# CoCloud: Enabling Efficient Cross-Cloud File Collaboration based on Inefficient Web APIs

Jinlong E*, Yong Cui*, Peng Wang†, Zhenhua Li‡, Chaokun Zhang*
* Department of Computer Science and Technology, Tsinghua University, China
† School of Computer Science, Carnegie Mellon University, USA
‡ School of Software, TNLIST, and KLISS MoE, Tsinghua University, China
{*ejl14, zhangck12*}*@mails.tsinghua.edu.cn*, {*cuiyong, lizhenhua1983*}*@tsinghua.edu.cn*, *pengwang@cmu.edu*

***Abstract*—Cloud storage services such as Dropbox have been widely used for file collaboration among multiple users. However, this desirable functionality is yet restricted to the "walled-garden" of each service. At present, the only effective approach to cross-cloud file collaboration seems to be using web APIs, whose performance is known to be highly unstable and unpredictable.**

**Now that using inefficient web APIs is inevitable, in this paper we attempt to achieve sound *user-perceived* performance for cross-cloud file collaboration. This attempt is enabled by two key observations from real-world measurements. First, for each cloud, we are always able to deploy one or several nearby (client) proxies which can efficiently access the web APIs. Second, during file collaboration, significant similarity exists among different versions of a file. This can be exploited to substantially reduce inter-proxy traffic and thus shorten the data sync time.**

**Guided by the observations, we design and implement an open-source prototype system called CoCloud. Currently, it supports file collaboration among four popular cloud storage services in the US and China. Its performance is well acceptable to users under representative workloads, even approaching or exceeding intra-cloud performance in many cases.**

## I. Introduction

Personal cloud storage services, such as Dropbox, Google Drive, Microsoft OneDrive, and Baidu PCS [1], have quickly gained tremendous popularity in recent years, for they provide convenient data backup and automatic cross-device/user synchronization (sync). They are seen as a great advance over, or a useful complement to, traditional network file services via NFS, HTTP/FTP, or P2P protocols. More recently, they have been widely used for more advanced, user-desired functionalities, in particular *multi-user file collaboration* such as collaborative document editing and team coding.

However, this desirable functionality is yet restricted to the "walled-garden" of each cloud storage service, *i.e.,* file collaboration happens inside either Dropbox or OneDrive, but not both. Meanwhile, users' preferences for clouds are different for a number of technical and non-technical reasons. In addition to personal habits, clouds show great spatial/temporal variations in performance [2], and some are even unavailable in certain regions (*e.g.,* Dropbox and Google Drive have been banned in China). The unsatisfactory status quo urges us to enable file collaboration across heterogeneous clouds.

Imagine Alice, a user of Dropbox in Los Angeles, and Bob, a user of Baidu PCS in Beijing, intend to collaboratively edit a paper. An intuitive approach goes as follows. Whenever Alice

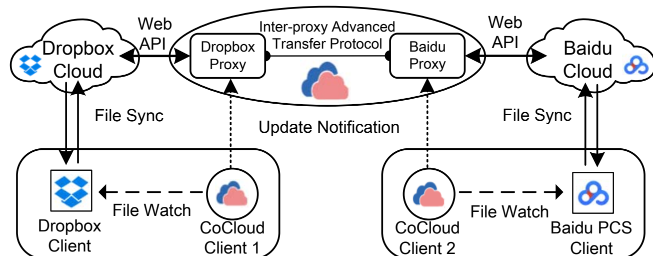


Fig. 1. Architectural overview of cross-cloud file collaboration with CoCloud.

finishes a version of the paper, she generates for Bob a URL to the version in Dropbox. Then, Bob downloads the version from the Dropbox URL by issuing an HTTP GET request. Likewise, Bob also returns to Alice the URL to his latest version in Baidu PCS after editing. Obviously, such a *manual sharing* approach is not only inconvenient but also inefficient, and is thus hardly adopted in the presence of frequent file edits.

An alternative approach, also the only seemingly effective approach at present, is to leverage the public web APIs provided by these cloud storage services, typically in a RESTful style for data access at *full-file* level. In the above example, Alice and Bob can avoid manual URL generation and sharing by invoking the web APIs of Dropbox and Baidu PCS, with the help of tools like IFTTT [3]. Unfortunately, the performance of the web APIs offered by popular cloud storage services is known to be highly unstable and unpredictable [2], let alone the lack of advanced data sync techniques such as delta compression, data deduplication, and small-file bundling (which are often supported by their PC clients and mobile apps) [4].

Now that using inefficient web APIs is inevitable, in this paper we attempt to achieve sound *user-perceived* performance for cross-cloud file collaboration, even under the workload of frequent file edits. This attempt is enabled by two key observations from our real-world measurements. First, for each popular cloud storage service, we are always able to deploy one or several nearby (client) proxies which can efficiently access the web APIs. Second, during file collaboration, significant similarity exists among different versions of a file. This can be exploited to substantially reduce inter-proxy traffic and

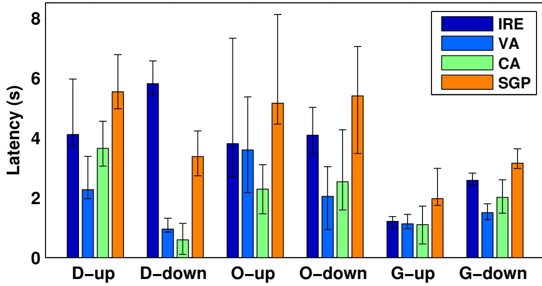


Fig. 2. Upload/download latency of a 10-MB file to/from Dropbox(D), OneDrive(O) and Google Drive(G) by four AWS nodes.

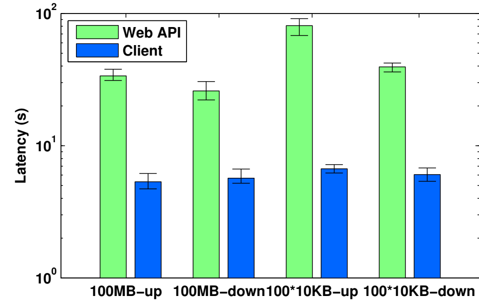


Fig. 3. Average upload/download latency of two typical file workloads by Dropbox web API and native client.

thus shorten the data sync time.

Guided by the observations, we design and implement a prototype system called CoCloud. As demonstrated in Fig. 1, a series of proxies (*i.e.,* rented virtual machines) are deployed close to the clouds involved, and a unified *inter-proxy advanced transfer protocol* is devised to take advantage of advanced data sync techniques (including deduplication, compression, bundling, and so forth). Besides, we also design relevant control mechanisms to guarantee timeliness and eliminate redundant updates during file collaborations.

Currently, CoCloud supports file collaborations among four popular cloud storage services in the US and China, namely Dropbox, Google Drive, Microsoft OneDrive, and Baidu PCS. Its source code is publicly available at *https://github.com/CoCloud/cocloud-demo*. Comprehensive real-world evaluation results confirm the efficacy and efficiency of CoCloud. In general, its performance is well acceptable to users under representative workloads. For example, an end-to-end file collaboration takes an average of about 30 seconds, approaching the performance of intra-Dropbox file collaboration. Sometimes, synchronizing a batch of small files from Dropbox to OneDrive takes less than 20 seconds, even exceeding the performance of intra-OneDrive file collaboration.

In summary, this paper makes the following contributions:

- From a number of real-world measurements, two key observations are drawn to overcome the inefficiency of cloud-storage web APIs (§ II).
- Based on the observations, we design CoCloud, an efficient cross-cloud file collaboration system that integrates a group of enabling solutions, including a cloud proxy deployment scheme, an inter-proxy advanced transfer protocol, and file collaboration control mechanisms (§ III).
- We implement an open-source CoCloud prototype, and extensive real-world evaluations demonstrate its well acceptable performance for cross-cloud collaboration (§ IV).

## II. Motivation

To synchronize data among devices in real time, most personal clouds provide a client to interact with the cloud server. Further, some clouds even support collaborative file editing among partners. As a great advance compared to the traditional static URL sharing and web access, the collaboration functionality attracts a multitude of non-computer professionals due to its simplicity, even overwhelming the widely used version control tools like Git and SVN.

TABLE I
IFTTT inter-cloud backup completion time

| Backup Service | File Size and Type | |
|---|---|---|
| | 30-KB Document | 10.1-MB Installer |
| Dropbox to OneDrive | 1min35s | 8min42s |
| Dropbox to Google Drive | 2min9s | 9min54s |

Taking the well-known Dropbox as an example, a typical collaboration protocol includes the interactions between client and both data storage server and control server. By analyzing SSL sockets hijacked with DynamoRIO [5], we find that before and after a client stores or retrieves data to/from the data storage server (the core file collaboration process), it needs to commit metadata (*e.g.,* hashes, file info) to the control server to either prepare or conclude the process.

On this basis, a number of capabilities are adopted by Dropbox client, optimizing both storage and transmission. These include chunking (*i.e.,* splitting data into certain size units for recovery simplification), deduplication (*i.e.,* transmitting only modified parts for storage and network bandwidth savings), bundling and data compression (*i.e.,* batching multiple small files or compressing large files for further traffic overhead reduction). Widely adopting these capabilities brings great gains to Dropbox's collaboration service.

Nevertheless, users' preferences for clouds are different due to both personal habits and performance considerations. Especially, as Dropbox is unavailable in some regions like China, a large number of users resort to local cloud services instead. These users may wonder how they can collaboratively edit papers or source codes with their remote Dropbox partners. Actually, data stored in all personal clouds are facing the vendor locked-in dilemma: external access is restricted to proprietary RESTful web APIs.

Based on these APIs, IFTTT [3] provides an intuitive inter-cloud backup approach: utilizing a proxy to unidirectionally forward files from one cloud to another. We test the performance under some typical workloads, and the results are

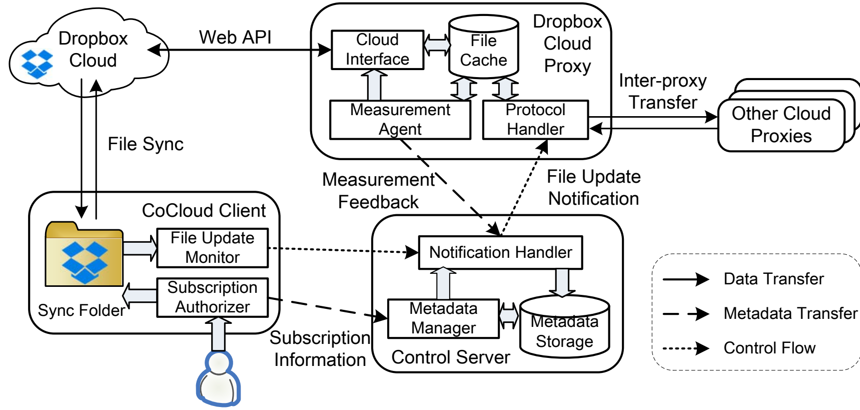


Fig. 4. System framework of CoCloud (including interaction among components and their internal module calls).

shown in Table I. Surprisingly, the completion time is overall extremely long (*e.g.,* merely a small document file may take several minutes). They can hardly satisfy the timeliness requirement of most file collaborations, nor do they provide file consistency guarantee or functionalities like file deletion and folder creation. As invoking web APIs is inevitable for external file access, we next try to deeply understand their inefficiency and possible improvements by further measurements.

Firstly, from four geo-distributed Amazon AWS nodes, we separately measured the latency of uploading to / downloading from a 10-MB file to four personal clouds (Dropbox, OneDrive, Google Drive, and Baidu PCS) for 50 times over a week. Fig. 2 illustrates the average, maximum and minimum latency with three clouds (Baidu PCS is not included because of extremely high latency). On the whole, cloud performance shows both spatial and temporal variations, in line with the measurements in [2]. It is worth mentioning that some nodes greatly outperform others on the latency to a certain cloud, and most of them only show temporal fluctuation within a narrow range (*e.g.,* California node for Dropbox download). By further constrasting with other virtual machine providers (*e.g.,* DigitalOcean [6]) and analyzing *traceroute* routing paths, we can conclude that *for each popular cloud, one or several proxies can be deployed nearby to efficiently access the web APIs*.

In addition, as the web APIs do not provide capabilities that are adopted in their native client counterparts, next we present the performance comparison between them to quantitize APIs' inefficiency. Measurements of Dropbox's upload and download are conducted on an Aliyun ECS Windows server in Silicon Valley. Fig. 3 describes the results (in log scale) of two typical cases: modifying a small fraction of a large file (100 MB) and transmitting a batch of small files ($100 \times 10$ KB). A remarkable latency gap is observed between the two approaches. Meanwhile, a large real-world data trace of cloud synchronization [4] indicates that $52\%$ files can be effectively compressed and $18\%$ can be deduplicated, and the optimal deduplication and compression settings tend to be similar among different versions of a given file. Therefore, we should adopt the capabilities found in native clients, and *further exploit the similar traits among different versions of a file to substantially reduce data traffic and thus sync time.*

## III. CoCloud Design

Guided by the above two key observations from real-world measurements, we design CoCloud to achieve sound user-perceived performance for cross-cloud file collaboration. The framework of CoCloud system along with the proxy deployment methodology is first described. Next, we specifically present *inter-proxy advanced transfer protocol* for inter-proxy transfer optimization and also *file collaboration control mechanisms* to guarantee timeliness and eliminate redundant updates.

### A. System Framework

CoCloud fulfills the whole file collaboration based on the interaction among components *CoCloud Client*, *Control Server*, and *Cloud Proxy*. The detailed system framework is shown in Fig. 4, and functionalities of each component are outlined as follows.

***CoCloud Client:*** The client program on users' terminals is lightweight, and it follows a subscription/push mode. A user is authorized by *Subscription Authorizer* based on OAuth 2.0 framework [7] when he subscribes to CoCloud. The *token* returned is stored along with the collaborator list in the control server. After the initial setup, *File Update Monitor* captures changes in the sync folder of cloud's native client, and it automatically synchronizes the updated files to other clouds.

***Control Server:*** As a central controller, it handles update notifications by selecting proxies to transfer the corresponding files. The work is mainly done by *Notification Handler*. In addition, *Metadata Manager* is in charge of maintaining metadata like user tokens and collaborator list as well as file version consistency.

***Cloud Proxy:*** *Cloud Interface* on each cloud proxy interacts with the corresponding cloud by APIs. *Measurement Agent* periodically measures link bandwidth, and proper proxies for each cloud are timely fed back to the control server based on their available bandwidths. On this basis, *Protocol Handler* is

repsonsible for transferring the notified file updates, where the data transfer optimization between peer proxies is achieved by an advanced transfer protocol.

### B. Cloud Proxy Deployment Scheme

According to our first key observation, one or several proxies are deployed nearby each cloud for efficient data access, and thus a *proxy-per-cloud* network architecture is built (Fig. 1). Specifically, for a personal cloud with centralized servers (*centralized cloud*, *e.g.,* Dropbox), which mainly shows spatial performance variation, CoCloud deploys two proxies with stably low upload and download transfer latency as the source and destination proxies respectively. For a personal cloud adopting multiple edge nodes (*multi-node cloud*, *e.g.,* Google Drive), whose best performance is achieved from different nodes temporally, CoCloud selects proxies for this kind of cloud according to the following approach.

Initially, we resolve the cloud's domain name from many DNSs to obtain a complete list of its edge nodes (their locations can be analyzed by the approach in [8]). Then a test file of size $S_{TF}$ is uploaded to and downloaded from these nodes and transferred among a number of geo-distributed CoCloud proxies, in order to measure the latency $T_l$ of every link. The overall bandwidth $BW_l$ of link $l$ can be estimated by $S_{TF}/T_l$.

To address the issue of temporal variation, the latency measurements are done *periodically*, and the values are then fed back to the control server in groups. Accordingly, some proxies with top download and upload bandwidth (lowest latency) to the nearest cloud node are picked out as source and destination proxies for each multi-node cloud. In practice, we select the proxies with latency less than $+10\%$ the lowest value.

### C. Inter-proxy Advanced Transfer Protocol

File creation and modification account for a remarkable proportion of file operations according to the real-world sync trace [4]. To boost the overall collaboration efficiency, cross-cloud data transfer should be well designed. Fortunately, on the basis of the aforementioned *proxy-per-cloud* architecture, we can focus on data transfer optimization between peer proxies. According to our second key observation, we exploit the similarity among file versions and propose *inter-proxy advanced transfer protocol* that integrates advanced data sync techniques, including *adaptively-chunked deduplication*, *wisely-adjusted compression*, and *multi-level bundling*.

**Adaptively-chunked Deduplication.** For a large proportion of files in collaboration, there is only slight modification from one version to the next. Therefore, the transfer performance can be greatly improved for large files, if only the modified parts are transferred. In view of this, deduplication techniques of both fixed-chunk and rolling-chunk like Rsync algorithm [9] were adopted by previous file transfer systems [10], [11], [12]. Though Rsync overcomes the ineffectiveness of fixed-chunk deduplication, which occurs when inserting bytes into a file [13], the pre-designated rolling chunk (window) size may not prove to be the best choice.

A typical process of Rsync algorithm can be described as follows: each cloud proxy maintains the metadata of files on the corresponding cloud. When a file is to be transferred, the source and the destination compare its metadata to decide if it is updated. Once successfully confirmed, the destination will send back the hash values of chunks in the old version with designated chunk size $c$. Thereafter, the source checks every continuous $c$ bytes in the updated file (shifting byte by byte) based on rolling hash values (4 bytes each), and further confirms the duplicated parts by comparing MD5 hash values (16 bytes each). Only the non-duplicated parts, along with the references of duplicated chunks (2 bytes each), are finally transferred to the destination. The destination then rebuilds the new version with such information and chunks from the old version. Finally, the metadata is updated according to the new version while the updated file is uploaded to the cloud storage server.

We define *deduplication ratio* $\gamma =$ *size of eliminated parts / original file size*. Then for a file of size $f$, the sizes of transferred data and the corresponding metadata are $(1-\gamma)*f$ and $\lceil f/c \rceil * (4+16) + \gamma * (f/c) * 2$, respectively. Note that the metadata size can be omitted here for the marginal overhead compared with the transferred data. Through the above analysis, we find that the deduplication ratio highly affects the actual traffic and transfer time, and it is determined by both the content (size, type, and modification scale) and *rolling chunk size* $c$.

For example, partners may revise different small files inside a tar file of Linux source code. Smaller chunk size for Rsync tends to bring higher deduplication ratio. In contrast, sometimes users may also operate on local databases and sync the backup files to their partners, for which larger chunk size can lead to lower overall overhead, as data tend to be appended to the end in most cases. As a consequence, a common chunk size suitable for all files can be hardly set.

Now that it is impossible to obtain $\gamma$ for a file unless it is actually transferred, we try to predict the best value based on our previous observation that the optimal chunk size is highly consistent among file versions. A specific process is: We pick a small collection of typical chunk sizes in advance, from hundreds of bytes to tens of kilobytes. When the file is updated for the first time, the rolling chunk size is chosen by the default setting of Rsync algorithm.

Thereafter, every time the destination cloud proxy receives an updated version, it runs the rolling-chunked deduplication *locally* with each chunk size $c_i$ in the collection to get the deduplication ratio $\gamma_i$ for this file. Note that the process do not have real time requirement, so it can be conducted whenever there is enough CPU resource, *e.g.,* in parallel with file uploading to destination cloud. Here we define the normalized deduplication ratios as the chunk size selection probability vector,

$$\vec{p}_{test} = \gamma_i / \sum_i \gamma_i \tag{1}$$

Then the correlation between currently recorded selection probabilities $\vec{p}_{cur}$ and predicted selection probabilities $\vec{p}_{next}$ is

$$\vec{p}_{next} = \vec{p}_{cur} * \alpha + \vec{p}_{test} * (1 - \alpha), \quad (2)$$

where $\alpha$ is a decay factor and is typically set as $n/(n+1)$ for the $n$-th adjustment. The chunk size corresponding to the *highest probability* in $\vec{p}_{next}$ will be adopted by the deduplication process next time.

**Wisely-adjusted Compression.** Data compression is deemed as a file transfer optimization technique for non-duplicate data as well as data after deduplication. However, an effective compression algorithm is somehow difficult to select, as any given compression algorithm like gzip, bzip2, or zlib achieves different compression levels for various files. We define *compression ratio $\beta$ = file size before compression / file size after compression*. Similar to deduplication, the compression ratio is determined by both compression algorithm and specific file structure.

Generally for a given file, high compression ratio generally concurs with long compression time. As the optimal compression setting (algorithm and level parameter) among versions of a file also tends to be consistent, we similarly predict the compression rate of each algorithm from history, and $\vec{\beta}$ denotes the recorded compression ratio list for a specific file.

Consider a complicated scenario where deduplication and compression are both enabled. We decide the scheme by comparing the computation rate and transfer rate. The transfer rate $r_t$, hash computation rate $r_a$, and compression rate $r_b$ can be inferred from the current network bandwidth and the recent computation. They are converted into relative rates with the aid of deduplication ratio $\gamma$ and compression ratio $\beta$,

$$(r'_a, r'_b, r'_t) = (\frac{1-\gamma}{\beta} r_a, \frac{r_b}{\beta}, r_t) \quad (3)$$

Then the overall computation rate (data generation rate) of CPU can be represented by

$$r_c = \frac{1}{1/r'_a + 1/r'_b} = \frac{(1-\gamma) r_a r_b}{\beta(1-\gamma) r_a + \beta r_b} \quad (4)$$

In (4), the best deduplication ratio $\gamma$ is adopted. Then each $\beta_i$ in compression ratio list $\vec{\beta}$ is evaluated. If $\forall \beta_i, r_c < r_t$, the calculation part is deemed as the bottleneck and thus compression will be disabled. Otherwise, we select the maximum $\beta_i$ that satisfies ${r_c}^i \geq r_t$ for compression.

**Multi-level Bundling.** To further boost transfer efficiency and reduce overhead, bundling mechanisms in multiple levels can be supplemented to the protocol. First, a persistent network connection is set up between peer cloud proxies to reuse for all the buffered files, instead of one connection per file. Moreover, large files are divided into a number of *transfer blocks* for data recovery consideration, whose size $\bar{S}_b$ can be typically set as 4MB based on network throughput as well as transmission failure rate [14]. Asynchronous application-layer acknowledgements are adopted instead of stop-and-wait ACK mode, as transfer blocks are not directly related to each other. Only unacknowledged blocks need to be retransmitted when network interruption or congestion occurs.

Finally, batching files smaller than the block size is designed as a fine-grained bundling mechanism. A *bundle block* is built with as many cached small files as possible. The hash values of the files are calculated, and then the contents of every file along with the corresponding hash and size are encapsulated into the bundle block. Moreover, a tag byte that indicates the special block is added at the head. The aggregate file size, along with the corresponding hash and file size set as well as the tag byte, should be less than $\bar{S}_b$. When the destination receives the bundle block, it retrieves the files based on size segments and check their hashes to confirm the transfer data. Compression will be further conducted on the bundle block, if the files all belong to types with high compression ratio.

We conclude the whole inter-proxy advanced transfer protocol as the following steps:

**Step 0:** The source cloud proxy is notified of the updated files, and it downloads them to its buffer.

**Step 1:** The source batches metadata of all buffered files and send it to the destination proxy to compare the file version.

**Step 2:** For files to update, different schemes are adopted, depending on the relation between file size $S_f$ and transfer block size $\bar{S}_b$:

- if $S_f \geq \bar{S}_b$ and it is an updated file, then do rolling-chunked deduplication and compression, and form transfer blocks;
- if $S_f \geq \bar{S}_b$ and it is a new file, then do compression and form transfer blocks;
- if $S_f < \bar{S}_b$, then bundle several files into a special block, and do compression.

**Step 3:** All the transfer blocks for the buffered files are transferred in a network connection, and async ACKs for the successful ones are returned from the destination.

**Step 4:** The destination handles the data correspondingly (decompression, rebuilding files with chunks, or de-bundling).

**Step 5:** Received files are uploaded to the destination storage server, and the best deduplication and compression parameters are predicted for future use when CPU is idle.

### D. File Collaboration Control Mechanisms

To further boost the collaboration efficiency, we next design file collaboration control mechanisms adopted in both the client and the control server. Fig. 5 depicts the complete collaboration control process.

We first consider control optimization in CoCloud clients. The collaboration directory is monitored by the local file system interface and file updates are notified to the control server in batch periodically. This avoids the considerable overhead of frequent notification with update inquiry poll or cloud callback mechanism [15]. More importantly, since file update operations are cached in *File Update Queue* during

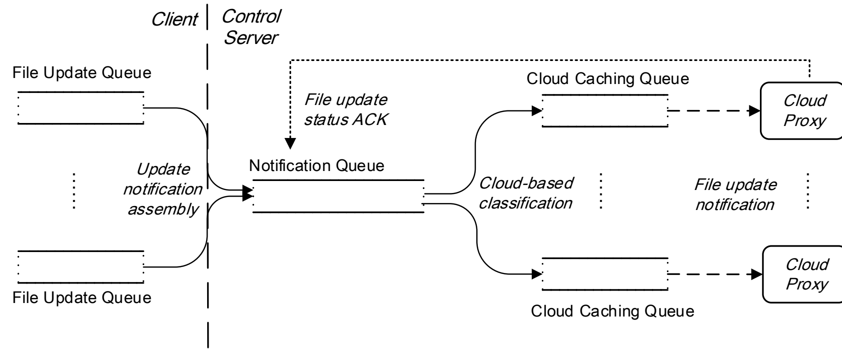


Fig. 5. File collaboration control process of CoCloud.

the interval, some redundant or unnecessary updates can be eliminated locally (*e.g.,* a file modification operation is removed once a deletion or another modification operation to the same file occurs subsequently). This *local redundant update elimination* mechanism can reduce the real update workloads to some extent.

When file update notifications arrive at the control server, they are added to a priority-based message queue named *Notification Queue*. Each notification includes *update operation*, *arrival timestamp*, and corresponding *file metadata*. These update notifications are then classified by source cloud (or destination cloud for folder creation and file deletion) and added into different *Cloud Caching Queues*. For a given cloud, proxies with top download/upload bandwidth (based on the periodical measurements in III-B) are used to serve operations to this cloud. The handling thread of every *Cloud Queue* notifies the corresponding *Cloud Proxy* of the file updates in sequence. Then *file update status* is fed back to the control server in real time.

## IV. Performance Evaluation

In this section, we first present the implementation of CoCloud prototype in details. On the basis of proxy deployment, we widely conduct measurements on the effectiveness of CoCloud, in different scenarios with a variety of typical realworld workloads. Finally, we study the end-to-end collaboration efficiency among the popular cloud services.

### A. CoCloud Prototype

We have implemented a prototype of CoCloud framework in approximately 5000 lines of both Java codes for cloud proxy and control server and C# codes for a lightweight Windows client. The prototype can provide efficient file collaboration service among users of four personal clouds: Dropbox, OneDrive, Google Drive, and Baidu PCS. The source code is available at *https://github.com/CoCloud/cocloud-demo*.

Particularly, the *Protocol Handler* module plays a key role in efficient sync. We implement the Rsync algorithm ourselves without calling libs, conveniently adding the mechanisms for optimization while avoiding the extra overhead when invoking a lib. To reduce storage overhead, the cached files are recycled periodically in idle time following LRU (Least Recently Used) scheme. In addition, the bandwidth measurement and feedback as well as the file update notification between a cloud proxy and the control server all rely on Apache MINA framework [16].

The *Cloud Interface* module supports four widely applied clouds currently, and new clouds can be easily added. When a number of files or transfer requests arrive simultaneously, CoCloud will start multiple threads to accelerate API upload or download corresponding to the designed control mechanisms. Besides, we implement CoCloud client on Windows platform, leveraging each cloud's OAuth interface for user authorization and *FileSystemWatcher* for file update monitoring.

### B. Experiment Setup

Based on our first key observation, we deploy cloud proxies in a number of geo-distributed AWS EC2 nodes (North Virginia, California, and so forth) and an Aliyun ECS node (in Beijing). The proxies corresponding to each cloud service are selected according to the periodical latency measurement in III-B. By this means, the sufficient throughput is guaranteed between cloud proxies and corresponding cloud storage servers.

Additionally, we take an Aliyun Silicon Valley ECS node as the control server, which can interact with all proxies with little latency. California EC2 node is also used to simulate IFTTT forwarding proxy for performance comparison, because it performs overall best among clouds for API access.

### C. Efficiency of CoCloud Data Transfer

We first evaluate the protocol performance on small files (i.e. $S_f < \bar{S}_b$), which is the most common scenario in collaboration. Typically, we conduct the evaluation by transferring a large batch of small files ($100 \times 10$KB), and measure the inter-cloud transfer time between three pairs of cloud (Dropbox-Baidu PCS, OneDrive-Baidu PCS, and Dropbox-OneDrive) in both directions, as shown in Fig. 6.

Servers in Beijing, California, and Virginia work as cloud proxies of Baidu PCS, Dropbox, and OneDrive respectively, since they are either along with or close to the corresponding cloud storage server. To avoid the influence of deduplication and compression, we randomly generate the contents of the

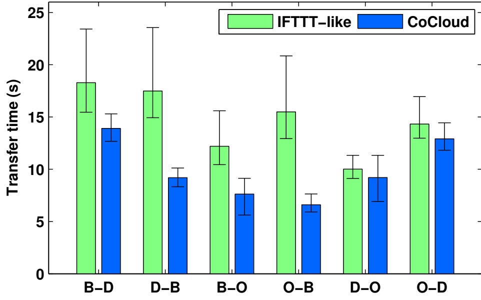


Fig. 6. Cross-cloud transfer time of a batch of small files among Dropbox(D), Baidu(B), and OneDrive(O).

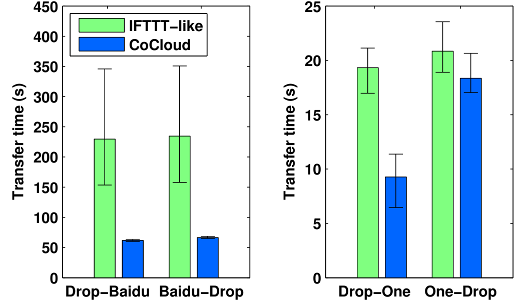


Fig. 7. Cross-cloud transfer time of FFmpeg source codes among Drop(box), Baidu, and One(Drive).

files. We also evaluate the transfer time of IFTTT-like forwarding approach (*i.e.,* a single server downloads files from a cloud and then uploads them to another cloud with only RESTful APIs) for contrast. Although performance disparities are found among different cloud peers due to network conditions, CoCloud always outperforms the intuitive scheme, with transfer time speedups up to $2.35\times$ and $1.56\times$ on average. The comparison confirms the effectiveness of multi-level bundling mechanism designed in the transfer protocol.

Besides small files like documents, partners also often collaborate on some relatively large files such as source codes, and even videos. Thus we also conduct the performance evaluation on large files, with FFmpeg [1] source codes [17] (about 50-60MB) as a sample. We collect 10 versions of codes (updated every 2 months) and transfer them between peer clouds sequentially according to the version number. Fig. 7 describes the transfer time between Dropbox and Baidu PCS, as well as between Dropbox and OneDrive. The IFTTT-like forwarding approach serves as a comparison. Similar to the transfer of small files, the improved deduplication and compression mechanisms in the transfer protocol can reduce the transfer time up to $73.1\%$ and $61.8\%$ on average in contrast to the forwarding approach.

We further evaluate the effectiveness of the adaptive selection mechanisms in the transfer protocol. For adaptively chunked deduplication, we conduct measurement on the network traffic incurred among the 10 different source code versions. Note that the time of deduplication varies little with different chunk size, so the network traffic is positively related to overall latency. The metric *Transfer Traffic Ratio (TTR)* is defined as the ratio of the traffic of a chunk size to the theoretically optimal one.

Fig. 8 shows the transferred data ratio of CoCloud in comparison with that of several typical chunk sizes, along with the theoretically optimal curve with $TTR = 1$. We can observe from the figure that the transfer traffic of CoCloud converges to the optimal curve very fast, outperforming all the fixed chunk sizes. According to our measurement, the real transfer traffic of CoCloud also keeps steady among different versions, which proves that the adjustment mechanism promotes the efficiency as well as robustness of inter-cloud data transfer.

In addition, we evaluate the wisely-adjusted compression mechanism by measuring and comparing the transfer and computation rates. For all the experimental proxies, compression is not the bottleneck and thus the algorithm with the highest compression ratio can be always adopted for compression. In reality, by overall considering network and computation overhead, the computation time for deduplication, compression and adjustment in CoCloud all occupy relatively small proportion compared with the whole transfer time, and in most time, computation is conducted in parallel with data transfer operations, well utilizing the relatively idle CPU resource. Therefore, the advanced transfer protocol adopted brings little overhead in practice.

For clouds adopting multiple edge nodes for API access, CoCloud provides multiple proxies for selection. We next take the representative multi-node cloud Google Drive as an example to evaluate the performance that multiple proxies work concurrently. Specifically, two AWS servers (California and Virginia) are selected as Google Drive proxies based on the latency measurement. Here we transfer the aforementioned FFmpeg source code files from Google Drive to the other three clouds, and dispatch the transfer workloads to the proxies according to their respective measured latencies.

Fig. 9 illustrates the overall transfer time of CoCloud policy, in comparison with adopting only one proxy (California node) for Google Drive ("Baseline" in the figure). While Baidu PCS proxy assembles workloads from both Google Drive proxies, Dropbox and OneDrive proxies each overlaps one Google Drive proxy. Among them, the transfer performance to Baidu PCS experiences the most obvious promotion ($27.9\%$ reduction in overall time), which well shows that concurrently utilizing multiple proxies can achieve efficiency promotion for multi-node clouds.

### D. End-to-end Collaboration Performance

We finally consider the performance of multi-user end-to-end file collaboration. To achieve this goal, we simulate the scenario that users of different cloud services use CoCloud for file collaboration simultaneously. It is manifest that the

[1]FFmpeg is an open-source program to record and convert audio and video.

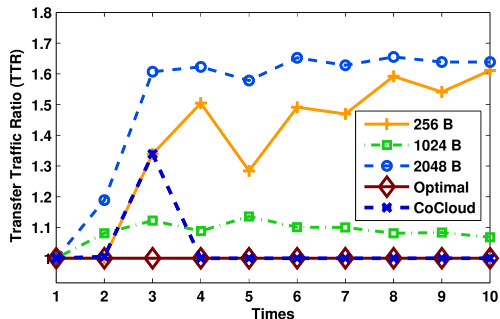


Fig. 8. Performance of adaptive chunk size selection (using one FFmpeg version at a time).

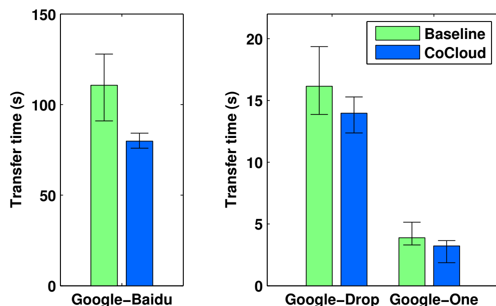


Fig. 9. Transfer time reduction with multiple proxies for multi-node cloud Google Drive.

end-to-end data sync time is highly influenced by the upload and download latency of its native client. As users in China are blocked from accessing Dropbox and Google Drive, and Baidu client sync is much slower than the other three, here we only show CoCloud performance among US users of Dropbox, OneDrive, and Google Drive.

Fig. 10 and Fig. 11 give their end-to-end collaboration time of two typical workloads (10-MB Zip file and a batch of document files around 10 KB each), in comparison with the performance of the native collaboration functionality (labelled by "Native client" bar). The figures indicate that an end-to-end CoCloud file collaboration for the above two typical workloads takes about 30 seconds on average, almost achieving the same level of efficiency as intra-cloud collaboration (only $1.42\times$ and $1.88\times$ collaboration time on average). Particularly, synchronizing $50\times$ 10-KB files from Dropbox to OneDrive takes less than 20 seconds, even outperforming the intra-OneDrive collaboration performance.

## V. Related Work

There has been a quantity of work on the increasingly popular cloud storage service, which our work is mainly related to in the following three aspects.

***Multiple cloud management:*** Some previous studies have proposed controlling multiple cloud services for redundant data backup. DepSky [18] builds a dependable cloud-of-clouds by distributing coded data into different public clouds, while MetaSync [19] and UniDrive [14] serve personal cloud users by adding more performance consideration and CYRUS [20] further considers privacy and reliability issues. However, these personal data backup managers require binding multiple clouds simultaneously as their backend and locally dividing every file into redundant chunks. In contrast, CoCloud is an efficient Dropbox-like end-to-end full-file collaboration service among heterogeneous personal clouds.

***Cloud storage capabilities:*** There have been quite a few relevant mature techniques these years, like Content Defined Chunking (CDC) [21], [12], [11], delta encoding, and deduplication [9], [22]. While these techniques are implemented in the native clients of some personal cloud services, the APIs provided for third-parties support none. According to our proxy deployment approach, CoCloud proxies are located near enough to API servers to overcome their inefficacy. Access to these proxies is very efficient in virtue of inter-proxy advanced transfer protocol, as if the cloud had provided the capabilities to third parties.

***Cloud measurement studies:*** A variety of previous research papers measure and benchmark performance of multiple clouds, from public clouds [23] to personal cloud services [8], [4]. In addition, [24] presents the architecture of mobile cloud storage services and their internal sync protocols, and QuickSync [13] further addresses the synchronization inefficiency problem of modern mobile cloud storage services. Some other papers elaborately study the well-performed Dropbox, by either pinning the inside architecture [10] or improving the inefficiency of some client capabilities [25]. Likewise, the internal structure of UbuntuOne is deeply studied by measurement in [26]. However, these measurements are all conducted on the cloud native clients. Besides, several papers have studied personal cloud web APIs, like [2], [27]. In contrast with them, we further make performance comparison between web API and native client, and more importantly, observe that proxies can be deployed nearby clouds to overcome API inefficacy.

## VI. Conclusion

In this paper, we address the cross-cloud file collaboration problem, attempting to achieve sound user-perceived performance based on the inefficient cloud web APIs. We first reveal by measurements that one or several proxies can be deployed nearby each cloud to overcome the web API inefficiency. On this basis, we propose CoCloud for file collaboration among heterogeneous clouds. It has a unified inter-proxy advanced transfer protocol and a collaboration control scheme. We implement an open-source CoCloud prototype to provide file collaboration service among four popular personal clouds. Extensive evaluations demonstrate that the system can well guarantee low cross-cloud transfer latency. Its performance even exceeds the intra-cloud collaboration performance in some cases.

We believe the CoCloud solution can be easily adopted as an efficient middleware among future clouds which has

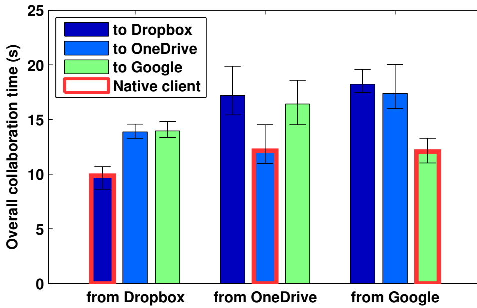


Fig. 10. End-to-end collaboration time of a 10-MB Zip file among three popular cloud services.

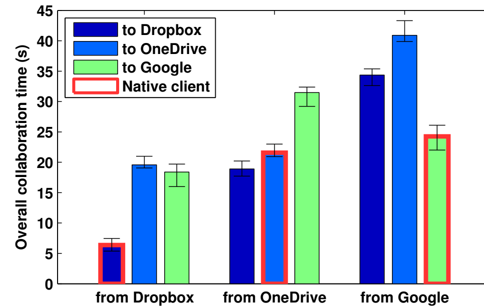


Fig. 11. End-to-end collaboration time of 50 documents (around 10 KB each) among three popular cloud services.

collaborative functionality and high client performance. In addition, as security and privacy protection becomes increasingly significant for cloud data services [28], security and privacy issues will be also considered for CoCloud system in our future work.

## Acknowledgment

This research is supported by the National High-Technology Research and Development Program ("863" Program) of China under grants 2015AA016101 and 2015AA01A201, National Natural Science Foundation of China under grant 61422206, Tsinghua University Initiative Scientific Research Program under grant 2014Z09103, and the CCF-Tencent Open Fund under grant RAGR20160105.

## References

[1] "Baidu PCS (Personal Cloud Storage)," http://developer.baidu.com/wiki/index.php?title=docs/pcs.
[2] G. Wu, F. Liu, H. Tang, K. Huang, Q. Zhang, Z. Li, B. Y. Zhao, and H. Jin, "On the Performance of Cloud Storage Applications with Global Measurement," in *Proc. of IEEE/ACM International Symposium on Quality of Service (IWQoS)*, 2016, pp. 31–40.
[3] "IFTTT," https://ifttt.com.
[4] 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 *Proc. of the 14th ACM SIGCOMM/SIGMETRICS Internet Measurement Conference (IMC)*, 2014, pp. 115–128.
[5] "DynamoRIO," http://www.dynamorio.org.
[6] "DigitalOcean," https://www.digitalocean.com/.
[7] D. Hardt, "The OAuth 2.0 authorization framework," *IETF RFC 6749*, 2012.
[8] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras, "Benchmarking Personal Cloud Storage," in *Proc. of the 13th ACM SIGCOMM/SIGMETRICS Internet Measurement Conference (IMC)*, 2013, pp. 205–212.
[9] A. Tridgell and P. Mackerras, "The Rsync Algorithm," *Joint Computer Science Technical Report Series, Australian National University*, 1996.
[10] I. Drago, M. Mellia, M. M. Munafò, A. Sperotto, R. Sadre, and A. Pras, "Inside Dropbox: Understanding Personal Cloud Storage Services," in *Proc. of the 12th ACM SIGCOMM/SIGMETRICS Internet Measurement Conference (IMC)*, 2012, pp. 481–494.
[11] B. Aggarwal, A. Akella, A. Anand, A. Balachandran, P. V. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese, "EndRE: An End-System Redundancy Elimination Service for Enterprises," in *Proc. of the 7th USENIX Symposium on Network System Design and Implementation (NSDI)*, 2010, pp. 419–432.
[12] A. Muthitacharoen, B. Chen, and D. Mazières, "A Low-bandwidth Network File System," in *Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, 2001, pp. 174–187.
[13] Y. Cui, Z. Lai, X. Wang, N. Dai, and C. Miao, "QuickSync: Improving Synchronization Efficiency for Mobile Cloud Storage Services," in *Proc. of the 21st ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2015, pp. 592–603.
[14] H. Tang, F. Liu, G. Shen, Y. Jin, and C. Guo, "UniDrive: Synergize Multiple Consumer Cloud Storage Services," in *Proc. of 16th ACM/IFIP/USENIX Annual Middleware Conference (Middleware)*, 2015, pp. 137–148.
[15] "Webhook," https://www.dropbox.com/developers/webhooks/tutorial.
[16] "Apache MINA," http://mina.apache.org/.
[17] "FFmpeg," http://ffmpeg.org/.
[18] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "DepSky: Dependable and Secure Storage in a Cloud-of-Clouds," in *Proc. of the 6th ACM European Conference on Computer Systems (EuroSys)*, 2011, pp. 31–46.
[19] S. Han, H. Shen, T. Kim, A. Krishnamurthy, T. Anderson, and D. Wetherall, "MetaSync: File Synchronization Across Multiple Untrusted Storage Services," in *Proc. of USENIX Annul Technical Conference (ATC)*, 2015, pp. 83–95.
[20] J. Y. Chung, C. Joe-Wong, S. Ha, J. W.-K. Hong, and M. Chiang, "CYRUS: Towards Client-Defined Cloud Storage," in *Proc. of the 10th ACM European Conference on Computer Systems (EuroSys)*, 2015, pp. 1–16.
[21] N. T. Spring and D. Wetherall, "A Protocol-Independent Technique for Eliminating Redundant Network Traffic," in *Proc. of ACM International Conference on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM)*, 2000, pp. 87–95.
[22] Y. Hua, X. Liu, and D. Feng, "Neptune: Efficient Remote Communication Services for Cloud Backups," in *Proc. of IEEE International Conference on Computer Communications (INFOCOM)*, 2014, pp. 844–852.
[23] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers," in *Proc. of the 10th ACM SIGCOMM/SIGMETRICS Internet Measurement Conference (IMC)*, 2010, pp. 1–14.
[24] Y. Cui, Z. Lai, and N. Dai, "A first look at mobile cloud storage services: Architecture, experimentation, and challenges," *IEEE Network*, vol. 30, no. 4, pp. 16–21, July/August 2016.
[25] 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 *Proc. of ACM/IFIP/USENIX International Middleware Conference (Middleware)*, 2013, pp. 307–327.
[26] R. Gracia-Tinedo, Y. Tian, J. Sampé, H. Harkous, J. Lenton, P. Garcá-López, M. Sánchez-Artigas, and M. Vukolic, "Dissecting UbuntuOne: Autopsy of a Global-scale Personal Cloud Back-end," in *Proc. of the 15th ACM SIGCOMM/SIGMETRICS Internet Measurement Conference (IMC)*, 2015, pp. 155–168.
[27] R. Gracia-Tinedo, M. Artigas, A. Moreno-Martínez, C. Cotes, and P. López, "Actively Measuring Personal Cloud Storage," in *Proc. of the 6th IEEE International Conference on Cloud Computing (CLOUD)*, 2013, pp. 301–308.
[28] 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*, vol. 49, no. 1, pp. 13:1–39, June 2016.