Enabling Conflict-free Collaborations with Cloud Storage Services

Minghao Zhao, Jian Chen, and Zhenhua Li*
School of Software, Tsinghua University
{zhaominghao.thu, chenjian1995.thu, lizhenhua1983}@gmail.com

***Abstract*—Cloud storage services (*e.g.,* Dropbox) have become pervasive in not only simple file sharing but also advanced collaborative file editing (*collaboration* for short). Using Dropbox for collaboration is much easier than SVN and Git, thus greatly facilitating common users. In practice, however, many Dropbox users are perplexed by unexpected collaboration conflicts, which severely impair their experiences. Through various benchmark experiments, we unveil the two root causes of collaboration conflicts: 1) Dropbox never locks an edited file during collaboration; 2) Dropbox only guarantees *eventual* data consistency among the collaborators, significantly aggravating the probability of conflicts.**

In this paper, we attempt to enable conflict-free collaborations with Dropbox-like cloud storage services. This attempt is empowered by three key findings and measures. First, although the end-to-end sync delay is unpredictable due to eventual consistency, we can always track the latest version of an edited file by actively resorting to the cloud via certain web APIs. Second, although all application-level data is encrypted in Dropbox, we can roughly deduce the sync status from traffic statistics. Third, applying a couple of useful mechanisms (*e.g.,* distributed architecture and data lock) learned from Git, we can effectively and efficiently avoid collaboration conflicts—of course, this requires re-implementing Git mechanisms in cloud storage services with minimum overhead and user interference. Integrating above efforts, we build the ConflictReaper system capable of helping users automatically avoid almost all collaboration conflicts with affordable network and computation overhead.

***Index Terms*—cloud storage, consistency, collaborations, network measurement, Dropbox**

I. Introduction

Cloud storage services, such as Dropbox, Microsoft OneDrive, Google Drive, and iCloud Drive, have quickly become pervasive in recent years. As a representative cloud storage service released in 2007, Dropbox has owned more than 500 million users working in over 200,000 companies and organizations [4]. These users store or update 1.2 billion files every day and *make around 4000 file edits every second* [3]. Moreover, Microsoft OneDrive, Google Drive, and iCloud Drive each has attracted more than 700 million users [2].

Conventionally, cloud storage services provide their users with reliable and ubiquitous file backup and retrieval functions. They further support *simple* file sharing across users; here simple means that the shared files are immutable. In addition, some services (*e.g.,* Dropbox) offer more advanced functions such as collaborative file editing (abbreviated as *collaboration*) where multiple users can edit a shared file via the cloud storage. Specifically, in Dropbox, every user owns a designated local folder (called a "sync folder") in which every data update is *automatically* noticed and synced to the cloud by the Dropbox client software [46]. Afterwards, the data update is *automatically* propagated by the cloud to the sync folder(s) of the other user(s) who share the file (called *collaborators*). When everything goes smooth, using Dropbox for collaborations is much easier than using VCS (Version Control Systems) like SVN and Git, since the users do not need to understand and manipulate the complex VCS primitives (*e.g.,* pull, push, comment, and clone). This greatly facilitates those computer non-professionals in online collaborations.

* Corresponding author

Despite the high usability, many Dropbox users are perplexed by unexpected collaboration conflicts in practice, which severely impair their experiences [19]. In particular, even if the collaborators (say Alice and Bob) always edit their shared file (say *f.txt*) in a sequential manner and possess fine network connections, they may be confronted with conflicting files in their sync folders, like *f.txt* and *f (Bob's conflicted copy 2021-11-11).txt*. When such conflicts happen, the collaborators have to manually fix the problem with considerable efforts, even with the help of Dropbox's "version history" feature [7].

Through various benchmark experiments, we unveil the two root causes of collaboration conflicts in Dropbox: 1) Unlike certain version control systems [60], Dropbox never locks an edited file to avoid conflicts during collaboration; 2) Dropbox only guarantees *eventual* data consistency among the collaborators, thus significantly aggravating the probability of conflicts. As shown in Figure 1, after Alice edits a shared file, her Dropbox client syncs the data update to the cloud *as soon as possible* [Step 1] [47], but the notification for the data update is then propagated by the cloud to Bob's Dropbox client *with an unpredictable delay* (ranging from a few seconds to several hours). According to the design principle of Dropbox's edge network [6], [13], we infer that the notification is first pushed to a message queue [Step 2] and then popped to Bob by the message queue [Step 3]. Dropbox's leveraging a message queue for notification propagation is quite understandable, since Dropbox often has to propagate an unpredictably large number of concurrent notifications and message queue turns out to be a cost-effective solution (that trades a part of sync speed for affordable infrastructure cost and guaranteed sync success).

With the above findings, we attempt to enable conflict-free collaborations with Dropbox-like cloud storage services. This

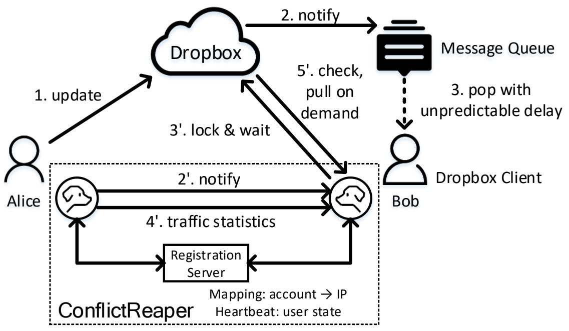

Fig. 1: Architectural overview of Dropbox and ConflictReaper on handling cross-user collaborations.

attempt is empowered by the following three insights. First, although the end-to-end sync delay is unpredictable due to eventual consistency, we can always obtain the latest version of an edited file by actively resorting to the Dropbox cloud via certain web APIs (*e.g.,* `list_folder`, `get_metadata`, and `download` [5]). This finding offers us the feasibility to convert unpredictably long sync delay into predictably short sync delay, with small network/computation overhead.

In principle, we can develop a *watchdog* program for each collaborator to periodically check the state of every shared file through web APIs. The length of the period, however, is hard to configure: a short period incurs too much system overhead while a long period leads to poor user experience. An intuitive approach is to replace periodical checking with on-demand checking, which requires the watchdog to understand the *sync status* (*e.g.,* start an update, finish an update, get a notification, begin a download, and finish a download) of its corresponding Dropbox client. Unfortunately, all application-level data is encrypted in Dropbox through HTTPS (TLSv1.2), including the sync status. Fortunately, through extensive measurement, we are able to deduce the sync status from the statistics of encrypted traffic (especially the traffic volume, upload/download speed, and bursty pattern). This information can help a watchdog to determine when to send its traffic statistics to other watchdog(s).

With the knowledge of file state and sync status, we still need considerable extra mechanisms to fulfill the whole collaboration process. Here we do not want to "re-invent the wheels" to achieve this goal; instead, we apply a couple of useful mechanisms (*e.g.,* distributed architecture and data lock) learned from Git to effectively and efficiently avoid collaboration conflicts. Of course, this requires re-implementing Git mechanisms in cloud storage services with minimum overhead and user interference. We distribute a watchdog program to locally assist each Dropbox client, coupled with a lightweight *registration server* for mapping user accounts to IP addresses and collecting heartbeat messages that report user states. Moreover, we implement *implicit data lock* by closely monitoring a user's file operations in a lightweight manner (*e.g.,* using `inotify` in Linux or `FileSystemWatcher` in Windows). Any file operation that may potentially bring conflicts will trigger a pop-up warning.

Integrating all above efforts, we build the ConflictReaper system to help users automatically avoid almost all collaboration conflicts in Dropbox with affordable network/computation overhead and little user interference. As illustrated in Figure 1, a user only needs to run the watchdog program which is the client side of ConflictReaper. As soon as Alice starts a data update on a shared file f [Step 1], Alice's watchdog notices it and sends a notification to Bob's watchdog [Step 2']. On receiving the notification, Bob's watchdog implicitly locks f and then waits for Alice's traffic statistics to come [Step 3']. Note that Alice's watchdog sends Alice's traffic statistics when it deduces that the data update has been successfully synced to the cloud [Step 4']. After receiving Alice's traffic statistics, Bob's watchdog constantly records and analyzes Bob's traffic statistics; if it is necessary, Bob's watchdog will actively check and pull the latest version of f through web APIs [Step 5'].

We implement ConflictReaper in ∼2500 lines of C# code for the watchdog and ∼650 lines of Java code for the registration server, all of which are publicly available at https://ConflictReaper.github.io. Hundreds of computer professionals and non-professionals have used our Windows clients and returned positive feedback. In typical realistic collaboration scenarios, the network overhead brought by ConflictReaper is ∼14 KBps, *i.e.,* less than 1% of the total sync traffic of Dropbox. Meanwhile, the computation overhead is ∼3% CPU utilization on a single core. Finally, ConflictReaper works independently of Dropbox in essence, so it also generalizes to other cloud storage services such as Microsoft OneDrive [12], Google Drive [10], and Amazon Drive [1].

II. Motivating Analysis

To explore the root causes of collaboration conflicts in Dropbox, we conduct black-box measurements to investigate the work flow of collaborative editing on a shared file by two users (say Alice and Bob). Alice uses a laptop in San Francisco while Bob uses a desktop in New Haven; they each use a 100-Mbps Internet connection.

Simultaneous editing. First, we let Alice and Bob open and edit a shared text file at roughly the same time. We observe either of them can *locally* finish both open and edit operations without any impediments or warnings from Dropbox. This clearly indicates that Dropbox never locks the shared file during collaboration, which is an essential reason for the occurrence of conflicts.

Alternate editing. In practice, Alice and Bob seldom edit a shared file at the same time. Instead, they typically edit it *alternately*. If Alice first makes an edit and then Bob keeps waiting for the corresponding sync notification to come, the state of the file will *eventually* become consistent for Alice and Bob. However, we notice a key issue that prevents such eventual consistency from working well, *i.e.,* the end-to-end sync time for Alice's edit to be propagated to Bob varies significantly (between a few seconds to hours) and can hardly be predicted. During the sync time (or says sync delay), once

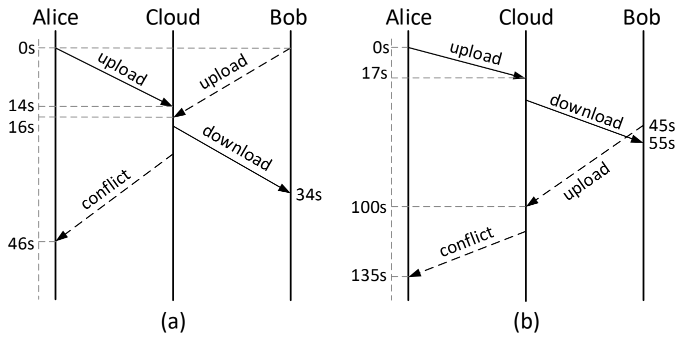

Fig. 2: Time sequence diagram of (a) a typical simultaneous edit and (b) a typical alternate edit.

Bob makes another edit to the file—this is fairly likely to happen in real-world scenarios—conflicts will occur.

Quantifying the sync delay. According to the above experiments, we plot the time sequence diagram of a typical simultaneous edit and a typical alternate edit in Figure 2(a) and Figure 2(b), respectively. From both figures, we find that it *always* takes a short period of time (usually just a few seconds) for Dropbox to upload the file edit to the cloud, except when the edit size is pretty large (note that Dropbox leverages the delta sync approach to reduce the sync traffic of a file edit [47]). On the contrary, downloading the file edit from the cloud often takes a longer period of time and the period seems unpredictable. As mentioned in §I, this can be reasonably ascribed to Dropbox's utilization of a message queue for propagating sync notifications [6], [13].

Status quo. In both Figure 2(a) and Figure 2(b), after Alice and Bob upload their own file versions to the cloud, the Dropbox server will retain both versions, and the later one arriving at the server will be marked as "the conflict version." Afterwards, both versions will be delivered to Alice and Bob, who then have to merge the different versions *manually*—definitely suffering experiences.

As a matter of fact, Dropbox has noticed the problem and launched Dropbox Badge [18], as a partially solution. We have studied the workflow of Dropbox Badge through measurement-based reverse engineering. We find that Dropbox Badge adopts the periodical checking (polling) to probe and retrieve the latest version of a shared file from the cloud, thus incurring heavy network traffic overhead. Note that Dropbox Badge is not a general solution but only works for documents in the format of Microsoft Office (*e.g.,* Word, PowerPoint, and Excel). In this work, we strive towards a generic solution that applies to any file types, while avoiding expensive polling.

III. ConflictReaper

Driven by the findings in §II, we design ConflictReaper to assist users automatically avoid collaboration conflicts in Dropbox with affordable overhead. This section presents the design of ConflictReaper (§III-A), and describes the implementation of each component (§III-B).

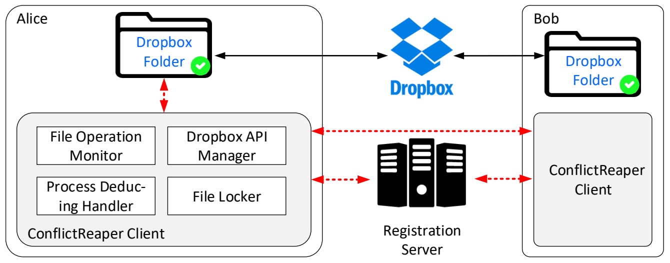

Fig. 3: System structure of ConflictReaper with modularized components.

A. System Design

Figure 3 gives an overview of ConflictReaper, which consists of *ConflictReaper Client* and *Registration Server*. Each ConflictReaper user (say Alice and Bob) installs *ConflictReaper Client* locally. As shown in Figure 3, a ConflictReaper client contains four modules: File Operation Monitor, Traffic Measurement Agent, Notification Handler, and Dropbox API Manager. File Operation Monitor is used to monitor the editing behaviors of users, and report file status to other clients. Traffic Measurement Agent aims to collect and analyze network traffic information, in order to deduce the sync status. Subsequently, the sync status is submitted to Notification Handler, which is responsible for triggering the corresponding file lock or unlock. Dropbox API Manager is designed to call Dropbox Web APIs to retrieve the latest version of a given file. Design details of these modules are elaborated in §III-B.

Registration Server stores user registration information, and mappings among users who share same files. When a user Alice logins, she submits the username along with her IP address to one of the registration servers, acquiring all the IP addresses of other users who share files with her. Using the acquired username-IP mappings, as shown in Figure 3, Alice can directly establish connection to other peers, say Bob.

B. Implementation

File Operation Monitor. This module aims to capture file modifications through file monitoring. We use *effective* keyboard typings—which exclude the typings not causing any file modifications such as a single keying of `control`—as signals of the beginning of modifications. Using specific APIs (*e.g.,* `setWindowsHookEx` in Windows and `input_event` in Linux), we can accurately monitor all the typing behaviors. When a keyboard typing is captured, ConflictReaper checks whether this typing affects files in the shared folder. If so, ConflictReaper identifies that the user is modifying a shared file and immediately informs collaborators to lock the file.

The file saving operation indicates the completion of file modification, so an update sync is needed. Normally, the file saving will be accompanied by storage device write-in, which is a simple file system operation and can be caught by file system monitor. Thus, we use the action of storage device write-in as the completion of file modification and the beginning of the file uploading process.

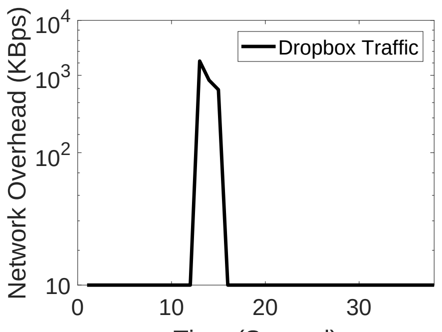

Fig. 4: Network traffic of Dropbox.

Traffic Measurement Agent and Notification Handler. Once the user saves the modified file in the shared folder, the Dropbox client immediately and automatically uploads this file to the cloud. ConflictReaper needs to know the status of this uploading process, because we want to track this updated file and trigger other peers' lock or unlock operations. However, the data traffic of the Dropbox client is strongly encrypted (with TLS); thus, it is difficult to acquire sync status from the traffic content.

We conduct an experiment to understand the variation feature of traffic in uploading a file. As shown in Figure 4, the traffic of Dropbox gradually increases (*i.e.,* chunk comparison for delta sync), followed by a sharp increase (*i.e.,* starting of data transmission). After retaining stable (maybe with slight fluctuation), a rapid decline appears (*i.e.,* completion of data transmission). Driven by this feature, we propose a traffic measure-based protocol, deployed in Notification Handler module, to determine when to unlock the shared file.

Specifically, Alice's Notification Handler uses Traffic Measurement Agent to record the traffic (denoted as F_a) and time (denoted as T_a) consumed during the file uploading process. Then, the handler sends the traffic F_a and time T_a to Bob's client. Until receiving F_a and T_a, Bob's client begins to count the total traffic (denoted as F_b), and at the same time records the elapsing time (denoted as T_b). If F_b gets equal to F_a before T_b reaches T_a (*i.e.,* $F_b = F_a$ and $T_b < T_a$), it can be indicated that the file has been successfully downloaded, as the downloading phase consumes roughly the same traffic as the uploading phase. ConflictReaper then unlocks the shared file. Otherwise, if T_b gets equal to T_a before F_b approaches at F_a (*i.e.,* $T_b = T_a$ and $F_b < F_a$), this indicates that the file has not been downloaded to the client in excepted time. This normally happens when the message queue in the Dropbox server is long, leading to long delay. ConflictReaper will check the consistency between the local file and the cloud file; if a newer version exists in the cloud, ConflictReaper will download it through Web APIs.

Dropbox API Manager. Dropbox provides a set of web APIs for developers, which allow programmable read/write access to the files stored in the Dropbox cloud. Although it takes a long time for Dropbox to push a file from the cloud to the client, only a short period is required to retrieve the file through the provided APIs. We conducted an experiment to accurately evaluate the efficiency gap of these two methods in acquiring files from the cloud. For Dropbox APIs, the sync time is the duration of download; whereas the sync time of automatic synchronization with Dropbox client is measured from when a file is successfully uploaded to the cloud to when it is synchronized to the client. As shown in Figure 5, the overwhelming percentage number of downloading tasks can be accomplished within 2 seconds through APIs, whereas automatic synchronization with Dropbox client requires at least 25 seconds, and most of the cases require 50 seconds. Thus it is feasible to "immediately" get the latest version at cloud by actively fetching it with Web APIs.

Nevertheless, active fetches can be done aggressively or conservatively. Fetches too aggressive (*e.g.,* once Bob is notified of Alice's successfully syncing her edits to the cloud, he immediately fetches from the cloud) incur considerable network overhead at both client and server side, as delta sync is not supported in web-based transmission in Dropbox [48], [67]. Fetches too conservative may not effectively prevent conflicts. Thus, we should make a tradeoff between conflict probability and network overhead. Our basic idea is: if a file (*esp.,* a large file) is updated by only a small part, it is preferable for Bob to wait for a certain period of time (T_{wait}), as the latest version of the file is highly likely to arrive (without active fetches) in T_{wait}. To this end, we devise:

$$T_{wait} = \frac{file\_size - sync\_traffic}{bandwidth} + T_{P99}, \tag{1}$$

where $file\_size$ denotes the file size, and $bandwidth$ denotes the transfer bandwidth in Bob's latest downloading from the cloud. *sync_traffic*, which is contained in the sync status sent from Alice, denotes the network traffic consumed in Alice's uploading the file update to the cloud. T_{P99} denotes the 99-percentile delivery time of file updates, which is 312 second according to our measurement study.

IV. Performance Evaluation

Experiment setup. Our experiment focus on the evaluation of the computation and network overhead of ConflictReaper. We do not evaluate the performance of *Registration Server*, because 1) the server is only connected and queried once by each client, and 2) the server is neither the throughput nor the scalability bottleneck.

As shown in §II, we run ConflictReaper on a laptop located in San Francisco (say Alice) and a desktop located in New Haven (say Bob), each with a 100-Mbps Internet connection. Text documents (*i.e.,* files with format of txt) with the size of roughly 4 MB are shared between them; once one of these files is changed by Alice, the revisions will be synchronized to Bob. We repeat the editing-sync procedure for 2000 times.

Evaluation results. We measure the network overhead of ConflictReaper and Dropbox. As shown in Figure 6, a typical sync procedure of Dropbox has two traffic spikes—the former is for uploading and the latter is for downloading.

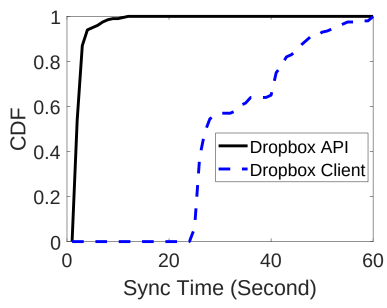

Fig. 5: Distribution of synchronization time used by Dropbox client and Dropbox Web APIs.

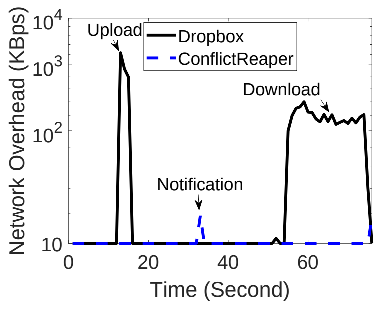

Fig. 6: Network traffic for synchronization using ConflictReaper and regular Dropbox during collaboration

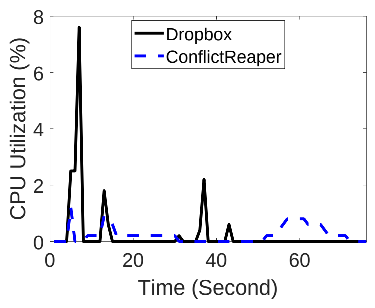

Fig. 7: CPU utilization for synchronization with ConflictReaper and regular Dropbox during collaboration.

The uploading phase takes relatively less time. In the middle of these two spikes, the traffic consumption for ConflictReaper can be found. In total, the network overhead brought by ConflictReaper is ∼14 KBps, *i.e.,* less than 1% of the sync traffic of Dropbox.

We also measure the CPU utilization of ConflictReaper and Dropbox. As shown in Figure 7, in a typical sync procedure incurs CPU spikes for both ConflictReaper and Dropbox. The first two CPU utilization crests for Dropbox are accompanied with that of ConflictReaper, as the first one appears at the chunk comparisons and the second one happens when uploading the file to the cloud. Both of these phases are coupled with traffic transmission and the ConflictReaper is monitoring the traffic simultaneously. After that, we can see the CPU utilization of Dropbox (*i.e.,* consistency checking at Bob's side), followed by the CPU consumption of ConflictReaper (*i.e.,* counting the traffic of Dropbox). In sum, ConflictReaper costs less than 3% CPU utilization on a single core. In conclusion, ConflictReaper involves negligible network and CPU overhead.

V. Related Works

This paper tends to enable conflict-free collaboration with cloud storage services. In this section, we first briefly introduce some research on cloud storage, and then describe the research work of measurements on cloud storage service, as well as synchronization and collaboration with cloud storage service.

Cloud storage and cloud storage services Recent years have witnessed the popularity of cloud storage. Among them, infrastructure level cloud storage, such as Amazon EBS and S3, are basic facilities in the cloud ecosystem. They provide different storage models (*e.g.,* block store, object store, file store, and structured database) and enable users to store or retrieve data with specific APIs [21], [51]. Since Amazon first launched its cloud storage product in 2006 (generally regarded as the beginning of the cloud era), numerous researches have been done on different aspects of clouds storage, such as performance [55], [56], power efficiency [31], security [39], [59], [64], [65], and reliability [52].

Cloud storage services, such as Dropbox and OneDrive, are normally built on top of cloud storage infrastructures [49], [54]. Generally, they store data contents in cloud object stores, and utilize specific cloud resources to maintain filesystem metadata (*e.g.,* constructing the directory tree of users' filesystem). For example, Dropbox used to build on top of AWS infrastructures (*e.g.,* store all data content in S3), and started to migrate data content to its own deployed MagicPocket object storage since 2016 [11]. In terms research products, Up to now, numerous research efforts have been made to design novel data structures to implement the file-to-object maps and maintain filesystem structures. These researches (and their corresponding systems) include Cumulus [62], Goofys [9], SVFS [17], S3FS [15], gcsfuse [8], YAS3FS [20], SCFS [24], S3QL [16], Agni [49], Panzura [14] H2cloud [70], and BlueSky [63], *etc.*.

Measurements on cloud storage service. Quite a number of works focus on benchmarking and measurement, of cloud storage service. These works cover both enterprise [23], [27], [37], [53] and personal cloud storage service [25], [29], [30], [33], [35], [58]. Specifically, Palankar *et al.* [53] conducted a comprehensive evaluation on whether Amazon S3 is qualified for today's scientific collaborations. Similarly, Bergen *et al.* [23] presented a performance analysis of the Amazon Web Services, and revealed that the user-perceived performance is heavily determined by network condition, rather than the computing ability of the data center. Hill *et al.* [37] conducted an experimental analysis on the performance of each component inside Windows Azure Platform, and Li *et al.* [44] measured the overall performance of elastic computing, persistent storage and networking services of some mainstream cloud storage services (*e.g.,* Amazon AWS, Microsoft Azure, Google AppEngine and Rackspace CloudServers). In [27], a standard benchmark and benchmarking framework were proposed to evaluate the performance of cloud data storage system.

In terms of measurement in personal cloud, Hu *et al.* [40] firstly evaluated and compared the performance of four popular commercial cloud service products (*e.g.,* Mozy, Carbonite, Dropbox, and CrashPlan). Gracia-Tinedo *et al.* presented the measurement study of REST interfaces provided by three personal clouds in [34] and internal structure of UbuntuOne in [35]. Drago *et al.* [29], [30], [33] conducted a series of research on the characteristics of Dropbox, covering the aspect of storage capability and client behavior. Li *et al.* [46] and Li *et al.* [45] investigated the network traffic in data sync among cloud storage services. Unlike the aforementioned works that concentrate on the performance of cloud platforms, the work [38] regards the cloud platforms as "black-box" and focuses on observation and analysis of the end-to-end performance from a client's perspective.

Synchronization and collaboration with cloud storage service. Data/file synchronization is a fundamental functionality in cloud storage system. It serves as the basic component of cloud-based collaboration. After years of research, many sync algorithms, such as Optimistic Deltas [22], `rsync` [61], Delta encoding [57], Content Defined Chunking (CDC) [42] and Efficient Delta Synchronization (EDS) [43] were proposed. Among them, the `rsync` has become the de facto delta sync protocol and been adopted by almost all the mainstream cloud platforms and operating systems. Many researchers have conducted a series of research on the respects of data security [36], [50], consistency [32], [69] and performance optimization [45], [47] of data/file synchronization in a cloud (file) system. Recently, the QuickSync [28] and DeltaCFS [68] were designed for data sync in mobile applications and the WebR2sync+ [66], [67] was constructed for efficient data sync in web browsers.

In terms of cloud-based collaboration, CoCloud [41] is a system for cross-cloud file collaboration. Our work differs from it in that, CoCloud focuses on efficient file transmission among different cloud storage providers (*e.g.,* collaborations between Baidu Cloud and Dropbox), whereas our work tends to provide collision-free collaboration editing services in cloud storage platforms. UFC2 [26] presents an alternative way for addressing the conflict issue in collaboration with cloud storage services. It achieves this by automatically merging conflicting versions after conflicts appear. Nevertheless, this approach needs cloud service providers to modify their system implementations. In contrast, we present a lightweight solution that solves this issue with plug-ins – it only needs the client to run our client-side software (*e.g.,* merely several Mb under Windows operating system). The only industrial solution to conquer the conflict issue is the Dropbox Badge [18], which only serves for Microsoft Office, thus has limited scalability.

VI. Future Work

In this paper, we explore the possibility of conflict-free collaborations with cloud storage service, and propose the ConflictReaper to achieve this goal. The primary experiment indicates that ConflictReaper only causes imperceptible overhead in CPU utilization and network traffic. Our future work includes evaluating its performance in more aspects and details, e.g., the accuracy of traffic-based sync status deduction, and CPU and traffic consumption in editing and sync files under real-world file editing and synchronization workloads.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China (NSFC) under grants 61822205, 61632020, 61632013 and 61902211.

References

[1] Amazon Drive. http://www.amazon.com/clouddrive.
[2] Apple Music passes 11M subscribers as iCloud hits 782M users. http://appleinsider.com/articles/16/02/12/apple-music-passes-11m-subscribers-as-icloud-hits-782m-users.
[3] By the Numbers: 22 Staggering Dropbox Statistics (July 2017). https://expandedramblings.com/index.php/dropbox-statistics/.
[4] Dropbox — Company Info - Dropbox. https://www.dropbox.com/news/company-info.
[5] Dropbox API Explorer. https://dropbox.github.io/dropbox-api-v2-explorer/.
[6] Evolution of Dropbox's Edge Network. https://blogs.dropbox.com/tech/2017/06/evolution-of-dropboxs-edge-network.
[7] File version history. https://www.dropbox.com/help/security/version-history-overview.
[8] gcsfuse. https://github.com/GoogleCloudPlatform/gcsfuse.
[9] Goofys. https://github.com/kahing/goofys.
[10] Google Drive. https://www.google.com/drive/.
[11] Inside the Magic Pocket. https://dropbox.tech/infrastructure/inside-the-magic-pocket.
[12] Microsoft OneDrive. https://onedrive.live.com/.
[13] Optimizing web servers for high throughput and low latency. https://blogs.dropbox.com/tech/2017/09/optimizing-web-servers-for-high-throughput-and-low-latency.
[14] Panzura Global Cloud File System. https://panzura.com/technology/.
[15] S3Fuse. https://github.com/s3fs-fuse/s3fs-fuse.
[16] S3QL. https://bitbucket.org/nikratio/s3ql/.
[17] SVFS. https://github.com/ovh/svfs.
[18] What is the Dropbox badge? https://www.dropbox.com/help/business/badge-overview.
[19] What's keeping you from dropping Dropbox? https://forum.syncthing.net/t/whats-keeping-you-from-dropping-dropbox/3022.
[20] YAS3FS. https://github.com/danilop/yas3fs.
[21] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010.
[22] G. Banga, F. Douglis, M. Rabinovich, et al. Optimistic Deltas for WWW Latency Reduction. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, page 289–304.
[23] A. Bergen, Y. Coady, and R. McGeer. Client bandwidth: The forgotten metric of online storage providers. In *Proceedings of the Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim)*, pages 543–548. IEEE, 2011.
[24] A. N. Bessani, R. Mendes, T. Oliveira, N. F. Neves, M. Correia, M. Pasin, and P. Verissimo. SCFS: A Shared Cloud-backed File System. In *Proc. of USENIX ATC*, pages 169–180, 2014.
[25] E. Bocchi, I. Drago, and M. Mellia. Personal Cloud Storage Benchmarks and Comparison. *IEEE Transactions on Cloud Computing*, 5(4):751 – 764, 2017.
[26] J. Chen, M. Zhao, Z. Li, E. Zhai, F. Qian, H. Chen, Y. Liu, and T. Xu. Lock-Free Collaboration Support for Cloud Storage Services with Operation Inference and Transformation. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, pages 13–27, 2020.
[27] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing (SoCC)*, pages 143–154. ACM, 2010.
[28] Y. Cui, Z. Lai, X. Wang, and N. Dai. QuickSync: Improving Synchronization Efficiency for Mobile Cloud Storage Services. *IEEE Transactions on Mobile Computing*, 16(12):3513–3526, 2017.
[29] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras. Benchmarking Personal Cloud Storage. In *Proceedings of the Internet Measurement Conference (IMC)*, pages 205–212. ACM, 2013.
[30] I. Drago, M. Mellia, M. M Munafo, A. Sperotto, R. Sadre, and A. Pras. Inside Dropbox: Understanding Personal Cloud Storage Services. In *Proceedings of the Internet Measurement Conference (IMC)*, pages 481–494. ACM, 2012.
[31] S. S. Gill and R. Buyya. A Taxonomy and Future Directions for Sustainable Cloud Computing: 360 Degree View. *ACM Computing Surveys (CSUR)*, 51(5):104, 2018.

[32] Y. Go, N. Agrawal, A. Aranya, and C. Ungureanu. Reliable, Consistent, and Efficient Data Sync for Mobile Apps. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST)*, pages 359–372. USENIX Association, 2015.

[33] G. Gonçalves, I. Drago, A. P. C. Da Silva, A. B. Vieira, and J. M. Almeida. Modeling the Dropbox Client Behavior. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pages 1332–1337. IEEE, 2014.

[34] R. Gracia-Tinedo, M. S. Artigas, A. Moreno-Martinez, C. Cotes, and P. G. Lopez. Actively Measuring Personal Cloud Storage. In *Proceedings of the 6th International Conference on Cloud Computing (CLOUD)*, pages 301–308. IEEE, 2013.

[35] R. Gracia-Tinedo, Y. Tian, J. Sampé, H. Harkous, J. Lenton, P. García-López, M. Sánchez-Artigas, and M. Vukolic. Dissecting UbuntuOne: Autopsy of a Global-scale Personal Cloud Back-end. In *Proceedings of the 2015 Internet Measurement Conference(IMC)*, pages 155–168. ACM, 2015.

[36] S. Han, H. Shen, T. Kim, A. Krishnamurthy, T. E. Anderson, and D. Wetherall. MetaSync: File Synchronization Across Multiple Untrusted Storage Services. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 83–95, 2015.

[37] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey. Early Observations on the Performance of Windows Azure. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC)*, pages 367–376. ACM, 2010.

[38] B. Hou, F. Chen, Z. Ou, R. Wang, and M. Mesnier. Understanding I/O Performance Behaviors of Cloud Storage from a Client's Perspective. *ACM Transactions on Storage (TOS)*, 13(2):16, 2017.

[39] C. Hu, Y. Xu, P. Liu, J. Yu, S. Guo, and M. Zhao. Enabling Cloud Storage Auditing with Key-exposure Resilience under Continual Key-leakage. *Information Sciences*, 520:15–30, 2020.

[40] W. Hu, T. Yang, and J. N. Matthews. The Good, the Bad and the Ugly of Consumer Cloud Storage. *ACM SIGOPS Operating Systems Review*, 44(3):110–115, 2010.

[41] E. Jinlong, Y. Cui, P. Wang, Z. Li, and C. Zhang. CoCloud: Enabling efficient cross-cloud file collaboration based on inefficient web APIs. *IEEE Transactions on Parallel and Distributed Systems*, 29(1):56–69, 2018.

[42] E. Kruus, C. Ungureanu, and C. Dubnicki. Bimodal Content Defined Chunking for Backup Streams. In *Proceedings of the 8th USENIX conference on File and storage technologies (FAST)*, pages 18–18. USENIX Association, 2010.

[43] G. Lee, H. Ko, and S. Pack. An Efficient Delta Synchronization Algorithm for Mobile Cloud Storage Applications. *IEEE Transactions on Services Computing*, 10(3):341–351, 2017.

[44] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: Comparing Public Cloud Providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement(IMC)*, pages 1–14. ACM, 2010.

[45] S. Li, Q. Zhang, Z. Yang, and Y. Dai. Understanding and Surpassing Dropbox: Efficient Incremental Synchronization in Cloud Storage Services. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2015.

[46] Z. Li, C. Jin, T. Xu, C. Wilson, Y. Liu, L. Cheng, Y. Liu, Y. Dai, and Z.-L. Zhang. Towards Network-level Efficiency for Cloud Storage Services. In *Proceedings of the Internet Measurement Conference (IMC)*, pages 115–128. ACM, 2014.

[47] Z. Li, C. Wilson, Z. Jiang, Y. Liu, B. Y. Zhao, C. Jin, Z.-L. Zhang, and Y. Dai. Efficient Batched Synchronization in Dropbox-like Cloud Storage Services. In *Proceedings of the International Middleware Conference (Middleware)*, pages 307–327. ACM/IFIP/USENIX, 2013.

[48] Z. Li, Y. Zhang, Y. Liu, T. Xu, E. Zhai, Y. Liu, X. Ma, and Z. Li. A Quantitative and Comparative Study of Network-level Efficiency for Cloud Storage Services. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 4(1):1–32, 2019.

[49] K. Lillaney, V. Tarasov, D. Pease, and R. Burns. Agni: An efficient dual-access file system over object storage. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 390–402, 2019.

[50] P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish. Depot: Cloud Storage with Minimal Trust. *ACM Transactions on Computer Systems (TOCS)*, 29(4):12, 2011.

[51] Y. Mansouri, A. N. Toosi, and R. Buyya. Data Storage Management in Cloud Environments: Taxonomy, survey, and Future Directions. *ACM Computing Surveys (CSUR)*, 50(6):91, 2018.

[52] R. Nachiappan, B. Javadi, R. N. Calheiros, and K. M. Matawie. Cloud storage reliability for big data applications: A state of the art survey. *Journal of Network and Computer Applications*, 97:35–47, 2017.

[53] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon S3 for Science Grids: A Viable Solution? In *Proceedings of the International Workshop on Data-aware Distributed Computing (DADC)*, pages 55–64. ACM, 2008.

[54] R. Pontes, D. Burihabwa, F. Maia, J. Paulo, V. Schiavoni, P. Felber, H. Mercier, and R. Oliveira. Safefs: A modular architecture for secure user-space file systems: One fuse to rule them all. In *Proceedings of the 10th ACM International Systems and Storage Conference (SYSTOR)*, pages 1–12, 2017.

[55] C. Qu, R. N. Calheiros, and R. Buyya. Auto-scaling Web Applications in Clouds: A Taxonomy and Survey. *ACM Computing Surveys (CSUR)*, 51(4):73, 2018.

[56] M. Ruan, T. Titcheu, E. Zhai, Z. Li, Y. Liu, E. Jinlong, Y. Cui, and H. Xu. On the synchronization bottleneck of openstack swift-like cloud storage systems. *IEEE Transactions on Parallel and Distributed Systems*, 29(9):2059–2074, 2018.

[57] N. Samteladze and K. Christensen. DELTA: Delta Encoding for Less Traffic for Apps. In *Proceedings of the 37th Conference on Local Computer Networks (LCN)*, pages 212–215. IEEE, 2012.

[58] Y. Shi, X. Meng, J. Zhao, X. Hu, B. Liu, and H. Wang. Benchmarking Cloud-based Data Management Systems. In *Proceedings of the second International Workshop on Cloud Data Management (CloudDB)*, pages 47–54. ACM, 2010.

[59] J. Tang, Y. Cui, Q. Li, K. Ren, J. Liu, and R. Buyya. Ensuring Security and Privacy Preservation for Cloud Data Services. *ACM Computing Surveys (CSUR)*, 49(1):13, 2016.

[60] W. F. Tichy. RCS – A System for Version Control. *Software: Practice and Experience*, 15(7):637–654, 1985.

[61] A. Tridgell, P. Mackerras, et al. The rsync Algorithm. The Australian National University, 1996.

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

[63] M. Vrable, S. Savage, and G. M. Voelker. Bluesky: A Cloud-backed File System for the Enterprise. In *Proceedings of the 10th USENIX conference on File and Storage Technologies (FAST)*. USENIX, 2012.

[64] H. Wang, H. Qin, M. Zhao, X. Wei, H. Shen, and W. Susilo. Blockchain-based Fair Payment Smart Contract for Public Cloud Storage Auditing. *Information Sciences*, 519:348–362, 2020.

[65] J. Wang, L. Wu, S. Zeadally, M. K. Khan, and D. He. Privacy-preserving Data Aggregation Against Malicious Data Mining Attack for IoT-enabled Smart Grid. *ACM Transactions on Sensor Networks (TOSN)*, 17(3):1–25, 2021.

[66] H. Xiao, Z. Li, E. Zhai, and T. Xu. Practical Web-based Delta Synchronization for Cloud Storage Services. In *Proceedings of the 9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*. USENIX, 2017.

[67] H. Xiao, Z. Li, E. Zhai, T. Xu, Y. Li, Y. Wang, Q. Zhang, and Y. Liu. Towards Web-based Delta Synchronization for Cloud Storage Services. In *Proceeding of the USENIX Conference on File and Storage Technologies (FAST)*. USENIX Association, 2018.

[68] Q. Zhang, Z. Li, Z. Yang, S. Li, S. Li, Y. Guo, and Y. Dai. DeltaCFS: Boosting Delta Sync for Cloud Storage Services by Learning from NFS. In *Proceedings of the IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 264–275. IEEE, 2017.

[69] Y. Zhang, C. Dragga, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. ViewBox: Integrating Local File Systems with Cloud Storage Services. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST)*, pages 119–132, 2014.

[70] M. Zhao, Z. Li, E. Zhai, G. Tyson, C. Qian, Z. Li, and L. Zhao. H2Cloud: Maintaining the Whole Filesystem in an Object Storage Cloud. In *Proceedings of the 47th International Conference on Parallel Processing (ICPP)*, pages 1–10, 2018.