PandaSync: Network and Workload aware Hybrid Cloud Sync Optimization

Suzhen Wu*, Longquan Liu*, Hong Jiang‡, Hao Che‡, Bo Mao†✉
*Computer Science Department of Xiamen University, China
‡Department of Computer Science and Engineering, The University of Texas at Arlington, USA
†Software School of Xiamen University, China
✉Corresponding author: maobo@xmu.edu.cn

***Abstract*—With the widespread use and increasing popularity of cloud storage, more and more data are moved to the cloud, making cloud storage a platform for both data sharing among users, devices, and data backup for data reliability. Thus, it is critically important to ensure data consistency through efficient cloud synchronization (sync). The existing cloud synchronization schemes are either delta sync, which sends only the updated portion of a file but incurs high compute overhead of data deduplication for small files, or full sync, which avoids data deduplication by sending the full file but wastes network bandwidth and lengthens sync time by transferring significant amount of redundant data over the networks for large files. In this paper, we propose a hybrid cloud sync scheme, PandaSync, that combines full sync and delta sync dynamically based on file size and network conditions. To further improve small-file sync performance, we propose an optimization, Full2Sync, that merges the sync request with the file-sending request to reduce the number of network round-trips between the client and the cloud servers. The experiments conducted on our lightweight prototype implementation of PandaSync show that PandaSync reduces the sync time by an average of 85.1% and 74.6% from the delta sync scheme and full sync scheme, respectively.**

***Index Terms*—Cloud Storage; Full Synchronization; Delta Synchronization; Performance Evaluation**

I. Introduction

Due to cost-effectiveness and ease of management, more and more companies and users have moved or planned to move data out of their own local storage into the cloud storage systems. The use of the cloud storage has become so pervasive that many organizations are adopting a "cloud-first" approach [12]. The most recent annual cloud computing survey by the venture firm North Bridge found that 50% of the organizations had either a cloud-first or cloud-only policy and more than 90% used the cloud in some way [31]. IDC predicts that the collective sum of the world's data will grow to 175ZB and 49% data will be stored in public cloud environments by 2025 [18]. In terms of the usages of cloud storage, file sharing and collaboration continue to be the category with the greatest variety of cloud services in use (e.g. Slack [29], Cisco WebEx [7], etc.), accounting for 20.9% of cloud services in use. Rounding out the top 5 categories are finance (7.5%), IT services (7.1%), cloud infrastructure (7.1%), and development (6.5%) [8].

One of the biggest advantages provided by cloud storage is file sharing and collaboration, which enables the files to be synchronized conveniently across multiple devices/users anywhere and anytime via Internet. The data synchronization (sync) is a primary and critical technique for cloud storage services that allows the clients to automatically make the local files consistent with the files stored in the remote cloud data centers. From the end user's perspective, higher sync performance can provide better perceived service experience and avoid inconsistency among different clients/devices and the cloud data centers [1], [28]. From the cloud storage provider's perspective, shorter sync latency means higher system throughput, which directly improves data center performance and cost efficiency. Thus, the sync performance is considered the most important factor for cloud storage services and directly affects the system consistency and throughput [9], [10], [32].

The existing cloud sync schemes can be characterized as either full sync or delta sync, with the latter being much more widely studied [9], [13], [36], [37]. A full sync scheme synchronizes the whole file no matter how many changes are made to the file; whereas, in a delta sync scheme only the changes made to the file are synchronized to significantly reduce network transmission, which explains why it is much more preferred in recent studies. However, the delta sync schemes incur significant computing overhead in the critical I/O path, also a key problem existing studies try to address. Our preliminary performance evaluations reveal that the computing and network overhead can be too high for a delta sync scheme to be adopted for synchronizing small files even when the delta is small, i.e., with high data redundancy in such files. For example, with 3KB content updated in a 10KB file, the delta sync scheme with fixed-size chunking increases sync latency by more than 20% over the full sync scheme. Moreover, the network latency accounts more than 50% of the sync latency for small-file synchronization.

On the other hand, previous studies on the workload characteristics of the enterprise and cloud environments have shown that small files dominate, meaning that small files account for the vast majority of all files [2], [19], [24], [33], [38]. Moreover, these small files also account for more than 80% of all the user operations [2], [24]. Thus, the performance of small-files accesses is critical to the overall system performance and directly affects user experience [16], [17]. However, most recent cloud sync schemes, including the state-of-the-art

QuickSync [9], DeltaCFS [37], and WebDelta [36], are delta-based sync schemes. This, combined with our experimental evaluations (Section IV), suggests that a sync scheme for cloud storage that leverages the strengths of both the full sync scheme and delta sync scheme in a sensible way adapting to the network performance and characteristics of files to be synchronized, is perhaps the most effective one. Moreover, the synchronization of small files should be governed in part by the Amdahl's law [15] to strike an optimal balance between performance and overhead.

To address the sync efficiency issue in cloud storage, we propose a hybrid data sync approach, called PandaSync, that dynamically switches between full sync and delta sync, adapting to the changes in both the network performance and the workload characteristics. Different from the existing sync approaches, PandaSync utilizes full sync for small files and delta sync for large files, where the size threshold between small and large files is based on the network round-trip times. By exploiting the workload characteristics and network conditions, the advantages of both full sync and delta sync schemes are exploited and their disadvantages alleviated. Moreover, PandaSync merges the sync request and file-sending request into a single request to further reduce the number of network round-trip interactions between clients and cloud servers. The extensive benchmark- and trace-driven experiments conducted on our lightweight prototype implementation of PandaSync demonstrate that PandaSync significantly outperforms both the state-of-the-art full sync and delta sync schemes.

More specifically, this paper makes the following main contributions:

- Our experimental analysis shows that while delta sync schemes incur high processing overhead for small files, full sync schemes are highly bandwidth inefficient and thus induce long latency by their repeated data transferring of redundant large files.
- We propose a hybrid sync scheme, PandaSync, which combines the strengths of delta sync and full sync by dynamically and judiciously switching between them based on network performance and sync file characteristics. Moreover, PandaSync further reduces the network latency by merging the sync request and file-sending request for small-file synchronization.
- We conduct experiments on our lightweight prototype implementation to evaluate the PandaSync performance and compare it with the existing state-of-the-art delta sync scheme and full sync scheme with both benchmark-driven and trace-driven experiments.

The rest of this paper is organized as follows. Background and Motivation are presented in Section II. We describe the PandaSync architecture and design in Section III. The performance evaluation is presented in Section IV. We review the related work in Section V and conclude this paper in Section VI.

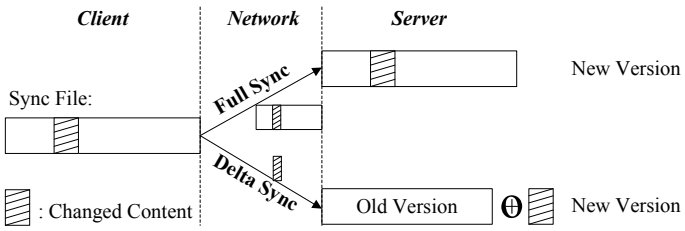


Fig. 1. The full and delta synchronization schemes.

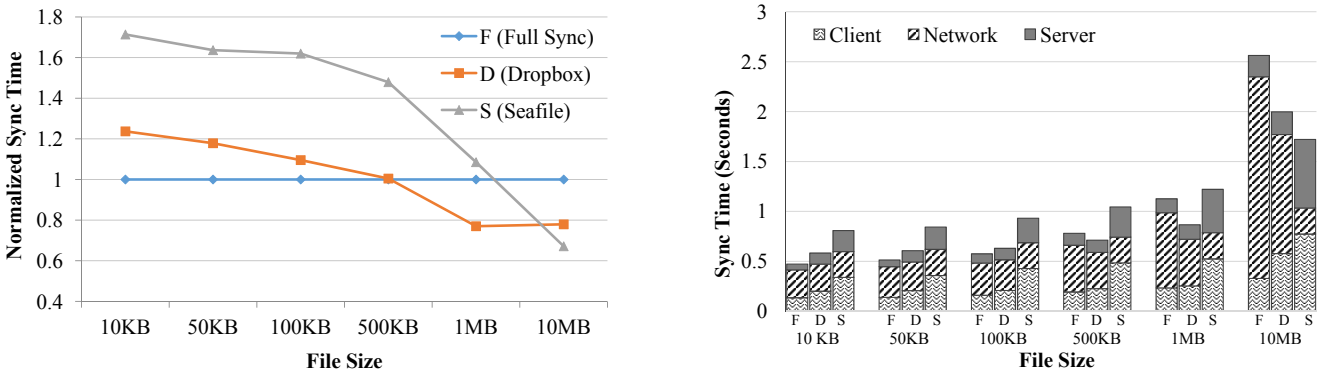
II. Background and Motivation

In this section, we first present key problems associated with cloud sync and some important observations drawn from our preliminary analysis and evaluation results. Then we present the workload characteristics to motivate the PandaSync study.

A. Cloud sync problems

Cloud synchronization is a process of keeping copies of files in multiple devices/users up to date and consistent throughout the cloud. For cloud sync, a user first sets up a cloud-based folder, to which the desired files are copied. This folder makes the files accessible via a cloud interface to multiple users, on whatever device they are using. When a user updates a file on the device, the modified file or changes made to the file are automatically synchronized with the cloud folder, along with the corresponding folders on other devices/users [5]. Figure 1 shows two sync schemes currently used by different cloud storage providers: (1) the full sync scheme which is used by Google Drive [14] and Microsoft OneDrive [25], and (2) the delta sync scheme which is used by Dropbox [11] and Seafile [27]. Moreover, two representative delta algorithms are used in delta sync, fixed-size chunking which is used by Dropbox and Content Defined Chunking (CDC) which is used by Seafile.

We examine the efficiency of the above cloud sync methods in terms of the sync time. Figure 2 shows the results as a function of file size assuming an updated content of 3KB located in the middle of the file and a network Round-Trip Time (RTT) of 30ms, where F, D and S denote full sync, Dropbox_rsync (Delta sync with fixed-size chunking scheme), and Seafile_cdc (Delta sync with CDC scheme) respectively. The results indicate that full sync performs better than delta sync when the file size is small. As the file size increases, delta sync with either fixed-size chunking or CDC becomes advantageous. To better understand the performance results, the sync time is broken down into that spent on the client side, network transmission and the server side, as shown in Figure 2(b). The network transmission dominates the total sync time for the small-file sync both for full sync and delta sync schemes, implying that the network transmission can be optimized by reducing the network interactive time between clients and cloud servers. Moreover, the computing overhead in delta sync overwhelms the network transmission time by a big margin as the file size increases, especially for the CDC-based delta sync scheme. This observation is not only consistent with those made in previous studies but also the key



(a) Sync Time

(b) Sync Time Profile

Fig. 2. The sync time as a function of file size with 3KB content of each being updated and 30ms network RTT.

problem they try to address [9], [36], [37]. In the evaluations, we also conduct experiments with 30% content of a file being updated and the results show similar trends.

In summary, two important observations are made from the performance results. First, the full sync scheme is much more efficient than the delta sync scheme for small-file synchronization. Moreover, delta sync is not effective for small files because they contribute relatively very little to redundancy detection and reduction, which is also the reason why that a lot of deduplication schemes simply skip small files. Second, the network transmission dominates the overall sync time for small-file synchronization, because in this case the round-trip time between the client and server substantially overshadows the data transfer time over the network. Unfortunately, none of these observed issues has been addressed by the previous studies.

B. The workload characteristics

To better support the upper layer applications, the characteristics of workloads generated by these applications must be considered in the storage system design. For example, the I/O request size distribution is an important factor in workload characterization. Knowing the I/O request size can directly help with appropriate configuration of certain parameters, such as the choice of data sync methods. Previous studies on the workload characteristics have clearly shown that about 50% of files are smaller than 4KB and over 80% of files are smaller than 128KB in enterprise and cloud environments [38]. A recent study of cloud traces show that a vast majority (77%) of files are small in size (less than 100KB) [19]. These small files also receive a vast majority (over 80%) of file references [2], [24], [34]. Moreover, the distribution of file size has changed very little over the years [24], [38], which indicates that optimizing storage systems for small files are extremely important.

However, the existing studies are all trying to address the problem of computing overhead associated with the delta sync scheme [9], [36], [37] even for the small-file sync. Our preliminary evaluation results reveal that delta sync is not effective for small files due to the relatively small data-volume saving but very high computational overhead, in addition to the network transmission and round-trip delays. In contrast, the full sync scheme outperforms the delta sync scheme when the file sizes are small. These important observations, combined with the urgent need to address the sync efficiency problem of cloud storage systems, motivate us to propose PandaSync, which dynamically switches between the two sync schemes adapting to changes in file size and network performance. PandaSync retains the desirable advantages of both the full sync and delta sync methods to improve the performance of cloud storage systems.

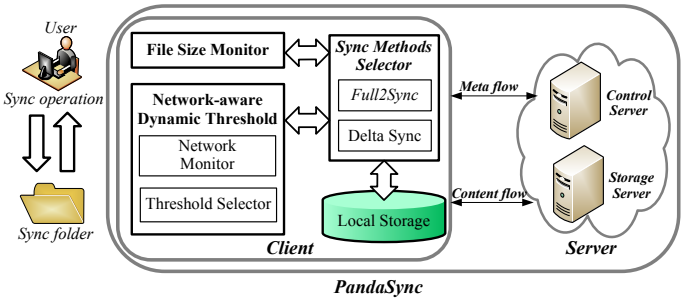


Fig. 3. System architecture of PandaSync.

III. The Design of PandaSync

In this section, we first present the system overview of PandaSync. Then we present its hybrid data synchronization scheme and the network optimization method for small-file synchronization, followed by the data consistency consideration.

A. PandaSync system overview

Figure 3 shows a system architecture of PandaSync. It is a set of enhanced and extended modules to the existing sync modules and mainly located on the client side. It is nontrivial to combine both full sync and delta sync into one system, because how to choose the specific data sync method is not predetermined and highly dependent on the workload characteristics and network performance, such as file size and current network RTT. PandaSync can be turned off and switched to either Full Sync scheme or Delta Sync scheme to provide much better flexibility.

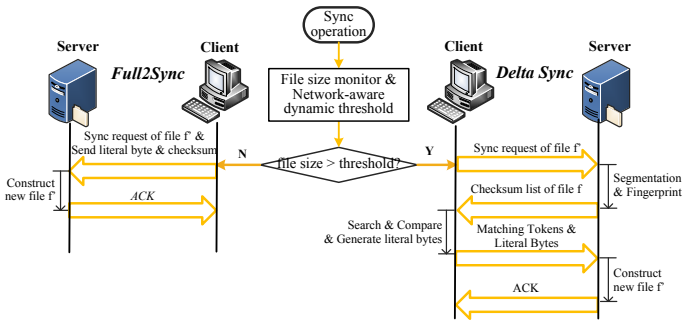


Fig. 4. The sync method selection workflow in PandaSync.

PandaSync has three main functional modules: File Size Monitor, Network-aware Dynamic Threshold and Sync Method Selector. The Network-aware Dynamic Threshold module is responsible for determining the size threshold between large files and small files based on the network performance which is detailed in the next subsection. Upon the initiation of a sync operation, the File Size Monitor first fetches the information regarding the size of the file and calculates the checksum values for later consistency checking. This information is sent and gathered in the Sync Method Selector module which chooses an appropriate sync method between full sync and delta sync. In the Figure 3, Full2Sync is an optimization to the full sync method for small files.

B. Hybrid data synchronization

As experimentally revealed in subsection II-A, no single, fixed sync method performs the best all the time. The most efficient sync method for a given file is highly dependent on the file size and the network performance. The file size information of a file is readily available when the file is being prepared for sync. However, the network performance information is dynamic and changes with time. With different network conditions, the best sync method for a given file may be different as well.

Figure 4 shows the sync method selection workflow in PandaSync. When a file is ready for sync, the file size information is obtained and compared with the size threshold determined by the Network-aware Dynamic Threshold module. If the file size is larger than the size threshold, the delta sync scheme is selected for the file. Otherwise, an enhanced full sync scheme for more efficient small-file full sync, Full2Sync, is used for the file. Thus, the size threshold is a key to sync method selection in PandaSync which in turn is highly dependent on the network performance.

Network-aware Dynamic Threshold module is responsible for tracking and monitoring the network performance to determine the appropriate size threshold. The cloud sync communication relies on TCP and its performance is affected by network factors such as round-trip time (RTT) [9]. The network RTT is the duration from the time when a packet is transmitted to the network until time when the acknowledgement of the packet is received by the sender, *e.g.,* the sum of the path latencies in the forward and reverse directions for the packet. The network RTT can be influenced by many

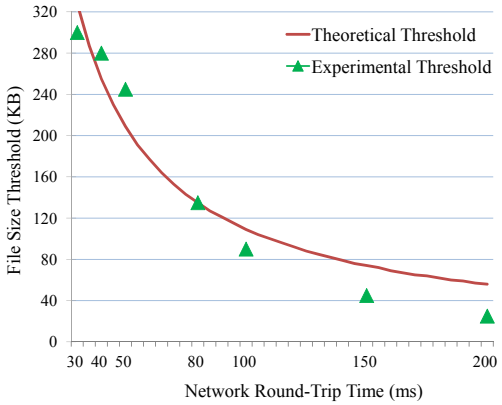


Fig. 5. A comparison between theoretical and experimental results on the file size threshold based on the Dropbox_rsync scheme and Full2Sync scheme.

factors, such as network queueing and path length, which may change over time. Thus, the Network-aware Dynamic Threshold module needs to adaptively track the network RTT values at the client side through the "ping" command. Once the network RTT value is obtained, the corresponding size threshold is determined and applied to the subsequent sync files.

The size threshold is essentially a file size value at which the sync latencies of the full sync and the delta sync scheme are expected to be the same. Here we assume that 30% new content is inserted in the middle of the file. Let R denote the network RTT and F the sync file size. Some parameters, such as processing and querying latencies, are derived from real experiments. The sync time is calculated within client, network and server parts. Based on these values and definitions, the file sync times for Dropbox_rsync and Full2Sync can be expressed by Formula (1) and Formula (2).

$$SyncTime_{Dropbox\_rsync} = 594 + F * (0.068 + \frac{0.392 * R}{64}) \quad (1)$$

$$SyncTime_{Full2Sync} = 262 + F * (0.2 + \frac{2.25 * R}{64}) \quad (2)$$

By making the two sync times equal with the two variables (network RTT and sync file size), we have the approximate relationship between network RTT and sync file size threshold, shown in Formula (3).

$$F = \frac{332}{0.141 + 0.029 * R} \quad (3)$$

To validate the correctness of Formula (3), we conduct real experiments to determine the size threshold. Figure 5 shows a comparison between theoretic and experimental results on the sync file size threshold, indicating a reasonably close match between the two. Based on the validation, we can use Formula (3) to calculate the size threshold once the network RTT value is measured.

C. Small-file sync optimization

Accesses to small files dominate user file accesses in the cloud, thus optimizing small-file sync is critically important.

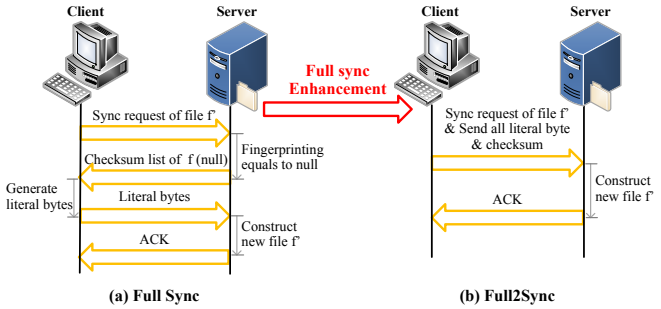


Fig. 6. A comparison of the sync workflow between the full sync method and the new full2sync method.

While full sync can provide better sync performance than delta sync, the sync performance is still dominated by the network delays for small files. The reason is that the sync process separates the sync request from sync data by default, which significantly increases the number of round-trips. To address this problem and reduce the number of round-trips for small-file sync, we propose an enhanced full sync, called Full2Sync, by properly piggybacking in the sync request the file data to maximize the utilization of network bandwidth by reducing the number of client and server interactions.

Figure 6 shows a comparison of the sync workflow between the original full sync method and the proposed Full2Sync method in PandaSync. In the original full sync method, when a user edits a file from f to f', the client instantly sends a request to the server for file sync. On receiving the request, the server directly removes the old file f and return a null checksum list of f to the client, telling the user that the modified file f' in the server is a new file and there is no similar files on the server side. After that, based on the null checksum list of f, the client will directly send the file data in its entirety to the server to create the new file f'. Finally, the server returns an acknowledgment to the client to conclude the process.

To facilitate Full2Sync, both the *sender* and *receiver* processes are enhanced to encapsulate the sync data within the sync request to reduce the number of round-trips during small-file sync operations. When a sync request is initiated in the client, not only the metadata, such as the sync file list and the files' checksums, but also the data of the small files to be synced, are enclosed in a single package. Upon receiving the package in the cloud server, the *receiver* process checks the files' checksums based on the sync file list and their checksums to determine which file should be updated. If it needs a replacement, a temporary file is created and reconstructed based on the sync data. The temporary file's checksum will be compared with the checksum in the sync request to make sure that the reconstructed file is correct. Otherwise the sync request will be resent. After the successful replacement, the new file's metadata is updated and the ACK is sent to the sender to indicate that the sync process is completed, as illustrated in Figure 6. Compared with the traditional full sync workflow, not only is the number of round-trips reduced, but also the network bandwidth efficiency is improved.

D. Data consistency

Data consistency in our PandaSync design refers to the following two requirements: (1) The sync data must be atomically stored on the cloud server, (2) The sync data shared by multiple devices/users must be consistent.

First, the synced data in the cloud server must be consistent with the data generated from the client. In the traditional sync schemes, data consistency is guaranteed by the checksums associated with the sync requests. In PandaSync, the sync data is encapsulated in the sync request, while the checksums are also packaged and double checked once the file is reconstructed in the cloud server. Thus, PandaSync provides the same data consistency guarantees between the synced data in the cloud server and the data generated from the client as the traditional sync schemes.

Second, PandaSync provides better data consistency guarantees for synced data that is shared by multiple devices/users. One of the biggest advantages of cloud storage is data sharing and accessibility anytime and anywhere via the Internet. However, in order to make the data consistently and timely shared by multiple devices/users anywhere, the sync latency must be as short as possible. Shorter sync latency is also the main objective of all the existing cloud sync optimizations [9], [36], [37]. Compared with the existing optimizations, PandaSync dynamically and judiciously switches between the full sync scheme and the delta sync scheme based on the file's characteristics and network performance. Moreover, with its Full2Sync enhancement, PandaSync further optimizes the small-file sync workflow, thus reducing the sync latency significantly.

IV. Performance Evaluations

In this section, we first describe the prototype implementation of PandaSync, followed by the experimental setup and methodology. Then we evaluate the performance of PandaSync through both benchmark-driven and trace-driven experiments.

A. Prototype implementation

The PandaSync prototype is built on top of the Rsync version 3.1.3 [26]. It adds and revises 2945 LOC on the client side and 1152 LOC on the server side. Both SSH and RSH can be used for the remote communications and SSH is used in our evaluations by default in Rsync. Once the connection between the client and the server is established, the SmokePing [30] module, embedded in the Network-aware Dynamic Threshold module, starts to monitor the network latency. To accelerate the data processing and transferring, a pool of concurrent threads is created to parallelize the sync process. In addition, Full2Sync in PandaSync simplifies the sync logic between the client and server by changing the original *sender*, *generator*, and *receiver* into *sender* and *receiver*. The *receiver* process on the server side is responsible for checking the file list and file content, which reduces the number of round-trip interactions between the client and the server. The source code of the PandaSync prototype is accessible on https://github.com/LongquanLiu/PandaSync.

TABLE I
THE WORKLOAD CHARACTERISTICS OF THE SIX TRACES.

Trace Name	*Volume*	*Min Size*	*Max Size*	*Redundancy*
sugarsvnc-chaos	1.2GB	2Byte	262.5MB	29.7%
dropbox-xiaohe	2.2GB	3Byte	163MB	19.1%
box-wcx	9.0GB	3Byte	509.1MB	33.7%
dropbox-y	6.7GB	2Byte	232.1MB	12.1%
ubuntuone-green072	3.2GB	1Byte	356MB	55.2%
dropbox-jeff	2.7GB	1Byte	105.1MB	19.2%

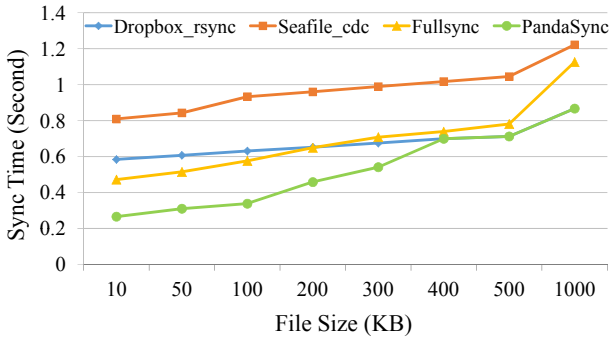
B. Experimental setup and methodology

Our tests are conducted on a desktop PC (client) with an Intel i5-3470 3.2 GHz quad-core processor and 4GB memory. The cloud server is Aliyun ECS [3] with Intel Xeon E5-2682V4 2.5 GHz processor and 2GB memory & 40GB SSD storage. The operating systems in both client and server sides are Ubuntu 16.04.3 (Linux kernel version 4.4). They are connected through the China Education and Research Network [6] between cities of Xiamen (client side) and Shanghai (cloud server side) in China. The network RTT is about 30ms normally and configurable to be variable. We compare PandaSync with both the full sync scheme (Fullsync) and delta sync schemes, including Dropbox_rsync with fixed-size chunking scheme and Seafile_cdc with CDC-based scheme. For Dropbox_rsync, we use rsync 3.1.3 to implement the delta sync method with 4KB fixed-size chunking. For Seafile_cdc, Seafile 6.0.1 is used with variable chunk size ranging from 2KB to 8KB, with an average of 4KB.

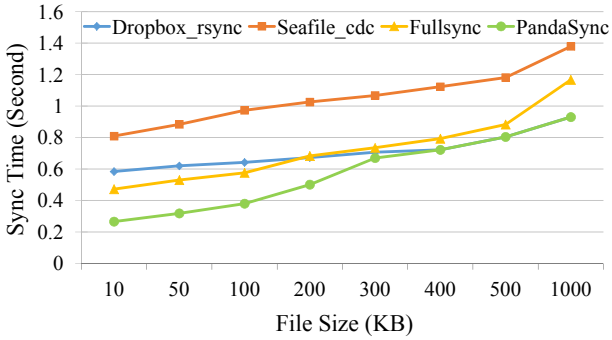
Our evaluation is based on both benchmark-driven and trace-driven experiments. For the benchmark-driven experiments, we configure the file update with one of the two redundancy levels, i.e., 3KB content or 30% content is inserted in the middle of a file. The file size ranges from 10KB to 100MB. The trace-driven experiments are used on six traces collected from real-world user activities of cloud storage services in several universities and companies in the US and China from July 2013 to March 2014 [19]. The file content in the traces are initiated according to the file name and sequences. Table I shows their key trace features of volume size, minimum and maximum file size, and redundancy. For cost-effectiveness assessment, the network traffic fee of Aliyun ECS is charged at a rate of USD$0.11/GB [3].

C. Performance results and analysis

Benchmark-driven experiments: Figure 7 shows the benchmark-driven sync times of different schemes as a function of file size. First, PandaSync consistently performs the best among all the schemes for all file sizes, except for files larger than 400KB when it performs the same as Dropbox_rsync. The file size threshold under the specific redundancy and network RTT values is between 300KB and 400KB. For files smaller than 400KB, PandaSync uses the Full2Sync scheme which reduces the number of the round-trip interactions between the client and the server, thus significantly reducing the network latency, as illustrated in Figure 8 which indicates a 50% network latency reduction from that of the fullsync scheme achieved by PandaSync.



(a) 3KB content insert

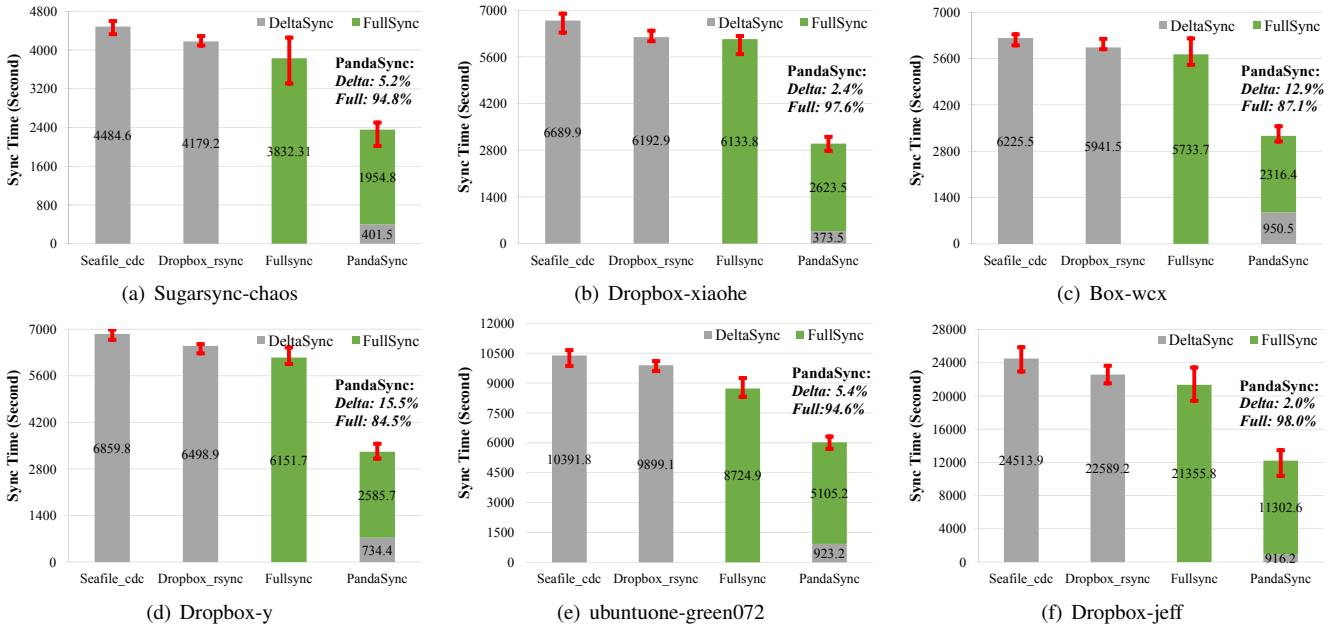


(b) 30% content insert

Fig. 7. Benchmark-driven results on sync times as a function of file size.

Second, full sync schemes (fullsync and full2sync) perform better than delta sync schemes (Dropbox_rsync and Seafile_cdc) when the sync files are smaller than 200KB. The reason is twofold: (1) The processing overhead associated with the delta sync schemes is high both on the client side and the server side; (2) The network latency for small-file sync is affected by the number of RTTs, not the data volume transferred over the network. Thus, it is not recommended to use delta sync schemes for small-file sync. It is this reason that motivated the PandaSync optimization for a hybrid sync scheme for cloud storage systems.

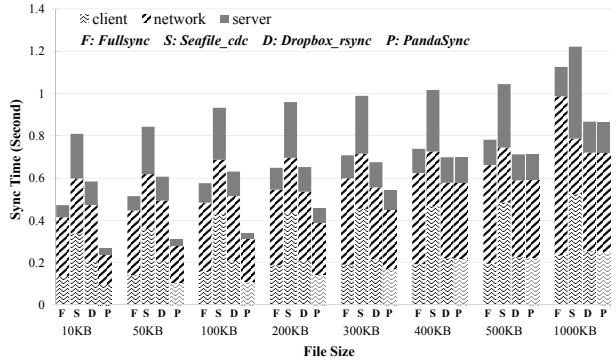
To investigate the time spent in each step in the file sync workflow, Figure 8 shows a breakdown of the sync latency into the client, network and server portions for different cloud sync schemes. We draw three important observations: (1) The cloud sync latency is dominated by the network latency in the fullsync scheme. The network portion of the latency remains almost unchanged when the sync files are smaller than 400KB. The reason is that the network latency is spent on context switching and the number of round-trip communications. The same phenomena is observed on the fixed-size based delta sync method (Dropbox_rsync) for small files. (2) The CDC-based delta sync scheme incurs significant latency overhead on both client and server sides, which dominate the sync time of the CDC-based delta sync. The reason is obvious because the redundancy detection overhead incurred by the CDC step is extremely high by itself, let alone the fingerprint transmission overhead between the client side and the server side. (3) PandaSync performs the best among all the cloud sync schemes. It improves the small-file sync performance



(a) Sugarsync-chaos (b) Dropbox-xiaohe (c) Box-wcx

(d) Dropbox-y (e) ubuntuone-green072 (f) Dropbox-jeff

Fig. 9. The cloud sync times driven by the six traces for the different cloud sync schemes.



(a) 3KB content insert

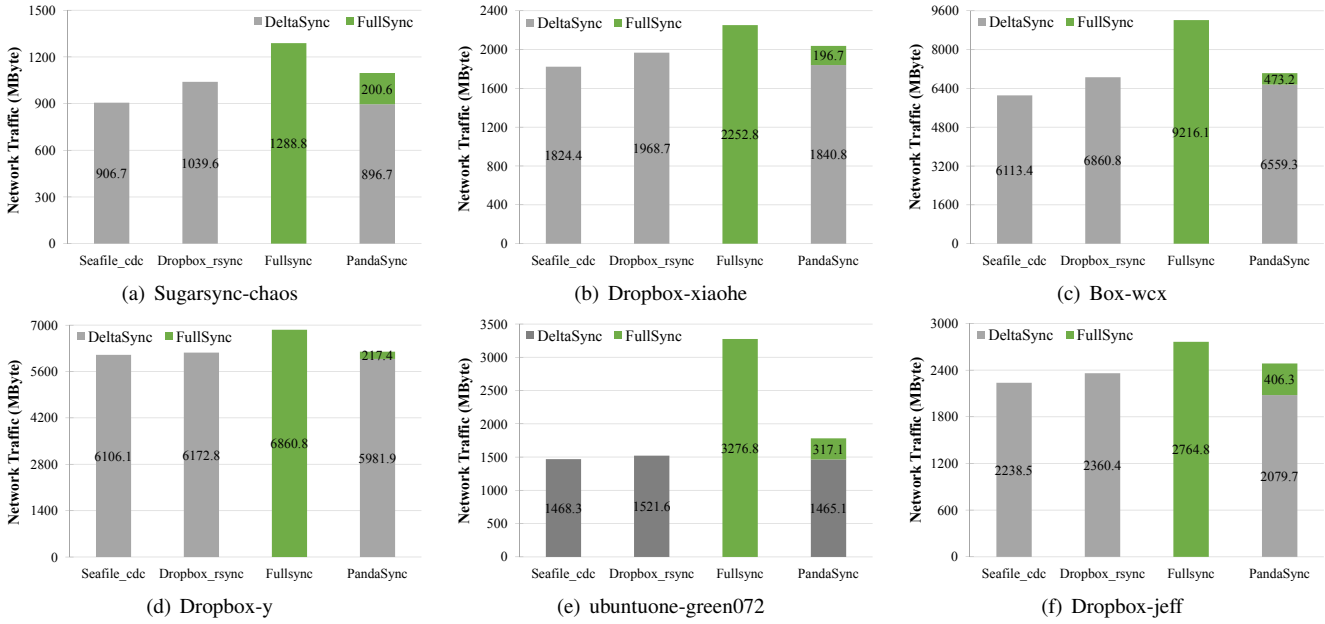
(b) 30% content insert

Fig. 8. Breakdown of the sync latency into client, network and server for different schemes.

by reducing the network latency. Moreover, for large files it chooses the fixed-size chunking based delta sync scheme and performs the same as the Dropbox_rsync scheme. To assess PandaSync's performance stability and consistency, we compare it with the Dropbox_rsync scheme by increasing the sync file size to 100MB, observing that the former continues to perform the same as the latter. During the benchmark-driven experiments, the file size is fixed thus the sync method in PandaSync is also fixed. To evaluate the efficiency of PandaSync in real environment, we also conduct trace-driven experiments.

Trace-driven experiments: Figure 9 shows the cloud sync times driven by the six traces for the different cloud sync schemes. Each experiment runs at least 3 times and the average cloud sync time, along with max and min, is shown in the Figure 9. PandaSync reduces the cloud sync time by an average of 97.3%, 85.1%, and 74.6% from the Seafile_cdc, Dropbox_rsync, and Fullsync schemes respectively. In PandaSync, a vast majority of files, above 84%, are synchronized by the Full2Sync method for all the traces. For four out of six traces this majority reaches a minimum of 94%, which indicates that an overwhelming majority of files in these traces are small files. Although Figure 10 shows that both the Seafile_cdc and Dropbox_rsync schemes transfer less data over the network than the FullSync and PandaSync schemes, this advantage of the former is substantially overshadowed by their much higher compute overhead in data deduplication, resulting in longer total sync times than the latter. In contrast, the reduction on the number of the network round-trips plays a much more important role in accelerating the cloud sync performance. With the Full2Sync optimization in the PandaSync scheme, a large number of network round-trips has been eliminated, as shown in Figure 11. In fact, the Fullsync scheme performs even better than both the Seafile_cdc and Dropbox_rsync schemes, because when small files dominate, Fullsync performs better than delta sync schemes.

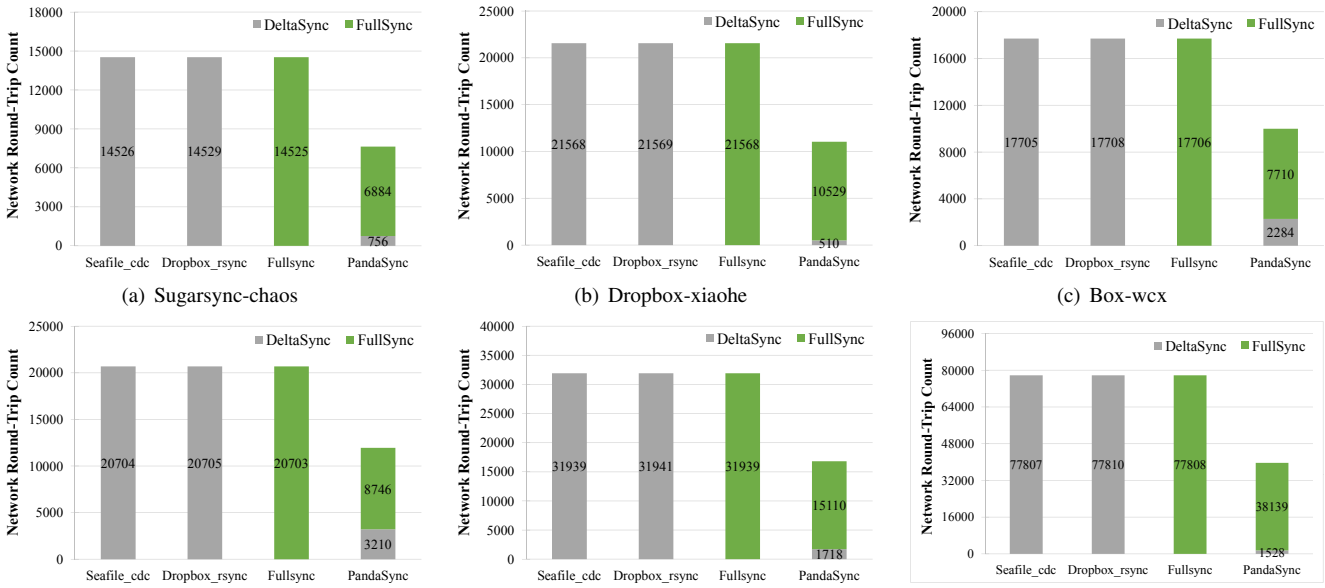
Figure 10 shows the total data volume transferred over the network driven by the six traces for the different cloud sync schemes. The Seafile_cdc and Dropbox_rsync schemes indeed



(a) Sugarsync-chaos (b) Dropbox-xiaohe (c) Box-wcx

(d) Dropbox-y (e) ubuntuone-green072 (f) Dropbox-jeff

Fig. 10. The total data volume transferred over the network driven by the six traces for the different cloud sync schemes.



(a) Sugarsync-chaos (b) Dropbox-xiaohe (c) Box-wcx

(d) Dropbox-y (e) ubuntuone-green072 (f) Dropbox-jeff

Fig. 11. The number of network round-trips, driven by the six traces for the different cloud sync schemes.

transferred over the network by up to 55.5% less data than the Fullsync scheme with an average of 27.9% and 22.6%, respectively. These reductions in data transferred stem mainly from the large files. Compared with the cloud sync time results shown in Figure 9, applying delta sync method to small files does not reduce the amount of data transferred over the network not only because small files account for a tiny fraction of the total data transferred, but also because delta sync incurs extra processing overhead and increases the number of the network round-trips, thus increasing the cloud sync latency.

Figure 11 shows the number of network round-trips driven by the six traces for the different cloud sync schemes. First, PandaSync requires a much smaller number of network round-trips than all the other schemes by merging the sync data with the sync request and thus halving the number of network round-trips for small-file synchronization. The fact that small files dominate all the six traces explains why the number of network round-trips is reduced significantly by the PandaSync scheme. Second, both the delta sync schemes and the full sync scheme have similar numbers of network round-trips because they share the same workflow in terms of the number of network round-trips as indicated in Figure 4 and Figure 6. This also implies that optimizing the workflow to reduce the number of network round-trips in the cloud sync process is an

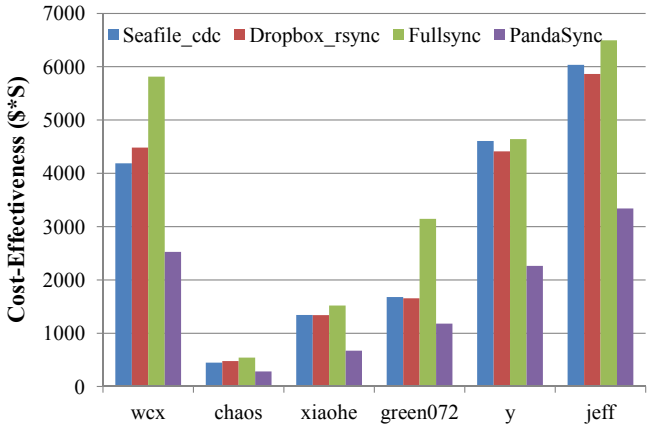


Fig. 12. Comparison of the cost-effectiveness of the different cloud sync schemes based on the latency and cost results obtained from the trace-driven experiments.

effective way to improve the cloud sync performance.

Assessment of cost-effectiveness: To reasonably estimate and quantify the cost-effectiveness of PandaSync relative to the state-of-the-art cloud sync schemes, we use the cost-latency product as a measure for cost-effectiveness [21], [23]. The lower the cost-latency product value of a scheme, the more cost-effective the scheme is.

Fig. 12 shows the cost-effectiveness, in terms of the cost-latency product, of the different cloud sync schemes based on the latency and cost results obtained from the trace-driven experiments. Among all the schemes, PandaSync is the most cost effective. It outperforms the Seafile_cdc, Dropbox_rsync, and Fullsync schemes by an average of 41.9%, 76.0%, and 118.8%, respectively. The reasons behind PandaSync's superiority in cost-effectiveness are twofold. First, by applying the Full2Sync optimization to small files to reduce the number of network round-trips, the overall cloud sync latency is reduced. Second, by applying the delta sync method to large files to reduce total data volume transferred over the network, the network traffic cost is reduced.

Sensitivity analysis: Network RTT is an important factor in the PandaSync scheme that affects the file size threshold. To examine its impact on PandaSync's efficiency, we conduct experiments on the Dropbox_rsync and PandaSync schemes driven by the dropbox_xiaohe and dropbox_y traces under different network round-trip times. Figure 13 shows the results and indicates that PandaSync consistently improves cloud sync efficiency with the increasing of the network round-trip time. However, the improvement is slightly reduced from 51.6% to 40.3% driven under the dropbox_xiaohe workload, and from 48.9% to 36.8% under the dropbox_y workload. The reason is that with a larger network round-trip time, the file size threshold in PandaSync is reduced accordantly, thus fewer files are optimized in PandaSync with the Full2Sync method. For files larger than the file size threshold, their sync times are the same as the Dropbox_rsync scheme. Thus, fewer files are synchronized by the Full2Sync method, and less improvement is achieved by the PandaSync scheme over the Dropbox_rsync scheme.

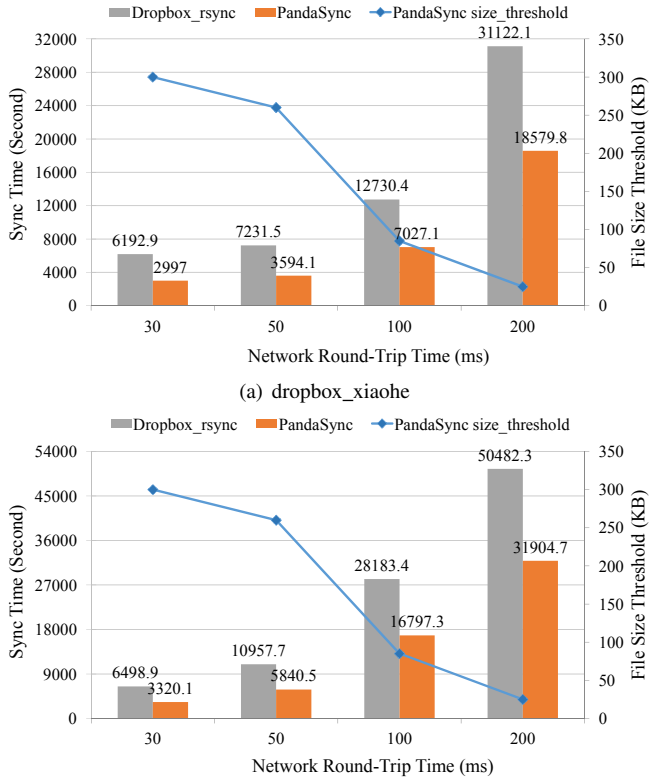


Fig. 13. Sensitivity analysis on the network round-trip time.

V. Related Work

Cloud storage has become a popular and cost-effective storage platform. More and more organizations and individual users are moving their data to the cloud, which makes the cloud sync efficiency increasingly more important and challenging for cloud storage users and providers.

There are two main cloud sync approaches, full sync and delta sync. The full sync approach, which sends the file in its entirety to the cloud during file synchronization, is simple and incurs little, if any, extra processing overhead. As a result, it is widely used in cloud applications on mobile devices where the processing and memory resources are limited [36]. Some main cloud storage products, such as Google Drive [14] and Microsoft OneDrive [25], also use the full sync approach for PC-based clients. In contrast, the delta sync approach only transfers from the client to the server the differences (deltas) between the client copy and the server copy of the sync file, rather than the complete content of the sync file. It is particularly useful for file modifications or incremental updates on files with sufficient content redundancy. However, detecting and eliminating the redundant content between copies of a sync file, a technology known as data deduplication, usually incurs significant compute overhead.

Dropbox [11] is the first cloud storage provider to adopt fixed-size chunking-based delta sync (rsync) in around 2009 in its PC client-based file sync process [20]. Then, iCloud Drive [4] and Seafile [27] followed the design choice of Dropbox by utilizing delta sync, using either fixed-size-based

chunking (FC) or content-defined chunking (CDC), to reduce cloud sync traffic. QuickSync [9] and DeltaCFS [37] implemented FC- and CDC-based delta sync respectively for mobile APPs. QuickSync [9] uses network-aware chunk size selection to adaptively select the proper deduplication strategy based on real-time network conditions. DeltaCFS combines delta sync with NFS-like file RPC by leveraging the hints of typical file update patterns, thus significantly cutting compute overhead on both the client side and the server side while preserving the network-level efficiency. Xiao et al. propose a web-based delta sync solution (WebR2sync+) by moving expensive chunk search and comparison operations from the client side to the server side [35], [36]. It further leverages locality-aware chunk matching and lightweight checksum algorithms to reduce the overhead, thus providing a practical solution of web-based delta sync for cloud storage services. In summary, all these three schemes are trying to alleviate the processing overhead associated with the delta sync approach. However, it is not efficient to apply delta sync to small files even if they have high redundancy.

Previous studies have found that delta sync suffers from both traffic and compute overhead problems in the presence of frequent, short data updates [9], [20]. To address these issues, they have designed efficient batched synchronization algorithms, such as batched sync [9] or UDS (Update-batched Delayed Sync) [20] to reduce the bandwidth usage and CPU usage by alleviating compute overhead. The delayed sync method is mainly designed for small files that dominate in the cloud storage systems [2], [19], [24]. However, delayed sync methods will induce longer sync time, leading to data consistency problems in cloud storage systems. Different from these approaches, our proposed PandaSync scheme takes the network conditions and the workload characteristics, specially the network RTT and the sync file size, into the design of the cloud sync strategy so that the advantages of both the full sync and the delta sync approaches are exploited while their disadvantages alleviated or hidden. As a result, the sync performance is improved with a better data availability guarantee.

VI. Conclusion

Sync performance is one of the main factors users consider when deciding whether or not to move their data to the cloud. This paper proposes a hybrid data sync approach, called PandaSync, that exploits the workload characteristics and the network conditions to improve the cloud sync performance. PandaSync utilizes full sync for small files and delta sync for large files, where the size threshold separating small files from large files is based on the network round-trip time at runtime. PandaSync further boosts sync performance for small files by merging the sync request and file-sending request into a single request to reduce the number of network round-trip interactions between clients and cloud servers. By exploiting the workload characteristics and the network conditions, the advantages of both the full sync and delta sync approaches are exploited while their disadvantages are alleviated. The experiments conducted on our lightweight prototype implementation of PandaSync show that PandaSync significantly outperforms existing cloud sync schemes.

PandaSync is an ongoing research project and we are currently exploring several directions for future research. First, the data redundancy characteristics is an important factor determining whether to apply delta sync for large files. However, the data redundancy characteristics is hard to measure or predict in advance for a specific file. We will further investigate how to reduce the sync latency of large files by exploiting the workload characteristics [22]. Second, we will extend the PandaSync design to consider the diversity features of cloud storage services, thus further improving the flexibility of PandaSync and the efficiency of cloud storage services.

VII. Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant No. U1705261, No. 61872305, No. 61772439, and No. 61472336, the US NSF under Grant No. CCF-1704504 and CCF-1629625.

References

[1] M. Abebe, K. Daudjee, B. Glasbergen, and Y. Tian. EC-Store: Bridging the Gap Between Storage and Latency in Distributed Erasure Coded Systems. In *Proceedings of the 38th IEEE International Conference on Distributed Computing Systems (ICDCS'18)*, Vienna, Austria, Jul. 2018.

[2] N. Agrawal, William J. Bolosky, John R. Douceur, and Jacob R. Lorch. A Five-Year Study of File-System Metadata. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST'07)*, pages 31–45, Feb. 2007.

[3] Aliyun Open Storage Service. http://www.aliyun.com/. 2014.

[4] Apple iCloud. https://www.icloud.com/. 2018.

[5] F. Chen, M. Mesnier, and S. Hahn. Client-aware Cloud Storage. In *Proceedings of the 30th International Conference on Massive Storage Systems and Technology (MSST'14)*, Santa Clara, CA, Jun. 2014.

[6] China Education and Research Network. http://www.edu.cn/english_1369/index.shtml. 2014.

[7] Cisco Webex: Online Meetings. https://www.webex.com/Webex/Meetings. 2018.

[8] Cloud Adoption and Risk Report. https://info.skyhighnetworks.com/WPCloudAdoptionRiskReport2019_BannerCloud-MFE.html?Source=Website&LSource=Website. 2018.

[9] Y. Cui, Z. Lai, X. Wang, and N. Dai. QuickSync: Improving Synchronization Efficiency for Mobile Cloud Storage Services. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom'15)*, Paris, France, Sep. 2015.

[10] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras. Benchmarking Personal Cloud Storage. In *Proceedings of the 13th ACM Internet Measurement Conference (IMC'13)*, Barcelona, Spain, Oct. 2013.

[11] Dropbox. https://www.dropbox.com/. 2018.

[12] From Cloud First to Cloud Smart. https://cloud.cio.gov/strategy/. 2018.

[13] Y. Go, N. Agrawal, A. Aranya, and C. Ungureanu. Reliable, Consistent, and Efficient Data Sync for Mobile Apps. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15)*, Santa Clara, CA, Feb. 2015.

[14] Google Drive. https://www.google.com/drive/. 2018.

[15] John L. Hennessy and David A. Patterson. Computer Architecture: A Quantitative Approach, Sixth Edition. *Elsevier Science Ltd*, Nov. 2017.

[16] B. Hou and F. Chen. Pacaca: Mining Object Correlations and Parallelism for Enhancing User Experience with Cloud Storage. In *Proceedings of the 26th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'18)*, Milwaukee, WI, Sep. 2018.

[17] B. Hou, F. Chen, Z. Ou, R. Wang, and M. Mesnier. Understanding I/O Performance Behaviors of Cloud Storage from a Client's Perspective. In *Proceedings of the 32nd International Conference on Massive Storage Systems and Technology (MSST'16)*, Santa Clara, CA, May 2016.

[18] IDC Study, sponsored by Seagate: Data Age 2025, the Digitization of the World. https://www.seagate.com/our-story/data-age-2025/. Nov. 2018.

[19] Z. Li, C. Jin, T. Xu, C. Wilson, Y. Liu, L. Cheng, Y. Liu, Y. Dai, and Z. Zhang. Towards Network-level Efficiency for Cloud Storage Services. In *Proceedings of the 2014 Conference on Internet Measurement Conference (IMC'14)*, Vancouver, BC, Canada, Nov. 2014.

[20] Z. Li, C. Wilson, Z. Jiang, Y. Liu, Ben Y. Zhao, C. Jin, Z. Zhang, and Y. Dai. Efficient Synchronization in Dropbox-like Cloud Storage Services. In *Proceedings of the 14th ACM/IFIP/USENIX International Middleware Conference (Middleware'13)*, Beijing, China, Dec. 2013.

[21] B. Mao, H. Jiang, and S. Wu. Improving Storage Availability in Cloud-of-Clouds with Hybrid Redundant Data Distribution. In *Proceedings of the 29th IEEE International Parallel & Distributed Processing Symposium (IPDPS'15)*, pages 633–642, May 2015.

[22] B. Mao, H. Jiang, S. Wu, and L. Tian. POD: Performance Oriented I/O Deduplication for Primary Storage Systems in the Cloud. In *Proceedings of the 28th IEEE International Parallel & Distributed Processing Symposium (IPDPS'14)*, pages 767–776, May 2014.

[23] B. Mao, S. Wu, and H. Jiang. Exploiting Workload Characteristics and Service Diversity to Improve the Availability of Cloud Storage Systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(7):2010–2021, 2016.

[24] Dutch T. Meyer and William J. Bolosky. A Study of Practical Deduplication. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST'11)*, San Jose, CA, Feb. 2011.

[25] Microsoft OneDrive. https://onedrive.live.com/about/en-us/. 2018.

[26] Rsync open source utility. https://rsync.samba.org/. 2018.

[27] Seafile: Enterprise file sync and share platform with high reliability and performance. https://www.seafile.com/en/home. 2018.

[28] A. Singh, X. Cui, B. Cassell, B. Wong, and K. Daudjee. MicroFuge: A Middleware Approach to Providing Performance Isolation in Cloud Storage Systems. In *Proceedings of the 34th IEEE International Conference on Distributed Computing Systems (ICDCS'14)*, Madrid, Spain, Jul. 2014.

[29] Slack: Team Messaging. https://www.slack.com/. 2018.

[30] SmokePing keeps track of the network latency. https://oss.oetiker.ch/smokeping/. 2018.

[31] The Future of Cloud Computing by North Bridge (6th Annual Survey). http://nbvp.northbridge.com/2016-future-cloud-computing-survey. 2016.

[32] T. Titcheu, E. Zhai, Z. Li, Y. Cui, and K. Ren. On the Synchronization Bottleneck of OpenStack Swift-like Cloud Storage Systems. In *Proceedings of the 35th IEEE International Conference on Computer Communications (INFOCOM'16)*, San Francisco, CA, Apr. 2014.

[33] A. Traeger, E. Zadok, N. Joukov, and C. Wright. A Nine Year Study of File System and Storage Benchmarking. *ACM Transactions on Storage*, 48(2):1–56, 2008.

[34] J. Wang, W. Gong, Peter J. Varman, and C. Xie. Reducing Storage Overhead with Small Write Bottleneck Avoiding in Cloud RAID System. In *Proceedings of the 13th ACM/IEEE International Conference on Grid Computing (GRID'12)*, Beijing, China, Sep. 2012.

[35] H. Xiao, Z. Li, E. Zhai, and T. Xu. Practical Web-based Delta Synchronization for Cloud Storage Services. In *Proceedings of the 9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'17)*, Santa Clara, CA, Jul. 2017.

[36] H. Xiao, Z. Li, E. Zhai, T. Xu, Y. Li, Y. Liu, Q. Zhang, and Y. Liu. Towards Web-based Delta Synchronization for Cloud Storage Services. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST'18)*, Oakland, CA, Feb. 2018.

[37] Q. Zhang, Z. Li, Z. Yang, S. Li, Y. Guo, and Y. Dai. DeltaCFS: Boosting Delta Sync for Cloud Storage Services by Learning from NFS. In *Proceedings of the 37th IEEE International Conference on Distributed Computing Systems (ICDCS'17)*, Atlanta, GA, Jun. 2017.

[38] S. Zhang, H. Catanese, and An-I Andy Wang. The Composite-file File System: Decoupling the One-to-One Mapping of Files and Metadata for Better Performance. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16)*, Santa Clara, CA, Feb. 2016.