A First Look at Mobile Cloud Storage Services: Architecture, Experimentation, and Challenges

Yong Cui, Zeqi Lai, and Ningwei Dai

Abstract

Mobile cloud storage services provide users a convenient and reliable way to store and share data on mobile devices. Despite increasing popularity, little work has focused on their architecture and internal sync protocol. In this article, we present a thorough architecture of mobile cloud storage services including sync protocols and key capabilities for speeding up transmission. Furthermore, we conduct a series of experiments to evaluate the sync performance of four popular commercial mobile cloud storage services. Our results show that the concrete deployment of capabilities has a strong impact on the performance of mobile cloud storage services. There is no clear winner, with all services suffering from some limitations or having potential for improvement. We pose issues and challenges to advance the topic area, and hope to pave a way for the forthcoming.

Cloud storage services facilitate the sync of local folders with servers in the cloud. In recent years they have gained tremendous popularity and accounted for a large amount of Internet traffic [1, 2]. This high public interest pushes various providers to enter the cloud storage market. Services like Dropbox, Google Drive, OneDrive, and Box are becoming pervasive in people's routines. The first one, active since 2007, currently accounts for over 300 million users and has reached US$10 billion market capitalization [3]. Apart from a convenient way to store, synchronize, and share personal files, cloud storage services also provide powerful application programming interfaces (APIs) that enable third-party applications to offload the burden of data storage and management to the server. By aggregating users' files or application data in the same place, cloud storage services are becoming the "data entrance" for personal users.

Meanwhile, the increasing number of wireless devices poses the demand of accessing, operating, and sharing user data from anywhere, on any device, at any time, and with any connectivity. To this end, cloud storage services are extended and deployed on mobile devices. File changes made on local files can be automatically uploaded to the cloud through wireless communications, and then be transferred to other devices. However, synchronizing and sharing data via wireless networks are more challenging than that in wired networks. Wireless networks often suffer higher delay and packet loss. Connections may also be interrupted because of the mobility and varied channel quality. Therefore, ensuring good sync performance while keeping data consistency is quite critical but challenging for mobile cloud storage services.

Previous works on cloud storage services have only focused on the server-side storage design [4] or the measurement in wired networks [1, 5–7]. Katsuya Suto *et al.* [8] introduced more general and effective principles for parallel data processing. However, there is scant research on the design and architectural choice of mobile cloud storage services. It is important for both service providers and research communities to understand the internal sync principle in order to provide optimization. To fill this gap, in this article we present the first study to shed light on the architecture, protocol, and sync performance of mobile cloud storage services. We provide a technical overview including the introduction of the system architecture, sync protocols, and key capabilities that can be implemented to optimize sync efficiency, which indicates how fast changes can be completely synchronized between the local file system and the cloud.

Further on, we conduct an experimental study for the four most popular commercial mobile cloud storage services. As previous work [5] only measures the sync time and the sync traffic in wired networks, we take the benchmark one step further. Our measurement focuses on the mobile platform, and we further consider the impact of various network conditions, the overhead of exception recovery that may be caused by intermittent connectivity, and the energy cost. Our results show that the concrete capability implementation has a strong impact on the sync performance, and some services cannot even work in high loss environments. In summary, all tested services suffer from performance limitations, and there remains room for improvement. As optimizing service quality for mobile cloud storage is significant and challenging, we also highlight the key challenges and give hints for future improvement. To the best of the authors' knowledge, this is the first article that provides a wide overview and experimental evaluation for mobile cloud storage services.

Mobile Cloud Storage: Architecture and Protocol

Architecture

Three major components can be identified in mobile cloud storage services: the *client*, the *control server*, and the *data storage server*. Figure 1 depicts these three components and their

The authors are with Tsinghua University.

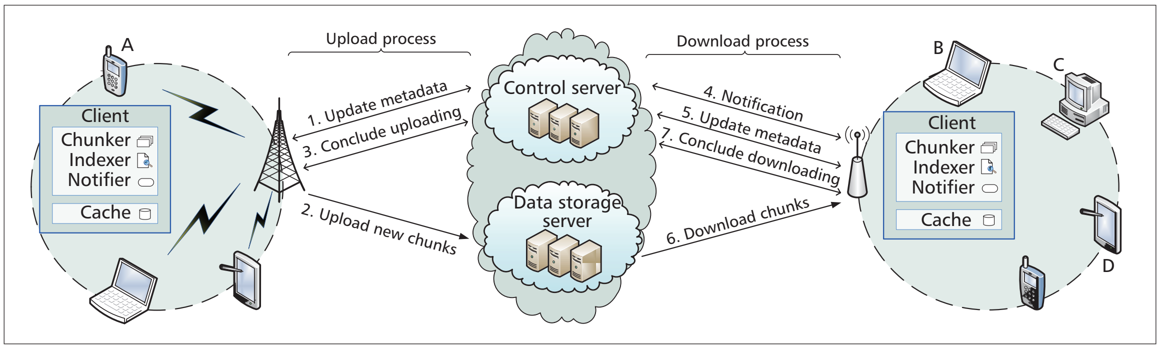

Figure 1. Sync architecture of mobile cloud storage.

interaction in a typical wireless network. All components are integrated and exchange data through the sync protocol. The client exchanges control information (including metadata and account information) as well as content with control and data storage servers, respectively, over wireless communications. To speed up transmission, several capabilities can be implemented in the sync protocol for optimization. Here we first describe the detailed functions of each component.

Client: Internally, the client consists of a variety of functional modules. Before a sync, files are often spilt into several chunks in fixed or variable size [9] by the `Chunker`. Each chunk is treated as an independent object and identified by a hash. The `Indexer` generates metadata, which include the hash value, version, and timestamp for each chunk. The `Notifier` connects to the control server and is used for polling changes performed elsewhere. Local file contents are stored in a `Cache`, and user modifications are first performed in the `Cache`. When connectivity is available, changes in the `Cache` are automatically updated to the cloud.

Control Server: The control server authenticates users, manages metadata, and notifies changes to clients. The file system in the control server has a different abstraction, which is presented by a big metadata database. In this database, no file contents are stored, only metadata including chunk lists, which are the sets of hashes uniquely identifying file contents.

Data Storage Server: This component maintains a key-value store of hashes to encrypted chunks. Generally, all files are treated as binary objects, which are partitioned into chunks. These chunks are stored in the data storage server with no knowledge of users and files, and how they fit together. Such a chunking mechanism provides scalability for storage and can also be leveraged to improve transmission efficiency, as discussed later. Data storage servers can be distributed in different geographical locations, meaning that the chunks combining a file may be stored in different storage nodes.

Sync Protocol

Typically, sync protocols are built on HTTP(S). We identify three main flows according to their function.

Notification Flow: The `Notifier` in the client continuously opens a persistent HTTP(S) connection to the control server, waiting for notification of changes performed elsewhere. Delayed responses are used to implement a push mechanism: notifications are pushed to the client once changes from other devices are updated to the cloud.

Control Information Flow: This flow is designed for the authentication and exchanging metadata between the client and the control server. Typically, a sync process starts with exchanging metadata with the control server, followed by a batch of either store or retrieve operations through storage servers. When data chunks are successfully transferred through data storage flows, the client updates metadata with the control server again to conclude the transmission.

Data Storage Flow: During this process, file contents are transmitted between the client and the data storage server. Generally, files are split into several chunks and transferred sequentially in storage flows. In certain circumstances when updating multiple files, the client may open several concurrent connections for transmission, and in this way chunks would be distributedly stored in different storage servers.

A typical sync process contains three key steps:
- *Sync preparation*, updating metadata with the control server
- *Data sync*, in which the client stores data to or retrieves data from the data storage server
- *Sync conclusion*, in which the client sends metadata to the control server again to conclude transmission

Now we illustrate the protocol by introducing a typical sync scenario. Assume that user A in Fig. 1 is outside of the office carrying her mobile device and wants to synchronize with workmates B, C, and D in the office. Once files in the local `Cache` are changed, the client follows the typical three steps to upload changes to the cloud. Metadata are first updated to the control server (step 1), and then the mobile device of A uploads chunks to the storage server (step 2). After transmission, the client communicates with the control server again to conclude the uploading process (step 3). Next, the control server will notify collaborators' devices in a WLAN via the notification flow (step 4). Devices of B, C, and D download chunks following the similar three processes to fetch and merge changes (step 5–7).

Capabilities for Performance Optimization

Wireless networks often suffer from limited bandwidth, higher packet loss, and intermittent connectivity, especially in poor signal strength environments. Moreover, mobile devices also have limited computation and storage resources. To improve sync efficiency and reduce traffic overhead in mobile networks, several key capabilities can be implemented in mobile cloud storage services.

Chunking: Chunking is the general and basic design consideration for data storage. Chunking helps to reduce the sync overhead when resuming the transmission from an interruption, which usually happens in an unstable wireless environment. During the sync process, the client creates a session with the storage server, and sequentially sends chunks tagged with the offset and length. The next chunk will be sent until receiving an application layer acknowledgment for the previous chunk. By this approach, the sender avoids transferring the acknowledged chunks if the sync session is interrupted.

Bundling: When a batch of small files needs to be transferred, a natural way is to treat each small file as a single chunk and submit them continuously. However, frequent acknowledgment for small chunks will seriously harm sync efficiency, especially in high delay networks. Moreover, too many HTTPS connections will also increase handshake overhead. It is beneficial to bundle small files in a single connection to reduce the application layer acknowledgment and the handshake overhead.

Client-Side Deduplication: The basic idea of client-side deduplication is to identify the contents that already exist in the cloud before a sync to avoid redundant transmission, and reduce both the sync traffic and the completion time [10]. In practice, such a process can be conducted in the metadata information flow. The control server can identify redundant chunks by checking existing hashes in the database.

Delta Encoding: Ideally, the client only needs to update the modified data to the cloud. To this end, delta encoding calculates file differences between two versions, allowing to only synchronize the delta content. Such technique provides significant benefit in bandwidth limited environments by reducing the sync traffic of frequent modifications. However for some encoded file types (e.g. jpeg or zip), delta-encoding should be carefully used since it will not provide benefit but only involve additional computation overhead.

Previous studies [1, 5] showed that cloud storage services can implement these capabilities in their desktop applications. In this article, we focus on cloud storage services in mobile platforms and check the capability implementation on mobile storage applications.

APIs for Connecting Everything in the Cloud

Modern mobile cloud storage services provide APIs that extend the access to the content management features in client software for use in third-party applications. In practical mobile operating systems (e.g., Android or iOS), these APIs take care of synchronizing data with cloud storage services through a familiar way like operations on the local file system. Behind the scenes, APIs synchronize local file changes to the cloud and automatically notify mobile applications when changes are made on other devices. Specifically, advanced functionalities like search, revision, or restoration of files can be implemented in APIs. The open APIs often implement the key capabilities like chunking or deduplication described above to optimize sync. The released client applications of mobile cloud storage services are also built on these content management APIs.

Experimental Evaluation

In this section we report the experimental results of evaluating the four most popular commercial mobile cloud storage services: Dropbox, Google Drive, OneDrive, and Box. Our goal is two-fold: to check the capability implementation and benchmark the sync performance, including the protocol overhead, sync completion time, and energy cost.

Testbed Setup

We deploy a testbed consisting of a smartphone (Galaxy Nexus, with Android 4.3.1) connecting to a controlled WiFi access point (AP). We use tc, a Linux traffic control tool, at the AP to tune the round-trip time (RTT) and packet loss of the wireless environment. We use tcpdump to collect the packet-level trace of a sync process, and extract the total sync time and traffic size. To measure the energy cost, we use `Monsoon Power Monitor` [11] to gain the power in each step of a sync process. Each test is performed 10 times to calculate the average result.

Checking the Capability Implementation

To check the capability in mobile cloud storage services, we design a specific test for each capability in the Android platform to observe if the given one is implemented.

Chunking: We determine whether files are transferred as single objects or split into chunks, each delimited by a pause, by monitoring the throughput during the upload process of files differing in size. We find that only Box does not perform chunking when uploading files. Dropbox uses 4 MB chunks, and OneDrive uses 1 MB chunks, while Google Drive uses 260 kB chunks. Since chunks are uploaded sequentially, using larger chunks reduces the acknowledgment overhead, but may involve more retransmission traffic when the sync process is interrupted. Chunking would be advantageous since it simplifies upload recovery in case of failures, especially in slow or intermittent networks.

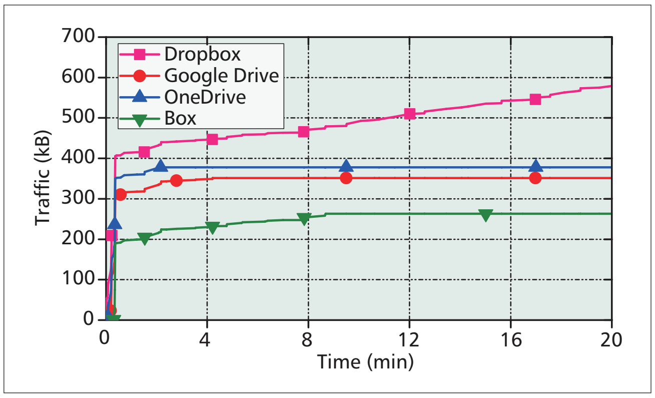

Figure 2. Background traffic in 20 minutes from startup.

Bundling: We identify bundling by analyzing whether a batch of small chunks is bundled and cumulatively acknowledged. Our results show that in the mobile platform, none of these services implement the bundling capability, with all chunks acknowledged sequentially.

Client-Side Deduplication: To check whether client-side data deduplication is implemented, we design two tests: uploading a file and its replica with a different name, and deleting a file in the server and uploading a copy of it. Our results show that only Dropbox implements data deduplication. All the other services will upload the same data even if it is available at the storage server. Interestingly, Dropbox can identify copies of files even after they are deleted.

Delta Encoding: By adding/deleting/updating a small amount of random data at the beginning, end, or random position of a file, we examine whether delta encoding is implemented. Our results show that in the mobile platform, none of the tested services implement delta encoding capability.

Sync Traffic Overhead

The traffic overhead is one of the most crucial design considerations for mobile cloud storage services. If not designed properly, the amount of data sync traffic can potentially cause financial pains to both providers and users. Now we evaluate the sync traffic overhead of the four services when the client starts, is idle, and synchronizes changes, as well as when the sync process is interrupted.

Startup and Idle Overhead: We measure the sync traffic in 20 min from startup without content updating to obtain the startup and idle overhead. Figure 2 reports the cumulative bytes exchanged. All four services have a burst when the client starts authentication and checking whether local files are up to date. Specifically, Dropbox has the largest startup traffic, which is 421.47 kB in 1.53 min. The background traffic keeps increasing because the clients continuously poll the server for new changes.

Sync Overhead: We then measure the total traffic size when uploading several benchmark sets differing in file number and size, as shown in Fig. 3. We find that in most cases the uploading traffic is close to the original file size. Specifically, we find that Google Drive has the largest uploading traffic, OneDrive and Dropbox come next, while Box has the smallest uploading traffic. This is because Google Drive implements the most aggressive chunking strategy (260 kB chunks) and hence involves much more acknowledgment overhead.

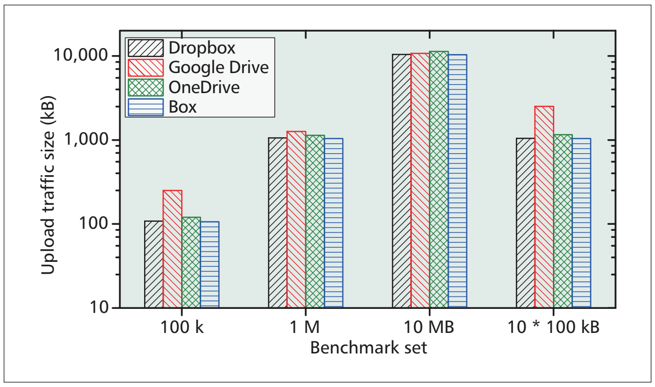

Figure 3. Sync overhead of uploading several benchmark sets.

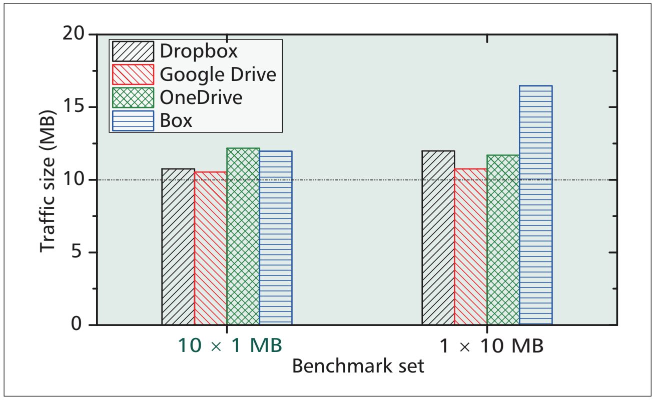

Figure 4. Total traffic overhead when the sync process is interrupted.

Furthermore, we measure the traffic overhead for updating local modifications to the server. Specifically, we use an editing tool to modify 10 bytes of a 5 MB word file, and then the client updates the modified content to the cloud. We observe that for all the services, the updating traffic is 5 MB even though we only change 10 bytes of data. Note that the sync overhead of Dropbox is much higher than the measurement results in [12] conducted in a desktop environment. The root cause is that in the mobile platform, all these services use the full-file sync mechanism instead of the incremental sync.

Overhead of Exception Recovery: Finally, we examine the total overhead when the sync process is interrupted. We upload a set of files with different sizes, and close the TCP connection when half of the file has been uploaded. After the restart of the program, the client will create new connections to finish the sync. The results of total traffic overhead are shown in Fig. 4. We find that Google Drive obtains the smallest total overhead. The aggressive chunking strategy used by Google Drive helps to reduce the overhead of exception recovery by avoiding the retransmission of chunks that are not received. Box generates the largest traffic because it retransmits the whole file when the connection is unexpectedly closed. The traffic of the four services when uploading 10 × 1 MB files are similar and close to the total file size (10 MB) because all services only send the unacknowledged files after the restart.

Sync Completion Time

Wireless or mobile environments often suffer from larger RTT and packet loss than wired networks. Now we evaluate the impact of various network conditions on the sync completion time. We measure the sync completion time as the duration from the first to the last packet with payload during the sync process, ignoring TCP tear-down delays.

The sync time results of uploading a set of files with

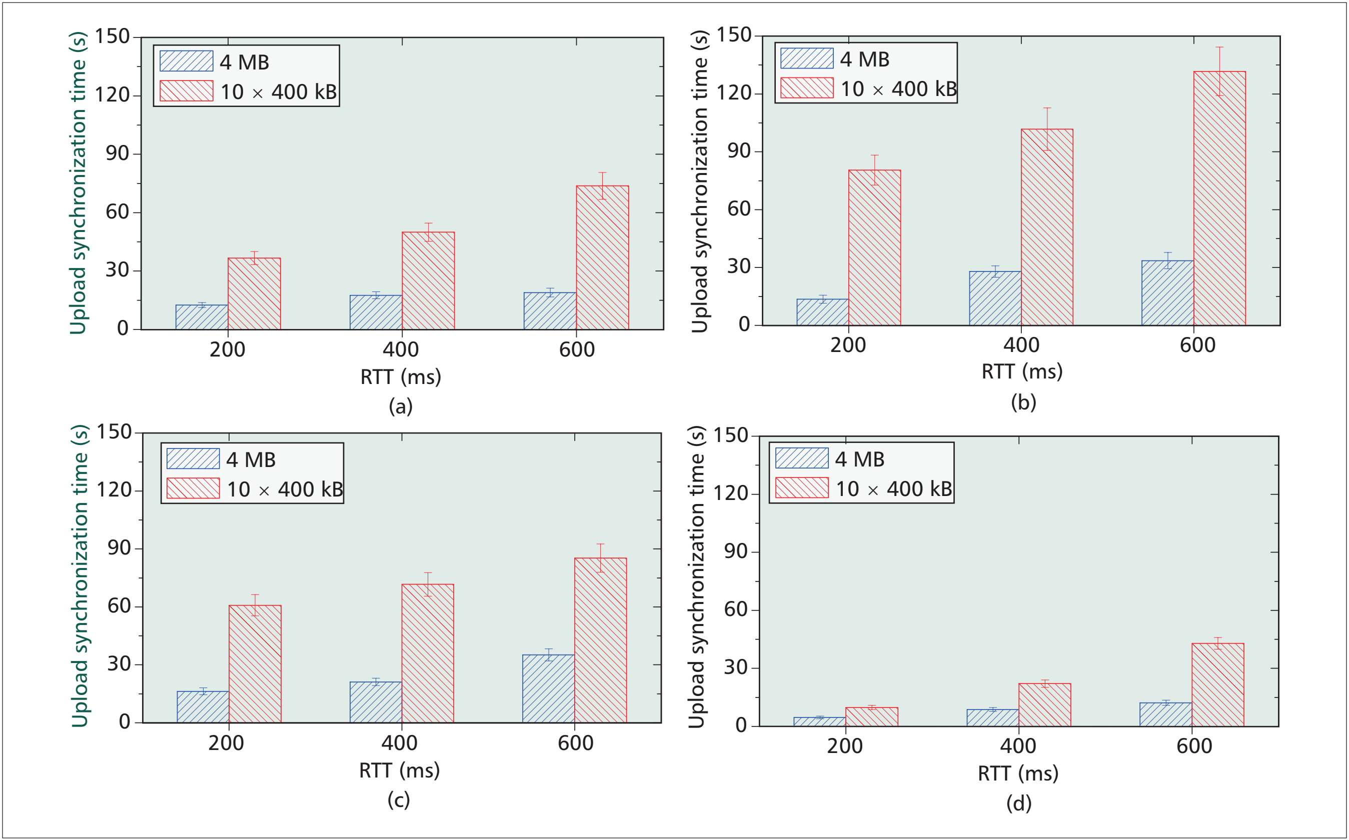

Figure 5. Sync complete time with various RTT settings: a) Dropbox; b) Google Drive; c) OneDrive; d) Box.

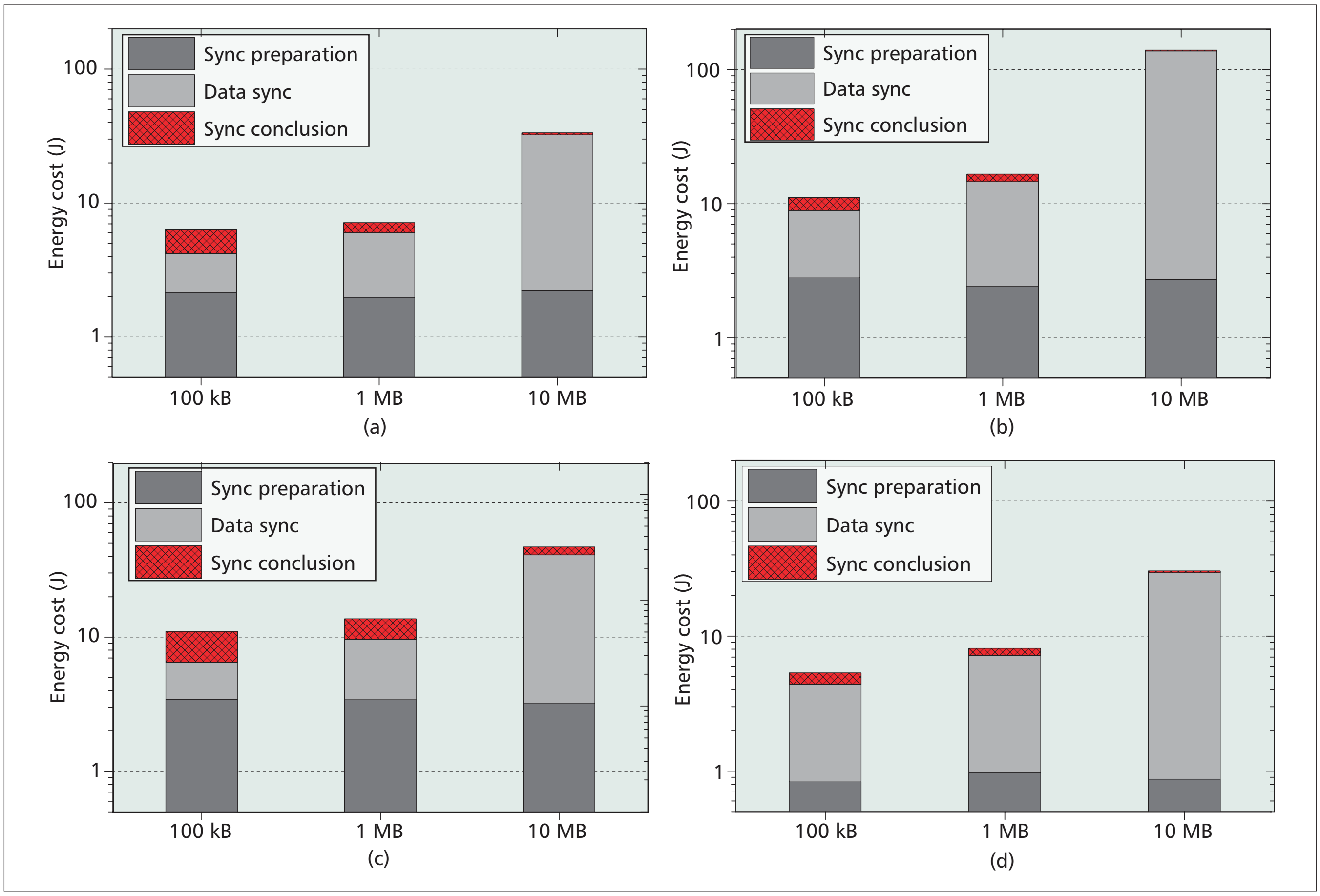

Figure 6. Breaking down energy consumption for four services during the sync process: a) Dropbox; b) Google Drive; c) OneDrive; d) Box.

different RTT using the four services are illustrated in Fig. 5. We only plot the results of 4 MB and 10 × 400 kB files due to the page limit, and we consider 4 MB to be a reasonable size on mobile devices because of their limited storage. Data sets in different volumes have also been tested and show similar results. In each case, Box is the fastest because without chunking, it has the smallest acknowledgment overhead and transfers chunks in a single TCP connection. Google Drive spends more time than others because of using the aggressive chunking strategy and the usage of separated TCP/SSL connections per file. Furthermore, we find that the completion time of multiple files is much longer than that of single files. This is because without bundling, a lot of small files involve more acknowledgment overhead. For all services, although it is common to see the sync be slower as the RTT rises, interestingly we find that for multiple small files' sync, the completion time increases faster. This is because many small files incur more acknowledgment overhead. Considering that the sequential acknowledgment scheme forces clients to wait one RTT between two chunks, the completion time is extended in slow networks because application-layer acknowledgments are transiting the network.

Results in various packet loss are similar and are omitted due to the page limit. Interestingly, we observe that when the packet loss reaches 10 percent, Google Drive can never synchronize data successfully! The client will get into a loop retransmitting the same chunk. We observe that Google Drive terminates the current connection and retransmits the chunks that are not received in high loss networks to ensure data consistency. Collectively, both the RTT and packet loss have strong impact on the sync completion time.

Energy Cost

We conduct experiments to evaluate the energy cost when synchronizing files differing in size. We break down the total energy consumption into the three sync steps described earlier, and the results are shown in Fig. 6. Box has the minimal energy cost in the Sync Preparation state because it does not split data into chunks before transmission. We observe that the energy cost of synchronizing a 100 kB file is similar to that of a 1 MB file. The reason is that because of the TCP slow start, the sync completion time of a 100 KB file is close to that of a 1 MB file. Specifically, we observe the energy consumption of Google Drive when synchronizing a 10 MB file is the highest. The average CPU usage of the four services when uploading 10MB files are 10.47 percent (Dropbox), 64.53 percent (Google Drive), 13.38 percent (OneDrive), and 13.80 percent (Box). Thus, the reason for such high energy cost may be that Google Drive uses small chunk size and continuously hashes the chunks during the entire sync process.

Challenges

According to our evaluation results of four mobile cloud services, we find that the concrete capability implementation has a strong impact on the sync performance in various network environments. Now we conclude and highlight three key challenges of future improvement for mobile cloud storage services.

The Conflict between Bandwidth Saving and Distributed Storage

Network bandwidth efficiency is the most significant but extremely challenging issue for sync in cellular networks. Ideally, only the modified parts of the file need to be synchronized to the cloud. However, our measurement results indicate that most mobile cloud storage services only use the simple full-file sync mechanism, wasting a large amount of traffic when synchronizing small changes. Implementing the incremental sync mechanism is very challenging in practice for two key reasons. On one hand, most of today's cloud storage services (e.g., OneDrive and Dropbox) are built on top of RESTful infrastructure, which only supports data access operations at the full-file (or full-chunk) level. On the other hand, the delta-encoding algorithm is the key technique for the incremental sync mechanism. However, most delta-encoding algorithms [13, 14] work at a file granularity, which means that the utility running on two ends must have access to the entire file. But for mobile cloud storage services, files are split into chunks and stored distributedly. Straightforward adaptation of delta-encoding requires piecing together all chunks and reconstructing the whole file, which would waste massive intra-cluster bandwidth. Since file modifications frequently happen, for future improvement it is worthwhile to address the challenges by proposing an improved delta encoding algorithm to reduce sync traffic overhead.

The Trade-off between Real-Time Consistency and Protocol Overhead

Real-time consistency requires that data among multiple devices should be correctly synchronized as soon as possible [15]. To this end, some services like Dropbox open a persistent TCP connection for polling and receiving notifications. However, such a polling mechanism causes a large amount of traffic and energy consumption since polling involves additional message exchanging, requesting the wireless network interface to be woken up and kept in a high power state. For mobile cloud storage services, the notification flow should be carefully designed to avoid too much traffic and energy waste.

The Balance between Sync Efficiency and Constrained Energy Resource in Mobile Devices

Sync efficiency, defined as how fast the local changes can be updated to the server, is an important metric for mobile cloud storage services. Generally, deduplication, delta-encoding, and compression are the key techniques for efficiency improvement. However, all these techniques may involve additional computational overhead and energy cost, as shown above. The emergence of more energy-efficient sync techniques is expected to improve mobile cloud storage services.

Conclusion and Outlook

Since cloud storage is becoming a more predominant way to store, synchronize, and share user data in mobile networks, understanding the design and performance of cloud storage services is significant for both service providers and research communities. In this article, we present a thorough overview of the mobile cloud storage services by introducing their architecture, sync protocols, and key capabilities, followed by an experimental study that evaluates the four most popular commercial services. Our experiment results show that the capabilities have a strong impact on the performance of mobile cloud storage services. All tested services suffer performance limitations, and there remains room for improvement.

Finally, we conclude and highlight three key challenges for future improvement of mobile cloud storage services. We hope our experiments and analysis can give insight for both service providers and research communities. To the best of our knowledge we are the first to study the design and performance of mobile cloud storage services. As future work, we intend to do research on improving network level sync efficiency for mobile cloud storage services.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (no. 61422206, 61120106008) and National 863 project (no. 2015AA015701).

References

[1] I. Drago *et al.*, "Inside Dropbox: Understanding Personal Cloud Storage Services," *Proc. ACM IMC*, 2012, pp. 481–94.
[2] Z. Li *et al.*, "Towards Network-Level Efficiency for Cloud Storage Services," *Proc. ACM IMC*, 2014, pp. 115–28.
[3] Dropbox, http://www.dropbox.com.
[4] M. Vrable, S. Savage, and G. M. Voelker, "Cumulus: Filesystem Backup to the Cloud," *TOS*, vol. 5, no. 4, 2009, p. 14.
[5] I. Drago *et al.*, "Benchmarking Personal Cloud Storage," *Proc. ACM IMC*, 2013, pp. 205–12.
[6] Y. Zhang *et al.*, "Viewbox: Integrating Local File Systems with Cloud Storage Services," *Proc. FAST*, 2014, pp. 119–32.
[7] A. Li *et al.*, "Cloudcmp: Comparing Public Cloud Providers," *Proc. 10th ACM SIGCOMM Conf. Internet Measurement*, 2010, pp. 1–14.
[8] K. Suto *et al.*, "Toward Integrating Overlay and Physical Networks for Robust Parallel Processing Architecture," *IEEE Network*, vol. 28, no. 4, July 2014, pp. 40–45.
[9] A. Muthitacharoen, B. Chen, and D. Mazieres, "A Low-Bandwidth Network File System," *Proc. ACM SIGOPS*, vol. 35, no. 5, 2001, pp. 174–87.
[10] B. Agarwal *et al.*, "Endre: An End-System Redundancy Elimination Service for Enterprises," *Proc. NSDI*, 2010, pp. 419–32.
[11] Mosoon, https://www.msoon.com/LabEquipment/PowerMonitor.
[12] Z. Li *et al.*, "Efficient Batched Synchronization in Dropbox-Like Cloud Storage Services," *Middleware 2013*, Springer, 2013, pp. 307–27.
[13] A. Tridgell *et al.*, "The Rsync Algorithm," 1996.
[14] A. Chitnis *et al.*, "Primary Neurogenesis in Xenopus Embryos Regulated by a Homologue of the Drosophila Neurogenic Gene Delta," *Nature*, vol. 375, no. 6534, 1995, pp. 761–66.
[15] T. Hobfeld *et al.*, "Challenges of QoE Management for Cloud Applications," *IEEE Commun. Mag.*, vol. 50, no. 4, Apr. 2012, pp. 28–36.

References

YONG CUI (cuiyong@tsinghua.edu.cn) received his B.E. and Ph.D. degrees in computer science and engineering from Tsinghua University, China, in 1999 and 2004, respectively. He is currently a full professor at Tsinghua University and Co-Chair of IETF IPv6 Transition WG Softwire. His major research interests include mobile wireless Internet and computer network architecture. Having published more than 100 papers in refereed journals and conferences, he received the National Award for Technological Invention in 2013, and the Influential Invention Award of the China Information Industry in both 2012 and 2004. Holding more than 40 patents, he has authored three Internet standard documents, including RFC 7040 and RFC 5565, for his proposal on IPv6 transition technologies. He serves at the Editorial Board on both *IEEE TPDS* and *IEEE TCC*.

ZEQI LAI (laizq13@mails.tsinghua.edu.cn) is now a Ph.D. student in the Department of Computer Science and Technology, Tsinghua University. His supervisor is Prof. Yong Cui. His research interests include cloud computing and cloud storage.

NINGWEI DAI (lemondnw@gmail.com) is now a Master's student in the Department of Computer Science and Technology, Tsinghua University. Her supervisor is Prof. Yong Cui. Her research interests include cloud computing and cloud storage.