UNIVERSITAT
ROVIRA I VIRGILI

Department of Computer Engineering and Mathematics
Architecture and Telematic Services Research Group

M.Sc. Thesis Dissertation

Benchmarking Personal Cloud Storage Systems

by

Eduard Hernàndez Lledó

Thesis Advisor:
Dr. Marc Sánchez Artigas

Dissertation submitted to the Department of Computer Engineering and Mathematics in partial fulfilment of the requirements of the degree of

Master in Computer Security and Intelligent Systems

September 2014

ABSTRACT

Despite the tremendous success of the so-called Cloud storage revolution, exemplified by Dropbox, Google Drive , OneDrive and the likes, little is known about the architecture and performance of these commercial solutions.

To support true live file synchronization, the systems may make use of numerous techniques to save bandwidth and improve throughput. For instance, these systems only transfer those parts of files that have been modified since the last file sync or achieve high throughput by bundling together several file changes.

We believe that with the current state of the personal Cloud systems, there is an interest to investigate more about the capacity of these systems. The results of this thesis will help to understand a little bit more how the Cloud storage works.

In this thesis, first we define the techniques that Cloud storage solutions can use to achieve a better performance when trying to synchronize files into the Cloud.

Second, we define a test suite to benchmark the different Cloud systems. We describe each test, explaining the goals researched, the metrics studied and the setups used.

Next, we test our benchmark tool with some of the most known commercial Cloud solutions. We show the results obtained and relate them with the techniques used.

Finally we give an overall conclusion about what we learned from the tests and the behavior of the Cloud clients.

ACKNOWLEDGMENTS

First, I would like to thank my parents for giving me the opportunity to study this master. I would also like to thank my girlfriend Cristina and my close friends not only for supporting me during the duration of the master but also for encouraging me to pursue my life objectives.

In second place, I want to thank my thesis advisor, Marc Sánchez Artigas, for all the advice and help he gave me in the completion of this thesis, and also for giving me the opportunity to work with his research group.

From this research group, I would like to give special thanks to Cristian Cotes for all the help he gave me during the most difficult moments. I would also like to mention Sergi and Edgar for the assistance they gave me during the start of this thesis.

Honestly, thank you everyone.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

Gdrive Google Drive

Cdrive Amazon Cloud Drive

Odrive Microsoft OneDrive

CPU Central processing unit

PC Personal Computer

MB Megabyte

KB kilobyte

RAM Random access memory

1

INTRODUCTION

In this chapter we are going to make a brief introduction on the current state of the Cloud services. We will also explain the motivation under this thesis and we will conclude with the thesis structure.

In the last years, the number of intelligent mobile devices has grown significantly. This caused a change in the personal storage necessities. While before we just needed a single computer in order to store all our files, now, we want to access to all our data from any posible device, wether a mobile phone, a tablet, a friend's computer, etc.

To this fact we also have to add the boom produced in collaborative editing projects. A clear example is *GitHub*[6], a company that provides collaborative code storage services to build software. *GitHub* accumulated 3 million users in the first five years of the company life (2008-2013)[8] and has just grown to 6.8 million users in the last year[9].

If we look at companies that offer a Cloud storage services, *Dropbox*[1], for instance, has over 300 million users that storage more than 1000 million files every day[10]. *Dropbox* plataform also has more than 300.000 applications, with *Facebook*[11], *Yahoo*[12] or *Vimeo*[13] integration among others.

With all these big numbers, we can state that that Cloud storage has a true impact in the life of millions of people and it is also a hot research topic[14][15].

It is not rare that big companies like *Google* or *Microsoft* have their own personal Cloud storage solution. All these commercial companies, tend to present a free mode with a limited storage space, and offer an space increment in exchange of a monthly fee.

Most Cloud storage services are cross-platform, allowing the user to access the data from multiple devices. Table 1 shows the cross-platform support offered by the Cloud systems we are going

Table 1: Cloud Storage cross-platform comparison

Provider	Windows	Linux	Mac	Android	BlackBerry	iOs
Box	Yes	No	Yes	Yes	Yes	Yes
Dropbox	Yes	Yes	Yes	Yes	Yes	Yes
Google Drive	Yes	No	Yes	Yes	No	Yes
Amazon Cloud Drive	Yes	No	Yes	Yes	No	Yes
Microsoft OneDrive	Yes	No	Yes	Yes	No	Yes

Table 2: Cloud Storage service free-version comparison

Provider	Storage size	Max. file size	Traffic limit
Box	10 GB	250 MB	10 GB/month
Dropbox	2 to 18 GB	10 GB	20 GB/day
Google Drive	15 GB	1 TB	None
Amazon Cloud Drive	5 GB	2 GB	Amazon S3 limits
Microsoft OneDrive	15 GB	2 GB	None

to analyze in this thesis. Although *Google Drive*[3] and *Microsoft OneDrive*[5] do not directly support *BlackBerry*, both have 3rd party apps to access your personal data.

As we can see in table 2, each company offers a different type of storage service, while some companies like *Dropbox* and *Google Drive* have a free version that can handle heavier workloads (more storage, bigger file size and more bandwidth), others like *Box*[2] limits you with a very little file size, which in some cases, for instance, can't be enough to upload a video file.

But the differences do not end here, storing the data into the Cloud is also a technique that every company implements in a different way. Therefore, there is a need to compare how the different Clouds store the data into the server, specifically, we are going to compare how the different desktop clients synchronize the data from the PC to the Cloud.

1.1 MOTIVATION

Despite the success showed by Cloud storage solutions, as we can see with companies like *Dropbox* or *Box*, little is known about how these services work.

Moreover, open source alternatives, where data con be accessed by anyone, like *ownCloud*, *SparkleShare* or *Syncany* can not be considered as a real alternative to commercial solutions, as these services lack some crucial requirements needed for an extended use, for

instance, scalability. Luckily, some open source projects have already addressed this scalability problem, as we can see with *StackSync*[24].

For these reasons, we believe it is necessary to depth study these commercial solutions, comparing how each of them perform in a bunch of predefined tests, in order to see which techniques they apply as well as the impact these techniques have in the efficiency and performance of a file synchronization with the Cloud.

Some studies performed on the subject only show the amount of data transmitted between the desktop client and the Cloud server in concrete scenarios. In this thesis, we will increase the amount of tests performed through the different Cloud services and we will also measure more metrics, for instance, we will take into consideration the processor consumption by the desktop Cloud clients and in some cases we will analyze the time it takes to start the synchronization with the Cloud.

Definitely, we want to create a benchmark tool that can be used to compare all the different Cloud services available in the market. To reach our goal, we will define a set of tests we believe are necessary to understand the functioning of the Cloud desktop clients. These tests will consist in different scenarios, some of them will be real, recreating workloads a Cloud storage can be put into, and some will be simulated, in order to understand how the client reacts to specific inputs.

Once the benchmark suite is defined, we will test our tool with some of the well known Cloud storage commercial solutions. Giving the results we obtain and trying our best to explain the behavior of each Cloud. We will also link the results obtained from the tests with the implementation of the different Cloud techniques that can be used to improve efficiency.

1.2 THESIS STRUCTURE

The thesis structure will be as follows:

In chapter 2 we will explain the background information needed to understand the concepts and definitions we will see through the thesis. We will enumerate the Cloud techniques available to improve efficiency and we will take a look into the different related work on the subject.

In chapter 3, the design of the benchmarking tool is explained. We will define the test suite we are going to create, explaining the goal

of each test and the setup used. We will also enumerate the metrics that we are going to take into account for our tests.

Chapter 4 will consist in the test of our benchmark tool. We will run this tool into a set of Cloud clients and analyze the results obtained in them. We will explain each of the results and relate them with the techniques applied.

To finish the thesis, chapter 5 will include the conclusions, we will take a look at the goals obtained and present some future work that can be done in relation with the thesis subject.

2

BACKGROUND

This chapter will include the different techniques a Cloud system can apply in order the improve the efficiency to synchronize data between the Cloud client and the Cloud server. The chapter will also explain the related work made on the subject.

2.1 CLOUD SERVICE TECHNIQUES

In this sections we will describe the different techniques available to the Cloud systems. It is necessary to note that not all the Cloud systems include each of these techniques, and every client can have a different way to implement them.

2.1.1 *Chunking*

The chunking technique consist in dividing the files (or objects) into multiple chunks. With this technique, the Cloud system does not have to deal with the original files but with the chunks created for each object. The chunks are objects up to 10 MB (depending on the chunking implementation), so when this technique is applied, the Cloud does not have to work with files that can be gigabytes of data but can work with smaller objects, improving efficiency.

Working with chunks is better in numerous cases, for instance, if we had to recovery from a synchronization failure, if the system applies chunking, we only need to sent the chunk lost, that means a maximum of 10 MB, otherwise, with no chunking, when we lose the synchronization connection while uploading a bigger file, all the file needs to be uploaded again. Moreover, working with chunks is a key point in the implementation of the rest of techniques we are going to explain next, as it drastically simplifies them implementation

There are two types of chunking, the first and most simple one is called *static chunking*, with this type of chunking, the file original file is divided into different chunks of a fixed size.

The second type, is called *content based chunking* and the most efficient one from this type is called *TwoThresholdTwoDivisors* or *TTTD*.

Brielfy, in *TTTD*, there are 2 thresholds, one is the *maxThreshold* and the other the *minThreshold*, these thresholds consists in the maximum and minimum size of the chunks we want. Apart from the thresholds, we also have two divisors, these divisors will help us to find the optimum points to apply chunking in our file. More of this chunking technique can be found in *A framework for analyzing and improving content-based chunking algorithims*[27].

In *Benchmarking Personal Cloud Storage*[20], *Idilio Drago* made a test to investigate what Cloud systems perform a bundling technique. His results showed that only *Amazon Cloud Drive*[4] does not perform a chunking technique, and also stated that *Google Drive* and *Dropbox* implement an static chunking of 8 and 4 MB respectively.

Upsides

As said before, applying chunking is favorable in different aspects. First, it provides a better failure recovery, because it allows a faster file recovery as we just need to continue the synchronization from the chunk lost without needing to upload the whole file.

For example, in the case we wanted to upload to the Cloud a file of various gigabytes and we lose the connection while synchronizing the last bytes of the file, without chunking we will need to upload the whole file again while applying chunking, we will only need to upload the lasts chunks (fer megabytes), saving lot of data transmission and lot of time.

As we will see next, applying chunking will also help the implementation of the rest of techniques, for instance, when a change is made in a file, if we have chunking implemented, we would be able to upload just the chunk modified.

Downsides

Applying chunking has little downsides. Probably if the size of the files uploaded is small (few kilobytes), chunking will be unnecessary.

With small files the synchronization with the Cloud is faster, and therefore, the risk of failure is lowered. And in case some failure happens, re-uploading the whole file should not be a problem.

2.1.2 *Bundling*

When a batch of files needs to be transferred, files could be bundled and pipelined so that both transmission latency and control overhead impact are reduced.

With this technique, if we want to synchronize with the Cloud more than one file at once, we do not need to open a connection with the server for each file, there is just the need to open one connection for all the files that need to be sent. Thus, applying bundling we can reduce the number of TCP/SSL connections needed to transmit a file.

Bundling can be implemented in some different ways. For instance, the Cloud services could reuse the TCP connections implementing a full bundling technique or a sequential technique.

In the sequential technique, one TCP connection is stablized with the server, next, files are uploaded one by one, waiting for application layer acknowledgments between each file upload. This method only needs one connection but can only send one file at the same time.

Full bundling technique not only allows to send the data with one TCP connection but also permits sending multiple files at the same time.

In *Benchmarking Personal Cloud Storage*[20], *Idilio Drago* stated that only *Dropbox* uses a full bundling technique. On the other hand, *Google Drive* and *Cloud Drive* do not implement bundling which causes them to open lots of TCP connections when sending more than one file. In some cases, considering management, *Cloud Drive* opens 3 TCP/SSL control connections per file operation.

Upsides

Bundling is really beneficial when we need to synchronize more than one file at once, for instance, if a user places a bunch of files into the Cloud synchronization folder, the client will detect it needs to upload more than one file and, using the bundling technique, the client will open only one connection with the Cloud server to synchronize all the files.

Downsides

Applying a bundling technique can produce a delayed startup time, as the client needs more time to prepare the data before the transmission. It could also lead to a more processor consumption as preparing the data has a computational cost.

2.1.3 *Compression*

Data compression, involves encoding information using fewer bits than the original representation. Compression can be either lossy or lossless.

Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression.

Lossy compression reduces bits by identifying unnecessary information and removing it. The process of reducing the size of a data file is popularly referred to as data compression, although its formal name is source coding (coding done at the source of the data before it is stored or transmitted).

Compression is useful because it helps reduce resource usage, such as data storage space or transmission capacity. Because compressed data must be decompressed to use, this extra processing imposes computational or other costs through decompression.[19]

Upsides

As expected, using compression will be favorable in the cases where the data format is subject to compression. Compressing a file will lead into less amount of data to be transmitted and therefore, the time needed to complete a synchronization with the Cloud server will also be smaller.

Downsides

On the other hand, some file formats are not subject to compression. For instance, files that do not include redundancy or truly random files (files with a sequence of random bits) and files files that are already compressed are not subject to compression.

If the compression implementation does not distinct if a file can be compressed or not, the Cloud client could try to compress an incompressible file, then, applying the compression algorithm will have no effect in size reduction but will induce some CPU consumption.

Some examples like *JPEG*, *GIF*, *MP3* or raw random binary files are not subject to compression.

2.1.4 *Client Side Deduplication*

The deduplication technique is used to avoid uploading to the Cloud a replica of an existing file. Deduplication is still more useful when combined with the chunking technique.

For example, imagine we just apply deduplication, if we have a full replica of a file, the system will use the deduplication technique and there will be no need to upload more data. However, if we change some part of the file, even if it is just one byte, the system will detect it is not a replica, and will have to upload the whole file again.

Combining deduplication with chunking, the Cloud system will not check if it has a replica of the file in the server but, will check if it has replicas from the chunks of the file. This check is performed by comparing the hash[36] of the chunks.

For example, following the example we gave before, now we have the file divided in 10 chunks, once we performed a modification of the file, the system would detect the exact chunk where the changes where performed and will only need to upload that specific chunk.

Upsides

Applying a deduplication technique clearly improves the synchronization when working with file or chunk replicas. For instance, the Cloud system can save lot of storage when applying this technique as it only needs to maintain the original file.

If we look at the transmission costs, when applying deduplication the system only needs to send the control data, which only consists in a few kilobytes, completely ignoring the data of the file, as it does not have to be uploaded.

When combined with the chunking technique, deduplication can be used to detect modifications in the files, allowing the Cloud to synchronize only the modified chunk.

Downsides

The reality is that no one uses Cloud storage to save replicas of the same file, therefore, if the deduplication technique is not combined with the chunking technique, the Cloud would not make much use of this technique.

There is a computational cost using this technique, as the client needs to check with the Cloud server the different chunks it has stored and check for hash compatibles.

2.1.5 *Delta Encoding*

Delta encoding is an special technique used to synchronize chunks modifications with the Cloud. It is used in combination with the chunk technique and not only detects the exact changes produced in the chunk but also permits sending only these changes instead of the whole chunk.

First, applying deduplication we were able to track the chunks that were modified, combining with delta deduplication, we can detect those changes and upload to the Cloud only the modified part of the chunk, reducing even more the data traffic.

Dropbox, for instance, implements delta encoding using the *librsync*[7] library.

This library, in order to uptade the versions of the Cloud server with the new ones coming form the Cloud client , uses a very simple commands (COPY and ADD) in order to reconstruct the new version of the chunk. So, when a chunk modification take place, the server will receive a list of instructions to perform.

If the server receives a COPY command, it means it has to copy a part of the chunk it already has in the server. On the other hand, if the server receives an ADD command followed by the chunk part, the system will add a new chunk part, coming from the client, to the new chunk structure.

In figure 1 we have an example of how this method works. In purple we have the parts constructed with the COPY command while with the red color we can see the modified parts the client sent with the ADD command.

So in this example, the server first received a COPY command, which means he can reconstruct that chunk part with a part it already has stored. Next, the server receives an ADD command with the new chunk part that needs to be added. Later another COPY command followed by an ADD command, and to finish, the same process happened again. In the whole chunk reconstruction, only the new chunk parts needed to be sent by the client which reduced the overall traffic communication.

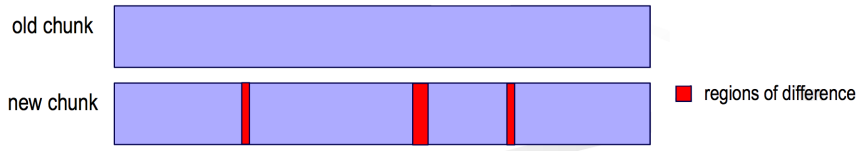

Figure 1: Delta encoding example

Upsides

Delta deduplication is the ultimate technique in order to reduce data transmission when synchronizing different file versions with the Cloud. When we have file modifications, we are able to synchronize the new versions just by sending the real data modifications, rather than sending the whole file or the whole chunk again.

Downsides

With the way delta encoding is implemented, some calculations need to be made in order to detect the exact changes a file suffered. Therefore, the downside of this technique if the processor consumption that these calculations have.

2.2 RELATED WORK

The performance evaluation of Cloud services is a current hot topic as can be indicated by the several papers recently appeared. Some work have focused on measuring distinct aspects of the Cloud service such the nature of the datacenter traffic[28], the service variability[29] or the computing performance[30].

The authors of *CloudCmp*[31], present a systematic performance and cost comparator of Cloud providers. *CloudCmp* measures the elastic computing, storage and networking services offered by different Cloud commercial services. The results show the performance delivered to costumer applications.

In a similar way, the authors of [32] compare *Dropbox*, *Mozy*, *Carbonite* and *CrashPlan* backup services. However, their analysis are kind of lightweight, and more work is needed to characterize these services with enough rigor.

Some research and measurements[33] has been made in order to determine if Cloud storage is suitable for scientific Grids or not. As we can find in [34], a performance analysis of the *Amazon Web Services* has been made, but it is not related with Cloud storage.

The authors of [35], tested the performance of *Microsoft Azure*, including storage measurements.

In [23], the authors present a measurement study on the behavior of the Cloud systems. They found these systems have high variability in transfer performance depending on the geographic location, the type of traffic, namely inbound or outbound, the file size, and the hour of the day. Authors also examined the Cloud failure patterns and found that these services are reliable, but susceptible to unpredictable changes in their quality of service.

Finally, the authors of [25], first presented an extensive measurement of the *Dropbox* service, analyzing the traffic transmitted by users as well as the architecture of the service. Later, in [20], the authors produce a serie of benchmarks to compare different Cloud services.

3

DESIGN A BENCHMARK TOOL FOR PERSONAL CLOUDS

In this section we are going to present our benchmarking tool. First, the current state problem is defined, later we define the design used to create the benchmark. To finish, we enumerate the test suite our benchmark tool is going to incorporate.

3.1 PROBLEM DESCRIPTION

As we have seen through all the *Related Work* section 2.2, there is some research in the subject of Cloud performance. These researches try to understand how the Cloud systems works and what are the techniques available to improve the system efficiency.

However, we lack any tools to perform a complete benchmark for the Clouds systems. Most of the tests already performed are specific in only one subject, and the ones that take an overall look, do only measure one metric or are not deep enough.

The most concern in this subject is about the total data used in the synchronization between the Cloud client and the Cloud server, but there are really other metrics that we should take into account. For instance, the processor consumption is an important metrics to understand if the Cloud clients can handle heavier workloads.

In addition, there are no studies done related with file modifications. Few researches try to identify if the Clouds perform deduplication or delta encoding but do not give any idea of how the Cloud system will perform under a constant modification environment.

Overall, we do not have a enough information on the subject to compare and benchmark the different Cloud systems. Therefore, there is a need to create a benchmark tool that let us not only study the different clients but also compare them to identify lack on performance.

3.2 DESIGN

Our benchmarking tool consists in a test suite that puts the Cloud client under different environments and is able to track the processor consumption or the data transmitted among others.

Specifically, our code generates the files needed in each benchmark and as each test is selected, the files needed are loaded into the synchronization folder of the cloud client.

Oracle VM VirtualBox

In order to isolate the CPU costs of launching the tests, processing the files and collecting information from the real CPU consumption of the benchmark, our tool uses a virtual machine, specifically we are going to use *Virtual Box*[38] version 4.1.18.

Isolating the tests is a good idea because this will let us know exactly what happened during the tests, as our benchmarking tool will not interfere with the Cloud syncronization.

In order to pass to the files to the virtual machine, we are going to use an *ftp*(file transmission protocol) connection.

Processor consumption

One of the metrics we believe it is important to measure is the CPU consumption. Performing some of the techniques explained in section 2.1 have a processor cost, therefore, we believe its important to know if applying these techniques can have a negative impact on the overall performance of the system.

Moreover, in the tests where no efficient techniques need to be applied, it is also important to measure the processor consumption, because each Cloud is implemented in a different way and consume different resources.

In our benchmarking tool, we are going to measure the processor consumption in the tests. In order to do it, we are going to take advantage of using a *Virtual Box* environment, as it already has some built-in tools to track this metric.

Specifically, we will be using *VBoxManage* metrics, which will retrieve us a list of the different metric values obtained during the test.

We want to remark that, in order to have a better idea of the processor consumption, the CPU metric will start 5 seconds before the file is added into the synchronization folder.

Traffic consumption

One of the most important metrics of a Cloud storage system is the amount of data transmitted it takes to complete a file synchronization. Therefore, it is a must to include this metric in our benchmark.

In order to measure the traffic from the virtual machine, where the Cloud client is installed and sends the data to the Cloud server, we are going to implement the same system as *Idilio* did in [20]. This system captures all the packets send by the Cloud application and stores them in a *pcap*(packet capture) format.

Once we have that data stored in that format, we will process it to generate graphics to show the total traffic consumption of the test, overhead tables or traffic cumulative graphics, in which we can see the exact evolution of the data transmitted during the benchmark.

Start up Time

The benchmark will also take a look at the start up time. This time is the amount of seconds the client needs to start the synchronization with the Cloud server once a new file (or a modified one) has been added into the synchronization folder.

With the data obtained from the transmission (previous section), we can calculate how much time past between the file insertion and the first byte transmitted.

This metric is super important in the cases where the Cloud client needs to perform some actions before sending the data, for instance, applying a bundling technique. This metric will help us determine if performing the mentioned Cloud techniques has an impact in the time it takes to start the transmission.

Total time and RAM consumption

Although these metrics will not be analyzed in the following chapter, our benchmarking tool is also able to monitor the Random access memory consumed by the server as well as the total time it takes to perform a test.

As seen in papers like [20], an important factor when comparing the total time needed to complete a synchronization with the Cloud is the data center location. Therefore, including this metric without being able to test multiple locations will not be fair.

3.3 TEST SUITE

In this part of the thesis we are going to define the test suite that our benchmarking tool includes. We are going to present the 6 tests studied in this thesis.

First we are going to define 3 different cases where the files were artificially created in order to create an specific scenario. In this 4 cases, the files are going to contain random binary data in order to negate the effect of data compression.

The next 2 tests represent real scenarios that a Cloud system can encounter. These workloads are created with real data, therefore, the files are subject to compression.

The last test will also be an artificial scenario, with random binary data files.

3.3.1 *File size benchmark*

In this first test we are going to analyze how Clouds handle a single file synchronization depending on the file size. It is a simple yet important test as it will helps us understand how the Cloud works in the simplest case possible.

The goal of this experiment is to test how the size of the file affects the Cloud client. Specifically we are going to look at the traffic each client generates to synchronize a single file.

What we want to know, is the overhead of traffic data generated when uploading a file, so for instance, if we want to synchronize a file of 1 KB and we need to sent 1.4 KB of data to the cloud, that 0.4 KB of extra information will be considered overhead.

The less the overhead, the more efficient the Cloud client is, as it needs less data to synchronize the same file with the cloud.

Table 3: Different file sizes tested

1 KB	10 KB	100 KB	1 MB	10 MB	100 MB

Setup

To test this environment, we are going to use a setup presentend in [21].

The setup of the test simply consists in sending one file to the Cloud at a time. Each time we sent a file, we are going to repeat the experiment but increasing the file size by an order of magnitude. The tests are going to be repeated 10 times in order to verify the results obtained. Table 3 shows the different file sizes we are going to test.

So our benchmark tool will start by creating one file of 1KB of random binary data, next, the file is sent into the synchronization folder of the virtual machine, the system waits until the synchronization is completed, then cleans the folder and repeats the operation again until it is done 10 times. Once that happens, the benchmark switches to the next size test, 10 KB, and repeats the operations until the whole test is completed.

For this benchmark, we are going to analyze the total amount of data transmitted to the Cloud. Once we have this data, we will know which Cloud client generates more extra traffic, allowing us to create an overhead table with the ratio of data transmitted depending on the original size.

As stated before, once we know how the cloud handles single files operations, we can advance to more complicated tests.

3.3.2 *Bundling benchmark*

This benchmark aims to understand how the Clouds work when different files need to be synchronized at the same time. As we studied before, some Clouds will implement a bundling technique reducing the amount of connections needed to upload the files to de Cloud server.

This benchmark will test how the bundling technique works in terms of CPU cost and start up time.

In this test, will not analyze the traffic data consumption, as the performance increase in bundling was already tested in *Benchmarking Personal Cloud Storage*[20]. The authors saw that implementing

a bundling technique reduces the total amount of connections we have to create with the Cloud server to send the files. Having to create less connections means less data needs to be send to the cloud. It also means a reduced total time to complete the synchronization, as the time lost to create new connections is deleted.

The test intention is to determine if the total amount of files the Cloud client has to synchronize has a resource impact (CPU and time). In other words, is the same synchronizing a single file of 1 MB and synchronizing 10 files of 100 KB? The total amount of data we need to synchronize is, in fact, the same, but as the data is separated in different files, we do not know how the Cloud will act.

Setup

To answer this question we are going to use a setup presented in *Benchmarking Personal Cloud Storage*[20].

In this benchmark, we will work with 4 different sets as indicated in table 4. We will repeat each test 10 times to guaranty the results obtained.

The file size utilized in the first set does not match the total data size of the other 3 tests, but, we will use set 1 and 2 to test the CPU difference when sending a single file with different size.

The other 3 tests contain the same amount of data (1 MB) divided in a different amount of files. Tracking the CPU and calculating the start up time of each set will give us a good idea oh how the bundling technique works.

Table 4: Experiment setup

Set number	Number of files to be sent	File size
1	1	100 kB
2	1	1 MB
3	10	100 kB
4	100	10 kB

So, when starting the benchmark, our tool will first create as single file of 100 KB. This file will be sent through ftp to the synchronization folder in the virtual machine, the system will wait until the transmission is completed, and will repeat the test 9 times more.

This process will be repeated for each of the sets, sending 1, 10 and 100 files of 1 MB, 100 KB and 10 KB respectively.

Once the test is completed we would be able to determine if bundling has a huge impact on the processor consumption. We will also determine if grouping the files in a bundle has a time cost.

3.3.3 *Deduplication and delta encoding benchmark*

In order to present the benefits of the deduplication as well as delta encoding, this test was created.

The goal of this test is to check if the different Cloud clients implement deduplication or delta encoding. We believe that both of these techniques are crucial in a Cloud storage environment.

The amount of data and time we can save when implementing them is really notorious. Some researches in this topic have already been done, and in *Benchmarking Personal Cloud Storage*[20], results showed that *Dropbox* is the only system implementing a full deduplication and delta encoding technique.

Delta encoding could be tested in three different ways: modifying the beginning of the file, the middle or just appending new data at the end. The authors of [20] shown the results of modifying a file in the middle and appending new data. To give new information, we are going to test delta encoding when the modifications take place at the beginning of the file.

Setup

To perform this setup we will work with files of 10 MB, this file size is motivated due to being 8 MB the maximum chunk size of *Google Drive*. We want the Clouds to work with atleast 2 chunks to perform the deduplication techniques.

First, we will create the files of the test. A file of 10 MB, a second file with the same data but a different name, and a third file with the exact same data but incorporating a new random byte at the beginning.

If the Cloud incorporates deduplication, it will detect that the chunks of the 2 first files are the same, and therefore, it will not upload the replica, as the Cloud already has the file.

If the Cloud incorporates delta encoding, the system will detect the exact bytes changed in the chunk (the first byte of the file) and the system will only need to upload the new byte.

We will synchronize the first file, wait until the upload its done, include the second file in the synchronization folder, wait again and repeat with the third file.

When the test ends, we will be able to generate a graphic with all the data sent in each set, allowing us to clearly visualize which systems performed deduplication or delta encoding.

With the Clouds that performed any of these techniques, we will create an start up table. We will do this to compare the time needed to start the synchronization when we upload the whole file with the time when we just upload the changes or do not upload anything at all. We believe that in the cases we do not have to sent all the file again, the start up time should be smaller.

3.3.4 *Wikipedia file benchmark*

As we stated at the beginning of this thesis, one hot topic is collaborative editing, although Cloud storage services serve the purpose of simply storing files, it is known that people also uses them as a way to share files or projects with others. Most of them have in-built options to directly share files among others. This is why we thought that performing a benchmark for this type of scenario was a must.

The goal of this scenario will be to analyze how Cloud storage handle the edition of a file simulating a real trace of a *Wikipedia*[37] page.

This scenario evaluates the behavior of the Cloud system when different users are continuously modifying a file. Although this benchmark will only be performed by one client, the Cloud server will still need to process all the updates performed.

We consider this test as a light-moderate workload, as the average of time between file modifications is around 5 minutes.

Setup

In order to reproduce this environment we are going to study how a *Wikipedia* page evolves during a period of time. In concrete the benchmark will focus on *Aleksandr Solzhenitsyn* page[39]. We will reproduce the changes that took place in a 4 hour interval, taking account of the total data transmitted as well as the CPU consumption.

To do this, we have a folder with all the different versions of the *Wikipedia* page and a trace indicating when each version was changed in the web. We will be replacing the old version with the new one

exactly as it happened in the reality.

So the test will begin sending the first version to the virtual machine. The system will read the trace file and will look for the time the next version was uploaded. The system will not perform any action until then. Once the the time of the new version come, the system will replace the old version with the new one, repeating the process through the 4 hours of the test.

During the 4 hours of test, there will be 50 changes in the *Wikipedia* page (therefore 50 synchronizations with the Cloud) and the total size of the file will vary between 140 and 150 KB depending on the type of change performed (deleting or adding new information).

With the data collected, first we will analyze the traffic consumption, plotting the total data each Cloud client sent to synchronize the file through the 4 hours of the test.

We will also take a look at the CPU consumption. We want to know if putting the Clouds into a light-moderate workload has a huge impact on the CPU or not.

3.3.5 *Log file benchmark*

This benchmark presents a similar scenario to the previous test, in this test, however, we are going to take a look at the synchronization of a log file, specifically a web server log file.

A server log file has some peculiarities that differ with the previous experiment. The first one is that the changes come in append mode, which means that all the changes are made at the end of the file, this is important because appending has less computational cost than doing modifications at the middle of a file.

The second one is the time between modifications. In a *Wikipedia*, file we stated that the average time between changes was 5 minutes. Here the changes are more frequent, with an average time of less than 10 seconds between modifications.

This scenario is interesting because we have a mix of inputs. As explained before, in one hand we have a heavier workload because the changes are more frequent, but on the other hand we have a less costly CPU operation (append).

It is also interesting to notice that with this test the file of the size will be increasing some bytes each change, in contrast with the "constant" size we got in the *Wikipedia* test.

Setup

To reproduce this test we are going to have an extract of web server log. We will run a program that will read through this log file and extract the date stamp of each logged event as well as the data saved. Next, we will recreate all the log events with the exact time difference they happened in the reality.

The test will be as follows. First, the system will read the first line of the log file, which includes the time stamp indicating when the change was performed as well as the connection received (we analyze a web server log file). Next, the system will create a new file including this line. The file will be sent to the virtual machine to start the synchronization with the cloud.

The system will continue by reading the next line and extracting the time stamp. This will let us know when the next change needs to be performed. The new line will be appended to the file created and when the system detects the change has to be made, the new file will replace the one the virtual machine has. These actions will be performed until the end of the test.

To concrete, this test will be 4 hours long, going from an original 129 bytes of size to a 127 KB size at the end of the test.

From this test we will extract the data transmission. First we will create a graphic with the total data consumed by each Cloud and then we will show the accumulated traffic consumption plotted over the time line of the test.

3.3.6 *Synthetic benchmark*

This benchmark presents a synthetic scenario where a file is modified under a heavy workload.

Before, we tested some real scenarios, but in both cases we consider the workload was not heavy enough. For this reason, in this scenario, the changes performed to a file will be done under 10 seconds intervals, and the modifications will not be in append mode, instead, the system would be able to perform the change in any place of the file.

This test will really put the Cloud client into a heavy workload and would be useful to see if any of the clients has problems dealing with heavy workloads.

Setup

In order to prepare this benchmark, we are going to use the setup presented in [26]. In this setup, an starting file of 10 MB, suffers modifications in random intervals from 0 to 10 seconds. These modifications incorporate 0 to 10 bytes to the file in a random position of the file. There is a 50% change to perform a modification and the test last for an hour.

In order to simulate the most real scenario possible, we are going to take a look at paper [21]. In this paper, authors studied how file modifications were performed in different operating systems. For instance, they calculated the probability of modifying a file at the beginning, at the middle, at the end, at the beginning and middle, beginning and end, etc.

For this experiment, we used the modifications probabilities of a MacOS system.

In this test, the system will start by creating the random file of 10 MB and sending it to the virtual machine to start the synchronization. Once its done the system will wait a random time from 0 to 10 seconds, and later, with a probability of 0.5 will decide if an action is done or not. If it has to perform an action, the system will then calculate how many bytes are going to be added, again, this will be a random number from 0 to 10.

The position where these bytes are going to be added is selected from the probabilities mentioned before, in case the modification needs to be performed in two or three places, for example at the beginning and at the end, the total bytes modified are equally distributed in each modification place. All this process is repeated until the test hour is reached.

It is important to notice, that the benchmarking tool has the ability to create a trace with the changes, in case we want to benchmark the exact test (same modifications) for different Cloud clients.

With the test completed, we are going to analyze the traffic consumed as well as the CPU consumption. We want to know if working under heavier workloads really makes an impact in the processor consumption.

3.4 DESIGN CLARIFICATIONS

With the creation of our benchmarking tool, we are able to perform the tests in any Cloud storage provider. Although we proposed 6 different tests to benchmark the Clouds, our tool is flexible enough to incorporate new tests to the benchmark.

It is also important to notice that although all the metrics can be obtained in the tests performed, as some of the cases were already studied in other researches, we avoided writing the same information again.

4

BENCHMARK

In this chapter we are going to use our benchmark tool. First we are going to define the environment in which the tests were performed. Next the results of the test will be presented and discussed. To conclude with the chapter, we are going to explain the conclusions learned from the tests.

4.1 GENERAL SETUP

The computers used in our benchmarks are located at the lab 210 of the *Universitat Rovira i Virgili*, Tarragona, Spain. In those computers, the host system (system where we launch the tests) have a 32-bit Debian 7.4 Gnome Version 3.4.2 with 7.8 GB of RAM installed and a Intel Core i5-3470 CPU operating at 3.20 GHz (quad core)

The virtual machine used in our test will have a 32-bit Windows7 with 1.95 GB of RAM and will only use one processor.

Our tests will be performed on the following Cloud storage clients versions:

Dropbox 2.10.3
Box 4.0.5116
Google Drive 1.16.7009.9618
Amazon Cloud Drive 2.4.2013.3290
OneDrive 17.0.4041.0512

4.2 TEST SUITE

This section will contain the results obtained when using our benchmark tool with the test suite described in chapter 3. In each test, we will show the most important data obtained and we will try to explain the results comparing the Clouds between them.

4.2.1 *File size benchmark*

In this part we are going to examine the results obtained from the File size benchmark.

Results

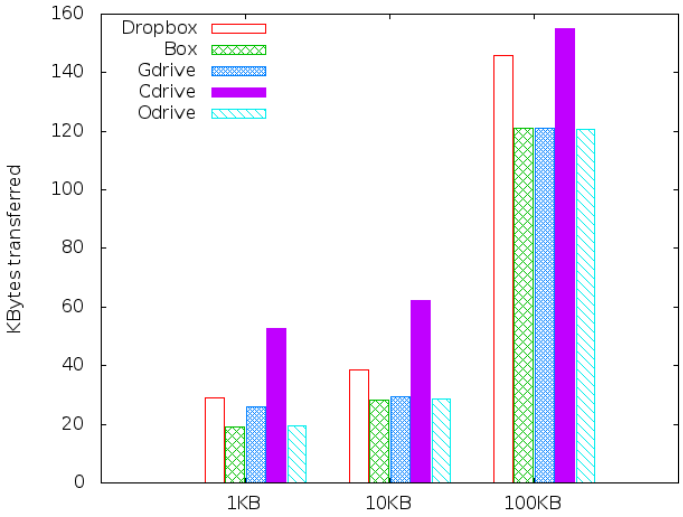

Figure 2: 1 kB, 10 kB and 100 kB files traffic

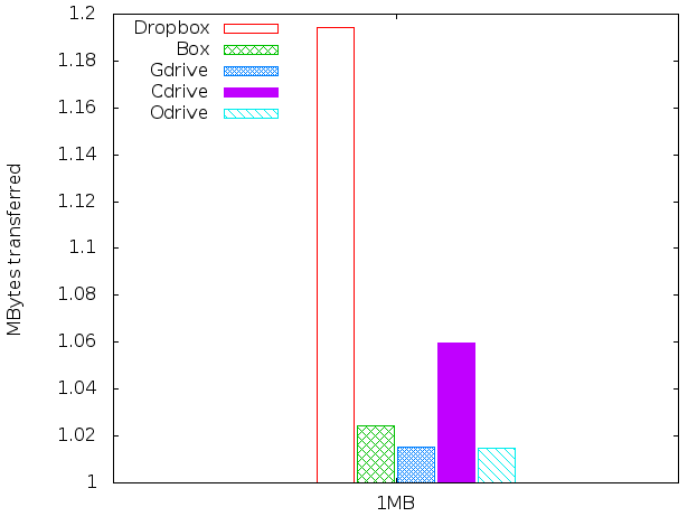

Figure 3: 1 MB file traffic

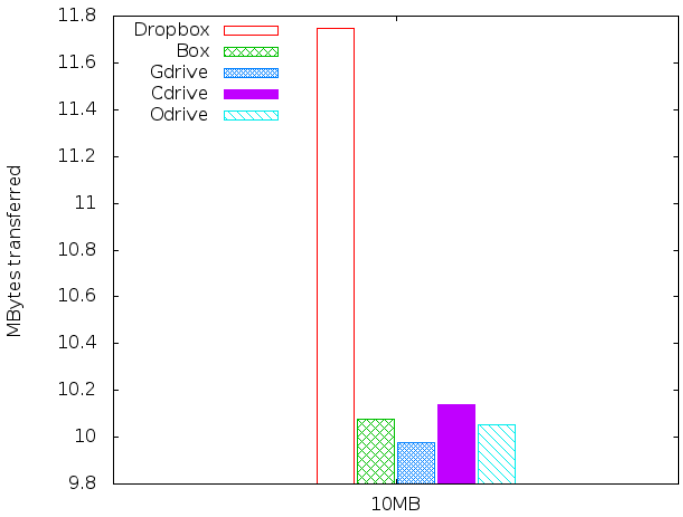

Figure 4: 10 MB file traffic

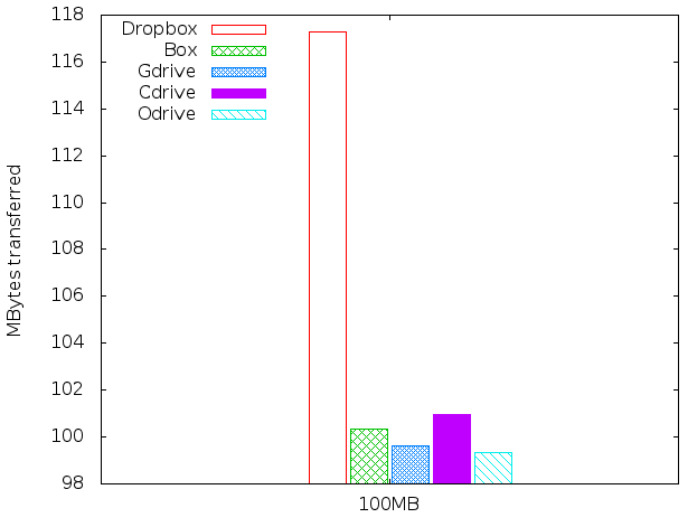

Figure 5: 100 MB file traffic

Table 5: Overhead table

File size	Dropbox	Box	Google Drive	Amazon Cloud Drive	OneDrive
1 KB	29.2α	19.2α	25.9α	52.8α	19.5α
10 KB	3.84α	2.84α	2.95α	6.23α	2.88α
100 KB	1.46α	1.21α	1.21α	1.55α	1.20α
1 MB	1.22α	1.05α	1.04α	1.08α	1.04α
10 MB	1.2α	1.03α	1.02α	1.04α	1.03α
100 MB	1.2α	1.02α	1.02α	1.03α	1.02α

Results Explanation

Graphics 2, 3, 4 and 5 show the results of the test. We can observe how sending a file of 1 KB or sending another of 10 KB results nearly in the same amount of data transmitted. This is because all of the metadata sent. Metadata is data that does not come from the file but that contains information related with it, for instance, it could be the file size, name or date among others. Each Cloud system stores different information in their metadata, so that is why each Cloud is sending a different total amount of data.

We can observe how metadata really impacts the total amount of data sent when we are working with low size files (figure 2). On the other hand, while sending large amounts of data (figures 4 and 5), the proportion of extra data sent is nearly irrelevant. Although that, *Dropbox* seems to store a large amount of extra data compared with the other Clouds.

In table 5, we have calculated the overhead obtained in each set of tests. The overhead is the fraction of the total data we have sent compared to the original file size. Ideally the perfect overhead should be 1, as it means that for each byte of data we have, we also just transfer 1 byte to the Cloud. Due to the metadata factor we explained before, there is always an extra data sent for each file.

We observed that sending files smaller than 1 MB means that a big part of the data transferred will not consist in the file we want to send. Specifically, if we are using *Amazon Cloud Drive* and we want to send 1 KB of data, we will need to transfer 53 KB, that is 53 times the size of the file.

As stated before, once the file size increases, the overhead is considerably reduced, and nearly reaches the ideall overhead of 1. Just in the case of *Dropbox*, we can see how this overhead remains relevant as we just achieved a 1.2 overhead factor when sending 100 MB.

4.2.2 *Bundling benchmark*

This part will show the results obtained in the Bundling benchmark.

Results

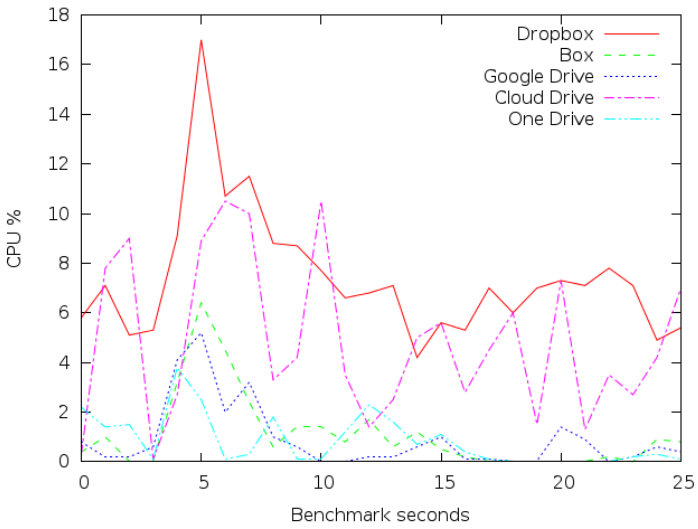

Figure 6: Set 1 CPU Results

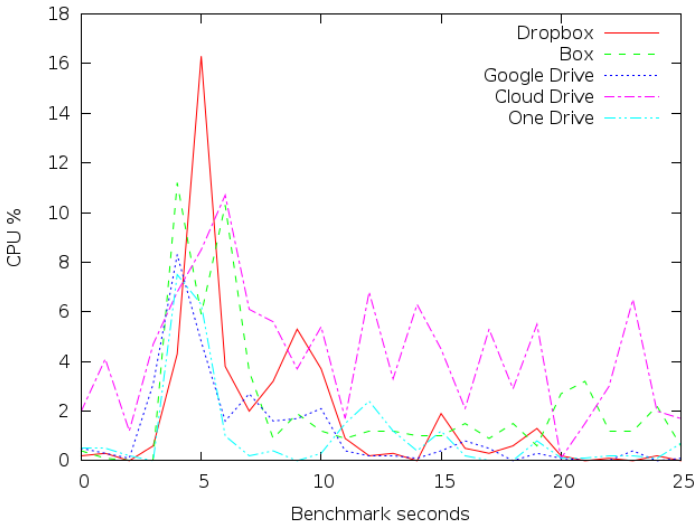

Figure 7: Set 2 CPU Results

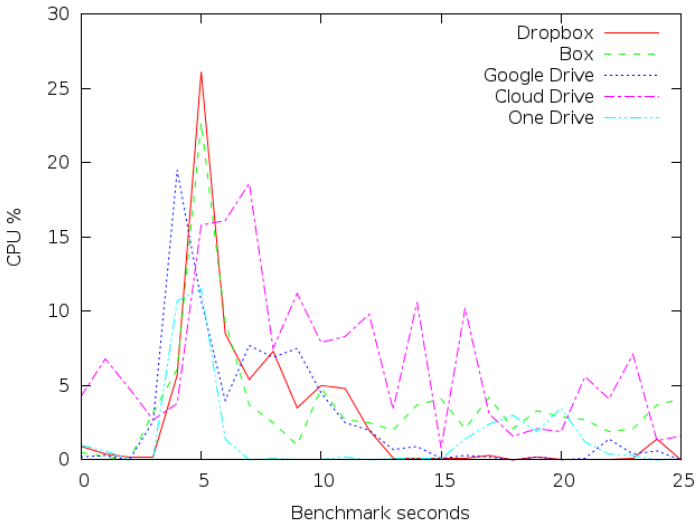

Figure 8: Set 3 CPU Results

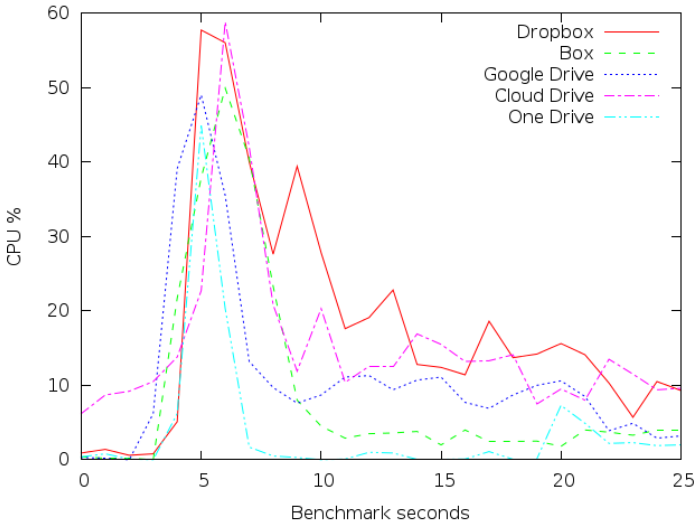

Figure 9: Set 4 CPU Results

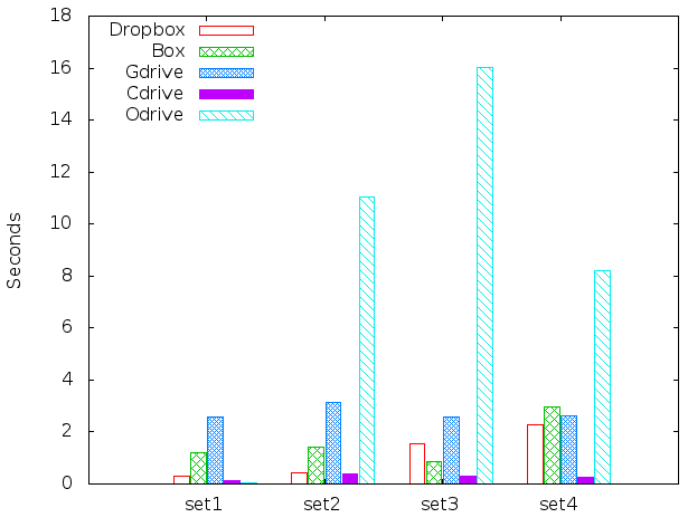

Figure 10: Start up time for the different Setups

Results Explanation

If we take a look at the CPU figures 6, 7, 8 and 9, sending more files really impacts the CPU cost although we are sending the same total output data. We can observe how *Dropbox* is the system that consumes more CPU in general, concretely, this consume is really exaggerated while we sent just one file (figures 6 and 7). On the other hand, *One Drive* is the system that consumes less CPU resources.

First, we will compare if sending a file with different size impacts the CPU cost. Looking at figures 6 and 7 we can see that systems like *Dropbox* and *Amazon Cloud Drive* do not have a CPU variation regardless of the file size they need to transmit. The others, on the other hand, need to consum 4% more CPU in other to handle the file size increase.

In general, we can say that sending 1 file (7) has an approximate CPU consumption of 12%, sending 10 files (figure 8) increases the CPU to 20% and sending 100 files (figure 9) increases the average to a 50% CPU consumption.

It is important to notice, that *Dropbox* and *Cloud Drive* consume an important amount of CPU not only before starting the file transmission but also during the whole transmission of the file.

For this test we also measured the amount of time that takes for each system to start transmitting data to the Cloud. As we can observe in figure 10, the average time is around 2 secs. *Dropbox* and *Box* increase the amount of time depending on the number of files to be transmitted. More files to syncronizate means more time before

Dropbox and *Box* start the transmission of data. *Google Drive* and *Cloud Drive* start up times remain constant regardless of the amount of files to be transmitted.

We can observe how *Cloud Drive* start up time is extremely low compared with the other systems. *One Drive*, on the other hand, is the system that takes more time to start sending files to the Cloud.

With all this results, we can conclude that *Google Drive* and *Cloud Drive* do not perform a bundling technique, we can say that because in all the scenarios, both Cloud services had the same start up time regardless of the amount of files being transmitted.

In terms of CPU, we do not have a clear result to say if bundling has a heavy impact or not. It seems that the CPU cost is more linked to the overall client implementation than to the bundling implementation.

On the other hand, talking about the start up time, its clear that not performing bundling will result in starting the transmission a couple of seconds before. However, we also need to take into account the total amount of time that will take place to finish all the transmission, and in *Benchmarking Personal Cloud Storage*[20] we could see how bundling clearly reduced the total time needed to complete the file synchronization.

4.2.3 *Deduplication and delta encoding benchmark*

This part contain the results obtained from the Deduplication and delta encoding benchmark. The original file will be considered the set 1, the replica under different name the set 2, and the replica with an extra byte the set 3.

Results

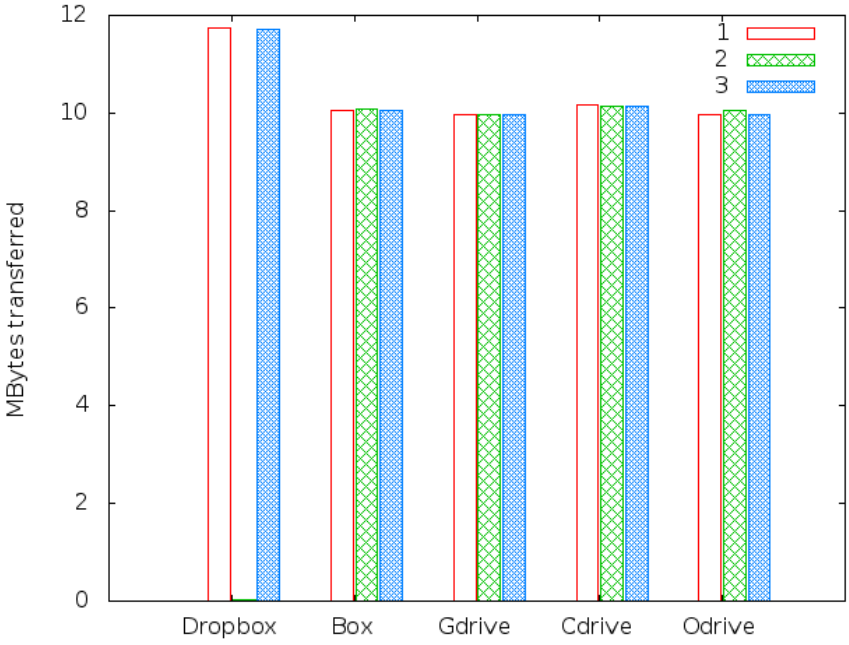

Figure 11: Deduplication and delta encoding results

Table 6: Dropbox start up time with deduplication

Set	Dropbox start up time
Original 10 MB file	7.56 seconds
Replica under different name	6.8 seconds
Replica with 1 byte at the beginning	7.4 seconds

Results Explanation

Once we look at the figure 11 we can see how *Benchmarking Personal Cloud Storage*[20] was right about *Dropbox* being the only system applying deduplication techniques. However, our results showed that contrary to what *Idilio* said in [20], it does no apply a fully delta encoding technique.

It is true that *Dropbox* uses delta encoding when the changes are performed in the middle or at the end of the file, but our results showed us that this technique is not used when the changes are performed at the very beginning of the file.

Although this discrepancy, *Dropbox* is still the only tested Cloud applying at least the deduplication technique and nearly a full supported delta encode.

In table 6 we have the results of the start up time *Dropbox* needs to start sending data to the Cloud. As we can observe, in both cases 1 and 3, where there was no data reduction, *Dropbox* takes 7.5 seconds in order to start the transmission. With deduplication, however, once the client detects the file is a replica, the transmission of the metadata starts 0.8 seconds earlier.

4.2.4 *Wikipedia file benchmark*

In this part, the results from the Wikipedia file benchmark are showed.

Results

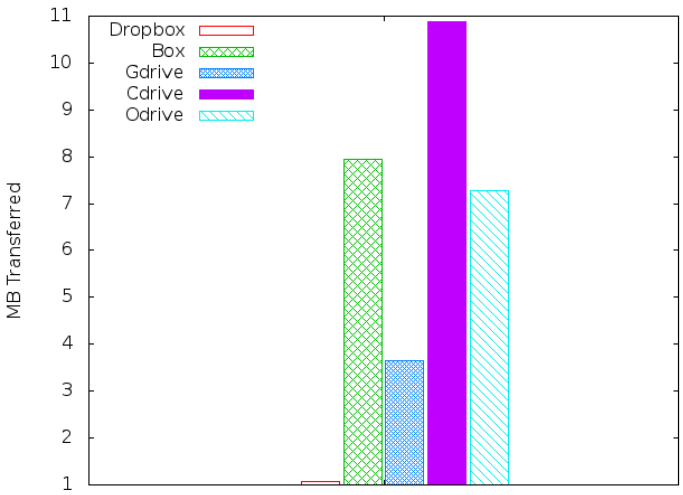

Figure 12: Wiki overhead Results

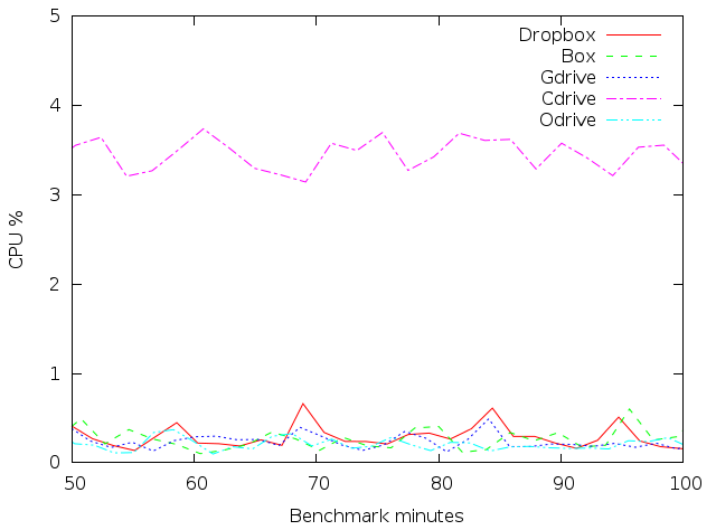

Figure 13: Wiki CPU Results

Results Explanation

First we are going to take a look at the data transmitted for each system in the figure 12. Here we can see how implementing deduplication really saves a lot of traffic, for instance, *Dropbox* does only

need 1 MB in order to synchronize the 50 different versions of the page. The other systems, not implementing deduplication, had to sent a bigger quantity of data to complete the 50 changes.

In order to understand the rest of the results, we will take a look back at the results we got in the "File size benchmark" 4.2.1, concretely between the 100 KB (figure 2) and 1MB (figure 3) file size. In both tests, excluding *Dropbox* (due to the deduplication technique), *Amazon Cloud Drive* is the system that needs more data to synchronize the files with the Cloud, in this Wikipedia test, it needs 11 MB to complete the test.

The rest of the Clouds also reproduce the results, as we have *Google Drive*, *OneDrive* and *Box* consuming the 2nd, 3rd and 4th less data respectively.

Figure 13 shows a cut of the CPU consumption we had through the test. *Amazon Cloud Drive* is the system that consumes more processor as it needs between a 3% and a 4% of the CPU. The others, on the other hand, do not even need a 1% of the processor.

With this results, we can state that collaborative editing does not require lot of processor potential, as the little modifications as well as the small file size do not consume that much CPU.

In conclusion, *Dropbox* is the best system handling a file-version workload due to the huge impact that deduplication has on the amount of data transmitted. On the other hand, *Amazon Cloud Drive* could be considered the worst at handling these situations, not only because it consumes the most CPU (it is still only a 4%) but due to the total amount of data transmitted.

4.2.5 *Log file benchmark*

This part will show the results obtained in the Log file benchmark.

Results

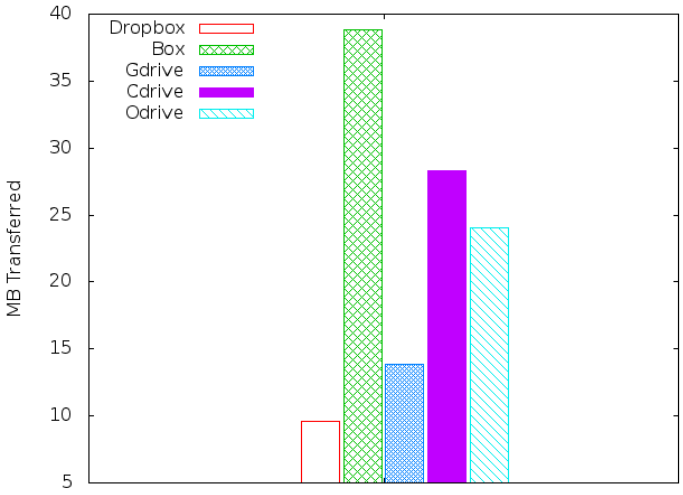

Figure 14: Log overhead Results

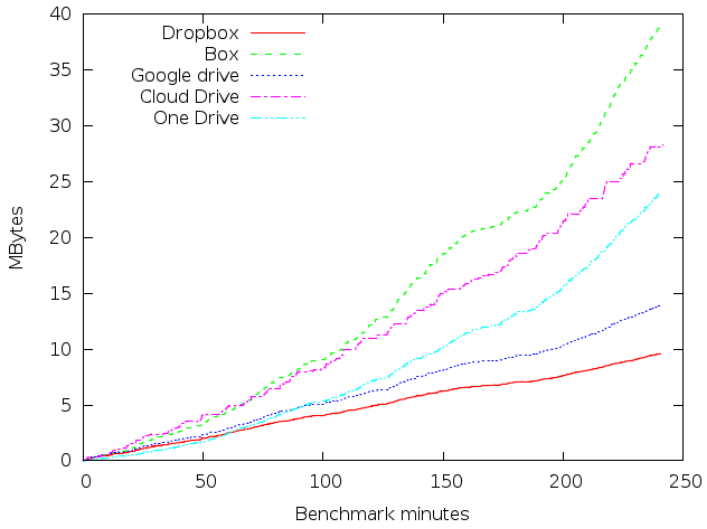

Figure 15: Log CPU Results

Results Explanation

In figure 14, we can see how again, *Dropbox* uses the deduplication technique to save large amounts of data, this time, however, the data reduced with this technique is not as impressive as we saw in previous section 4.2.4. This is because in this test we have much

more file changes (more than 10 times) and also because the versions have more data modified.

The rest of the systems behave as expected with the exception of *Box*. We can see how *Box* is the system with more overhead in this test, this is due to the client behavior of *Box*. This client, creates a queue list with all the changes performed in the synchronization folder, and goes one by one through this list, performing the synchronization with the server for each change.

The problem with this list is the following. Imagine we add a file "A" to our Cloud folder, and during the next 10 seconds we perform 2 modifications. The *Box* client behavior will be as follows. The client will sent the original "A" file to the Cloud, lets assume this synchronization cost 20 seconds. Next, the client will start with the first modification synchronization, we assume again 20 more seconds to sync. Finally, the client will start with the last and current modification of the file, 20 seconds more. At the end, we used 60 seconds to complete the 3 synchronizations.

Other Clouds, with the same test will act as follows: start transmitting the original "A" file, and as soon as the client detects the first modification, stop the synchronization and send the last version of the file. At the end, these Clouds will just need the first 10 seconds plus 20 seconds more to complete the synchronization, saving lot of data transmission.

Box behavior is super inefficient when the workload performs changes in short intervals because the client will be adding tasks to the queue faster than the server will handle the synchronization, and as explained before, we will have a bottleneck.

Figure 15 shows this bottleneck, while at the beginning of the test *Box* can handle the workload (consuming the 2nd most data as showed in the previous section 4.2.4), once the file size starts to increase (around 70 minutes), the amount of time each synchronization takes is higher than the time between file updates.

We can conclude that under workloads with small time between updates in the files, *Box* shows a poor performance compared with the other systems.

4.2.6 *Synthetic file benchmark*

This part shows the results of the last test, the Synthetic file benchmark.

Results

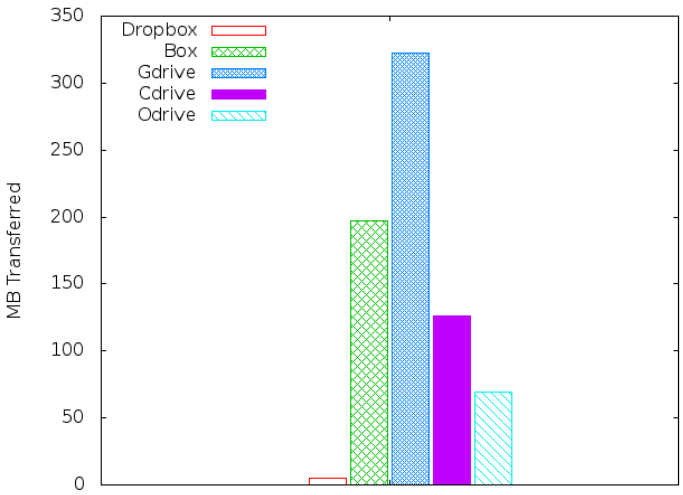

Figure 16: Synthetic Overhead

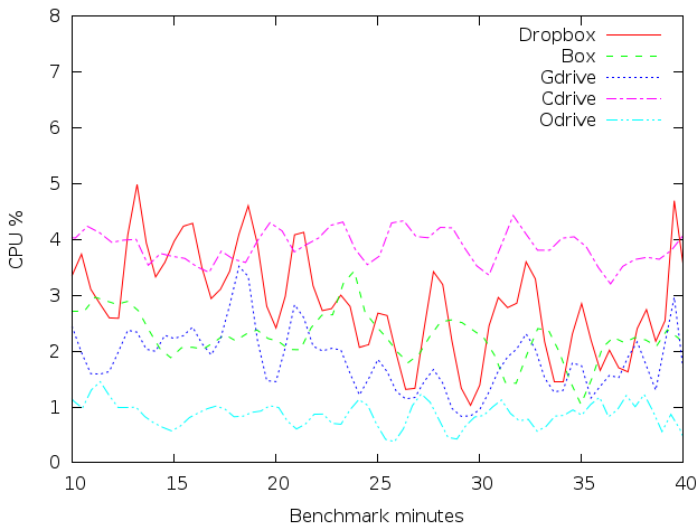

Figure 17: Synthetic CPU

Results Explanation

The results in figure 16 show that, again, applying deduplication results in less data transmitted, as *Dropbox* is the system with less data sent. The other systems show similar results as the ones

obtained in the *Log file benchmark* 4.2.5 with the exception of *Google Drive*.

Results show that modifying a file with intervals from 0 to 10 secs (small intervals) and applying this modifications in different parts of the file makes *Google Drive* send an incredibly amount of data. We don't know what can cause this situation.

Looking at the CPU consumption in the figure 17, we can conclude that this workload takes no more than a 5% of the processor capacity. *Dropbox* and *Cloud Drive* are the systems that need more CPU to complete the test while *One Drive* is the system that requires less processor, between 1 and 2% of the total capacity.

4.3 LESSONS LEARNED

In this section we are going to give the conclusions learned from the previous benchmark results.

First, we saw how metadata makes a huge impact in the overall data transmitted when working with small size files, however, as the size increases, the overhead was nearly negated. Only *Dropbox* showed to sent an important extra data when synchronizing big files.

Sending more than one file at the same time impacts the processor consumption, but this consumption is not clearly related with the application of bundling. However, we could see how bundling increases the start up time by a couple of seconds.

Deduplication and delta encoding are the key techniques when working with file modifications, as the data transmitted is nearly negated. Only *Dropbox* implemented deduplication but our test showed that a full delta encoding was not achieved. Deduplication also seems to reduce the start up time by near a second.

Modifying a file do not have a huge impact in processor consumption as long as the changes made and the file of the size is relative small (changes of some bytes and files of hundreds of KB).

When the intervals between modifications is small, *Box* queue technique acts as a bottleneck increasing the amount of data transmitted.

When the modifications are made in different parts of the file and we have small intervals between modifications, *Google Drive* showed a really poor performance, however, we were not able to identify the reason for this behavior.

5

CONCLUSION

In this thesis we generated a benchmarking tool to benchmark personal Clouds. We explained the reasons why this tool was needed and defined the features it incorporates. We also described the test suite it incorporates.

We also tested the benchmark with various Cloud storage providers. We analyzed our results an linked them with the different Cloud techniques available.

To finish with this thesis, we are going to give some conclusions:

First, as we have seen during the whole thesis, each client works completely different from others. Even when we tried the most simple benchmark possible, we obtained different results for each Cloud client.

Second, we can conclude that applying the Cloud techniques described in this thesis has a positive impact into the overall synchronization process. To summarize:
Chunking really benefits the way we handle data, as we can be more precise and reduce recovery time after synchronization failures.
Bundling has an start up time cost but ends up resulting beneficial because reduces the total data transmitted as well as the total time of the transmission.
Deduplication and **delta encoding** have a huge impact on the data transmitted when working with modified files.
Compression helps reducing the amount of data transmitted if the format accepts compression.

Third, the commercial solutions have a nice interface and a good cross-platform software but lack in applying all the real Cloud storage techniques.

Fourth, we need to invest in open source solutions and try to make them viable and able to compete with the big companies, because this

way commercial solutions will have to improve (apply Cloud storage techniques) in order to be competitive.

5.1 FUTURE WORK

We believe there is still a lot of work that can be done related with the topic. For example:

Testing the benchmark with more Cloud storage solutions, specially with Open Source alternatives. This way we can be able to pick the best technique implementation and overall produce a better Cloud storage system.

The test suit can be increased. There are still plenty of new tests that could be done and incorporate to the benchmark tool, for instances, testing how each Cloud handles files collisions or evaluating how the client synchronizes the files when the changers are being made in another computer.

The tests can also be repeated with different real traces, for example, instead of using a *Wikipedia* page, we could recreate the test with a full *Github* repository.

BIBLIOGRAPHY

[1] Dropbox, http://www.dropbox.com

[2] Box, http://www.box.com

[3] Google Drive, http://drive.google.com

[4] Amazon Cloud Drive, http://www.amazon.com/clouddrive

[5] Microsoft OneDrive, http://onedrive.live.com

[6] Github, https://github.com

[7] Librsync, http://librsync.sourceforge.net

[8] Github 3 million users, https://github.com/blog/1382-three-million-users, 28/8/2014.

[9] Github 6.8 million users, https://github.com/about/press, 28/8/2014.

[10] Dropbox info, http://www.dropbox.com/news/company-info, 29/8/2014

[11] Facebook, http://facebook.com

[12] Yahoo, http://yahoo.com

[13] Vimeo, http://vimeo.com

[14] Gartner consumer research: Personal Cloud, http://gartner.com/technology/research/personal-cloud

[15] F. E. Gillet, The personal cloud Transforming personal computing, mobile and web markets. In *Forrester Research, BT Futures Report*, 2011.

[16] ownCloud, http://www.owncloud.org

[17] SparkleShare, http://www.sparkleshare.org

[18] Syncany, http://www.syncany.org

[19] Wikipedia Data Compression http://www.wikipedia.org/wiki/Data_compression

[20] Idilio Drago, *Benchmarking Personal Cloud Storage*, 2013.

[21] Zhenhua Li, *Efficient Batched Synchronization in Dropbox-Like Cloud Storage Services*, 2013.

[22] Pedro Garcia, *StackSync: Architecturing the Personal Cloud to Be in Sync*, 2013.

[23] Raúl Gracia, *Actively Measuring Personal Cloud Storage*, 2013.

[24] Pedro Garcia, *StackSync: Bringing Elasticity to Dropbox-kie File Synchronization*, 2013.

[25] Idilio Drago, *Inside Dropbox: Understanding Personal Cloud Storage Services* , 2012.

[26] Vasily Tarasov, *Generating realistic dataset for deduplication analysis* , 2012.

[27] Kave Eshghi, *A framework for analyzing and improving content-based chunking algorithims*, 2005.

[28] Srikanth Kandula, *The nature of data center traffic: measurements and analysis*, 2009.

[29] A. Iosup, *On the performance variability of production cloud services*, 2011.

[30] E. Walker, *Benchmarking amazon ec2 for high-performance scientific computing*, 2008.

[31] A. Li, *Cloudcmp: comparing public cloud providers*, 2010.

[32] W. Hu, *The good, the bad and the ugly of consumer cloud storage*, 2010.

[33] Mayur R. Palankar, *Amazon s3 for science grids: a viable solution?*, 2008.

[34] A. Bergen, *Client bandwidth: The forgotten metric of online storage providers*, 2011.

[35] Zach Hill, *Early observations on the performance of windows azure*, 2010.

[36] Hash, `http://www.wikipedia.org/Hash_function`

[37] Wikipedia, `http://www.wikipedia.org`

[38] Virtual Box, `http://www.virtualbox.org`

[39] Aleksandr Solzhenitsyn Wikipedia, `http://en.wikipedia.org/wiki/Aleksandr_Solzhenitsyn`