workload generated from filebench [28]

Table IV
STATISTICS OF THE OFFICE WORKLOAD TRACE.

Statistics	Begin	End
Directories No.	109	118
Files No.	782	815
Small Files No. (<4KB)	333	348
Medium Files No.	423	441
Large Files No. (>1MB)	26	26
Total Capacity	222.33MB	230.67MB

The first office workload trace is collected from the desktop computer of an author of this paper, which lasts for a week. During the time, all system calls on the author's home directory were logged. The total size of the home directory is roughly 15 GB. Because the user was focusing on a project during the time, only a small portion of the files was actively updated. After filtering out directories that have no changes, the active files were about 222.3 MB initially. The statistics of this active file set are shown in Table IV, where files less than 4 KB are categorized as small, and those larger than 1 MB are identified as large. Updates of small and medium sized files dominate in the workload. Most modifications are performed on text files. Some binary files may be created as intermediate files and will finally be removed from the file system or be renamed [10].

The second workload is from filebench [28], which simulates a heavily loaded mail server, much like the Postmark benchmark [12]. The mail server workload consists of large number of small files, in the pattern of creating, writing, reading, and finally deleting files.

B. Storage Usage

This experiment studies the storage usage on the clouds. Figure 6 plots storage changes on cloud for the office workload. Initially, the compressed data size on the cloud is 34.33 MB. As time progresses, the cloud storage in use

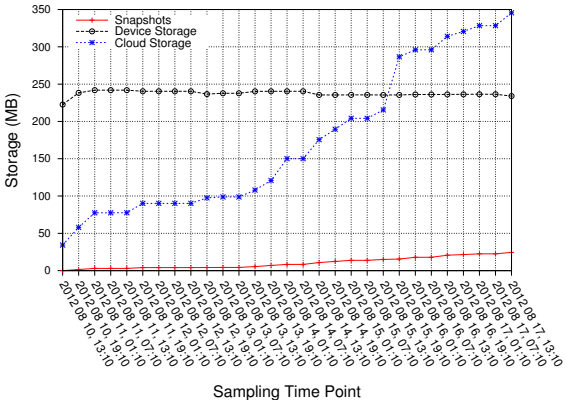

Figure 6. Cloud storage changes in the office trace.

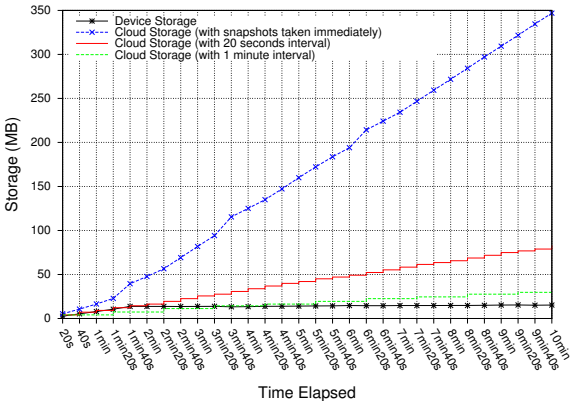

Figure 7. Cloud storage changes in a mail server workload generated by filebench.

increases gradually, which is expected given the copy-on-write update model. At the end of the trace, the total data size is 345.6 MB, which is larger than the size of 230.7 MB on local devices. This is because the cloud storage keeps all versions of modified files. Figure 6 also shows the total size of snapshot objects, which grows steadier with time and only consumes a small portion of cloud storage. This is because snapshots objects only contain metadata information.

Figure 7 illustrates storage changes for the mail server workload, which runs filebench [28] for ten minutes. Because the workload is composed of a series of file creation, read, and deletion operations, the local storage remains flat. However, the cloud storage increases linearly, because deleted files are kept as a version in the cloud. As we can see taking snapshots immediately does increase cloud storage requirement. A user can limit the number of snapshots by enlarging backup time interval to control and reduce storage requirement. As shown in the figure, when we increase the backup time interval to 20 seconds, significant cloud storage space can be saved. If backing up every one minute, even more space is saved. Similar to the office workload case (not shown in the figure), snapshot consumes little storage due

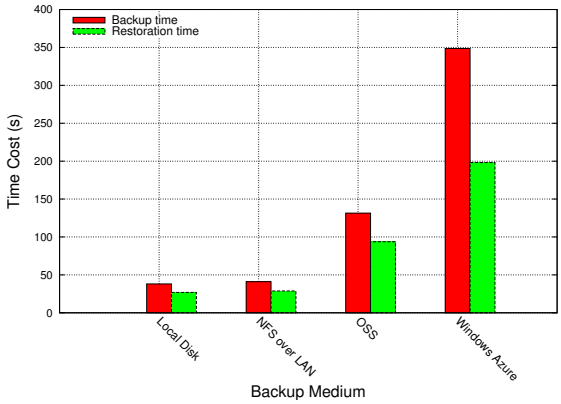

Figure 8. The time cost of full backup and restoration operations for the office workload.

to the small size of `Snapshot` objects.

C. Time Cost

1) Backup and Restoration Time: This experiment studies the temporal cost for backup and restoration operations. The backup starts with an empty cloud and RosyCloud will transfer decorated data to cloud storage. The restoration reverses the backup process, where the local file system is empty and all data is synchronized from the cloud, mimicking the initial synchronization from an empty end device.

We compare four different backup storage media: another hard disk on the same machine, an NFS share on a gigabit LAN, Aliyun's OSS, and Windows Azure. OSS is a local cloud storage provider, which has better network performance than Windows Azure residing abroad.

Figure 8 illustrates the time costs of these four approaches with the office workload. Using another disk and NFS perform the best, because of low delays from source disk to destination storage. OSS performs better than Windows Azure, for the better network connections. For all four approaches, full restore time is less than backup. For OSS and Windows Azure, the reason is that writing to cloud storage is slower than reading. Because writes must be successful on multiple replicas [7], it takes more time than reads that can be returned when any one replica succeeds. For local disk and NFS, the time difference is due to different amount of data is read from disks. The backup operation needs to read 222.3 MB, while restore reads only 34.3 MB compressed data.

2) Synchronization Time: This experiment measures the time for synchronizing different devices using the office workload. In the experiment, a laptop at home is synchronized with a desktop in office through OSS. The laptop is set to synchronize with OSS every fifteen minutes. The cumulative distribution function (CDF) of time elapsed from a file is updated on the office desktop to the time the file

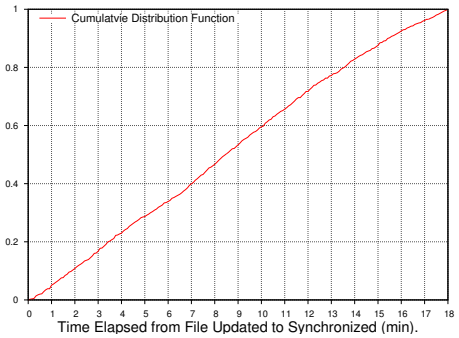

Figure 9. CDF graph of synchronization time for the office workload.

is synchronized on the laptop is shown in Figure 9. All files are synchronized within 18 minutes after updating the original file and the average delay is 8.6 minutes. Because the user takes about 20 minutes to commute between home and office, the system is always able to synchronize files between computers before the user arrives, allowing the user to continue the work left.

D. Monetary Cost

Table V
CLOUD STORAGE PRICING MODELS.

Provider	Data Request	Egress Bandwidth	Storage
OSS	$0.01/10,000	$0.129/GB*M	$0.129/GB*M
Azure	$0.01/100,000	0	$0.093/GB*M

Table VI
REQUEST DISTRIBUTION OF HTTP METHODS FOR THE OFFICE WORKLOAD.

Method	Count (K)	Size (MB)
`GET`	103.3	165.6
`PUT`	31.9	340.6
`HEAD`	0.4	N/A

Cloud storage is not a "free-lunch" and we study the monetary cost of using RosyCloud for the office workload. As of November 2012, the pricing models (in US dollars) for OSS and Azure are shown in Table V, based on the number of data access requests to cloud, egress bandwidth per month, and monthly storage usage. Table VI illustrates the distribution of different HTTP request methods. Most of the requests are HTTP `GET` method with a total download of 165.6 MB. HTTP `PUT` is second in terms of request numbers, but first in bandwidth usage (340.6 MB), due to the uploading of changed files. Table VII shows the cost for the office workload. The combined cost of using OSS and Windows Azure is $0.15 for a week, indicating that RosyCloud is financially feasible.

Table VII
COST OF THE OFFICE WORKLOAD.

Provider	Cost ($)			Total Cost
	Request	Bandwidth	Storage	($)
Aliyun OSS	0.1	0.02	0.01	0.13
Windows Azure	0.01	0	0.01	0.02

VI. CONCLUSIONS AND FUTURE WORK

This paper presents the design, implementation, and evaluation of RosyCloud, a versioned file backup and synchronization tool in multiple clouds with a thin HTTP-based interface. RosyCloud supports concurrent file updates through copy-on-write snapshots. Conflicts by concurrent writes from different devices are detected and are resolved with a DAG dependence model.

To the best of our knowledge, this is the first tool supporting both backup and synchronization for a multi-cloud environment with no server computing requirement. Our evaluation shows that RosyCloud is flexible to support different clouds in a reasonable amount of time with a low monetary cost.

Future work of RosyCloud includes using erasure codes to distribute file content on different clouds to save storage space and cost. Another way to reduce cloud storage usage is to garbage collect unused snapshots stored on the cloud. To this end, we plan to implement different policies on reclaiming snapshots based on time and importance [20].

ACKNOWLEDGMENT

We would like to thank Wei Zhang and all the anonymous reviewers for their insightful comments. This work was supported in part by NSFC (Grant 61003012 and 61261160502) and 863 program of China (Grant 2011AA01A202). Yang was supported in part by NSF IIS-1118106. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these sponsors.

REFERENCES

[1] Dropbox. https://www.dropbox.com/.

[2] Megaupload file-sharing site shut down. http://www.bbc.co.uk/news/technology-16642369, 2012.

[3] Amazon Web Services. Amazon simple storage service. http://aws.amazon.com/s3/.

[4] B. Berliner. CVS II: Parallelizing software development. In *Proc. of the USENIX Winter Conference*, 1990.

[5] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa. Depsky: dependable and secure storage in a cloud-of-clouds. In *Proceedings of the sixth conference on Computer systems*, pages 31–46. ACM, 2011.

[6] C. Cachin, R. Haas, and M. Vukolic. Dependable storage in the intercloud. *Research Report RZ*, 3783:1–6, 2010.

[7] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, et al. Windows azure storage:

a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157. ACM, 2011.

[8] L. Cox, C. Murray, and B. Noble. Pastiche: Making backup cheap and easy. *ACM SIGOPS Operating Systems Review*, 36(SI):285–298, 2002.

[9] S. Ghemawat, H. Gobioff, and S. Leung. The google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[10] T. Harter, C. Dragga, M. Vaughn, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. A file is not a file: understanding the i/o behavior of apple desktop applications. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 71–83, New York, NY, USA, 2011. ACM.

[11] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS)*, 6(1):51–81, 1988.

[12] J. Katcher. Postmark: A new file system benchmark. Technical report, Technical Report TR3022, Network Appliance, 1997. www. netapp. com/tech_library/3022. html, 1997.

[13] J. Lemon. Kqueue: A generic and scalable event notification facility. In *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*, pages 141–153, 2001.

[14] J. Loeliger. *Version control with Git*. O'Reilly Media, 2009.

[15] R. Love. Kernel korner: Intro to inotify. *Linux Journal*, 2005(139):8, 2005.

[16] J. Marc. The source code control system. *IEEE Transactions on Software Engineering. SE-1 (4)*, pages 364–470, 1975.

[17] C. Pilato, B. Collins-Sussman, and B. Fitzpatrick. *Version control with subversion*. O'Reilly Media, 2008.

[18] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *Proceedings of the FAST 2002 Conference on File and Storage Technologies*, volume 4, 2002.

[19] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the sun network filesystem. In *Proceedings of the Summer USENIX conference*, pages 119–130, 1985.

[20] D. Santry, M. Feeley, N. Hutchinson, A. Veitch, R. Carton, and J. Ofir. Deciding when to forget in the elephant file system. In *ACM SIGOPS Operating Systems Review*, volume 33, pages 110–123. ACM, 1999.

[21] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *Computers, IEEE Transactions on*, 39(4):447–459, 1990.

[22] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. Ieee, 2010.

[23] H. Tang, A. Gulbeden, J. Zhou, W. Strathearn, T. Yang, and L. Chu. Sorrento: A self-organizing storage cluster for parallel data-intensive applications. In *SC2004: High Performance Computing, Networking and Storage Conference*, Pittsburgh PA, 2004.

[24] M. Vrable, S. Savage, and G. Voelker. Cumulus: Filesystem backup to the cloud. *ACM Transactions on Storage (TOS)*, 5(4):14, 2009.

[25] M. Vrable, S. Savage, and G. Voelker. Bluesky: A cloud-backed file system for the enterprise. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST)*, 2012.

[26] H. Wang, R. Shea, F. Wang, and J. Liu. On the impact of virtualization on dropbox-like cloud file storage/synchronization services. In *IEEE/ACM International Workshop on Quality of Service (IWQoS)*, Coimbra, Portugal, 2012.

[27] H. Weatherspoon, C. Wells, P. Eaton, B. Zhao, and J. Kubiatowicz. *Silverback: A global-scale archival system*. Computer Science Division, University of California, 2001.

[28] A. Wilson. The new and improved filebench. In *Proceedings of 6th USENIX Conference on File and Storage Technologies*, 2008.

[29] L. Wingerd. *Practical perforce*. O'Reilly Media, Inc., 2005.

[30] A. Yip, B. Chen, and R. Morris. Pastwatch: A distributed version control system. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation*, pages 381–394, 2006.