vfcBOX: Multi-User Consistent File Sharing

Jean-Pierre Ramos
INESC-ID / Instituto Superior Técnico / Technical University of Lisbon
jean-pierre.ramos@ist.utl.pt

ABSTRACT

The emerging of cloud file sharing systems has been motivated by real user needs for data sharing. There are many solutions providing such sharing support all having the common goal of being widely scalable while providing users with consistent shared data. However, offering consistent data is at odds with scalability as it requires many messages and available network bandwith for file transfer.

Network bandwidth can be minimized using several techniques such as compression, deduplication[10], delta encoding[9], etc. However, these approaches do not take into account that not all files must be fully consistent at all times for all users.

In this paper we further increase the scalability of a cloud file sharing system, called vfcBOX, by taking into account the notion of users interest. This means that vfcBOX considers users' consistency needs regarding shared files, to avoid sending useless (or unnecessary) data through the network. As a matter of fact, some files do not need to be constantly propagated to all users, because some of them do not require such immediacy given the particular semantics of the shared data.

vfcBOX uses not only deduplication techniques to minimize network usage but also a consistency model that takes into account the users' interests. The result is a scalable and efficient cloud file sharing system that fulfills users needs regarding data sharing.

Keywords: File Sharing, Data Deduplication, Adaptability, Interest Management.

1. INTRODUCTION

Cloud file sharing systems are becoming an emergent solution to the problem of sharing/updating data among multiple users[1]. These systems require new methods to synchronize data in a more efficient way, specially with respect to bandwidth, which is still a scarce resource and is decisioning for scalability. The goal of this work is to design and build a system called **vfcBOX** that efficiently manages the consistent sharing of data. For scalability, the system is required to be efficient specially in terms of network bandwidth usage.

To attemp such goal, vfcBOX has to deal with the following challenges: i) minimize the amount of data to be transfered through compact forms of representing data; ii) allow concurrent access to files, while preserving replica consistency; iii) ensure correct data synchronization, while minimizing the use of network resources; iv) deal with conflicts, supporting ways of detecting and resolving them through the merge of concurrent data updates; v) support disconnected work.

Current solutions to minimize network bandwidth (or data stored) in distributed systems, such as Semantic-chunks[16], LBFS[11], Rsync[15], redFS[4], SVN[1], Dropbox[2] and many others, take advantage of some form of either compression or data redundancy.

In particular, deduplication[10] techniques take advantage of the similarity between portions of data. Efficient synchronization may be achieved by not sending data that is found as redundant between two sites. Other solutions are based in **Optimistic Replication**[14], which enables the bounded divergence of data consistency. More specifically, systems such as Semantic-chunks[16] use the notion of **interest management**[8](or locality-awareness) of a user to avoid propagating (useless) updates. These solutions try to reason about the importance of each update, performing an intelligent management of updates and performing a selective scheduling based on this importance.

vfcBOX combines deduplication with an interest-based optimistic replication that adapts consistency guarantees to users needs. In other words, the system is able to identify the user's interest over each data set. Thus, updates regarding more relevant data (i.e. requiring strong consistency) are rapidly propagated to users; other updates, regarding data not required to be consistent, are delayed and batched, in order to prioritize file and data transfer according to users' interests. This is specially important when synchronizing "dropboxes" with large number of files, number of updates, and file sizes.

Additionally, deduplication techniques allow the system to minimize the redundant data transfered between sites.

This paper is organized as follows. Section 2 presents the vfcBOX's architecture. Section 3 describes some implementation details of the vfcBOX system. Section 4 presents the obtained results of the evaluation procedure performed to the implemented solution. Finally, Section 5 describes the

This article has been published in
MGC'2011, December 12th, 2011, Lisbon, Portugal.

[1] http://subversion.tigris.org/
[2] http:www.dropbox.com

related work.

2. ARCHITECTURE

vfcBOX provides file storage and synchronization among multiple users and it appears to users as a synchronized folder where files/folders may be dropped and updated without having to explicitly synchronize them. It uses deduplication to minimize storage space and network bandwidth on client-server communication. Additionally, vfcBOX defines a relaxed consistency model that takes into account the interests of users over certain files or certain parts of a file, in order to create multiple consistency levels, and use this to prioritize and schedule (delay, batch, reorder, omit) data transfers. These two aspects minimize network bandwidth while ensuring that users consistency needs are fulfilled.

2.1 Baseline Architecture

vfcBOX is based on a client-server architecture. Figure 1 is a simplified illustration of the system in which clients submit their updates and consistency needs (i.e. data interests) to the server (for simplicity of the presentation, with no lack of generality, we consider just one server, that could reside inside a data center, or cloud infrastructure). For a user, such interests specify files or parts of files which are most relevant to him, thus requiring strong consistency. Taking these interests into account, the vfcBOX server is then able to enforce multiple consistency guarantees over multiple data subsets. In particular, it does not propagate useless or unneeded updates (i.e. those regarding data which is not relevant for a user, or that can be subsumed by a later update sent).

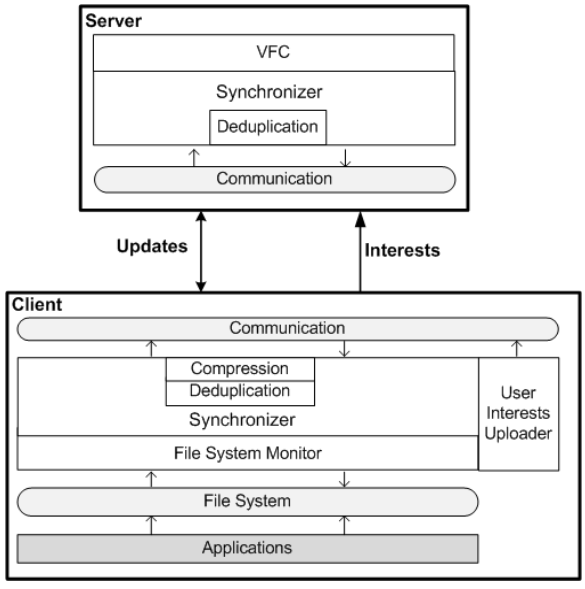

Figure 1: vfcBOX's global overview.

Briefly, Client nodes are composed by the following modules:

- File System Monitor: detects new file updates;
- Synchronizer: manages the versioning information;
- Deduplication: avoids the transfer to the server of already transferred data parts;
- Compression: compresses the transfered data;
- User Interests Uploader: uploads to the server the user interests.

The Server node is mainly composed by the following modules:

- Deduplication: avoids the storage/transfer of already stored/transferred data parts;
- Synchronizer: manages the versioning information;
- VFC: according to users' interests, it ensures critical updates to be immediately propagated to clients and less critical to be postponed.

The vfcBOX consistency model is based on three main concepts: Pivots, Consistency Rings, and Consistency Degrees.

Pivots are special entities within a document which represent the focal point of interest of a particular user. A pivot is used by the vfcBOX to calculate the distances between a user's interest focal point within a file/folder and other entities (e.g. sections, chapters) in that file or in other files.

Consistency rings are formed around pivots. The number of rings may be set by the user. For example, to calculate the distance between a pivot and a file section we just check on which ring the section is on.

Each consistency ring has a consistency degree associated. Consistency degrees vary with the distance so that rings closer to a pivot have a stronger consistency degree; conversely, rings that are far away from a pivot provide weaker consistency degrees.

A consistency degree is directly associated to the number (and rate/frequency) of messages that will be exchanged. If sections/chapters/etc. in a file become closer to a user's pivot, the number of messages sent to the user increases and, vice versa, as file entities become farthest from the pivot, the rate of updates sent will be reduced.

2.2 Consistency Requirements

vfcBOX consistency model supports two granules for users interests specification: **i)** whole file, and **ii)** file parts interests. The first type is related to the interests that a certain user may have over certain files within a folder. The consistency guarantees are thus applied over the whole file. The second type is related to the interests that a certain user may have over parts (which we call semantic zones) of a certain file. Consistency guarantees are stronger when closer to the selected pivot zone, which is manually selected by users.

To accomplish the above mentioned, a file is divided in parts (which are in fact, semantic zones). These parts may be seen as natural parts of a document, for instance, chapters and sections of a Latex document. Given that, each semantic zone is composed by data that is deduplicated; a file is seen as a list of semantic zones in which each zone is composed by a list of references (chunks pointers) to data chunks. The data chunks are stored in a chunk repository.

In order to apply different consistency guarantees over multiple semantic zones of a file, we consider each semantic zone an independent object.

Zone	Sequence (σ)	Time (θ)
Very Important	1 update	0 min.
Important	5 updates	5 min.
Not Important	10 updates	10 min.

Table 1: vfcBOX model: semantic zone limits.

Client nodes contain version vectors with two entries, one entry to its own version and one entry to the server version. Server nodes contain version vectors with N+1 entries, one entry to its version and N entries to N clients (only the clients that share the file are included). Each file has one version vector for the structure of the file (Structure VV) and one for each zone (Zone VV). This allows the identification of modifications on the file structure (e.g. insertion of one zone) or on the content of each semantic zone.

By representing parts of a file as independent objects, vfcBOX is not only capable of enforcing multiple consistency degrees but also of avoiding conflicts, since it enforces the versioning information independently for each semantic zone. As such, modifications to different parts of a file are viewed as updates to different objects and thus not considered as a (false-sharing) conflict.

2.3 Consistency Levels

Each semantic zone update is assigned to a consistency degree, i.e. a set of divergence bounds/limits and a set of clients to whom this update may concern. We define three different consistency degrees (note that the system is prepared to be configured to have more consistency degrees): **i)** *very important data zones*; **ii)** *important data zones*; **iii)** *not important data zones*.

Based on user defined consistency needs, the vfcBOX assigns a consistency degree to each semantic zone; each semantic zone is then inserted in a list of semantic zones. Periodically, the server iterates each list in order to propagate to clients only those updates that are effectively needed so that consistency requirements are fulfilled.

2.4 Consistency Bounding

To bound consistency divergence, a set of limits are imposed. Ensuring the respect of such limits is the task of the vfcBOX server by propagating updates to clients when needed. There are two divergence criterias: Sequence (σ) and Time (θ). These two criteria form a consistency vector.

The Sequence criteria indicates the number of updates that have already been performed over a certain file (each file save is considered to be as an update). Thus, the Sequence limit is defined as the maximum number of unseen updates from a certain semantic zone. The Sequence limits are checked in consequence of either the arrival of an update, or a change in the user pivot. This limit is checked simply by comparing the number of undelivered updates in a certain semantic zone with the maximum sequence limit defined with the sequence divergence limit.

To create these sequence events, we associate a function to a counter signal in the server. This function counts the number of updates that have been performed over a semantic zone and checks, for every consistency zone, if the sequence limit has been exceeded, as follows.

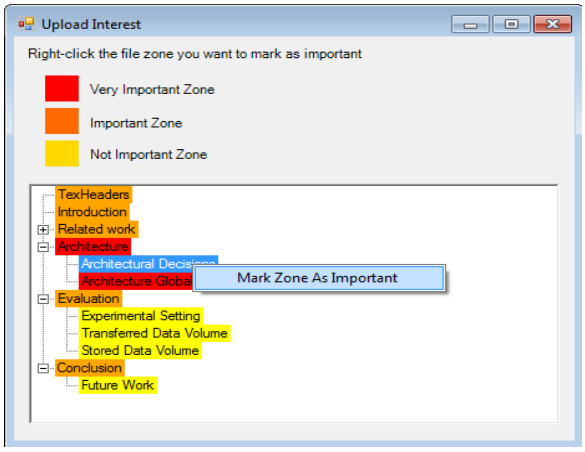

Figure 2: Latex document example.

$$\#Updates[Semantic\_Zone] > Sequence[distance(Semantic\_Zone; Pivot)]$$

The Time limit of a vfcBOX consistency vector defines the maximum time during which no updates are applied to a semantic zone (or its staleness). To enforce the time limits, the vfcBOX server uses a timeout mechanism for each consistency zone.

To create these timeout events, we associate a function to a clock signal in the server, with a period equal to the minimum unit of measurement, i.e. one second. This function counts the elapsed time and checks, for every consistency zone, if the time limit has been exceeded, as follows.

$$Elapsed\_Time[Semantic\_Zone] > Time[distance(Semantic\_Zone; Pivot)]$$

Table 1 presents the vfcBOX default limits of divergence values (obviously, these can be configured). These limits are enforced over each semantic zone. According to a certain level, each zone has assigned some divergence bounds. For instance, a semantic zone that is considered very important, has associated a sequence bound of 1 update and a time bound of 0 minutes. This means that an update to that semantic zone is immediately propagated to the concerning client(s). On the contrary, a semantic zone considered as important, has associated a sequence bound of 5 updates and a time bound of 5 minutes. This means that only after 5 updates or only after a delay of 5 minutes, the semantic zone content is transferred to the concerning client(s).

Figure 2 shows an example of a Latex file, for which several semantic zones have been defined. A darker color shows a zone that is more relevant for the user thus requiring stronger consistency than other (lighter gray) zones. A similar interface is provided for the user to express (and visualize) which files within a folder are more/less relevant and thus require stronger/weaker consistency. In this case, the consistency granule is the whole file.

2.5 Deduplication Process

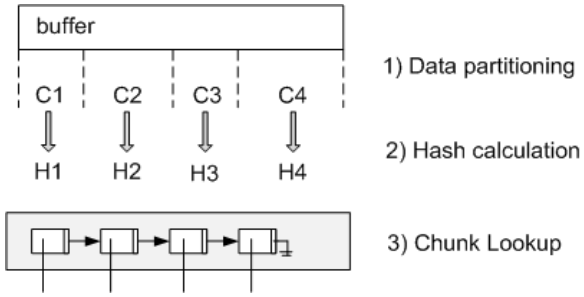

Figure 3: Example of deduplication process.

In this section we describe the deduplication process used to reduce the redundant data transferred over the network. The deduplication process explores the redundant data either from the client to the server side or from the server to the client side. Through comparison of variable-size hashes we explore both cross-file and cross-version redundancy.

In short, the deduplication process is composed by three main steps (see Figure 3): I) **Data partitioning**; II) **Hash Calculation**; III) **Chunk Lookup in chunk repository and references table**.

The first step regards the process of partitioning data that is found to be written. This partitioning is accomplished by using Rabin Fingerprints[13] to calculate the chunk boundaries according to data's contents. It is based on the examination of every 48-byte regions of the file and respective calculation of a rolling hash. The rolling hash is then compared with a pre-defined value. When a match occurs the current region is marked as a chunk boundary.

The second step of the process constitutes the hash calculation (using SHA-1[6]) of each chunk provided by step 1.

The third and last step regards the process of writing data in a compact form. The compact form is constituted by a list of references to chunks. For this, it is required a lookup over the chunks repository and over the references table in order to detect redundant chunks. If the chunk already exists, only a reference is added to the chunk contained by the repository. Otherwise, the new chunk has to be added to the repository, creating for it a new entry that is identified by its hash-value.

The chunk repository stores chunks and provide the possibility of performing lookups, in order to detect already existing chunks. Each entry of the repository contains a unique hash-value of a chunk and a reference to the actual chunk.

The references tables consists on the set of hash values that each site knows to be found at a given site. The references table stores references to chunks that have been sent to a certain site. These references are associated with the synchronizing site identification. This provides the possibility of performing lookups, in order to detect if a certain chunk has already been sent to a certain site.

3. IMPLEMENTATION

vfcBOX has two main modules: upload and download pipelines.

3.1 Upload Pipeline

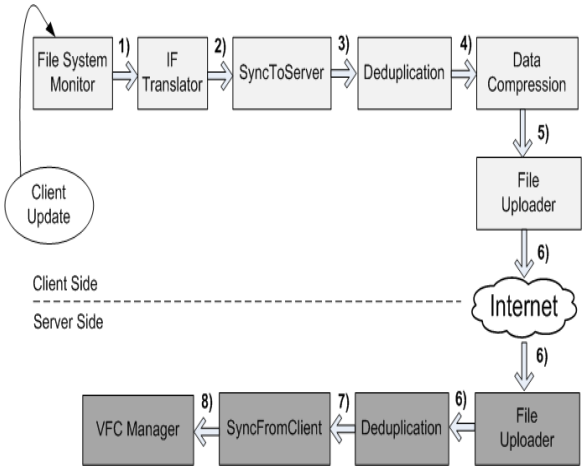

Figure 4: vfcBOX's data upload pipeline. The figure illustrates the modules of the baseline architecture that are involved in the data upload.

The **upload pipeline** (Figure 4) detects new updates and uploads them from the client to the server. There are eight steps as follows.

- **1)** ***File System Monitoring***: The process of uploading a file update starts with an update event, triggered by the file system monitor (when a file saving occurs).
- **2)** ***File Translation to the Intermediate Format***: After the identification of a new update, the update is translated to an Intermediate Format (IF). This allows using the same format and same semantic zone definition for any type of files, since all files are translated to the IF.
- **3)** ***Client Synchronization***: In this phase the new update is synchronized with the server. This step is responsible for managing all the versioning information of updates.
- **4)** ***Client Deduplication***: Before transferring the new update's data to the server, the process of deduplicating redundant data (described in Section 2.5) is triggered, resulting in a hash value for each chunk. Then, the hash value is searched in the client's chunk repository and in the client's references table in order to detect if the concerning chunk has already been sent to the server in previous uploads.

  The chunks repository is responsible for storing each data chunk. The references table is responsible for maintaining information about the data chunks that have already been sent to the server. If the concerning chunk has already been sent to the server, the chunk is replaced by a single reference containing the chunk's hash value. Additionally, the client adds to the references table a reference to each chunk that is sent to the server, in order to find in the future if that chunk has already been synchronized.
- **5)** ***Data Compression***: After the deduplication, the remaining literal chunks (chunks with their associated

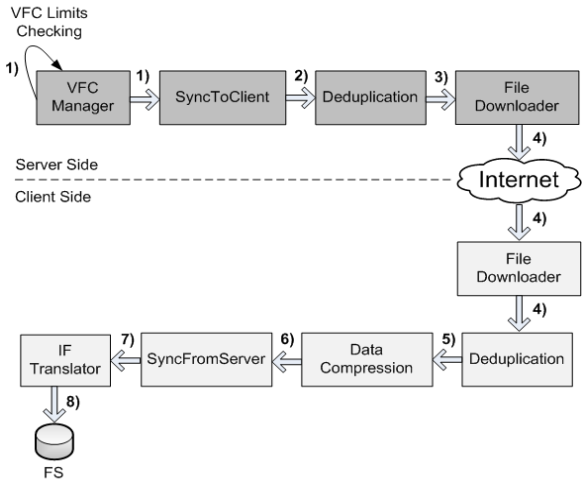

Figure 5: vfcBOX's data download pipeline. The figure illustrates the modules of the baseline architecture that are involved in the data download.

contents) are compressed using the BZip2[3] compression algorithm.

- **6)** ***Data Transfer***: The upload process then continues with the data transfer of the new update.
- **7)** ***Server Deduplication***: On the server side and for each chunk the server receives, it searches the chunks repository in order to avoid the storage of already existing chunks (e.g., one since transferred by another client). Additionally, the server adds to the references table a reference to the received chunk, indicating that the uploading client contains a copy of it.
- **8)** ***Server Synchronization***: Finally, the server updates the versioning information of the incoming update, determining if there were any conflicts. For each received semantic zone and for each client, a consistency level is assigned in accordance to client's interests. This phase allows further synchronization of incoming updates with the rest of the clients.

3.2 Download Pipeline

The **download pipeline** supports the download of new files and new file updates from the server to each of the clients. The download of a file update is initiated on the server side. Thus, this pipeline periodically checks if there are updates to be sent from the server to clients. This synchronization is done according to the consistency requirements of each client; thus, it determines if a certain update will be delayed or immediately propagated to a certain client. Some updates may be dropped altogether as they have been made obsolete by subsequent updates to the same file (or to same chunk), and need neither be further stored nor sent.

The download process (Figure 5) is composed by eight steps as follows.

- **1)** ***vfcBOX Limits Checking***: The download of a file update starts when the divergence bounds associated to a certain file or file semantic zone are exceeded.
- **2)** ***Server Synchronization***: The set of updates that are found to be synchronized with a client are passed to the synchronization module that is responsible for updating the versioning information.
- **3)** ***Server Deduplication***: Before transferring the contents of each update to the concerning client, the server performs the deduplication process. To accomplish this, the server searches its references table in order to find if the current client already contains a certain chunk of data. If a reference of a chunk is found, the chunk is replaced by a reference containing the chunk's hash value.
- **4)** ***Data Transfer***: The download process then continues with the data transfer of the current update.
- **5)** ***Client Deduplication***: When the client receives a literal chunk, it stores the chunk in the chunk repository and adds a reference to it in the references table. When the client receives a non literal chunk, it adds a reference to it in its references table and searches for the chunk's content in the chunk repository.
- **6)** ***Data Decompression***: After the deduplication process, the compressed data chunks are decompressed.
- **7)** ***Client Synchronization***: The new update is synchronized with the client, updating the versioning information.
- **8)** ***File Translation to the File Format***: The client translates the synchronized update to the actual file format.

4. EVALUATION

In this section we evaluate vfcBOX by comparing it to other systems (Dropbox, SVN and LBFS).

We simulate a user performing modifications to shared data. These modifications are propagated to the server, which is then in charge of synchronizing, if required by the consistency zones specified, the new updates with other users.

We use the consistency divergence limits already described in Table 1.

4.1 Workload Description

For the evaluation, the set of files used are MSc thesis samples composed by 10 different Latex files, representing the content of some MSc dissertations of students from IST/Technical University of Lisbon. These samples were composed by the regular chapters of a dissertation, namely *Introduction*, *Related Work*, *Architecture*, *Implementation*, *Evaluation* and *Conclusion*, and several sections in most chapters. The medium size of each Latex file is 200KB. Thus, this workload has a total size of 2000KB.

4.2 Bandwidth Usage Analysis: Real Workload Stress Test

The goal of this test is to measure the effects on bandwidth usage of both vfcBOX consistency model and deduplication. We used thesis samples (described in Section 4.1) and requested real users to perform some minor changes to these file samples, in order to simulate typical editing of the documents. To accomplish this, each user shared his MSc

[3]http://www.bzip.org

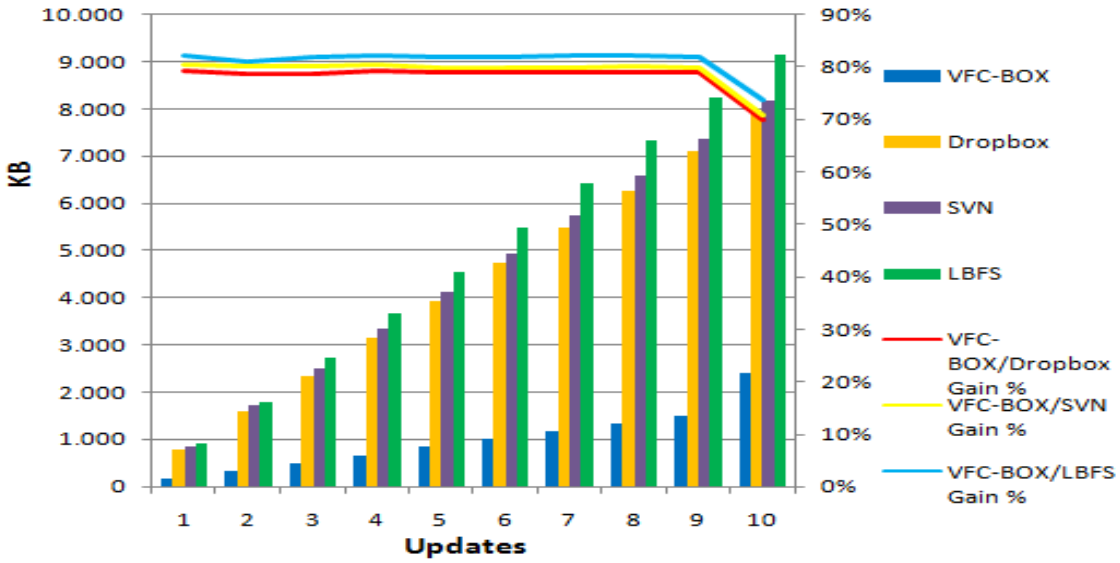

Figure 6: Bandwidth usage (in KB) sum of the stress test illustrated in Figure 7.

thesis with a user that was downloading new updates. This user assigned a special interest over the *Architecture* chapter of each file sample. Then, each user performed small changes (insertion of 1 byte) on each chapter of the sharing file.

Figure 7 illustrates the results of the used bandwidth (in KB) to download each file update. Dropbox, SVN and LBFS make use of more or less the same bandwidth to download each file update, given that for each update the multiple modifications to the content of the file are transmitted.

Analyzing the results of vfcBOX, we may find a huge difference between the 10th update and the others. This happens because the current user has assigned a special interest over the Architecture chapter of each file. Thus, only this chapter and its sections are considered as important zones to vfcBOX server. Therefore, on each update, only the modifications to this specific chapter are transferred, avoiding to transfer updates over the rest of the chapters. In the 10th update, the consistency limits (sequence limits) of all chapters/sections are exceeded and all modifications are transferred to the concerning client.

Bandwidth is saved because delayed updates may be rendered unnecessary to be transferred, for being subsumed by more recent ones when these are effectively transferred at a later time.

In comparison to the other three solutions, vfcBOX is capable of reducing the total amount of used bandwidth up to 82% on each update. The exception is in the 10th update, where all solutions propagate all the modifications.

Figure 6 illustrates the sum of the used bandwidth of the above mentioned results (Figure 7). In comparison to the three systems, the vfcBOX results show a total saving from 70% to 74% in bandwidth resources after 10 updates to 10 files.

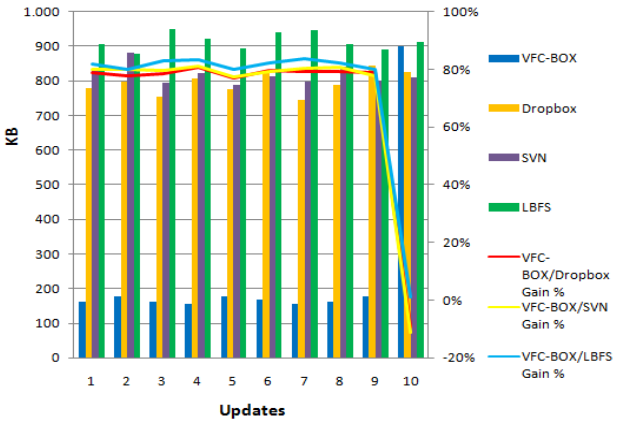

Figure 7: Bandwidth usage (in KB) for downloading 10 file updates of a set of 10 different thesis files.

5. RELATED WORK

There are several types of services that may be provided by cloud computing. Regarding vfcBOX, we focus on cloud computing as a system to provide large-scale storage (Cloud Storage). Many systems such as Dropbox[4], SkyDrive[5], SpiderOak[6], Box.net[7], SOS Online Backup[8] and SugarSync[9] use cloud systems in order to provide high available and reliable storage.

Such a service is offered by software running on a collection of servers, with data from client machines stored at the hard disks of multiple server nodes. Typically, a cloud storage process on a client node transfers (part of) the data available in local storage back and forth to an entry point of the cloud storage service. This entry point makes sure that the data from the client is distributed over other server

[4]https://www.dropbox.com
[5]http://skydrive.live.com/
[6]https://spideroak.com/
[7]http://box.net
[8]http://www.sosonlinebackup.com
[9]https://www.sugarsync.com/

nodes. The cloud storage process keeps local data synchronized with data stored at the cloud storage service: new data generated locally by the user is uploaded to the cloud, data is retrieved from the cloud when local data is lost.

5.1 Amazon S3

Amazon S3 Simple Storage Service[12] is an Amazon's system based on cloud computing to provide storage for the Internet. It provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. Through Amazon S3, developers may achieve highly scalable, reliable, secure, fast and inexpensive infrastructures to store data, without having to be concerned about any internal issues. Amazon S3 is supported by a large number of data centers in the United States and Europe and is expected to offer low data access latency, infinite data durability and 99.99% of availability[12]. Data stored in Amazon S3 is organized in a two level namespace: buckets and object names. Buckets are similar to folders and allow users to organize their data. Object names correspond to objects that are stored into buckets. Regarding the data access protocols, Amazon S3 supports 3 main protocols: SOAP[10], REST[11] and BitTorrent[12].

5.2 DropBox

Dropbox is a commercial backup and file synchronizer[2] system that enables users to share data with others across the Internet. It is designed to achieve high performance regarding the transfer of data between clients and servers. To provide and support storage to large-scale networks, Dropbox actually uses Amazon S3 internally to store files. Each Dropbox client has in his computer a "Dropbox Folder" where he can modify and upload new files/folders. This folder is managed by a background process that is responsible for the correct synchronization of the folder between clients and servers. This application can also be used by several users as a collaboration tool as a user can allow others to access specific folders inside his "Dropbox folder". To accomplish the high performance with respect to data transfer, Dropbox uses delta-enconding techniques to produce a binary diff between new and previous versions of a file. As such, it enables an efficient syncing, only uploading changes made to a file. It also enables a file versioning allowing clients to fetch previous versions. To detect the existence of modified data, Dropbox also uses Compare-by-Hash[11, 15, 5, 3] techniques over folders to find out which folders have been modified. For this, the system exchanges folder hashes in order to find if the contents of a given folder have been modified. With delta-encoding Dropbox is able to reduce the use of bandwidth, by only uploading the changes performed over a file. However, it does not perform deduplication to reduce the amount of stored data. Also, it does not use any kind of technology to specify multiple consistency levels, which could improve even more the efficiency in data transfer. Moreover, Dropbox does not provide any kind of tools to resolve updating conflicts, delegating to clients this task when any concurrent operations to the same files have been performed.

[10]http://www.w3.org/TR/soap/
[11]http://www.ics.uci.edu/ fielding/pubs/dissertation/rest_arch_style.htm
[12]http://www.bittorrent.com

5.3 Microsoft Live SkyDrive

SkyDrive[13] is a file hosting service that allows users to upload files to a cloud storage and then access them from a Web browser. It is based on cloud storage services that allow users to store and share data. It does not provide synchronization functionality between devices, and the primary interface is web browser based. A number of tools exist, however, that make uploading and syncing of local data more convenient. SkyDrive does not support versioning of objects. Individual items can be shared with others through the web.

5.4 LBFS

LBFS[11] is a network file system designed to perform in low-bandwidth networks. The main goal of this system is to avoid the transmission of data that may already be found at the receiver's site. To accomplish this, this system makes use of compare-by-hash techniques in order to improve the bandwidth usage. To deal with the problem of shifting file offsets and overlapping chunks, this system uses Variable-size Block Hashing[11, 5], basing chunk boundaries on file contents.

To make the chunk comparison possible both client and server store chunks in a database indexed by chunks hashes. When reading a file from the server, the client makes a request to the server in order to retrieve the hashes of the chunks to be read. Further, the client compares the received hashes with the already detained, labeling the chunks that were not found as missing. After this, the client requests the missing chunks to the server, receiving by this the missing data. As these operations are all pipelined, downloading a file only incurs in two network round-trips plus the cost of downloading the data.

Summing up, LBFS is a system that efficiently synchronize data, saving resources regarding the use of network bandwidth. Although it is clear the improvement in the transfer protocol, this system has to exchange sets of hash values between client and server, which in case of low redundancy may not compensate the gains and introduce a substantial overhead. Moreover, it does not explore data redundancy to efficiently store data. This system could use the same deduplication techniques to explore this last issue, making this a system that could efficiently transfer and store data. Additionally, LBFS also does not have into account multiple consistency guarantees, which could guarantee a multi-level consistency according to the bandwidth constraints and user needs. This could even improve more the efficiency with respect to data transmission.

5.5 Subversion (SVN)

Subversion (SVN)[7] is a revision control system typically used to synchronize and store multiple versions of source code files. SVN is based on a client-server architecture. It supports disconnected operations and provides to clients tools for handling conflicting updates, since normally there are multiple clients making concurrent changes to the same files.

In order to achieve higher performances with respect to data transfer protocol, SVN tries to reduce the use of network resources through the use of delta-encoding techniques. Through this technique it compares file versions with their

[13]http://skydrive.live.com/

previous versions, detecting *cross-version redundancy*. This redundancy exploitation is not only used to reduce the use of network bandwidth, but also to achieve better performance with respect to data storage. As specified, the delta-encoding technique needs one new version and one old version to encode data, which forces the client to use extra space to store old versions. Furthermore, this method also imposes the limitation that each file is encoded only against one other file, which makes SVN unable to exploit *cross-file redundancy*.

In short, SVN is able to perform efficient data transfer, improving concurrency and providing tools to reconcile conflicting updates.

6. CONCLUSION

The vfcBOX system efficiency comes from the use of both deduplication techniques and the vfcBOX consistency model. Through deduplication, the vfcBOX is able of detecting redundant data between multiple versions of the same file or even between multiple files. With this redundant data detection, network bandwidth may be saved since the redundant data has no longer to be transferred.

vfcBOX also performs a selective scheduling of updates based on its importance, determined through user specification. Therefore, multiple consistency degrees are applied over multiple data sets (files or file parts), which gives the system the ability of postponing certain updates, and even avoiding sending some altogether.

In conclusion, vfcBOX is an efficient file sharing system that uses low bandwidth to synchronize file updates by using several techniques such as data deduplication, data compression and a relaxed consistency model that makes an intelligent schedule of updates according to its importance to each user.

7. REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, and R. Katz. A view of cloud computing. *In Magazine Communications of the ACM*, Volume 53 Issue 4:50–58, 2010.

[2] S. Balasubramaniam and B. Pierce. What is a file synchronizer. *In MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, 1998.

[3] J. Barreto and P. Ferreira. A replicated file system for resource constrained mobile devices. *In Proceedings of IADIS International Conference on Applied Computing*, 2004.

[4] J. Barreto and P. Ferreira. Efficient locally trackable deduplication in replicated systems. *In Middleware'09: Proceedings of the ACM/IFIP/USENIX 10th international conference on Middleware*, 2009.

[5] L. Cox, C. Murray, and B. Noble. Pastiche: Making backup cheap and easy. *In OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 285–298, 2002.

[6] D.E. Eastlake and P.E. Jones. Us secure hash algorithm 1 (sha1). *http://www.ietf.org/rfc/rfc3174.txt?number=3174*, 2001.

[7] Collins-Sussman et al. Version control with subversion. *O'Reilly*, 2004.

[8] K. Morse et al. Interest management in large-scale distributed simulations. *Information and Computer Science, University of California, Irvine*, 1996.

[9] J. J. Hunt, K.-P. Vo, and W. F. Tichy. An empirical study of delta algorithms. *In ICSE '96: Proceedings of the SCM-6 Workshop on System Configuration Management*, pages 49–66, 1996.

[10] N. Mandagere, P. Zhou, M. Smith, and S. Uttamchandani. Demystifying data deduplication. *In Companion '08: Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, 2008.

[11] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. *In SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, Volume 35 Issue 4:174–187, 2001.

[12] M. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon s3 for science grids: a viable solution? *In DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing*, 2008.

[13] M. Rabin. Fingerprinting by random polynomials. *Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University*, 1981.

[14] Y. Saito and M. Shapiro. Optimistic replication. *In Journal ACM Computing Surveys (CSUR)*, Volume 37 Issue 1(1):42–81, 2005.

[15] A. Tridgell and P. Mackerras. The rsync algorithm. *Australian National University*, 1998.

[16] L. Veiga and P. Ferreira. Semantic-chunks: A middleware for ubiquitous cooperative work. *In ARM '05 Proceedings of the 4th workshop on Reflective and Adaptive Middleware Systems*, 2005.