Indo American Journal of Multidisciplinary Research and Review (IAJMRR)
ISSN: 2581 - 6292 / Impact Factor: 6.885 / Website: www.iajmrr.com

INDUCTING DESIGN PROVISIONING SYSTEM FOR UNIVERSITY EXAMINATION USING OPTIMIZATION ALGORITHM

A. Barveen*, M. Kamali & S. Selvi*****
Department of Computer Science and Engineering, MIET Engineering College, Tamilnadu

Cite This Article: A. Barveen, M. Kamali & S. Selvi, "Inducting Design Provisioning System for University Examination Using Optimization Algorithm", Indo American Journal of Multidisciplinary Research and Review, Volume 5, Issue 1, Page Number 9-13, 2021.

Abstract:

Mobile cloud storage services have gained phenomenal success in recent few years. We identify, analyze and address the synchronization (sync) inefficiency problem of modern mobile cloud storage services. Our measurement results demonstrate that existing commercial sync services fail to make full use of available bandwidth, and generate a large amount of unnecessary sync traffic in certain circumstance even though the incremental sync is implemented. For example, a minor document editing process in Drop box may result in sync traffic 10 times that of the modification. These issues are caused by the inherent limitations of the sync protocol and the distributed architecture. Based on our findings, we propose Quick Sync, a system with three novel techniques to improve the sync efficiency for mobile cloud storage services, and build the system on two commercial sync services. Our experimental results using representative workloads show that Quick Sync is able to reduce up to 73.1% sync time in our experiment settings.

Introduction:

Personal cloud storage services are gaining tremendous popularity in recent years by enabling users to conveniently synchronize files across multiple devices and back up data. Services like Drop box, Box, Sea file have proliferated and be-come increasingly popular, attracting many big companies such as Google, Microsoft or Apple to enter this market and offer their own cloud storage services. As a primary function of cloud storage services, data synchronization (sync) enables the client to automatically update local file changes to the remote cloud through network communications. Synchronization efficiency is determined by the speed of updating the change of client files to the cloud, and considered as one of the most important performance metrics for cloud storage services. Changes on local devices are expected to be quickly synchronized to the cloud and then to other devices with low traffic overhead.

More recently, the quick increase of mobile devices poses the new demand of ubiquitous storage to synchronize users' personal data from anywhere at any time and with any connectivity. Some cloud storage providers have extended and deployed their services in mobile environments to support Mobile Cloud Storage Services, with functions such as chunking and DE duplication optionally implemented to improve the transmission performance.

Despite the efforts, the sync efficiency of popular mobile cloud storage services is still far from being satisfactory, and under certain circumstances, the sync time is much longer than expected. The challenges of improving the sync efficiency in mobile/wireless environment are threefold. First, as commercial storage services are mostly closed source with data encrypted, their designs and operational processes remain unclear to the public. It is hard to directly study the sync protocol and identify the root cause of sync difficulty.

Although some existing services try to improve the sync performance by incorporating several capabilities, it is still unknown whether these capabilities are useful or enough for good storage performance in mobile/wireless environments. Finally, as a mobile cloud storage system involves techniques from both storage and network fields, it requires the storage techniques to be adaptive and work efficiently in the mobile environment where the mobility and varying channel conditions make the communications subject to high delay or interruption.

To address above challenges, we identify, analyze and propose a set of techniques to increase the sync efficiency in modern mobile cloud storage systems. Our work consists of three major components: 1) identifying the performance bottlenecks based on the measurement of the sync operations of popular commercial cloud storage services in the mobile/wireless environment, 2) analyzing in details the problems identified, and 3) proposing a new mobile cloud storage system which integrates a few techniques to enable efficient sync operations in mobile cloud storage services.

We first measure the sync performance of the most popular commercial cloud storage services in mobile/wireless networks (§2). Our measurement results show that the sync protocol used by these services is indeed inefficient. Specifically, the sync protocol cannot fully utilize the available bandwidth in high RTT environment or when synchronizing multiple small files. Furthermore, although some services, e.g. Drop box, have implemented the incremental sync to reduce the traffic size, this technique is not valid in all scenarios. We observe that a document editing process the two major factors that contribute to the

inefficiency are the inherent limitations of the sync protocol and the distributed storage architecture. Specifically, the de-duplication to reduce redundant data transmissions does not always contribute to the sync efficiency. The distributed nature of storage services poses a challenge to the practical implementation of the delta encoding algorithm, and the failure in the incremental sync may lead to a large traffic overhead. The iterative sync scheme suffers from low throughput when there is a need to synchronize a set of files through a slow network.

Based on our observation and analysis, we propose Quick Sync, a system with three novel techniques to improve the sync efficiency for mobile cloud storage services (§4). To reduce the sync time, we introduce Network-aware Chunkier to adaptively select the proper chunking strategy based on real-time network conditions. To reduce the sync traffic overhead, we propose Redundancy Eliminator to correctly perform delta encoding between two similar chunks located in the original and modified files at any time during the sync process. We also design Batched Synced to improve the network utilization of sync protocol and reduce the overhead when resuming the sync from an interruption.

We build our Quick Sync system on Drop box, currently the most popular cloud storage services, and Sea file, an popular open source personal cloud storage system (§5). Collectively, these techniques achieve significant improvement in the sync latency for cloud storage services. Evaluation results (§6) show that the Quick Sync system is able to significantly improve the sync efficiency, reducing up to 73.1% sync time in representative sync scenarios with our experiment settings. To the best of our knowledge, we are the first to study the sync efficiency problem for mobile cloud storage services.

Synchronization Inefficiency:

Sync efficiency indicates how fast a client can update changes to the cloud. In this section, we conduct a series of experiments to investigate the sync inefficiency issues existing in four most popular commercial cloud storage service systems in wireless/mobile environments. We will further analyze our observed problems and explain their root causes in Section 3.

Architecture:

A typical architecture of cloud storage services includes three major components [1]: the client, the control server and the data storage server. Typically, a user hash designated local folder (called sync folder) where every file operation is informed and synchronized to the cloud by the client. The client splits file contents into chunks and indexes them to generate the metadata (including the Hashes, modified time etc.). The file system on the server side has an abstraction different from that of the client. Metadata and contents of user files are separated and stored in the control and data storage servers respectively. During the sync process, metadata are exchanged with the control server through the metadata information flow, while the contents are transferred via the data storage flow. In a practical implementation, the control server and the data storage server may be deployed in different locations. For example, Drop box builds its data storage server on Amazon EC2 and S3, while keeping its own control server. Another important flow, namely notification flow, pushes notifications to the client once changes from other devices are updated to the cloud.

Low DER Not Equal to Efficiency:

To evaluate the effectiveness of DE duplication in reducing the original transmission data size, the metric De duplication Efficiency Ratio (DER) is defined as the ratio of the DE duplicated file size to the original file size. Intuitively, lower DER means more redundancy can be removed and the total sync time can be reduced. However, our experiment indicates that lower DER may not always make sync efficient.

As only Drop box and Sea file incorporate the DE duplication function, to study the relationship between the sync time and DER, we use Wire shark to measure the packet level trace of the two services in a controlled Wi-Fi environment. We use to tune the RTT for each service according to the typical RTT values in mobile/wireless networks [3]. We only perform measurements on the Windows platform because most services did not implement the DE duplication on the Android platform. We collect about 500MB user data from a Drop box user and upload these fresh data via the tested services. From the trace captured we can get the sync time and calculate the DER as a ratio of the transmission traffic size and the original traffic size.

The DER for Drop box and Sea file are 87% and 65% respectively under each RTT setting. Intuitively, a higher DER value would take more time to complete a sync operation. However, we find that in a better network condition (e.g. when the RTT is 200ms), it costs more time for Sea file to complete the sync.

Bandwidth Inefficiency:

Sync throughput is another critical metric that reflects the sync efficiency. The sync protocol relies on TCP and its performance is affected by network factors such as RTT or packet loss. Because of different system implementations, it is unreasonable to evaluate how the underlying bandwidth of a storage service is used by directly measuring the throughput or latency [2]. To characterize the network usage of sync protocol, we introduce a novel metric, Bandwidth Utilization Efficiency (BUE), which is defined as the ratio of the practical measured throughput to the theoretical TCP bandwidth.

The latter indicates the available bandwidth in Where can the observed average congestion window size during the transmission. The BUE metric is a value between 0 and 1. Rather than measuring the end-to-end throughput, we apply BUE to evaluate how well the cloud storage service can utilize the

available network bandwidth to reduce the sync time.

To investigate how the sync protocol utilizes the underlying network bandwidth, we have the Windows and Android clients of Drop box, Google Drive, One Drive and Sea file run in Wi-Fi and cellular networks (UMTS) respectively.

Reusing Existing Network Connections:

The second technique is to reuse the existing network connections rather than making new ones in storing files. While it may be easy and natural to make a new network connection for each chunk, the handshake overhead for establishing a new connection is not negligible, and creating many new connections also extends the period in the slow start state especially for small chunks. The Batched Synced reuses the storage connection to transfer multiple chunks, avoiding the overhead of duplicate TCP/SSL handshakes. Moreover, cloud storage services maintain a persistent notification flow for capturing changes elsewhere. Hence we reuse the notification flow for both requesting notification and sending file data to reduce the handshake overhead and the impact of slow start. Specifically, both the request and data are transferred over HTTP(S), so we use the Content-Type field in the HTTP header to distinguish them in the same TCP connection

System Implementation:

To evaluate the performance of our proposed schemes, we build the Quick Sync system over both Drop box and Sea file platforms. Implementation over Drop box: Since both the client and server of Drop box are totally closed source, we are unable to directly implement our techniques with the released Drop-box software. Although Drop box provides APIs to allow user program to synchronize data with the Drop box server, different from the client software, the APIs are Restful and operate at the full file level. We are unable to get the hash value of a certain chunk, or directly implement delta-encoding algorithm via the APIs.

To address this problem, we leverage a proxy in Amazon EC2 which is close to the Drop box server to emulate the control server behavior. The proxy is designed to generate the Virtual Chunks, maintain the map of file to the chunk list and calculate the hash and the sketch of chunks. During a sync process, user data are first uploaded to the proxy, and then the proxy updates the metadata in the database and stores the data to the Drop box server via the APIs. Since the data storage server of Drop box is also built on Amazon EC2, the bandwidth between our proxy and Drop box is sufficient and not the bottleneck.

To make our Network-aware Chunkier efficient and ad-just able, we use the SAMPLEBYTE [11] as our basic chunking method. Like other content defined chunking methods, the sample period p set in SAMPLEBYTE also determines both the computation overhead and DE duplication ratio. We leverage the adjustable property of p to generate a suite of chunking strategies with various DE duplication ratio and computation overhead, including the chunk-based DE duplication with the average chunk size set to 4MB, 1MB, 512KB and 128KB. Each Virtual Chunk contains a 2-byte field for chunk length.

We use librsync [13] to implement delta encoding. We use a tar-like method to bundle all data chunks in the sync process, and a client communicates with our proxy at the beginning of a sync process to notify the offset and length of each chunk in the sync flow. The timer of our Synced is set to 60s. We write the Quick Sync client and proxy in around 2000 lines of Java codes. To achieve efficiency, we design two processes to handle chunking and transmission tasks respectively in the client. The client is implemented on a Galaxy Nexus smart phone with a 1.2 GHz Dual Core CPU, 1 GB memory and the proxy is built on an Amazon EC2 server with a 2.8 GHz Quad Core CPU and 4 GB memory.

Implementation over Seafile: Although we introduce a proxy between the client and the Drop box server, due to the lack of full access of data on the server, this implementation suffers from the performance penalty. For instance, to perform delta encoding, the proxy should first fetch the entire chunk from the Drop box server, update its content and finally store it back to Drop box. Even though the bandwidth between the proxy and the Drop box server is sufficient, such an implementation would inevitably involve additional latency during the sync process.

In order to show the gain in the sync efficiency when our system is fully implemented and can directly operate over the data, we further implement Quick Sync with Sea file [14], an open source cloud storage project. The implementation is similar to that using Drop box but without the need of a proxy. Specifically, we directly modify source codes at both the client and server sides. We modify the client in a Linux laptop with a 2.6 GHz Intel Quad Core CPU and 4 GB memory. We build the server on a Linux machine with a 3.3 GHz Intel Octal Core CPU and 16 GB memory, as only the Sea file software on Linux platform is open source. Techniques in Quick Sync can also be implemented in the similar way on other mobile platforms.

Performance Evaluation:

To evaluate the performance of our schemes, we first investigate the throughput improvement of using the Network-aware Chunkier, and then show that the Redundancy Eliminator is able to effectively reduce the sync traffic. We further evaluate the capability of the Batched Synced in improving the bandwidth utilization efficiency. Finally, we study the overall improvement of the sync efficiency using real-world workloads. In each case, we compare the performances of the original Sea file and Drop box clients with those when the two service frameworks are improved with Quick Sync

Impact of the Redundancy Eliminator:

Next we evaluate the sync traffic reduction of using our Redundancy Eliminator with the average chunk size set to 1MB to exclude the impact of adaptive chunking. We are editing process, as temporary files whose sizes are close to that of the original. Ppt file are created and deleted. Changes are automatically synchronized. Our solution significantly reduces the traffic size, with Quick Sync to execute the delta encoding on the temporary files in the middle of the sync process to reduce the traffic. The Data Backup workload on Windows is a typical usage for large data backup. This workload contains 37655 files, with various file types (e.g. PDF or video) and sizes (from 1KB to 179MB). Our Quick Sync achieves 37.4% sync time reduction by eliminating the redundancy and reducing the acknowledgement overhead to improve the bandwidth utilization.

We also play the workload on Android platform. The Document Editing workload on Android is similar to that generated in the above experiment but contains fewer modifications. Our implementation reduces 41.4% of the total sync time. The Photo Sharing is a common workload for mobile phones. Although the photos are often in the encoded format and hard to be DE duplicated, our implementation can still achieve 24.1% time saving through the batched transmission scheme. The System Backup work-load is generated to back up all system settings, app binaries together with app configurations in a phone via a slow 3G network. As our implementation adaptively selects aggressive chunking strategy to eliminate larger amount of the backup traffic and bundles chunks to improve the bandwidth utilization, 52.9% sync time saving is achieved.

App Data Backup is a workload generated when we walk in the outdoor environment while using a phone in a LTE network to back up the data and configurations of several specified applications. As the network condition changes during our movement, Quick Sync dynamically selects the proper chunking strategy to eliminate the redundant data, which reduces 45.1% sync traffic and 73.1% total sync time.

Interestingly, for most workloads in our experiment LBF-S achieves the lowest traffic size in the sync process, but the total sync time of LBFS is larger than other solutions. This is because LBFS leverages a very aggressive DE duplication strategy that chops files into small chunks and identifies redundant data by checking hash values. However, the aggressive strategy does not always improve the sync efficiency since it is computation-intensive in the resource-constraint mobile platform. In addition, the effectiveness of DE duplication degrades for compressed workloads (e.g. photo sharing). Quick Sync outperforms LBFS and End RE by adaptively selecting the proper chunking strategy according to current network conditions, and wisely executing delta encoding during file editing.

Focusing on the enterprise cloud storage services, Cloud-Cmp [15] measures the elastic computing, persistent storage, and networking services for four major cloud providers. The study in [16] provides a quantitative analysis of the performance of the Windows Azure Platform. Works in [17] perform an extensive measurement against Amazon S3 to elucidate whether cloud storage is suitable for scientific grids. Similarly, [18] presents a performance analysis of the Amazon Web Services. However these studies provide no insights into personal cloud storage services, while our measurement study focuses on the emerging personal cloud storage services in mobile/wireless environments.

Some literature studies also attempt to analyze the performance of personal cloud storage services. To our best knowledge, Hu et al. first analyze personal cloud storage services by comparing the performance of Drop box, Mozy, Carbonate and Crash Plan [24]. However, they only provide an incomplete analysis on the backup/restore time for several types of files. Gracia-Tinedo et al. study the REST interfaces provided by three big players in the personal cloud storage arena [22], and more recently they conduct a measurement study of the internal structure of Ubuntu One [21]. Dragoet al. gives a large-scale measurement for Drop-box [19], and then compares the system capabilities for five popular cloud storage services in [2]. However, all these previous studies only focus on the desktop services based on black-box measurement. Li et al. give the experimental study of the sync traffic, demonstrating that a considerable portion of the data sync traffic is wasteful [20]. Our work steps closer to reveal the root cause of inefficiency problem from the protocol perspective, and we are the first to study the sync efficiency problem in mobile/wireless networks where the network condition (e.g. RTT) may change significantly.

Conclusion:

Despite their near-ubiquity, mobile cloud storage services fail to efficiently synchronize data in certain circumstance. In this paper, we first study four popular cloud storage services to identify their sync inefficiency issues in wireless networks. We then conduct the in-depth analysis to give the root causes of the identified problems with both trace studies and data decryption. To address the inefficiency issues, we propose Quick Sync, a system with three novel techniques. We further implement Quick Sync to support the sync operation with Drop box and Sea file. Our extensive evaluations demonstrate that Quick Sync can effectively save the sync time and reduce the significant traffic overhead for representative sync workloads.

References:

1. Y. Cui, Z. Lai, and N. Dai, “A first look at mobile cloud s-torage services: Architecture, experimentation and challenge,” http://www.4over6.edu.cn/others/techn ical report.pdf.
2. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras, “Bench-marking personal cloud storage,” in IMC. ACM, 2013.

3. Balasubramanian, R. Mahajan, and A. Venkataramani, “Aug-menting mobile 3g using wifi,” in Mobi Sys. ACM, 2010.
4. Tridgell, P. Mackerraset al., “The rsync algorithm,” 1996.
5. D. Kholia and P. Wegrzyn, “Looking inside the (drop) box,” in USENIX Workshop on Offensive Technologies (WOOT), 2013.
6. “Dynamorio,” http://dynamorio.org.
7. Muthitacharoen, B. Chen, and D. Mazieres, “A low-bandwidth network file system,” in SIGOPS. ACM, 2001.
8. Zhu, K. Li, and R. H. Patterson, “Avoiding the disk bottleneck in the data domain deduplication file system.” in FAST. USENIX, 2008.
9. P. Shilane, M. Huang, G. Wallace, and W. Hsu, “Wan-optimized replication of backup datasets using stream-informed delta compression,” TOS, 2012.
10. Y. Hua, X. Liu, and D. Feng, “Neptune: Efficient remote communication services for cloud backups,” in INFOCOM. IEEE, 2014.