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

Jinlong E, Yong Cui, *Member, IEEE*, Peng Wang, Zhenhua Li, *Member, IEEE*, and Chaokun Zhang, *Student Member, IEEE*

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 feasible 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 that of intra-cloud collaboration in many cases.

Index Terms—Cloud storage, cloud computing, cross-cloud file collaboration, transfer efficiency

1 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 or paper editing.

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.

- *J. E, Y. Cui, and C. Zhang are with Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: {ejl14, zhangck12}@mails.tsinghua.edu.cn, cuiyong@tsinghua.edu.cn.*
- *P. Wang is with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213. E-mail: pengwang@cmu.edu.*
- *Z. Li is with the School of Software, TNLIST, and KLISS MoE, Tsinghua University, Beijing 100084, China. E-mail: lizhenhua1983@gmail.com.*

Manuscript received 4 Feb. 2017; revised 21 Aug. 2017; accepted 31 Aug. 2017. Date of publication 8 Sept. 2017; date of current version 8 Dec. 2017. (Corresponding author: Jinlong E.)
Recommended for acceptance by X. Gu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2017.2750161

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 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

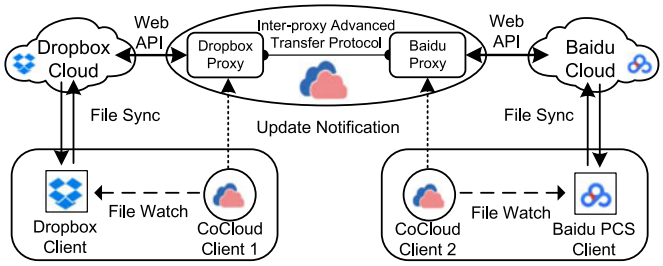

Fig. 1. Architectural overview of cross-cloud file collaboration with CoCloud.

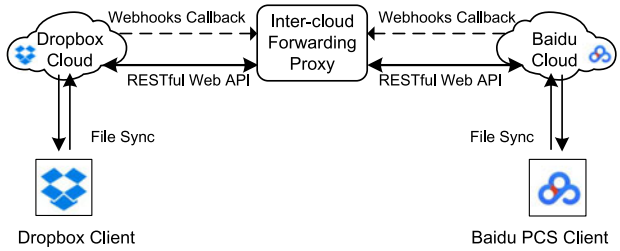

Fig. 2. IFTTT-like centralized forwarding proxy architecture.

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 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). To further optimize the cross-cloud data transfer with multiple proxies, a near-optimal algorithm is presented to evenly dispatch the request workloads to proxies and properly determine the transfer paths.

Besides, we design an online *inter-proxy dataflow scheduling algorithm* to achieve a moderate balance between the timeliness of data sync and system overhead. When the former is satisfied, the algorithm tries to minimize system overhead (particularly in terms of bandwidth). Otherwise, more bandwidth or proxies will be adaptively added to the system to deal with bursty workloads. We also design relevant control mechanisms to guarantee data consistency 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 25 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 (Section 2).
- Based on the observations, we design CoCloud, an efficient cross-cloud file collaboration system that integrates a bundle of enabling solutions, including a proximity-aware proxy deployment scheme, a unified inter-proxy advanced transfer protocol, and a cross-cloud data transfer optimization algorithm with multiple proxies (Sections 3.1, 3.2, 3.3, and 3.4).
- To boost collaboration timeliness while reducing system overhead, we further design an online inter-proxy dataflow scheduling algorithm. Additionally, we design file collaboration control mechanisms to guarantee data consistency and eliminate redundant updates (Sections 3.5 and 3.6).
- We implement an open-source CoCloud prototype. Extensive evaluations including both real-world experiments and trace-driven simulations demonstrate the well acceptable performance and high scalability of our system (Section 4).

2 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.

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 need 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. A typical architecture is depicted by Fig. 2, in which the proxy leverages Webhooks callback provided by personal clouds [6] to acquire file update notifications. We test the performance under some

TABLE 1
IFTTT Inter-Cloud Backup Completion Time

Backup Service	File Size and Type	
	30-KB Document	10.1-MB Installer
Dropbox to OneDrive	1 min 35 s	8 min 42 s
Dropbox to Google Drive	2 min 9 s	9 min 54 s

typical workloads, and the results are shown in Table 1. 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 other 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.

First, from four geo-distributed Amazon AWS nodes, we separately measured the latencies of uploading and downloading a 10-MB file to/from four personal clouds (Dropbox, OneDrive, Google Drive, and Baidu PCS) for 50 times over a week. Fig. 3 illustrates the average, maximum and minimum latency results with three clouds (Baidu PCS is not included because of extremely high latency). On the whole, cloud performance shows both spatial and temporal variations (i.e., the latency varies from different nodes and also fluctuates over time), 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 (*CA*) node for Dropbox download). By further contrasting with other virtual machine providers (e.g., DigitalOcean [7]) 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*.[1]

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 quantify APIs' inefficiency. Measurements of Dropbox's upload and download are conducted on an Aliyun ECS Windows server in Silicon Valley. Fig. 4 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 percent files can be effectively compressed and 18 percent 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 shorten sync time*.

3 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 *cloud proxy deployment scheme* is first described. Next, we specifically present *inter-proxy advanced transfer protocol* and *data transfer optimization algorithm* to optimize cross-cloud transfer, and *inter-proxy dataflow scheduling algorithm* and *file collaboration control mechanisms* to balance the data sync timeliness and system overhead.

1. The best download and upload nodes from the above measurement are selected as source and destination proxies respectively (details are described in Section 3.2).

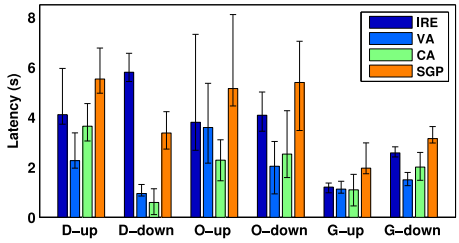

Fig. 3. Upload/download latency of a 10-MB file to/from Dropbox(D), OneDrive(O) and Google Drive(G) by four AWS nodes.

3.1 System Framework

CoCloud fulfills the whole file collaboration based on the interaction among components *CoCloud Client*, *Control Server*, and *Cloud Proxy*. A detailed system framework is shown in Fig. 5, 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 [8] 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 (instead of the server waiting for callback from clouds), and it automatically requests synchronization of the updated files to other clouds.

Control Server. As a central controller, it handles update notifications by scheduling proper proxies to transfer the corresponding files. The work is mainly done by the two-side interfaces *Notification Handler* and *Dataflow Scheduler* together. According to the scale of workloads and available bandwidths, the scheduler either periodically selects one or multiple proxies from a candidate set or dynamically adjusts the number of proxies. On this basis, *Metadata Manager* is in charge of tackling file inconsistency among collaborators, both to maintain the correct version and to restore the conflicting versions.

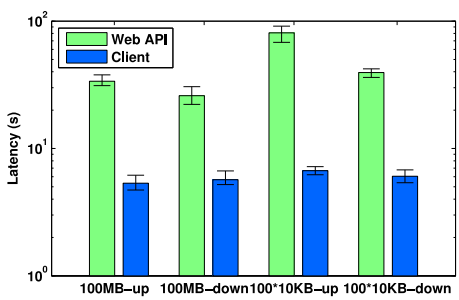

Fig. 4. Average upload/download latency of two typical file workloads by Dropbox web API and native client.

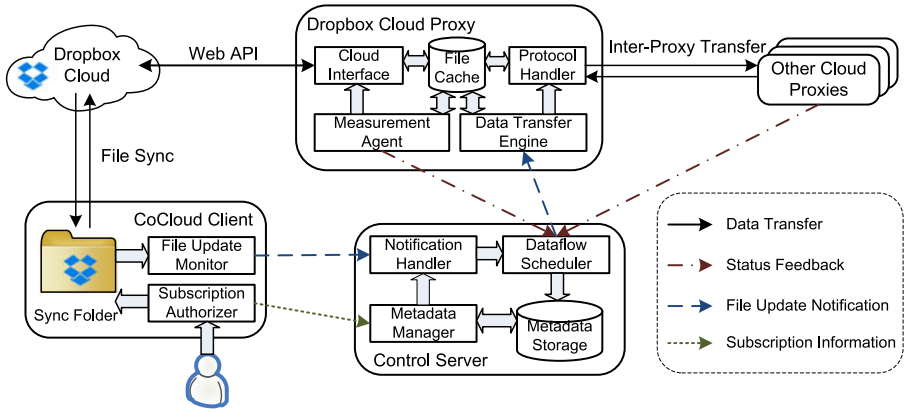

Fig. 5. System framework of CoCloud (including interaction among components and their internal module calls).

Cloud Proxy. Cloud Interface on each cloud proxy interacts with the corresponding cloud by either RESTful APIs of the cloud or some internal APIs we have detected. *Protocol Handler* is repsonsible for data transfer optimization between peer proxies with an advanced transfer protocol. In addition, *Measurement Agent* periodically measures link bandwidths to pick out proper cloud proxies, and timely feeds back proxies' available bandwidths to the control server for transfer workload dispatch. *Data Transfer Engine* is scheduled accordingly to transfer the notified file updates.

3.2 Cloud Proxy Deployment Scheme

According to our first key observation, one or several proxies are deployed close to 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 download and upload 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 by the following approach.

Initially, we resolve the cloud's domain name by a large number of DNSes to obtain a complete list of its edge nodes (their locations can be analyzed based on the approach in [9]). Then a test file of size S_{TF} is uploaded to and downloaded from these nodes and transferred among 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 or upload bandwidth (i.e., lowest latency) to the nearest cloud node are picked out as the source or destination proxies for each cloud. In practice, we select the proxies with latency less than $+10$ percent the lowest value. Based on these proxies, cross-cloud data transfer optimization is further designed for multi-node clouds (in Section 3.4).

3.3 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, the 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 [10] were adopted by previous file transfer systems [11], [12], [13]. Though rsync overcomes the ineffectiveness of fixed-chunk deduplication, which occurs when inserting bytes into a file [14], the predesignated rolling chunk (window) size may not prove to be the best choice.

Deduplication ratio γ (*= size of eliminated parts / original file size*), which highly affects the actual traffic (e.g., $(1-\gamma)*f$ for a file of size f) and transfer time, is determined by both the content (size, type, and modification scale) and *rolling chunk size* c.[2] Smaller chunk size brings higher deduplication ratio when collaborators revise different small files inside a Linux source code tar file, while larger chunk size can lead to lower overall overhead when synchronizing local database backup files among partners, as data tend to be appended in the latter case.

Now that it is impossible to obtain γ 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

2. Note that the corresponding metadata size is $\lceil f/c \rceil * (4+16) + \gamma * (f/c) * 2$ for a typical rsync process, which is omitted here for its marginal overhead compared with the transferred data.

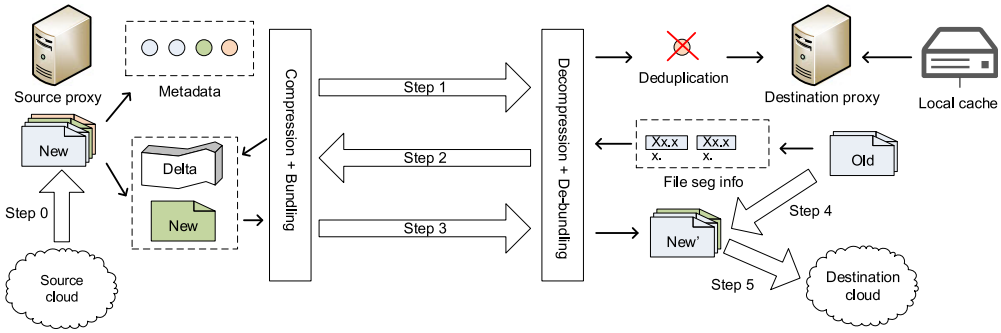

Fig. 6. The whole process of inter-proxy advanced transfer protocol.

transferred 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 γ_i for this file. Note that the process does not have a real-time requirement, so it can be conducted whenever there is enough CPU resource, e.g., in parallel with file uploading to the destination cloud. Here we define the chunk-size selection probability vector as the normalized deduplication ratio list,

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

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

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

where α is a decay factor and is typically set as $n/(n+1)$ for the nth 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 β = 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 settings (compression algorithms and compression level parameters) among versions of a file also tend to be consistent, we 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 the 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 recent computations. They are converted into relative rates with the aid of deduplication ratio γ and compression ratio β,

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

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

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

In (4), the best deduplication ratio γ is adopted. Then each β_i in the 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 β_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, which is reused for all the buffered files, instead of one connection per file transfer. 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 4 MB based on network throughput and transmission failure rate [15]. 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 proxy receives the bundle block, it retrieves the files based on size segments and checks 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 Fig. 6. The process can be described by steps as follows:

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

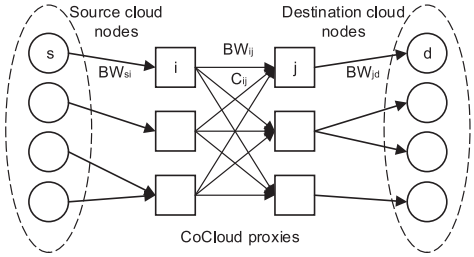

Fig. 7. A network topology example of cross-cloud transfer with multiple proxies.

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

Step 2: For files to be transferred, different schemes are adopted, depending on the current status and relation between the file size S_f and the 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 proxy.

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

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

3.4 Cross-Cloud Data Transfer Optimization Algorithm

Considering that cloud proxies may suffer from temporally poor performance of network from/to multi-node clouds, CoCloud deploys a group of proxies with low latencies for every multi-node cloud, and the best one can be selected periodically (see Section 3.2). In fact, if we adopt multiple proxies concurrently to dispatch the requested workloads, the cross-cloud data transfer can be further optimized. As the control server has collected three groups of latencies (namely the download latency from the source cloud, the upload latency to the destination cloud and the inter-proxy transfer latency), it can calculate the bandwidth of each link and then build up a Directed Acyclic Graph (DAG) like Fig. 7 with link bandwidths marked as edge weights. Then the goal is to determine a number of proxies that minimizes the overall transfer latency.

It seems that the problem can be reduced to Maximum Flow problem as optimal flow (or workload) allocation is involved. However, we show that the solutions to Maximum Flow are not available here. First, those solutions (algorithms) focus on the flow between source and destination cloud nodes, whereas addressing all the available cloud nodes will result in considerable overheads. More importantly, the aforementioned inter-proxy transfer protocol requires a file to be downloaded to the proxy before it is transferred, and the probable dataflow reduction after deduplication or compression violates the flow conservation principle. Rather, we design a two-step heuristic transfer optimization algorithm here to reduce the transfer latency to a near-minimum extent.

Here we mainly consider a general network topology between two multi-node clouds (just as depicted in Fig. 7), where a proxy may work for both clouds. Suppose that the size of current workload is w and the total number of cloud proxies is n. For two proxies i and j, we define the download bandwidth of proxy i from the nearest source cloud node as BW_{si}, the upload bandwidth of proxy j to the nearest destination cloud node as BW_{jd}, and the inter-proxy transfer bandwidth between i and j as BW_{ij}.

The first step is to use multiple proxies to concurrently download parts of the workload from the closest cloud node, which are then converged to a certain proxy. The workload allocation can be expressed as $w = \sum_i w_i$, $i = 1, 2, \ldots, n$. As the aforementioned transfer protocol (in Section 3.3) is adopted between proxies, here we also define the calculation rate as C_{ij} (generally all C_{ij}'s are equal), and $\eta = (1 - \gamma)/\beta$ represents the transfer ratio forecasted by file history traits according to the protocol. Then the transfer time of the workload from source cloud to proxy j through proxy i ($i = 1, 2, \ldots, n$) is

$$t_{ij} = w_i * (1/BW_{si} + \eta/BW_{ij} + 1/C_{ij}). \tag{5}$$

Note that when a proxy i serves for both clouds, the inter-proxy transfer is not needed (i.e., $w_i/BW_{ii} = 0$). It is manifest that the workload allocation achieves optimal only when all t_{ij}'s are equal and they can be represented by t_j.

Then the second step is to determine the proper proxy for converging and uploading the workload to the destination cloud. Since there are only a few deployed proxies, we can simply enumerate them for optimal full path estimation. Concretely, for the selected transfer server j, the overall transfer time is $t_j + w/BW_{jd}$. By this means, proxy j with minimum overall transfer time is determined as the destination cloud proxy for workload w.

Multi-Copy Transfer to One Cloud. In most cases, a file need be collaborated with a number of users from the same destination cloud. For some personal clouds that support file collaboration service, the best way is only transferring one file copy to the share folder among these destination users, and then they can leverage the cloud own collaboration mechanism. However, there are still a few personal clouds that are mainly used for data backup and do not well support file collaboration. For these collaboration-unsupported clouds, multiple file copies are inevitable to be transferred to the same destination cloud. Particularly, CoCloud reuses the download proxies and only transfers multiple copies by the selected upload proxy. This mechanism can well reduce the overall transfer time, which is only $t_j + m * w/BW_{jd}$ for uploading m copies to the destination cloud.

Transfer with Centralized Cloud. The above transfer optimization algorithm can be adopted between a *multi-node cloud* (cloud with distributed nodes) x and a *centralized cloud* (cloud with centralized servers in one location) y with little

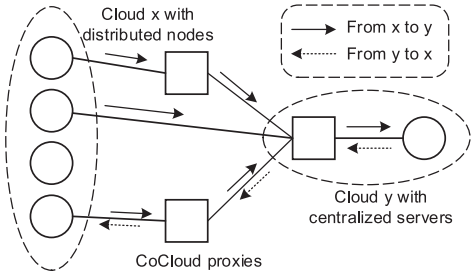

Fig. 8. Transfer between multi-node cloud and centralized cloud.

modification. For data transfer from cloud x to cloud y, the only difference is that CoCloud need not select the proxy for data uploading, but just leverages the previously deployed destination cloud proxy. Conversely, for data transfer from cloud y to cloud x, the whole file is downloaded by the unique source cloud proxy, and only one destination cloud proxy is determined according to the minimum overall transfer time. The corresponding transfer process is illustrated in Fig. 8.

3.5 Inter-Proxy Dataflow Scheduling Algorithm

During practical file collaboration, data sync is always expected to occur as soon as a user updates files, for better user experience as well as version conflict reduction. However, the massive cross-cloud dataflow induced by a large scale of file workloads (especially those cannot be optimized with the protocol in Section 3.3) may bring heavy overhead to all proxies. The above transfer optimization (in Section 3.4) aims at a given number of proxies without considering system overhead, and thus it may not fulfill the scalability requirement, especially for bursty workloads. Therefore, we next design a two-stage online algorithm named *inter-proxy dataflow scheduling algorithm* including update priority assignment and file workload dispatch to guarantee well acceptable collaboration experience while reducing the system overhead as much as possible.

The first stage starts when file update notifications arrive. Each notification includes *update operation*, *arrival timestamp*, and corresponding *file metadata*. They are added to a priority-based message queue named *Notification Queue*. In this stage, we optimize the overall performance by adjusting the order in which the notifications are handled (i.e., assigning priority).

First Come First Served (FCFS) and Shortest Task First (STF) are two typical scheduling policies. They try to optimize the overall completion time by arranging some tasks to be completed earlier. However, directly adopting either policy may strongly degrade performance: with FCFS, small tasks have to wait long even if they come just a little later than a large task; with STF, a large task is probably starved when many small tasks come continuously.

In our scenario, users are less sensitive to the collaboration timeliness of large files than that of small files. Therefore, we take both file update arriving time and file size into account. Define File Update Priority (FUP) as the following virtual completion time metric, and the file update with the *smallest* FUP value will be handled first

$$FUP(k) = t_A^k + t_{OT}^k, \forall\ file\ update\ k, \tag{6}$$

where t_A corresponds to the *arrival timestamp* attached in the file update notification, and t_{OT} represents the overall cross-cloud transfer time for a file of size S_f.

Considering the general cross-cloud transfer topology in Fig. 7, with two designated proxies i and j (workload dispatching to proxies is described later), t_{OT} can be estimated as,

$$t_{OT} = S_f * (1/BW_{si} + \eta/BW_{ij} + 1/C_{ij} + 1/BW_{jd}), \tag{7}$$

among which download bandwidth from source cloud BW_{si}, upload bandwidth to destination cloud BW_{jd}, inter-proxy transfer bandwidth BW_{ij}, calculation rate C_{ij} and transfer ratio η are acquired from the periodical measurements or forecasted by history traits of a file (as shown in Section 3.4). Note that when a proxy i serves for both clouds, the inter-proxy transfer is not needed, and t_{OT} can be calculated by $S_f * (1/BW_{si} + 1/BW_{id})$.

It is worth mentioning that collaboration operations like *folder creation* and *file deletion* do not involve cross-cloud data transfer, so they can be simply finished by the destination cloud proxy. Correspondingly, CoCloud sets a fixed small t_{OT} value for them, and thus these low-cost operations are most likely to be handled before common file operations (creation or modification).

In the second stage, the FUP-ordered update notifications are classified by the source cloud (or destination cloud for the folder creation and file deletion operations) and added into different *Cloud Caching Queues*. For a given cloud, when the queue is relatively short, only two proxies with top download/upload bandwidth are used to serve operations to the cloud. The handling thread of every *Cloud Queue* notifies the corresponding proxy of the file updates in sequence. All proxies upload and download different files with multiple threads concurrently, to make the best of their available bandwidths. Meanwhile, a background thread monitors the real-time available bandwidth BW_a in each proxy to avoid congestion caused by too many concurrent file workloads.

Adaptive Proxy Adoption Scheme. To balance the timeliness of bursty workloads and system overhead, here we further design an adaptive proxy adoption scheme. Additional proxies will be adopted when bandwidth shortage occurs in the working proxies without fixed periodicity. Specifically, whenever $BW_a \leq BW_o * \theta_c$, where BW_o represents the initially measured overall bandwidth of the proxy and θ_c is a congestion threshold, a proxy in the same location (for *centralized cloud*) or another proxy in the selected proxy set (for *multi-node cloud*) is added to balance the load. Suppose that there are currently n source (or destination) proxies ($P = \{p_i\}, i = 1, 2, \ldots, n$) for a certain cloud. To fully utilize the available bandwidth of the existing proxies, only when

$$BW_a^i \leq BW_o^i * \theta_c,\ \forall p_i \in P, \tag{8}$$

this number increases to $n + 1$. In addition, we also set a leisure threshold θ_l to save proxy resource. Here only when

$$BW_a^i \geq BW_o^i * \theta_l,\ \forall p_i \in P, \tag{9}$$

the proxy with lowest BW_o will be recycled after finishing the current file transfer (but not considered for new file workload dispatch), and then the number of proxies decreases to $n - 1$. We adopt such a lazy decrease mechanism since it can reduce

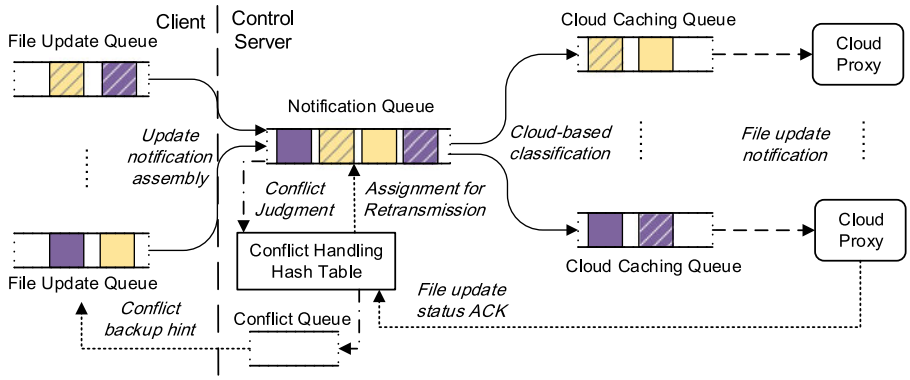

Fig. 9. The complete file collaboration control process of CoCloud (file updates from different clients are scheduled in the control server).

the possibility of proxy number fluctuation. Note that, to reduce network overhead, all proxies are scheduled by the control server separately and no information is transferred among proxies.

Based on the in-use proxies, we finally consider workload dispatching. As all the proxies (source or destination) for a *centralized cloud* are in the same location, the source and destination proxies with highest BW_a are selected for the next file workload. In contrast, the proxies for a *multi-node cloud* are geo-distributed, and we estimate the real-time bandwidth of each cloud-proxy-proxy-cloud path (in Fig. 7) as

$$BW_p = \frac{1}{1/BW_i^s + \eta/\min(BW_o^s, BW_i^d) + 1/BW_o^d}, \qquad (10)$$

among which η is the aforementioned transfer ratio, and BW_i^s, BW_o^s, BW_i^d, BW_o^d represent the input and output bandwidths of each pair of source and destination proxies. The source and destination proxy pair corresponding to the largest BW_p will be selected for the next transfer. Note that the above process of proxy selection is conducted by the control server based on real-time proxy bandwidths piggybacked in *file update status* application-layer acknowledgements. Particularly for an updated file, we give priority to transferring it to those destination proxies with its previous versions for deduplication, and thus other destination proxies can be eliminated before the bandwidth-based proxy selection. With this *multi-level proxy selection mechanism*, we are able to not only balance the workloads among each proxy but also save their transfer bandwidths to a large extent.

3.6 File Collaboration Control Mechanisms

To further boost the collaboration efficiency and guarantee data consistency, additional control mechanisms are adopted in both the client side and the control server. Fig. 9 depicts the collaboration control process of CoCloud, involving both the above scheduling algorithm and the following control mechanisms.

Local Redundant Update Elimination Mechanism. We first consider control optimization in CoCloud clients. The collaboration directory is monitored by the local file system interface and the control server are notified of file updates in batch periodically. This avoids the considerable overhead of frequent notifications with update inquiry poll or cloud callback mechanism [6] which is often adopted by the IFTTT-like centralized forwarding architecture (in Fig. 2). More importantly, since file update operations are cached in *File Update Queue* during 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 elimination mechanism can reduce the real update workloads to a considerable extent.

Hash-Based Conflict Handling Mechanism. In addition, as an advanced functionality, file versioning is restricted to most clouds' native collaboration without public APIs provided. However, version conflict is very likely to occur in collaboration, when each user successfully updates a file version in his own cloud and these versions are notified for cross-cloud transfer concurrently. To reduce control overhead and guarantee scalability, CoCloud tackles the consistency problem by a lightweight hash-based conflict handling mechanism, instead of database locking mechanism or Paxos-based qurum locking protocol [16]. We elaborate the mechanism as follows.

A hash table named *Conflict Handling Hash Table* (shown in Fig. 9) is maintained in the memory, storing hash indexes of file update notifications being processed. Before an arrived update notification is added to *Notification Queue*, its hash value is calculated by MD5 algorithm based on file name and collaboration group ID, which are attached in *file metadata*. When multiple update notifications of the same file arrive simultaneously from users of different clouds, the earliest-arrived one will possess an item in *Conflict Handling Hash Table*. This one is deemed as the correct version, and other later-arrived versions to the same hash item all experience hash conflicts. Such a conflicting version will be renamed and moved to the owner's *backup* folder to avoid *lost update problem* [17]. Meanwhile, a hint of file conflict and the corresponding backup will be sent to the owner in time. Then the conflicting file can be merged with the original correct version manually.

On the other hand, the correct file version will keep possessing the corresponding hash index (defined as a *lease*) until file updates to all collaborators are fed back from the destination cloud proxies (through *file update status ACKs* in Fig. 9). The lease ends and the hash item is removed then, thus unlocking the subsequent operations to the same file. Note that the new versions of the file arrive during the lease are also viewed as conflicting versions, and thus they are not transferred to collaborators but backed up to owners. In addition, a failed update can be tackled by reputting the update

notification into *Notification Queue* for retransmission and extending the lease of hash index, if no new update notification of the file arrives when the current lease ends. However, the extended lease is unprotected, which ends at once (along with cancel of retransmission) whenever a new file update arrives. Such a design can prevent a hash index from being obsolete for a long time due to transfer errors, and it also guarantees fairness among different file versions. Besides handling conflict, this hash-based mechanism can also remove redundant file updates when previous updates are cached (similar to the above local elimination mechanism).

4 Performance Evaluation

In this section, we first present the implementation of CoCloud prototype in details. On the basis of proxy deployment and real-world data trace analysis, we widely conduct measurements on the effectiveness of the above designed protocol and algorithms, in different scenarios with a variety of typical real-world workloads. Finally, we study the end-to-end collaboration efficiency among the popular cloud services.

4.1 CoCloud Prototype

We have implemented a prototype of CoCloud framework in approximately 8,000 lines of both Java codes for the control server and cloud proxies 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 *Data Transfer Engine* module (shown in Fig. 5) plays a key role in efficient data 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 Least Recently Used (LRU) scheme. In addition, the bandwidth measurement and feedback as well as workload dispatch rely on the interaction between each cloud proxy and the control server with Apache MINA [18] framework.

The *Cloud Interface* module supports four widely applied clouds currently, and new clouds can be easily added. It is worth mentioning that by capturing and analyzing network traffic of Baidu NetDisk APP, we have detected a number of Baidu PCS URIs adopted internally, which can better utilize upload and download bandwidths. We encapsule the internal functions into APIs, and adopt them instead of Baidu RESTful APIs to boost overall collaboration efficiency. Besides, when a number of files or transfer requests arrive simultaneously, CoCloud also leverages multiple threads to accelerate upload and download APIs as well as inter-proxy transfer corresponding to the system design. At last, we implement CoCloud client on Windows platform, utilizing each cloud's OAuth interface for user authorization and *FileSystemWatcher* for file update monitoring.

4.2 Experiment Setup

Based on our first key observation, we deploy cloud proxies in a number of geo-distributed AWS EC2 nodes (in North Virginia, California, Oregon, Ireland and so forth) and an Aliyun ECS node (in Beijing). The proxies corresponding to each cloud service are selected according to the periodical latency measurements in Section 3.2. By this means, 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.

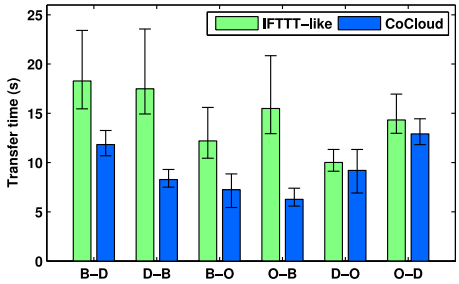

Fig. 10. Cross-cloud transfer time of a batch of small files among Dropbox(D), Baidu(B), and OneDrive(O).

In addition, to evaluate the scalability of our scheduling algorithm, we take the aforementioned cloud synchronization data trace [4] for deep analysis. It was collected from 153 users of six popular cloud services with 222,632 files over half a year. Particularly, we select the records on June 25, 2013 from 16:19 to 18:01 for scalability evaluation of dataflow scheduling algorithm, which has totally 8,260 files ranging from 3 Bytes to approximately 35 MB, and mainly includes two file modification (task arriving) bursts corresponding to two intensive task scheduling periods.

4.3 Performance of Inter-Proxy Advanced Transfer Protocol

To evaluate the efficiency of inter-proxy advanced transfer protocol, we first evaluate the protocol performance on small files (the size threshold $\bar{S}_b$ is set as 4 MB here according to [15]), which is the most common scenario in collaboration. Typically, we conduct the evaluation by transferring a large batch of small files (100×10 KB), and measure the inter-cloud transfer time between each pair of clouds (Dropbox, OneDrive, Baidu PCS) in both directions, as shown in Fig. 10.

Cloud proxies in Beijing, California, and Virginia serve for Baidu PCS, Dropbox, and OneDrive respectively, since they are either along with or close to the corresponding cloud storage servers. To avoid the influence of deduplication and compression, we randomly generate contents of the files. We also evaluate the transfer time of IFTTT-like forwarding approach for contrast. Although performance disparities are found among different cloud peers due to network conditions, CoCloud always outperforms the contrastive approach, with transfer time reduction up to 59.54 and 36.53 percent on average. The comparison confirms the effectiveness of the 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 source codes (an open-source program to record and convert audio and video) [19] as a sample. We collect 10 versions of code tar files (about

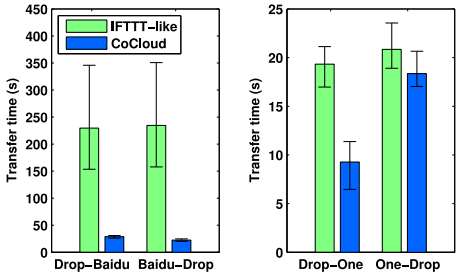

Fig. 11. Cross-cloud transfer time of FFmpeg source codes among Drop (box), Baidu, and One(Drive).

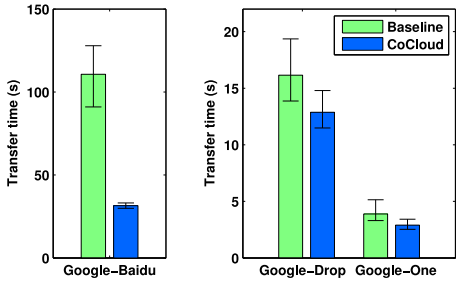

Fig. 13. Transfer time reduction with multi-proxy optimization for multi-node cloud Google Drive.

50-60 MB in size, updated every 2 months) and transfer them between peer clouds sequentially according to the version number. Fig. 11 describes the transfer time between Dropbox and Baidu PCS, as well as between Dropbox and OneDrive. Likewise, 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 speed up the transfer up to $10.40\times$ and $6.38\times$ on average in contrast to the forwarding approach.

We further evaluate the effectiveness of the adaptively chunked deduplication mechanism in the transfer protocol. Concretely, we conduct measurement on the network traffic incurred among the 10 different versions of FFmpeg source codes. Note that the time of deduplication varies little with different chunk sizes, so the network traffic is positively related to the 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. 12 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 above mechanism promotes the efficiency as well as robustness of the cross-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, computation is not the bottleneck, and thus we are able to adopt the algorithm with the highest compression ratio for compression every time. In reality, by overall considering network and computation overhead, we find that the computations for all deduplication, compression and adjustment in CoCloud occupy a relatively small proportion of the whole transfer time. Besides, in most time, computation can be conducted in parallel with data transfer operations, well utilizing the idle CPU resource. Therefore, the advanced transfer protocol adopted brings little overhead in practice.

4.4 Performance of Cross-Cloud Data Transfer Optimization

For clouds adopting multiple edge nodes for API access, CoCloud deploys multiple proxies for transfer optimization. We next take the representative multi-node cloud Google Drive as an example to evaluate performance of the transfer optimization algorithm. Specifically, three AWS servers (California, Virginia and Ireland) are selected as Google Drive proxies based on the initial measurement. Here we transfer the aforementioned FFmpeg source codes from Google Drive to the other three clouds, and dispatch the transfer workloads to the three proxies according to their respective available bandwidths.

Fig. 13 illustrates the overall transfer time of CoCloud algorithm, in comparison with adopting only one proxy (California node) for Google Drive ("Baseline" in the figure). While Baidu PCS proxy assembles workloads from all three 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 (63.82 percent reduction in overall time), which well shows the efficiency promotion for multi-node clouds.

In addition, CoCloud can guarantee considerably low overall time when a file is transferred to a number of collaborators, who are users of the same cloud. A typical scenario is when a Dropbox user transfers the edited files to several of his Baidu PCS partners. Fig. 14 describes the comparison between CoCloud and the IFTTT-like forwarding approach on the total transfer time of a 10-MB Zip file or 200 × 10-KB

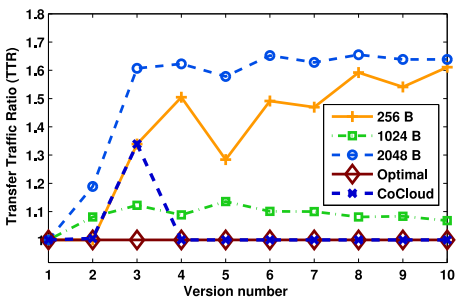

Fig. 12. Performance of adaptive chunk size selection (using one FFmpeg version at a time).

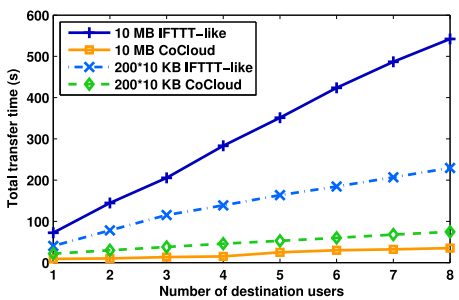

Fig. 14. Transfer time of two typical file workloads to multiple destination users.

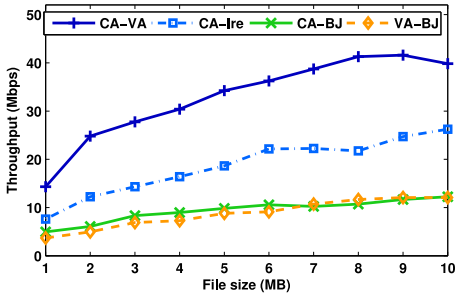

Fig. 15. Impact of file size on the inter-proxy network throughput.

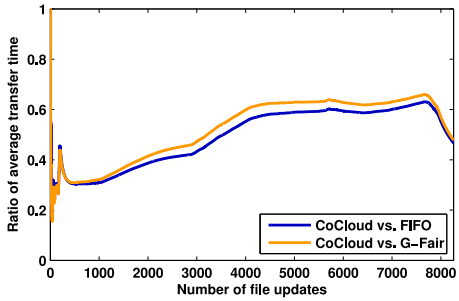

Fig. 16. Performance of CoCloud scheduling in comparison with typical scheduling algorithms.

small document files (the proxy settings are the same as above). As the user of the destination cloud increases, the transfer time of IFTTT-like forwarding approach rises dramatically, whereas CoCloud shows a quite slow and steady increasing trend. Particularly, the promotion for the 10-MB file is up to $15\times$ when the number of users is only 8. This is because we only need to send one replica of the file to the destination proxy, and then upload one copy for each collaborator, which is much faster than simply forwarding the file n times for n collaborators.

4.5 Performance of Inter-Proxy Dataflow Scheduling

Here we first measure the throughput when files of different sizes are transferred between peer cloud proxies. The results of some typical peers are plotted in Fig. 15. The figure indicates that higher throughput is achieved, as the size of each file increases while the increasing rate drops. This can well support not only the aforementioned setting of transfer block size threshold, but also mechanisms of multi-thread file transfer and congestion avoidance in CoCloud dataflow scheduling algorithm, i.e., an appropriate quantity of files are transferred concurrently. According to the bandwidth measurement and the file sync trace analysis, the congestion threshold θ_c and the leisure threshold θ_l can be set as 20 and 60 percent respectively.

We next conduct a large-scale simulation with the selected file update trace (in Section 4.2) to evaluate the scalability of dataflow scheduling algorithm. For sync timeliness consideration, the average cross-cloud file transfer time is an important metric. Thus we focus on the ratio of CoCloud average transfer time to that of two contrastive scheduling algorithms, FIFO and group-based fair sharing (*G-Fair*). In our scenario, the G-Fair algorithm schedules a group of tasks (up to 10) concurrently to share the bandwidth resource. Note that here we adopt one pair of source and destination proxies for data transfer, and the average path bandwidth is set as 10 Mbps according to the real-world measurement of deployed proxies.

Fig. 16 plots the variation trend of two ratios as the number of file updates increases. We observe that the ratio curves drop dramatically at the end of the test duration when a few large files arrive. This is because a number of following small files need wait for available bandwidth resource by both two contrastive algorithms, while CoCloud boosts the resource utilization by postponing the large files to proper places. Accordingly, the ratio curves experience a slowly increase trend in the middle period, since the previous large files should be handled at that time. As a whole, the average transfer time ratios of CoCloud to both algorithms are around 50 percent on average and just over 60 percent to the maximum. The obvious promotion indicates that CoCloud can well guarantee the timeliness of a large number of concurrently arriving file updates.

We also conduct another large-scale simulation (with the same data trace as above) to evaluate the effectiveness of the adaptive proxy adoption scheme. Here we focus on the number of proxies adopted by a cloud during the workload transfer period, which can well reflect the system overhead. According to the actual situation of EC2 nodes, the available bandwidth at a proxy is 50 Mbps to the maximum, which is shared by the transferred workloads and varies over time.

Fig. 17 depicts how the number of proxies varies as the workloads are scheduled for cross-cloud transfer. During the whole process, CoCloud keeps quite low number of adopted proxies (less than 2 on average and up to 3 only for a short period). At the same time, the number variation shows an overall stable trend with enough long periodicity of proxy number changes (about 31 percent of the total transfer time on average), which indicates the cost brought by proxy increase and decrease is also well acceptable. As shown in the figure, in contrast with adopting only one proxy for each cloud, the adaptive proxy adoption scheme can achieve 26.5 percent reduction in the overall transfer time for the typical workloads.

On the basis of the above simulations, we further evaluate the system scalability of CoCloud based on real-world service deployment and the whole file update trace of 153 users as illustrated in Section 4.2. To better utilize the data trace, we have removed some dirty data like zero-size files, and also revised the sparse file update timestamps to make them more intensive and closer to the reality. In addition, the users are divided into collaboration groups at random,

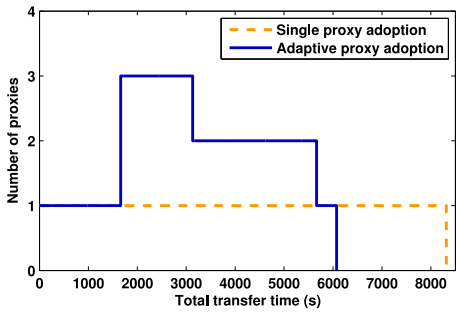

Fig. 17. Proxy number variation and transfer time reduction with adaptive proxy adoption.

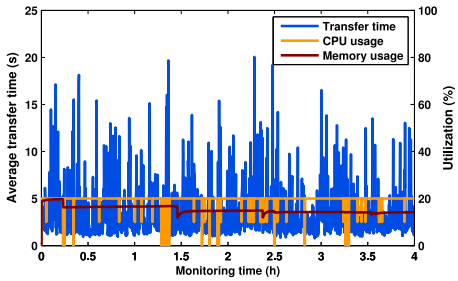

Fig. 18. Transfer time variation and resource utilization of control server in real-world service deployment.

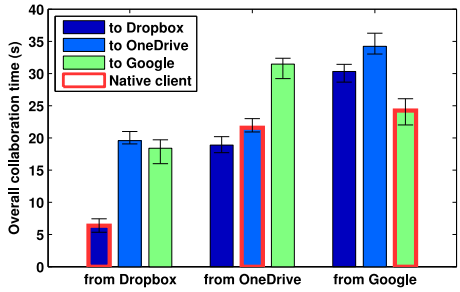

Fig. 20. End-to-end collaboration time of 50 documents (around 10 KB each) among three popular cloud services.

and we simulate their sync behaviors by a number of concurrent threads. Next we focus on both key factors for scalability, i.e., sync timeliness and system overhead.

By monitoring the control server for 4 hours during the collaboration process, we measure both variations of the average file transfer time (calculated every 10 seconds) and the server's CPU and memory usages. As shown in Fig. 18, the average file transfer time (just over 3 seconds) is well acceptable by adopting our scheduling algorithm with adaptive proxy adoption scheme. According to our experiment on the whole data trace, only when file updates arrive in bursts does the average file transfer time increase, and correspondingly the proxy number of a cloud increases up to 3 (in accordance with the above simulations). At the same time, both CPU and memory utilization rates are quite low (both below 20 percent). Given the control server is not in charge of any data transfer, deploying one server is enough to handle file updates from a very large scale of users.

4.6 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 end-to-end data sync time is highly influenced by the upload and download latencies 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 clouds, here we only show CoCloud performance among US users of Dropbox, OneDrive, and Google Drive.

Figs. 19 and 20 give their end-to-end collaboration time of two typical workloads (a 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 the "Native client" border). The figures indicate that an end-to-end CoCloud file collaboration for the above two typical workloads takes about 25 seconds on average, almost achieving the same level of efficiency as the intra-cloud collaboration (only $1.39\times$ and $1.82\times$ 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.

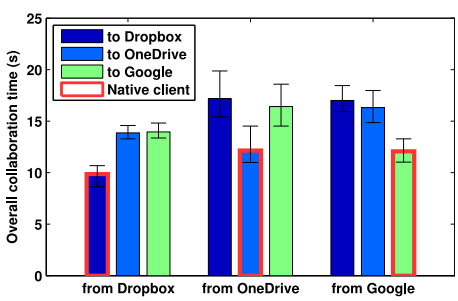

Fig. 19. End-to-end collaboration time of a 10-MB Zip file among three popular cloud services.

5 Discussion

In addition to the aforementioned system design and implementation, a few more issues also need to be considered, and we discuss them as follows:

Architecture of Cross-Cloud Collaboration. To provide efficient cross-cloud file collaboration, building up a unified interface upon the cloud RESTful APIs is obviously the ideal way. However, such advance is impeded by competitors in the market of personal cloud storage due to its low benefit-cost ratio. An intuitive alternative is to build up a personal cloud service, whose interface is compatible with all existing clouds. A number of issues such as sync protocol between the cloud and its client, client capabilities, should be well considered then, which increases implementation complexity as well as system costs. Correspondingly, our solution CoCloud mainly works as a transfer middleware among clouds. Except for a lightweight client that serves as a monitor, we utilize the sync capabilities of cloud native clients. With the design of per-cloud nearby-proxy deployment and advanced transfer protocol, CoCloud can be expected to efficiently serve among clouds with collaborative functionality and high user-perceived performance.

Security and Privacy Issues. Apart from the architecture, the data security and privacy of such a cloud service is also often concerned, as a survey shows that the protection becomes increasingly significant [20]. The OAuth 2.0 framework [8] is currently adopted for authorization by most personal cloud services, whenever users operate the stored data through a third-party service. However, a considerable number of users may wish to prevent the third parties or even cloud providers from accessing their confidential data. As OAuth 2.0 authorization is not enough in this scenario, encryption or erasure coding (also for redundant backup) can be conducted on these files to further improve security. Specifically, the confidential files can be encrypted before uploaded to the cloud (the file owner need keep the encryption key), or encoded at the client and uploaded to different clouds. Unfortunately, all the security protection approaches will sacrifice the overall collaboration performance, and they can hardly be adopted for cross-cloud collaboration. We will deeply consider the security and privacy issues in our future work.

System Cost-Effectiveness. At last, for providing such a cross-cloud collaboration service, the high traffic caused by the frequent collaboration requests is innegligible but inevitable. Although the advanced transfer protocol can reduce the network traffic while the dataflow scheduling algorithm can reduce the bandwidth overhead to some extent, they also bring about extra storage and computation overheads. Nevertheless, as CoCloud can break through the "walled-garden" of cloud services and provide an efficient cross-cloud collaboration service, we believe the benefit brought by the high requirement of a certain proportion of users (and maybe some new clouds) overwhelms the cost of those overheads.

6 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 four aspects.

Multiple Cloud Management. Some previous studies have proposed controlling multiple cloud services for redundant data backup. DepSky [21] builds a dependable cloud-of-clouds by distributing coded data into different public clouds, while MetaSync [22] and UniDrive [15] serve personal cloud users by adding more performance consideration and CYRUS [23] further considers privacy and reliability issues. However, these personal data backup managers require binding multiple clouds simultaneously as their backends 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) [12], [13], [24], delta encoding, and deduplication [10], [25]. 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 scheme, CoCloud proxies are located near enough to API servers to overcome their inefficacy. Access to these proxies is very efficient in virtue of the inter-proxy advanced transfer protocol, as if the cloud had provided the capabilities to third parties.

Optimization in Cloud CDN. A dozen of studies work on the performance of cloud CDN, to reduce tail latency [26], [27], [28], to provide replication cost-effective placement and consistency [29], [30], [31], or to balance workloads and guarantee inter-node bandwidth [32]. Unlike the previous work aiming at improving the performance of target clouds, CoCloud can optimize the overall transfer latency between two (multi-node) clouds. In addition, FUP defination involves virtual time, which is also adopted by the previous work [33], [34]. Aimed at the collaboration scenario, CoCloud leverages such a mechanism to improve the scheduling efficiency. Based on the priority assignment and file workload dispatch schemes, the inter-proxy dataflow scheduling algorithm well balances the collaboration timeliness and system overhead.

Cloud Measurement Studies. A variety of previous research papers measure and benchmark performance of multiple clouds, from public clouds [35] to personal cloud services [4], [9]. In addition, the architecture of mobile cloud storage services and their internal sync protocols are presented in [36], and QuickSync [14] further addresses the synchronization inefficiency problem of these mobile cloud services. Some other papers elaborately study the well-performed Dropbox, by either pinning the inside architecture [11] or improving the inefficiency of some client capabilities [37]. Likewise, the internal structure of UbuntuOne is deeply studied by measurements in [38]. Note that the above measurements are all conducted on the clouds' native clients. Besides, several papers have studied personal cloud web APIs, like [2], [39]. In contrast with them, we further make performance comparisons between web APIs and native client capabilities, and more importantly, observe that proxies can be deployed close to clouds to overcome API inefficacy.

7 Conclusion

In this paper, we address the cross-cloud file collaboration requirement, 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 close to each cloud to overcome the web API inefficiency. On this basis, we propose CoCloud for file collaboration among heterogeneous clouds. It includes a unified inter-proxy advanced transfer protocol and a cross-cloud data transfer optimization algorithm, as well as an online inter-proxy dataflow scheduling algorithm and collaboration control mechanisms. 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 as well as high scalability. Its performance even exceeds the intra-cloud collaboration performance in some cases.

Acknowledgments

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 AGR20160105.

References

[1] Baidu PCS (Personal Cloud Storage), [Online]. Available: http://developer.baidu.com/wiki/index.php?title=docs/pcs
[2] G. Wu, et al., "On the performance of cloud storage applications with global measurement," in *Proc. IEEE/ACM Int. Symp. Quality Serv.*, 2016, pp. 31–40.
[3] IFTTT, [Online]. Available: https://ifttt.com
[4] Z. Li, et al., "Towards network-level efficiency for cloud storage services," in *Proc. ACM Conf. Internet Meas.*, 2014, pp. 115–128.
[5] DynamoRIO, [Online]. Available: http://www.dynamorio.org
[6] Dropbox Webhooks Tutorial, [Online]. Available: https://www.dropbox.com/developers/reference/webhooks#tutorial
[7] DigitalOcean, [Online]. Available: https://www.digitalocean.com
[8] D. Hardt, "The OAuth 2.0 authorization framework," *IETF RFC: 6749*, Oct. 2012.
[9] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras, "Benchmarking personal cloud storage," in *Proc. ACM Conf. Internet Meas.*, 2013, pp. 205–212.
[10] A. Tridgell and P. Mackerras, "The Rsync Algorithm," *Joint Comput. Sci. Tech. Rep. Series*, Australian Natl. Univ., Tech. Rep. TR-CS-96-05, 1996.

[11] I. Drago, M. Mellia, M. M. Munafò, A. Sperotto, R. Sadre, and A. Pras, "Inside dropbox: Understanding personal cloud storage services," in *Proc. ACM Conf. Internet Meas.*, 2012, pp. 481–494.
[12] B. Aggarwal, et al., "EndRE: An end-system redundancy elimination service for enterprises," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2010, pp. 419–432.
[13] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in *Proc. ACM Symp. Operating Syst. Principles*, 2001, pp. 174–187.
[14] Y. Cui, Z. Lai, X. Wang, N. Dai, and C. Miao, "QuickSync: Improving synchronization efficiency for mobile cloud storage services," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 592–603.
[15] H. Tang, F. Liu, G. Shen, Y. Jin, and C. Guo, "UniDrive: Synergize multiple consumer cloud storage services," in *Proc. ACM/IFIP/USENIX Annu. Middleware Conf.*, 2015, pp. 137–148.
[16] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, May 1998.
[17] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen, "HTTP extensions for distributed authoring—WEBDAV," *IETF RFC: 2518*, Feb. 1999.
[18] Apache MINA, [Online]. Available: http://mina.apache.org
[19] FFmpeg, [Online]. Available: http://ffmpeg.org
[20] J. Tang, Y. Cui, Q. Li, K. Ren, J. Liu, and R. Buyya, "Ensuring security and privacy preservation for cloud data services," *ACM Comput. Surveys*, vol. 49, no. 1, pp. 13:1–39, Jun. 2016.
[21] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "DepSky: Dependable and secure storage in a cloud-of-clouds," in *Proc. ACM Eur. Conf. Comput. Syst.*, 2011, pp. 31–46.
[22] S. Han, H. Shen, T. Kim, A. Krishnamurthy, T. Anderson, and D. Wetherall, "MetaSync: File synchronization across multiple untrusted storage services," in *Proc. USENIX Annu. Techn. Conf.*, 2015, pp. 83–95.
[23] J. Y. Chung, C. Joe-Wong, S. Ha, J. W.-K. Hong, and M. Chiang, "CYRUS: Towards client-defined cloud storage," in *Proc. ACM Eur. Conf. Comput. Syst.*, 2015, pp. 1–16.
[24] N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," in *Proc. ACM Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2000, pp. 87–95.
[25] Y. Hua, X. Liu, and D. Feng, "Neptune: Efficient remote communication services for cloud backups," in *Proc. IEEE Annu. Joint Conf. IEEE Comput. Commun.*, 2014, pp. 844–852.
[26] Z. Wu, C. Yu, and H. V. Madhyastha, "CosTLO: Cost-effective redudancy for lower latency variance on cloud storage services," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2015, pp. 543–557.
[27] L. Suresh, M. Canini, S. Schmid, and A. Feldmann, "C3: Cutting tail latency in cloud data stores via adaptive replica selection," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2015, pp. 513–527.
[28] Z. Lai, Y. Cui, M. Li, Z. Li, N. Dai, and Y. Chen, "TailCutter: Wisely cutting tail latency in cloud CDN under cost constraints," in *Proc. IEEE Annu. Int. Conf. Comput. Commun.*, 2016, pp. 1845–1853.
[29] F. Chen, K. Guo, J. Lin, and T. L. Porta, "Intra-cloud lightning: Building CDNs in the cloud," in *Proc. IEEE INFOCOM*, 2012, pp. 433–441.
[30] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "SPANStore: Cost-effective geo-replicated storage spanning multiple cloud services," in *Proc. ACM Symp. Operating Syst. Principles*, 2013, pp. 292–308.
[31] G. Liu and H. Shen, "An economical and SLO-guaranteed cloud storage service across multiple cloud service providers," in *Proc. IEEE INFOCOM*, 2016, pp. 2689–2697.
[32] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea, "Chatty tenants and the cloud network sharing problem," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2013, pp. 171–184.
[33] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica, "Multi-resource fair queueing for packet processing," in *Proc. ACM Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2012, pp. 1–12.
[34] C. Zhang, Y. Cui, R. Zheng, E. Jinlong, and J. Wu, "Multi-resource partial-ordered task scheduling in cloud computing," in *Proc. IEEE/ACM Int. Symp. Quality Serv.*, 2016, pp. 1–6.
[35] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing public cloud providers," in *Proc. ACM Conf. Internet Meas.*, 2010, pp. 1–14.
[36] Y. Cui, Z. Lai, and N. Dai, "A first look at mobile cloud storage services: Architecture, experimentation, and challenges," *IEEE Netw.*, vol. 30, no. 4, pp. 16–21, Jul./Aug. 2016.
[37] Z. Li, et al., "Efficient batched synchronization in dropbox-like cloud storage services," in *Proc. ACM/IFIP/USENIX Int. Conf. Distrib. Syst. Platforms Open Distrib. Process.*, 2013, pp. 307–327.
[38] R. Gracia-Tinedo, et al., "Dissecting UbuntuOne: Autopsy of a global-scale personal cloud back-end," in *Proc. ACM Conf. Internet Meas.*, 2015, pp. 155–168.
[39] R. Gracia-Tinedo , M. Artigas, A. Moreno-Martínez, C. Cotes, and P. López, "Actively measuring personal cloud storage," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2013, pp. 301–308.

Jinlong E received the BE and MSc degrees in computer software from Nankai University, Tianjin, China, in 2007 and 2011, respectively. He is currently working towards the PhD degree in computer science and technology at Tsinghua University, supervised by Prof. Yong Cui. His current research interests include cloud storage, content distribution, scheduling, and mobile cloud computing.

Yong Cui received the BE and PhD degrees from Tsinghua University, China, in 1999 and 2004, respectively. He is currently a full professor with Tsinghua University, and a co-chair of IETF IPv6 Transition Softwire WG. He has published more than 100 academic papers in refereed journals and conferences, and won two Best Paper Awards. He also authored a number of RFCs on IPv6 transition technologies. His major research interests include mobile/wireless Internet and network architecture. He is a member of the IEEE.

Peng Wang received the BSc degree in mathematics from Tsinghua University, China, in 2012. He is currently working towards the MSc degree in computer science at Carnegie Mellon University. His current research interests include cloud computing/storage and big data analysis.

Zhenhua Li received the BSc and MSc degrees from Nanjing University, in 2005 and 2008, and the PhD degree from Peking University, in 2013, all in computer science and technology. He is currently an assistant professor at the School of Software, Tsinghua University. His research areas mainly consist of cloud computing/storage, big data analysis, content distribution, and mobile Internet. He is a member of the IEEE.

Chaokun Zhang is currently working toward the PhD degree in computer science and technology at Tsinghua University, Beijing, China, supervised by Dr. Jianping Wu and Dr. Yong Cui. He is also doing his research in McMaster University, Canada as a visiting PhD student from 2016 to 2017, supervised by Dr. Rong Zheng. He is a student member of the IEEE. His current research interests include scheduling, resource management, and optimization algorithm design and analysis in computer networks.