NTNU – Trondheim
Norwegian University of
Science and Technology

Benchmarking Cloud Storage Systems

Xing Wang

Master in Security and Mobile Computing
Submission date: July 2014
Supervisor: Yuming Jiang, ITEM
Co-supervisor: Julien Beaudaux, ITEM
Markus Hidell, KTH Royal Institute of Technology

Norwegian University of Science and Technology
Department of Telematics

Title: Benchmarking Cloud Storage Systems

Student: Xing Wang

Problem description:

Over the years, electronic components became smaller and cheaper. As a consequence, in the close future, connected devices will not only take the form of traditional terminals (e.g. computers, tablets, smartphones), but also of any everyday-life objects. This is what the Internet of things (or IoT) paradigm is about, a wide range of connected objects able to monitor our life space, and interact with it accordingly. Driven by the emergence of this new paradigm, the number of connected things is expected to be multiplied by 3 in the next few decades. This rapid growth will most probably be followed by an increased amount of retrieved data (e.g. sensor readings, status updates). While such data will have to be stored as they are collected, it is yet unknown whether or not current personal cloud storage systems are able to cope with it. This Master thesis should aim at providing a complete study of personal storage systems in the specific context of IoT deployments. The contributions to be displayed are manyfold, and include:

- An analysis of relevant storage systems: It should provide a complete analysis of the capabilities implemented by each personal cloud storage system, and how they can be put to good use to improve the overall performance of the system when coupled with an IoT application.

- A thorough evaluation based on a realistic IoT context: An experimental IoT house-monitoring testbed has to be deployed, in order to provide a realistic dataset. A thorough evaluation should then be conducted to evaluate several relevant cloud storage systems in this context.

- A comparative study of each system: In the light of the results, we should explore to what extent present cloud storage systems can cope with data generated by IoT applications. In parallel, the security and privacy guarantees proposed by each system should be investigated and put in perspective with the experimental results. As a result, we should detail which solutions are best suited for IoT applications, and provide future directions for further developement of IoT-integrated storage systems.

Responsible professor: Yuming Jiang, ITEM; Markus Hidell, KTH

Supervisor: Julien Beaudaux, ITEM

Abstract

With the rise of cloud computing, many cloud storage systems like Dropbox, Google Drive and Mega have been built to provide decentralized and reliable file storage. It is thus of prime importance to know their features, performance, and the best way to make use of them. In this context, we introduce BenchCloud, a tool designed as part of this thesis to conveniently and efficiently benchmark any cloud storage system.

First, we provide a study of six commonly-used cloud storage systems to identify different types of their features. Then existing benchmarking tools for cloud systems are presented, and the requirements, design goals and internal architecture of BenchCloud are studied. Finally, we show how to use BenchCloud to analysis cloud storage systems and take a series of experiments on Dropbox to show how BenchCloud can be used to benchmark and inspect various kinds of features of cloud storage systems.

Acknowledgements

This thesis is written as a part of the Erasmus Mundus NordSecMob (Nordic Security and Mobile Computing) Master's Program. I am glad to have taken up the challenging project. The thesis work has been a great learning experience.

I would like to thank my thesis instructor, Dr. Julien Beaudaux, and supervisor, Professor Yuming Jiang, for their invaluable guidance throughout the thesis work. I am also thankful to Professor Markus Hidell from KTH Royal Institute of Technology for supervising the thesis remotely. Finally, I am very grateful to my friends and family for their support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Problem Description

Over the years, cloud computing has become one of the most influential topics in IT industry. It introduces revolutionary innovation with respect to IT resource management and utilization. Based on distributed computing, virtualization and many other technologies, cloud computing offer us extensible and highly reliable on-demand services to reduce infrastructure cost, installation cost, and management cost.

As a part of cloud computing, various (personal) cloud storage systems keep coming to the market, providing a way to store files on the cloud instead of on a user's local file system. Such cloud storage systems free us from building and maintaining file storage systems ourselves, providing a convenient way to store and access our files, as well as other useful functions like file synchronization[1] and sharing. In addition, cloud storage systems can provide resilient and secure file storage, which is one of the main advantages when compared to local storage.

However, there are many cloud storage systems and choosing the right one among them is the first thing to take into consideration before start using it. Average users may need to pick the one with the best performance in file uploading and downloading. Industry and academic users may need to evaluate different cloud storage systems in terms of various features. Thus systematic way should be proposed and convenient tools should be developed to benchmark these systems both conveniently and efficiency, minimizing the effort for performing benchmarks.

In this thesis, we provide a study of benchmarking cloud storage systems. The contributions displayed in the present thesis are manifold, and include:

[1] File synchronization is the process to ensure that files stored in two or more computers are updated when changes have been made to the files.

- **An analysis of relevant cloud storage systems:** It provides an analysis of the capabilities implemented by each cloud storage system. Basic features as well as advanced features of the systems are studied like file chunking, bundling, deduplication, and delta encoding. Security and privacy features and resiliency features are also taken into consideration because they play an important role in ensuring the safety of users' data.
- **An introduction to BenchCloud, a tool for benchmarking cloud storage systems:** BenchCloud is a benchmarking tool developed for this thesis to provide a convenient, efficiency, and flexible way to benchmark cloud storage systems. The background and requirements of benchmarking tools are analyzed, and the design and architecture of BenchCloud are described.
- **A description of the process to benchmark cloud storage systems with BenchCloud:** Several experiments were conducted to study a wide range of features of cloud storage systems, and experiment process as well as the analysis of the results are described.

1.2 Motivation

Over the last decade, a wide range of personal cloud storage systems emerged. These solutions became more and more popular both for individual and industries, as they provide remote access to large storage capacities at low cost, together with robustness guarantees (as the storage company is held responsible for the data it stores).

Following the success of pioneers such as Dropbox [11], many companies developed their own cloud storage system (e.g. Microsoft Azure [20], Google Drive [14], LaCie Wuala [2]). Several experimental studies were conducted, quantifying the performance of each solution depending of the selected usage pattern (e.g. datasets size, access frequency). In [9], five popular storage system offers (managed by companies) are benchmarked.

However, no mature solution has yet been developed and provided to the large public to benchmark cloud storage systems with convenience, efficiency, and flexibility. Many benchmarks have to be performed manually, which are quite time-consuming and hard to be reproduced. Automatic benchmarks always require testers to have programming skills, and programs/scripts have to be made for different types of benchmarks. An automatic benchmark tool should be made to minimize the effort of benchmarking, and that is why we developed BenchCloud. Different kinds of users can benefit from BenchCloud, for example:

1. Average users can find the fastest cloud storage system for daily use.
2. Industry users like mobile and web application developers can find out the best way to make use of cloud storage systems for data storage of their applications.

3. Academic users can inspect different features of cloud storage systems, and may evaluate the feasibility to use one as a backend storage system for a larger system, etc.

1.3 Thesis Organization

The organization of the rest of the thesis are as follows.

Chapter 2 gives a study of six cloud storage systems. Various features are studied and detailed to give a deep view into the systems.

Chapter 3 introduces BenchCloud, a tool for benchmarking cloud storage systems developed for this thesis. The background, requirements, design goal, and internal architecture of BenchCloud are presented.

Chapter 4 presents the process and results of several experiments to benchmark cloud storage systems using BenchCloud. Each of the experiments are designed to study a specific feature of cloud storage systems, and their results are analyzed providing an insight how BenchCloud can be used to benchmark and analyze various kinds of features of the systems.

Chapter 5 makes a summary of the whole thesis and provides some possible work for the future.

Appendix A provides sample configuration files for performing benchmarks described in Chapter 4.

Chapter 2
An Analysis of Cloud Storage Systems

In this chapter, we will make an analysis of six (personal) cloud storage storage systems and study their features as advertised. First we will give a brief introduction to the six systems. Then we will describe the security and privacy features, the resiliency features and other features of the systems like file bundling, file compression and so on. Note that the conclusions from this chapter should not be taken for granted and should be confirmed by experiments later on.

2.1 Overview

Dropbox

Founded in 2008, Dropbox has now been one of the most popular file hosting services in the world. It is operated by Dropbox, Inc, which is headquartered in San Francisco, USA. It offers cloud storage, file sharing, file synchronization over different devices and it provides clients on both desktop platforms and mobile devices. Dropbox provides comprehensive features to make it very efficient and easy to use, while it may not be suitable for users which are very sensitive to the security and privacy of their data, as explained in 2.2.2.

MEGA

MEGA[1] is hosted by Mega, Ltd. which is headquartered in New Zealand. It is founded in 2013 after its predecessor, Megaupload[2], was shut down by the United States Department of Justice in 2012 [3]. MEGA has some features that shared by many other cloud storage systems, such as file synchronization and file sharing, but its most notable feature is so called "end-to-end encryption", which means the data uploaded by a user are encrypted on client side before transmitted to any storage

[1] https://mega.co.nz/
[2] http://en.wikipedia.org/wiki/Megaupload
[3] http://www.bbc.com/news/technology-16642369

server managed by MEGA. Because of this, users sensitive to their data security and privacy may like to choose MEGA, although client-side encryption and decryption will lead to low performance in file uploading and downloading. It can also share files without needing to create an account.

Wuala

Wuala is hosted by LaCie AG headquartered in Switzerland. It is a cloud storage service targeted at security. Apart from providing services like file synchronization, file sharing, and file versioning, Wuala provides extra security enhancements like encrypting data before uploading them to servers. Besides that, Wuala claims that no passwords will be transmitted to the server [33] and there is no way for them to get the decrypted content of users. Like MEGA, Wuala is also suitable for users sensitive to data security and privacy.

Google Drive

Operated by Google, Google Drive is released in 2012 and provides common cloud storage services like file hosting, file sharing, file synchronization, etc. It also offers collaborative editing on documents, spreadsheets, presentations, and more, which are very suitable for users with requirement of document making and collaborative editing. Based on the large user base of Google services, it is convenient to share documents with other people with Google Drive.

Tahoe-LAFS

Tahoe-LAFS[4] is a free and open source secure cloud storage system. It is featured by its provider-independent security, which means that users do not rely on storage servers to provide confidentiality or integrity for their data; instead, data are encrypted by a Tahoe-LAFS gateway before uploaded to the server. Tahoe provides redundancy so that even if some of the servers fail, the entire file system still functions correctly. A big advantage of Tahoe-LAFS is that it is open sourced, which means the internals of it can be checked by the public, and people can build their own secure cloud storage systems based on Tahoe-LAFS.

Tamias

Tamias[5] is a free and open source secure cloud storage built upon Tahoe-LAFS that has some important features including full encryption of every object and redundancy in storage. It provides a user identification and authentication system that are not included in Tahoe-LAFS. Besides that, Tamias implements capability signing and

[4]https://tahoe-lafs.org
[5]https://tamias.iijlab.net/

encryption, and a user-centric repository for in-band exchange of URIs. This provides decentralized multi-user management.

2.2 Security and Privacy

We have special interest in security and privacy, because we consider it a very important part of cloud storage systems. Some users need to store credential and sensitive data into the systems and they need to know that their data is secure and privacy can be guaranteed, and what level of security and privacy the systems can provide. The six cloud storage services we studied in this thesis have different levels of security and privacy implemented by different mechanisms and in this section we try to describe the security and privacy features of the systems.

2.2.1 Aspects of security and privacy

By security, we mean "information security", and it has three attributes: Confidentiality, integrity and availability. Confidentiality means the sensitive information can not be reached by wrong people. Integrity maintains the consistency, accuracy, and trustworthiness of data over its entire life cycle. Availability ensures that the information is available when you need it.

By privacy we mean that personally identifiable information is protected. Privacy concerns exist wherever personally identifiable information is collected and stored, and can be a major concern when choosing a personal cloud storage system.

For the security and privacy of personal cloud storage systems, we are especially interested in the following aspects:

- **Is Transport Layer Security [8], or TLS, is used when transmitting and downloading files?** TLS is a protocol ensuring privacy of communications on the Internet. When a server and client communicate, TLS ensures that no third party may eavesdrop or tamper with any message.
- **Are the data saved on the server encrypted or not?** If it is encrypted, where is the encryption made? Is it made in the client side or the server side? Is if possible for the system operator to get the content of users' files?
- **Is file metadata encrypted on the server?** Is a system operator able to get file metadata? We care about this because sometimes we want to make sure not only the file content but also file metadata like file names are not accessible to others.
- **Does the server keeps users' account information?** Are the passwords of users saved on server?
- **What information the system operator can get from users?** System operators often claims that they will not do something, for example, to claim

they will not view the files uploaded. But we would like to know whether it is trustable. Or, are the system operators have the ability to get sensitive information?

2.2.2 Security and privacy of the systems

Dropbox

Dropbox uses TLS when transmitting and downloading files between clients and servers. It claims that they encrypt files on the server side using AES-256 [5]. However, because the clients have no control over encryption/decryption, even if files are encrypted on the server, we can conclude that the system operator is able to decrypt the files and get their content. For the file metadata, we do not know and cannot know whether it is encrypted or not, but we can conclude that Dropbox is able to get them, because according to their privacy policy, they are "permitted to view file metadata" [10]. Besides, the users' passwords are saved on the server side, and we can not know if they are encrypted or not, or how they are encrypted, although nowadays it is a common practice to encrypt passwords saved on server. So basically, when using Dropbox, the content of files, the file metadata, and the users' account information are all accessible by the system operator.

MEGA

MEGA claims that they use AES-128 for bulk transfers, but it is to be verified. MEGA takes client-side encryption, which means the file content is encrypted before sent to the server and files we downloaded are encrypted and we will decrypt them after they are downloaded to the client side. This forms the so called "end-to-end encryption". According to MEGA's official website [18], the file names and folder structures are not encrypted. They claims that no usable encryption keys, with the exception of RSA [1] public keys, ever leave the client computers, but it is to be verified. The client machines are responsible for generating, exchanging and managing the encryption keys. The corresponding part for this, User Controlled Encryption (UCE), is open sourced, meaning we can know exactly how the client-side encryption/decryption works. As they claimed [19], the only key that MEGA requires to be stored on the user side is the login password, in the user's brain. This password unlocks the master key, which in turn unlocks the file/folder/share/private keys. So basically, MEGA cannot get the content of files and users' passwords, but is capable of getting file metadata.

Wuala

Data are encrypted before being sent to the server. As they claimed [35], they "can only see how many files you have stored and how much storage space you occupy.

The files themselves, as well as all metadata (folder names, file names, comments, preview images, etc.), are encrypted." But still, we need to find out whether we can verify that. For the algorithms, they claim [34] that they use AES-256 for encryption, RSA 2048 for signatures and for key exchange when sharing folders, and SHA-256 [28] for integrity checks. It is worthy to point that, although they claim that they will not store any encryption key from users, there is one circumstance that when a user makes a file public or share it by secret weblink, "the encryption key is temporarily sent to web server as part of the URL for the purpose of serving the requested data" [32].

Google Drive

TLS is performed when using Google Drive. We do not know whether files are encrypted, and we do not have the ability to verify whether Google Drive does server-side encryption. We do have the ability to check whether it does client-side encryption and it is to be done by future experiments. We do not know and have no way to check whether file metadata are encrypted or not. The users' passwords are not saved on the client side so they must be kept on the server side.

Tahoe-LAFS

Tahoe uses the capability access control model [7] to manage access to files and directories. A capability is a series of bits that uniquely identify a file or a directory, and can be used to gain access to the file or directory. In Tahoe, there are two kinds of files, i.e. mutable and immutable files. Mutable files have three capabilities, the read-write-cap, the read-only-cap and the verify cap. [31] A read-write-cap allows a user to read and write a mutable file, while a read-only-cap only allows reads to the file but no modification to it. While a verify-cap only allows the file to be checked for integrity. For immutable files, there are read-only-cap and verify-cap. One interesting part of capabilities is capability diminishing, which means a verify-cap can be derived from a read-only-cap, and a read-only-cap can be derived from a read-write-cap. The capability of a file is derived from two pieces of information: the content of the file and the upload client's "convergence secret". By default, the convergence secret is randomly generated by the client when it first starts up and re-used after that. So the same file content uploaded from the same client will always have the same cap. The convergence secret is saved in the user's own computer and is not uploaded to the server.

TLS is used when transmitting files between clients and servers. In Tahoe-LAFS, users do not rely on the server for security; all data are encrypted in a Tahoe-LAFS gateway which is typically deployed on the user's own machine, which means Tahoe-LAFS uses client-side encryption instead of server-side encryption. Besides, because the key to encrypt files is stored on the user's own machine and is not uploaded to

the server, the system operator is not able to decrypt the user's files. Every object stored in Tahoe-LAFS is encrypted so the file metadata is encrypted, too.

Tamias

The capability-based storage solution will introduce some problems when sharing files. Because of the lack of user identity, all the sharing is based on passing the capability from one user to another. If the capability is passed on and on, from user to user, the original owner of the file cannot control who has access to the file. Tamias overcomes this disadvantage of capability-based system by associating an identity to every user, to make itself capable of providing fine-grained sharing features, delegation and revocation. Identity is at the core of Tamias. Tamias implements identity based on public-key cryptography.

Based upon Tahoe-LAFS, TLS is also used in communications between clients and servers. Tamias takes client-side encryption as well, and system operator cannot get the content of a file without knowing the private key of its owner. Like Tahoe-LAFS, Tamias also encrypts files' metadata. Besides, the users information is not kept on the server.

Summary

We can summarize the security and privacy features of the systems described above into Table 2.1.

2.3 Resiliency

By resiliency, we mean the ability of a cloud storage system to continue operating and keep the data stored safe even when there has been an node failure. We care about resiliency because we need our cloud storage system to be dependable, and we want to make sure our data stored in the cloud will not lost due to system failures.

The resiliency of Dropbox, MEGA, Wuala and Google Drive is not clear because the system is not open-sourced and therefore we do not know how they store their data internally. However, experiments can be conducted and to infer some aspects about how the systems work. This is out of the scope of this thesis and could be put in future work.

Tahoe-LAFS and Tamias uses erasure coding [30]. Erasure code is a forward error correction (FEC) code that transfers information, e.g. a file, into a series of file chunks with redundant information and thus the original file can be recovered from a subset of the encoded chunks. The ciphertext is erasure-coded into N shares distributed across at least H distinct storage servers so that it can be recovered from

Table 2.1: Summarization of system security features

	Dropbox	MEGA	Wuala	Google Drive	Tahoe-LAFS	Tamias
TLS	Yes	To be studied	To be studied	Yes	Yes	Yes
File encryption	Server-side	Client-side	Client-side	To be studied	Client-side	Client-side
File meta data encrypted?	Cannot know	Yes	Yes	Not known	Yes. File meta-data are saved in directories, which are encrypted	To be studied
User account credentials saved on server?	Yes	No (to be verified)	No (to be verified)	Yes	No	No
System operator able to access the content of users' files?	Yes	No (to be verified)	No (to be verified)	Yes	No	No

any K of these servers. Therefore only the failure of $H - K + 1$ servers can make the data unavailable. The parameters N, H, K can be set according to the number of the servers and the level of resiliency.

2.4 Other System Features

In this section, we propose some other features of cloud storage systems that are also considered to be important, and describe whether the six storage cloud storage systems have these features and how they behave on the features if applicable.

2.4.1 Chunking

When a user is uploading a large file to the server, some systems will not upload the file as a whole, instead the large file will be split into smaller pieces and uploaded to the server separately. This feature is called file chunking.

According to [9], Dropbox uses file chunking with a fixed size of chunk, which is 4 MB. Google Drive also performs a fixed size of chunking which is 8 MB. Wuala

use variable-sized chunking as well. According to the source code, Tahoe-LAFS and Tamias uses file chunking. Whether MEGA implements file chunking is to be studied and can be done in future work.

2.4.2 Bundling

When uploading many files together, some systems will combine the files into larger bundles before uploading them to the server. This feature is called file bundling and is a way to avoid the overhead of making large amounts of connections between clients and servers, so that transmission latency can be reduced.

According to [9], Wuala, Google Drive do not perform file bundling. Google Drive will open a separate connection for each file uploading. Wuala reuse TCP connection and transmits file sequentially, so each file uploading will wait for the previous file to be uploaded. Dropbox performs file bundling. According to the source code, Tahoe-LAFS and Tamias do not perform bundling. Whether MEGA implements file bundling is to be studied by future experiments.

2.4.3 Compression

Before a file is transmitted to the server, it can be compressed. This is called data compression. Although data compression will take extra processing time, it is a way to reduce traffic and storage requirements.

According to [9], Dropbox and Google Drive performs data compression. Wuala, Tahoe-LAFS and Tamias does not perform data compression. Whether MEGA performs data compression is to be studied. 4.2.6 gives an example of testing file compression with BenchCloud.

2.4.4 Deduplication

A file uploaded to a cloud storage system can be deduplicated, meaning that if an exactly same file uploaded by the user or another user in the system, only a link is kept to avoid transmitting the same file to the server again, thus network traffic and storage requirements can be reduced. Deduplication can happen in chunk level, if the system's storage unit is a file chunk, instead of a file.

Among the six systems, only Google Drive does not perform deduplication [9]. Dropbox performs file deduplication among all users, which is considered to be vulnerable to "Confirmation-of-a-File Attack" and "Learn-the-Remaining-Information Attack". [29] MEGA uses deduplication, but it claims deduplication only performs on the same key encrypted with the same random 128-bit key. Or, if a file is copied between folders or user accounts through the file manager or the API, all copies point

to the same physical file. Similar to MEGA, Wuala only performs deduplication on the same file under the user's personal cloud with the same decryption key. Tamias and Tahoe-LAFS perform deduplication only among clients who have the same convergence secret. To enable deduplication between different clients, a user should securely copy the convergence secret file from to all the others. 4.2.6 gives an example of testing file deduplication with BenchCloud.

2.4.5 Delta-encoding

Delta encoding is to compress the file to be uploaded by calculating and uploading only the difference between its previous version if this file is not a new file but a revision of another file stored in the server.

According to [9], Dropbox performs delta encoding while Wuala and Google Drive do not. According to the source code, Tahoe-LAFS and Tamias do not use delta-encoding. Whether MEGA uses delta-encoding or not is still to be studied by future experiments. 4.2.6 gives an example of testing delta-encoding with BenchCloud.

2.4.6 File sharing

File sharing is the ability to share files to other users. When talking about file sharing, we are interested in the following questions:

- Whether or not file sharing is supported?
- If you share a file to another user, are you able to modify the file? Is he able to modify the file?
- Can the sharing be revoked?

All the six systems support file sharing. For Dropbox, MEGA and Google Drive, the user who are shared a file to can have read-only, read-write or full access to the file. In Wuala, only full access is supported for shared files. In Tahoe-LAFS, file sharing is enabled by passing the capability to other users. In Tamias, file sharing is based on user identities.

Dropbox, Google Drive, Tamias supports privilege revocation. In MEGA, files and folders can be shared using a URL with a decryption key, and if files and folders are shared in this way, there is currently no way to revoke it. Folders in MEGA can be shared in another way, by assigning a contact for the folder to be shared with, and in this way, the sharing can be revoked. In Wuala, only folders can be shared. Revocation is possible in Wuala. In Tahoe-LAFS, sharing is achieved by sending the capability and thus cannot be revoked because it is not possible to change a capability of an encrypted file.

Table 2.2: Summarization of other system features

	Dropbox	MEGA	Wuala	Google Drive	Tahoe-LAFS	Tamias
Chunking	Yes	To be studied	Yes	Yes	Yes	Yes
Bundling	Yes	To be studied	No	No	No	No
Compression	Yes	To be studied	No	Yes	No	No
Deduplication	Yes, among all users	Yes, must be same file encrypted with same key	Yes. Under user's personal cloud; by recognizing the same decryption key	No	Yes, but only with clients who have the same convergence secret	Yes
Delta-encoding	Yes	To be studied	No	No	To be studied	To be studied
File sharing	Yes	Yes	Yes. Only folders can be shared	Yes	Yes	Yes
Open source	No	UCE is open sourced	No	No	Yes	Yes

2.4.7 Open source

Open source means the source code of the system is accessible to public. We care about it because if a system is open-sourced, we can know exactly how the internal of the system works, and we can verify whether what they claimed is true or not. Among the six systems, only Tahoe-LAFS and Tamias are open-sourced. They use the licence GNU General Public License, version 2 [17].

2.4.8 Summary

We can summarize the above features of the six systems described above into Table 2.2:

Chapter 3
BenchCloud - a benchmarking tool for cloud storage systems

In this chapter we will introduce BenchCloud, which is a tool developed for this Master thesis and is aimed at providing a convenient and flexible way to benchmark cloud storage systems.

3.1 Background

3.1.1 The requirements for a cloud storage system benchmarking tool

As we mentioned in Chapter 2, there are many cloud storage systems nowadays, and new players keep coming to the market. Therefore we need some guidance to choose the appropriate system that satisfies our requirements best. Since many cloud storage systems share similar functions, the performance of the systems is a big issue we need to take into consideration, and that is why we need to benchmark them. Some possible scenarios are listed below where a benchmark is considered helpful.

- **Choose the fastest cloud storage system for daily use.** Suppose a user is going to have a try with some cloud storage system to store his files in the cloud and synchronize the files between the laptops in his home and office, and his main concern is that the service should upload/download files as fast as possible. Since different cloud systems have different network bandwidth and different locations for their data centers, a benchmarking has to be made to determine which system has the best performance in file uploading and downloading.
- **Find out the best way to make use of a cloud storage system as a backend storage system for web and mobile applications.** With the development of SaaS [4] and mobile computing [13], many web applications we use nowadays store their user data in the user's own personal cloud system, instead of storing in a dedicated server maintained by the developer himself. There are some advantages of this kind of cloud-based web applications. First,

the developer of the application does not need to maintain any dedicated storage servers, so the cost can be reduced greatly. Second, because the data is saved in the users' own cloud space maintained by a trusted cloud storage service provider, the user can feel safe about their data, which will make the application more attractive to the users who care much about the safety of their data. Third, the data stored in the cloud enjoy some extra functions provided by the cloud such as file synchronizing and sharing. One example of such application is site44[1], which is a dropbox-based web application that can turn Dropbox folders into publicly accessible websites. As a developer of a cloud-storage-service-based application, he may need to know the best way to use the service. For example, can multithreading be used when uploading files to the cloud? And if the answer is yes, how many threads should be used to achieve the best performance? And if we need to get the best performance when uploading a large data, should the data be split into smaller files before uploaded? To answer such questions, a benchmark is often considered helpful for comparing the performance of different strategies of using the cloud storage service.

- **Study the performance of cloud storage systems for a special use case.** Most of the cloud storage systems we have today are designed for normal daily uses like storing photos, music tracks, and documents in a casual way. However, as a general purpose cloud storage system, it may be used in some scenarios other than daily casual usage. For example, it is possible to use a cloud storage service as a backend storage system of a Internet of Things project with many sensors keep capturing data from environment and sending to the backend concurrently. Such use case differ from others with its special character of generating large amounts of small files and uploading concurrently. To study whether a cloud storage system can be used in such scenario and the performance, a benchmark is always needed.

In a word, benchmarking cloud storage systems is helpful in many ways. In reality, we can make ad-hoc benchmarking manually, but it will be very time consuming and it is not easy to reproduce the benchmarking process. Besides, writing scripts and programming is usually not avoidable if one needs to perform complex benchmarking, like multithreaded uploading with random file generation. Because these shortcomings of manual benchmarking, an automatic benchmarking tool is the key to improve the efficiency of benchmarking tasks, and that is why BenchCloud is made.

[1] http://www.site44.com/

3.1.2 Existing studies and tools for benchmarking cloud storage systems

[3] discussed the requirements for a good benchmark on cloud services in general. [9] presented the methodology to study cloud stogare systems and studied various features of five popular cloud storage systems. [16] presented CloudCmp, a systematic comparator of the performance and cost of cloud providers. [26] conducted comprehensive experiments on several representative cloud-based data management systems like HBase[2] and Cassandra[3] to explore relative performance of different approaches. [24] provides benchmarks on Amazon Elastic Computing (EC2)[4] services in terms of CPU performance, memory performance and so on. [27] presents a load tester for Web 2.0 applications on a variety of Amazon EC2 configurations.

[6] presents Yahoo! Cloud Serving Benchmark (YCSB), a benchmarking tool/framework for cloud storage systems. However, the target of YCSB is to benchmark the read/write performance of large-scale distributed database systems like BigTable, HBase and Cassandra, and can not be used directly for benchmarking the personal cloud storage systems like Dropbox and Google Drive.

[36] presents a benchmarking tool for cloud storage developed by Intel, however the target of it is cloud object storage systems like Amazon S3 [25] and OpenStack Swift [23], which are different from the systems that are under discussion in this thesis.

Apache Benchmark[5], or "ab" for short, is a popular tool for benchmarking HTTP server, and in theory it can be used to benchmark anything that is accessible from a HTTP(S) server. However, it lacks important features if it is used as a benchmarking tool for cloud storage systems, like service authentication, API wrapper and file generation.

Until the time this thesis is written, we have not found a mature tool aimed at benchmarking cloud storage systems like Dropbox with convenience and flexibility, which does not require testers to have knowledge of programming. Because of this, we consider the development of BenchCloud is a nice-to-have try in this area.

3.2 Software Requirements

From a software engineering perspective, it is a good practise to think clearly about the requirements of a product before trying to design and implement it. The functional

[2] https://hbase.apache.org/
[3] http://cassandra.apache.org/
[4] https://aws.amazon.com/ec2/
[5] http://httpd.apache.org/docs/2.2/programs/ab.html

requirements of BenchCloud are listed as follows:

1. Cloud service authentication and authorization.
2. Support different cloud storage service providers/products.
3. Support different kinds of file operation, including uploading, downloading, and sharing.
4. Support different file generators to generate files with different patterns.
5. Support for multithreaded operations.
6. Make statistics of benchmarking results.
7. Log and save benchmarking results automatically.
8. Capture network packets during the process of benchmarking.
9. Be able to test on both web APIs and native clients of cloud storage systems.

3.3 Design Goals

3.3.1 Flexibility

By flexibility we mean BenchCloud can be used for different purposes. The requirement for flexibility is derived from the fact that different benchmarking tasks have different target and purpose, and a general benchmarking tool should be able to support as many kinds of benchmarking tasks as possible. To achieve this, BenchCloud should be highly configurable and extensible.

Highly configurable means that a user is able to customize the settings of a benchmarking task in details. Such settings can include for example the cloud storage system he would like to benchmark, the type of operations to be performed (download or upload, etc.), the number of operations to be performed, the number of threads used to perform the operations, and so on.

Besides, BenchCloud should be extensible, which means it should be easy to extend the different parts of BenchCloud. As we mentioned, the purposes for different benchmarking tasks can be very different, and it is possible that even a highly configurable task cannot satisfy the needs of the user. For example, if a user needs to test a new cloud storage system which is not supported by BenchCloud, he should be able to extend BenchCloud easily to make it support the new service, without having to modify the existing parts of BenchCloud.

3.3.2 Usablity

By usability we mean BenchCloud should provide an easy way to use for most users. Although BenchCloud is written in Python, a user should not be forced to have knowledge of programming in Python before using BenchCloud. To achieve that, BenchCloud was designed to support configuration file, and almost every settings

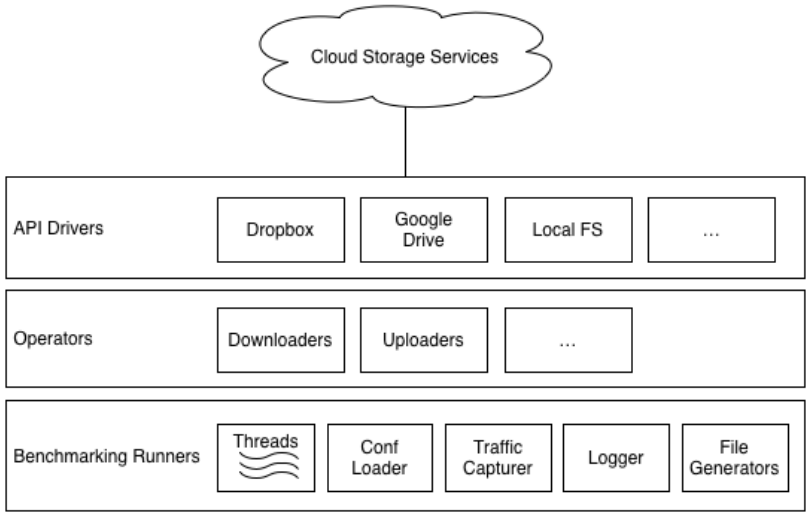


Figure 3.1: System Architecture of BenchCloud

of a benchmark can be set in the configuration file, which is a plain text file that is quite easy to understand and produce.

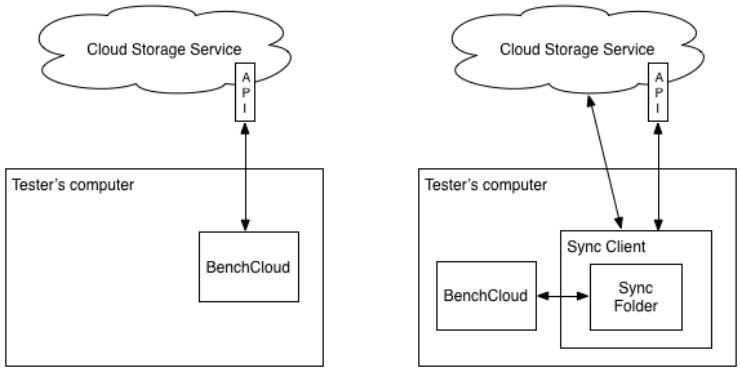
3.4 System Architecture

BenchCloud takes a layered architecture. As is shown in Figure 3.1, it has three main layers.

3.4.1 The API Driver Layer

The API Driver layer provides communication end points to cloud storage services. It has cloud service wrappers to be invoked by the Operators layer. A cloud service wrapper communicates with cloud storage services via RESTful APIs [12], and provides functions like service authentication and authorization, file metadata acquiring, file upload and download, file sharing, etc.

Local FS driver is a special driver for “uploading”/”downloading” files to/from the tester’s local file system. Unlike other drivers that make use of web APIs opened from cloud storage systems, Local FS driver just perform normal file copy operations in the scope of local file system. Local FS driver is used in the scenario when you do not want to test against web APIs but to the native clients of some cloud storage systems. Such systems provide synchronization client running on users’ computers and synchronizes local files (usually in a specific synchronized folder) to the cloud.



(a) Test via web APIs

(b) Test via synchronization client

Figure 3.2: Two styles of test architecture

Such client may have interesting features that can not be found by testing against web APIs directly, and by "uploading" files to the synchronized folders and let the synchronization client does the processing and real uploading operation, we can study in some way how the client works and what kinds of optimization it performs.

Based on whether web API or client is to be tested, the high level testing architecture can be divided into two styles, shown in Figure 3.2.

3.4.2 The Operators Layer

The Operators layer provides a higher abstraction based on the functions provided by the API Drivers layer. It provides generic downloaders and uploaders which are not bound to any specific API drivers.

3.4.3 The Benchmarking Runner Layer

The Benchmarking Runner Layer is responsible for parsing and loading configuration files and executing the benchmark based on the configuration. The logger is responsible for logging the exact steps and time consumed in detail when performing benchmarks. A file generator is a utility used by benchmarking runners to generator files based on given configuration and is often used when performing benchmarks for uploading files.

There are four types of file generators providing different kinds of file content patterns:

1. **RandomFileGenerator**. It generates files with random content, which are hard to be compressed efficiently and are highly impossible to have the same content with other generated files.
2. **IdenticalFileGenerator**. An IdenticalFileGenerator generates a series of files with exactly the same content. It plays a key role in testing the file deduplication feature of a cloud storage system, described in 4.2.6.
3. **SparseFileGenerator**. It generates files with sparse content. Sparse content is content with repeated strings. Files generated by a SparseFileGenerator can be effectively compressed with a high compression rate. SparseFileGenerator plays a key role in testing the file compression feature of a synchronization client, descirbed in 4.2.6.
4. **DeltaFileGenerator**. A DeltaFileGenerator generates a series of files with the same size and a certain amount of identical content. The other parts of the files are random content and are not identical. DeltaFileGenerator plays a key role in testing the delta encoding feature of a synchronization client, described in 4.2.6.

A traffic capturer is provided in the Benchmarking Layer to capture and dump network packets during a benchmark. The data format of the resulting dump file is PCAP[6], which is a very common format for recording network packets and it can be read and analyzed by many packet capturing and analysis tools, like Wireshark[7]. The PCAP format record the packets generated in detail, and thus can be used in post-analysis to study the characters of the network traffic.

3.5 Cooperation with other tools

As described in the previous section, BenchCloud provides a way to benchmark cloud storage systems easily and can log the time consumed in each steps during the benchmark process. However, users may need more information apart from time consumption and they may use some other tools to study the packets captured to get more insights. BenchCloud does not provide a detailed packet analysis tool, because there are mature tools for this purposes. In this section we introduce some packet analysis tools that can be used along with BenchCloud.

3.5.1 Wireshark

Wireshark [21] is an open-source network packet analyser. It is a powerful tool for troubleshooting and analysing network communications. It is cross-platform, which means it can be installed on GNU/Linux, Mac OS X, Solaris, Microsoft Windows, etc. Wireshark has both a graphics user interface and a command line tool. Some important features of Wireshark includes:

[6] http://en.wikipedia.org/wiki/Pcap
[7] http://www.wireshark.org/

1. Capture live packets from a network interface
2. Import/Export packets
3. Show packet data in a detailed and structured way
4. Show the protocol-specific information of packets
5. Filter packets according to various rules
6. Make various kinds of statistics

3.5.2 tcpdump

tcpdump [15] is a packet analysis tool similar to Wireshark. But unlike wireshark, tcpdump has only a command line tool and lacks graphics front-end. tcpdump can run on most Unix-like systems and is often distributed along with these systems. There is also a Microsoft Windows port of tcpdump called WinPcap[8].

3.6 Open Source

The best thing we can do to BenchCloud is to make it contribute it to the open source community. The source code[9] of BenchCloud is hosted at GitHub and it is open sourced under the Apache License Version 2.0[10].

3.7 Possible Improvements

While BenchCloud can be used for many purposes, some useful functions can also be built into BenchCloud to make it more convenient and efficient, such as:

1. More types of file generators can be made, for example, a JPEG file generator to generate JPEG image files. The right type of file generator can be the key to study some special features of a system.
2. Drivers for more cloud storage systems. So far Dropbox, Google Drive and Mega are supported and we would like to see more.
3. More realistic ways to simulate the sequence of events. So far, a fixed sleeping time can be assigned between consecutive operations. This can simulate events that occur in a fixed interval. However, in real scenarios the time between consecutive operations can be variant, and may follow some statistical distribution. Thus it is useful to make BenchCloud support variant time intervals.

Besides, so far BenchCloud can not be used directly to test security and privacy features of cloud storage systems, although it may be used as a tool to help with

[8] http://www.winpcap.org/

[9] The source code of BenchCloud can be found at https://github.com/zenja/benchmarking-cloud-storage-systems.

[10] http://www.apache.org/licenses/LICENSE-2.0.html

the process. Considering the importance of security and privacy, it would be nice to support the testing for it.

Chapter 4
Use BenchCloud to Analysis Cloud Storage Systems

In this chapter, we introduce the basic steps to benchmark cloud storage systems with BenchCloud, and give case studies of using BenchCloud in different kinds of scenarios.

4.1 Benchmarking Process

Assuming BenchCloud is already installed correctly, the typical steps to benchmark a cloud storage system with BenchCloud are:

Step 1 Make the configuration file for the benchmark.

a) Choose the service driver for the system to be benchmarked. If BenchCloud does not have a driver for the system, a new one should be made before you can start benchmarking.

b) Choose the operation to be performed (uploading, downloading, etc.).

c) Select the appropriate benchmark runner for the operation. Currently there are two runners support by BenchCloud, UploadTaskRunner and DownloadTaskRunner, to run benchmarks for uploading operations and downloading operations separately. If you choose UploadTaskRunner, you also need to specify the file generator to be used to generate files, and some attributes like the size of generated files, the prefix/suffix of file names, the remote directory, etc.

d) Choose the location where the logging file will be saved to.

e) Set the configuration for concurrency by setting the number of threads to be used.

f) Write the above configuration into a configuration file, along with some other settings such as the name of the benchmark, the number of operations, whether to sleep between operations, etc.

Step 2 Execute BenchCloud via command line. You may need to authorize BenchCloud to access your data in the cloud first.

Step 3 Wait for the benchmark to finish, and check the statistics, logs, and possibly captured packet data.

In BenchCloud, most of the settings of a benchmark are specified by a configuration file. And because a configuration file defines a benchmark, it is very easy to reproduce a benchmark. An example of configuration file for an uploading benchmark is shown in Source code 4.1. It is a benchmark for uploading 10 binary files of size 102400 bytes for each with random data to the remote directory "/benchmark-test" of dropbox, using 3 threads to upload concurrently.

Source code 4.1 Sample configuration file for a benchmark uploading files to Dropbox

```
[test]
description = Upload random binary files to dropbox
times = 10
sleep = False
#sleep_seconds = 1

[logging]
enabled = True
log_file = /tmp/benchmarking.log

[driver]
class = benchcloud.drivers.dropbox_driver.DropboxDriver

[operator]
class = benchcloud.operators.uploader.Uploader
operation_method = upload
operation_method_params = {
  "local_filename": "<generated_file.name>", "remote_dir": "/benchmark-test"}

[file_generator]
class = benchcloud.file_generators.random_file_generator.RandomFileGenerator
#directory =
prefix = benchmarking-
#suffix = .test
delete = True
size = 102400

[concurrent]
threads = 3
```

A typical configuration file for a downloading task is shown in Source code 4.2.

Source code 4.2 Sample configuration file for a benchmark downloading files to Dropbox

```
[test]
description = Download all files (not recursively)
 in a remote directory to a local directory
sleep = False
#sleep_seconds = 1
remote_dir = /CV_CL
local_dir = /tmp/CV_CL_test

[logging]
enabled = True
log_file = /tmp/benchmarking-dropbox-download.log

[driver]
class = benchcloud.drivers.dropbox_driver.DropboxDriver

[concurrent]
threads = 3
```

Example commands to start benchmarking based on a configuration file and to ask for authorization before benchmarking is shown in Source code 4.3.

Source code 4.3 Commands for running a benchmark and asking authorization for cloud service

```
cd <BENCHCLOUD_HOME>
python -m benchcloud.benchcloud <RUNNER> -f <CONFIGURATION_FILE> -a
```

4.2 Benchmarking Results and Analysis

In this section, we provide several examples of benchmark and analysis of Dropbox with BenchCloud. Although the experimental target is Dropbox, the same process should be easily made on other cloud storage systems.

4.2.1 Environment of benchmarking

The environment for running a benchmark should be specified beforehand, for it will greatly affect the experienced performance of cloud storage systems. The most important environment for benchmarking cloud storage systems includes the

Table 4.1: Environment of benchmarking

Location	United States, California, San Francisco
Public IP Address	198.199.97.195
Operation System	Ubuntu 12.10
CPU	1 Core, 2 GHz
RAM	512 MB
Disk	20GB SSD Disk
Network bandwidth[3]	Download: 741.78 Mbits/s Upload: 164.75 Mbits/s
Python Version	2.7.3

location of the client, the hardware and operating system of the client machine, and the network bandwidth of the client machine. By client we mean the machine to send API requests to cloud storage services. Table 4.1 specifies our benchmarking environment. The client is a VPS[1] from DigitalOcean[2]. The server is dedicated to run the benchmarking and is not used for any other network communications during the benchmarks.

4.2.2 The impact of concurrency on the performance of file uploading/downloading

As a software developer, one may need to build applications for file uploading/downloading with some cloud storage system like Dropbox. Choosing a proper number of threads to be used may increase the performance of file operations and thus optimize the user experience of the application. In this section we will study the impact of concurrency on the performance of file uploading and downloading for Dropbox. The benchmarks divides into two groups. In the first group, 50 files of size 100 KB for each will be uploaded to and downloaded from Dropbox, with different number of threads, i.e. 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 threads for each benchmark. The second group is the same as the first group except that the size of each file is 1 MB instead of 100 KB. The reason of benchmarking on both 100 KB and 1 MB files is that smaller files will be influenced more by the cloud system's latency than throughput, and larger files will be influenced more by throughput than latency, and we would like to check the performance in both scenarios to try to get a general conclusion.

[1] http://en.wikipedia.org/wiki/VPS

[2] https://www.digitalocean.com/

[3] The network bandwidth is tested via speedtest.net, against a server located in San Jose, CA, USA, which is near to the location of the testing client.

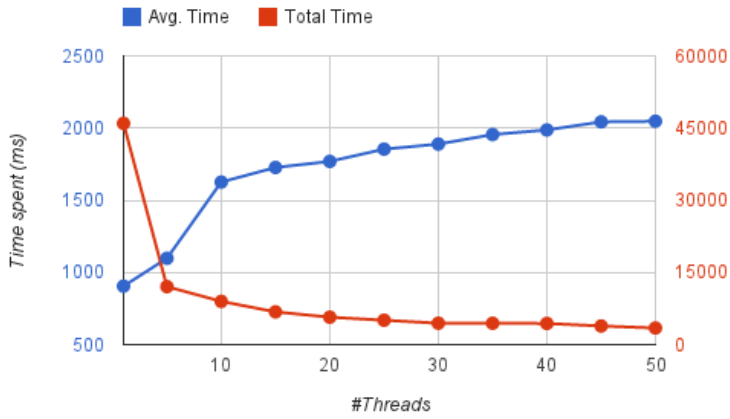


Figure 4.1: Time spent for uploading 50 files of 100 KB each with different number of threads

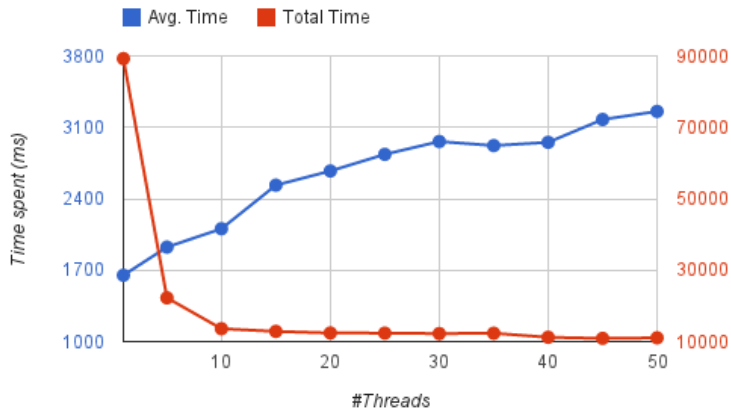


Figure 4.2: Time spent for uploading 50 files of 1 MB each with different number of threads

A sample configuration file for the benchmarks is A.1. The performance of uploading for Group 1 and Group 2 is plotted in Figure 4.1 and Figure 4.2: The blue line plots the average time spent in milliseconds for each upload/download operation, and the red line plots the total time spent for all the 50 operations.

From Figure 4.1 and Figure 4.2 we can get some interesting results:

1. The average time for each file operation increases with the number of threads

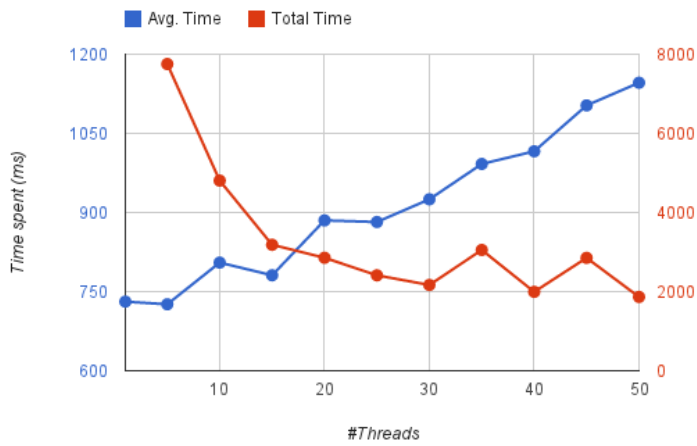


Figure 4.3: Time spent for downloading 50 files of 100 KB each with different number of threads

used. This is possibly due to the fact that waiting time for thread scheduler to wake up a certain blocked thread increases with the number of threads.

2. The total time spent decreases significantly once multithreading is used, even when the number of threads is just five. This tells us that the use of multithreading will generally decrease the total time spent significantly, compared to single-threaded uploading.
3. For Group 1 with 100 KB files, the total time spent did not decrease much after around 25 threads; for Group 2 with 1 MB files, the total time spent did not decrease much after around 10 threads. And we can tell from the graph that there was diminishing marginal utility of increasing the number of threads.

Figure 4.3 and Figure 4.4 are corresponding results for file downloading. A sample configuration file for the benchmarks is A.2. From the figures we can get similar conclusions. Note that there are some jitters in the graph data, and it may be caused by network congestion.

For application developers, one may be more interested in the total time than the average time, and the suggestion based on the observed results is to use multithreading. Although we can tell from the above results that the more threads used the shorter time we will get, the effect of diminishing marginal utility cannot be ignored, because threads are system resources that will introduce overhead for creation and maintenance, and moreover many operating systems have limits on the maximum number of threads that can exist at the same time.

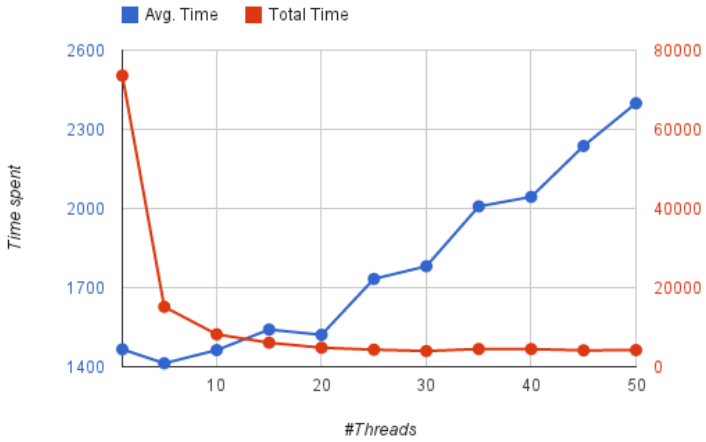


Figure 4.4: Time spent for downloading 50 files of 1 MB each with different number of threads

4.2.3 The impact of file size on the performance of file uploading/downloading

Sometimes we may want to know if a certain cloud storage system can handle both small files and large files well. In this sections we present the results of uploading and downloading files of different sizes in Dropbox, and by doing so we can find if Dropbox cannot handle certain size of files well.

A sample configuration file for the benchmarks is A.3. We uploaded/downloaded 10 files of different sizes (1 KB, 10 KB, 100 KB, 1 MB, 5 MB, 10 MB, 15 MB, 20 MB) with only one thread for each benchmark. Figure 4.5 shows the time spent for uploading files of different sizes to Dropbox, and Figure 4.6 shows the time spent for downloading.

From the graphs we can see that there were no circumstances that files of certain sizes took much more time or much less time than files of other sizes. So basically we can say Dropbox is able to handle files of different sizes (from 1 KB to 20 MB) reasonably.

4.2.4 Study the feasibility of using cloud storage system as a storage backend for IoT systems

Engineers and scientists from various backgrounds may make use of cloud computing to built more efficient and reliable systems and solutions. Cloud storage systems

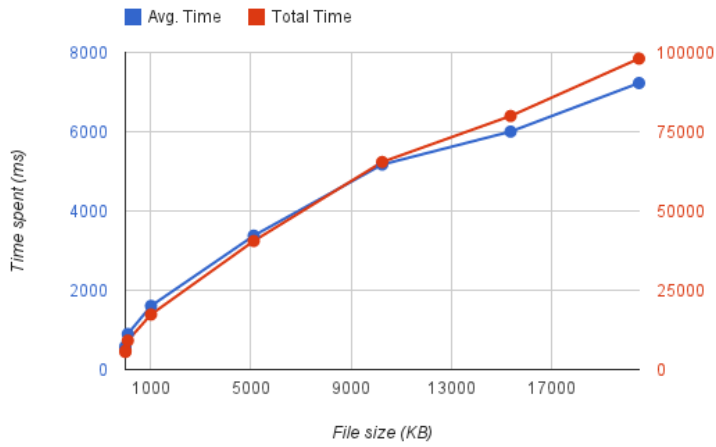


Figure 4.5: Time spent for uploading files of different sizes to Dropbox

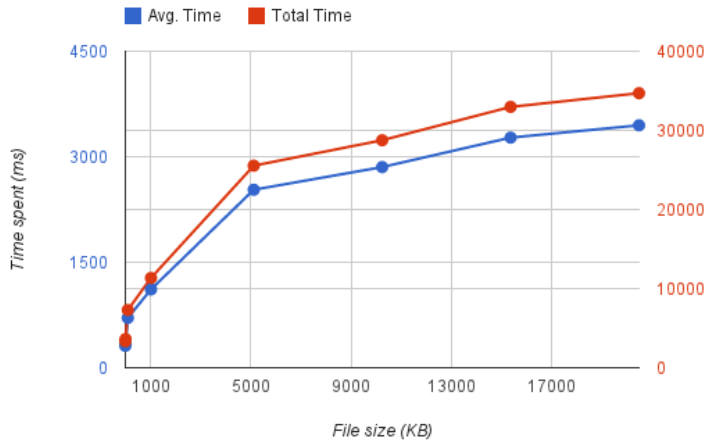


Figure 4.6: Time spent for downloading files of different sizes to Dropbox

provide such a possibility to relive engineers and scientists from building and maintaining backend storage systems of their own. For example, some Internet of Things (IoT) systems may need to consistently and concurrently store information gathered from many sensors. The storage servers for such IoT systems should at least satisfy the following requirements:

1. The free disk space is big enough to store incoming data
2. The number of files stored will not exceed the limit of the storage system
3. The storage server can handle concurrent uploads and will not reject connections

Building such systems may not be an easy task, depending on the scale of the IoT system. Because of this, migrating the storage system to the cloud may be a cost-saving choice. However, before migration we have to find out whether a cloud storage system satisfies the above requirements or not.

We now assume that we have an IoT system consisting of many sensors keeping collecting temperature data from various locations at certain fixed interval and sending each data in the form of small files to the storage system to be stored for future analysis. Such system is shown in Figure 4.7. We want to test the feasibility of using Dropbox as the storage system. We first check if Dropbox satisfies the requirements described before:

1. Does Dropbox has enough space for storing data? Yes, Dropbox has different pricing plans for different size of storage. [4]
2. Does Dropbox has limit on the maximum number of files that can be stored? No, there are no such limit if there is any free space. [5]
3. Is Dropbox able to handle the concurrency generated by the sensors? We do not know, and it depends on different parameters of the system: the size of generated data, the interval between every data transmission for each sensor, and the number of sensors of the system.

We use BenchCloud to simulate benchmarks for the temperature data collection system. Each uploading thread simulates an individual sensor. The interval between consecutive uploads is simulated by telling the thread to sleep a while between uploads. We assume the size of each file storing temperature data is 5 KB. A sample benchmark configuration file simulating 100 sensors sending 1000 files in total to Dropbox with an interval of 5 seconds between each operations is shown in A.5.

We tried benchmarking with different number of simulated sensors and different simulated intervals. Table 4.2 shows the benchmarking results. Note that the results were for the worst case, because in all these benchmarks, the concurrent threads would start uploading files simultaneously. And because the size of files uploaded was the same, the time spent for each file upload would take the same amount of time

[4] https://www.dropbox.com/plans
[5] https://www.dropbox.com/help/5/en

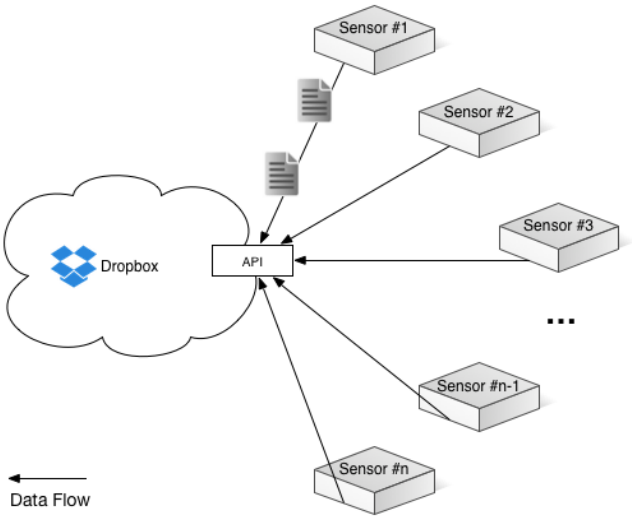


Figure 4.7: Overview of a temperature data collection system

approximately, the next round of file uploads would happen almost simultaneously, so and forth till the end of the benchmark. So Table 4.2 actually represents the worst case when the maximum concurrency was achieved, i.e. almost all the simulated sensors would send data to Dropbox at the same time. The column "#lost files" represents the number of files failed to be saved in Dropbox during the benchmark. The loss of files was caused by rejection from the Dropbox server due to the request rate exceeding maximum limit.

We can see from the result that the loss rate increased when the number of sensors increased and when the interval decreased. In this worst case, heavy loss rate was experienced in many benchmarks.

To simulate the average case where each sensor would not try to send data at the same time, we made another series of benchmarks based on the previous benchmarks. We introduced a random amount of sleep time for each thread before uploading the first file. Table 4.3 shows the result. The column "Thread sleep time (s)" represents the range of random sleep time for every thread. The column "Loss rate decrease" represents the amount of decrease of file loss rate comparing to the the loss rate in Table 4.2.

We can see from Table 4.3 that the loss rate decreased significantly for the benchmarks with interval of 10 seconds and 60 seconds. The random sleep time does not affect the loss rate much for the benchmarks with interval of 1 second.

Table 4.2: Benchmarking results for simulated sensor data collection system

#Files in total	#Sensors/threads	#Interval (s)	#Lost files	Loss rate
100	10	1	0	0.00 %
100	10	10	5	5.00 %
100	10	60	5	5.00 %
300	30	1	166	55.33 %
300	30	10	52	17.33 %
300	30	60	57	19.00 %
500	50	1	386	77.20 %
500	50	10	195	39.00 %
500	50	60	166	33.20 %
800	80	1	748	93.50 %
800	80	10	656	82.00 %
800	80	60	493	61.63 %
1000	100	1	967	96.70 %
1000	100	10	795	79.50 %
1000	100	60	678	67.80 %

Table 4.3: Benchmarking results for simulated sensor data collection system (worst case)

#Files in total	#Sensors/ threads	#Interval (s)	Thread sleep time (s)	#Lost files	Loss rate	Loss rate decrease
100	10	1	$0 \sim 5$	0	0.00 %	NA
100	10	10	$0 \sim 10$	0	0.00 %	100 %
100	10	60	$0 \sim 60$	0	0.00 %	100 %
300	30	1	$0 \sim 5$	157	52.33 %	5 %
300	30	10	$0 \sim 10$	1	0.33 %	98 %
300	30	60	$0 \sim 60$	0	0.00 %	100 %
500	50	1	$0 \sim 5$	366	73.20 %	5 %
500	50	10	$0 \sim 10$	19	3.80 %	90 %
500	50	60	$0 \sim 60$	0	0.00 %	100 %
800	80	1	$0 \sim 5$	734	91.75 %	2 %
800	80	10	$0 \sim 10$	194	24.25 %	70 %
800	80	60	$0 \sim 60$	0	0.00 %	100 %
1000	100	1	$0 \sim 5$	955	95.50 %	1 %
1000	100	10	$0 \sim 10$	355	35.50 %	55 %
1000	100	60	$0 \sim 60$	0	0.00 %	100 %

Table 4.4: Hardware and software details of downloader node

Location	Amsterdam
Public IP Address	188.226.158.203
Operation System	Ubuntu 12.10
CPU	1 Core, 2 GHz
RAM	512 MB
Disk	20GB SSD Disk
Network bandwidth[6]	Download: 330.37 Mbits/s Upload: 99.69 Mbits/s
Python Version	2.7.3

Table 4.2 and Table 4.3 represents the benchmarking results for the worst case and average case of the simulated weather data collection system. Based on the results, the scale and parameters of the system, and the requirements for the reliability of storage server, engineers and scientists can make a decision about whether or not Dropbox shall be used as the storage backend for such an IoT system.

4.2.5 Study the readiness time for uploaded files

Readiness time is normally defined as the length of time required to obtain a stabilized system ready to perform its intended function [22]. In the context of this thesis, we define the readiness time of a file as the length of time from the end of the file uploading to the time the file is ready for downloading. Based on the internal architecture and implementation of a cloud storage system, readiness time may be different depending on the size of the file and the location of upload/download requests. Short average readiness time is a good feature of a cloud storage system because users will not have to wait a long time before their files are ready to be downloaded. Experiments to study the readiness time of dropbox are described in details in this section.

The node described in 3.4.3 was used as the uploader to upload files to Dropbox. Another node was introduced in the experiments to act as the downloader to download the same files from Dropbox. The details of the downloader node is described in Table 4.4.

The steps performed to test the readiness time are listed as follows:

Step 1 The time of uploader node and downloader node was synchronized so that the time difference of the two nodes was within one second.

[6]The network bandwidth is tested via speedtest.net, against a server located in Dronten, Amsterdam, which is near to the location of the testing client.

Table 4.5: Readiness time of files with different size

File size	Readiness time (ms)
200 MB	255
100 MB	21
50 MB	-125
10 MB	102
1 MB	-83
100 KB	412
10 KB	285
1 KB	-198

Step 2 A script was made based on BenchCloud to try downloading a target file from Dropbox repeatedly with a time interval of 0.5 second, until the file was ready and downloaded to local file system. Timestamps of each operation were recorded during the process. See A.6 for the script.

Step 3 The downloader node started running the script described in Step 2.

Step 4 The uploader node started to upload the target file to Dropbox.

Step 5 After the target file was downloaded in the downloader node, the readiness time of the file, i.e. the time between the file being ready to be downloaded and the file being uploaded successfully, was calculated.

Step 6 Step 1 and Step 3 to Step 5 were repeated, using files of different sizes.

The results of experiments are shown in Table 4.5. All the readiness time records were within 0.5 second, which was less than the time difference of the uploader node and downloader node. This means the files were instantly ready to be downloaded once they were uploaded to the server. Note there were some negative readiness time records, and this might be caused by the time difference between the two nodes.

4.2.6 Study the features of synchronization clients

As described in 3.4.1, BenchCloud can be tested against not only the web APIs but also the synchronization client of a cloud storage system. Synchronization clients may have some interesting features that are not provided by public web APIs. In this section, we describe the results of experiments for studying whether the synchronization client of Dropbox has the following features described in 2.4: file compression, delta encoding, and file deduplication.

Table 4.6: Hardware and software details of the machine for testing synchronization client

Machine type	MacBook Air 13', 2013 Mid
Location	Lianyungang, Jiangsu, China
Operation System	Mac OS X 10.9.2 (13C1021)
CPU	1.7 GHz, Intel Core i7
RAM	8 GB, 1600 MHz, DDR3
Disk	251 GB, SSD Disk
Network bandwidth[7]	Download: 17.34 Mbits/s Upload: 2.31 Mbits/s
Python Version	2.7.5

A computer running a desktop operating system was used. The details of it were described in Table 4.6. The version of Dropbox synchronization client used was v2.6.33.

To test against synchronization client, the local file system driver is used instead of real Dropbox driver to "upload"/"download" generated files to/from the local synchronization folder. Once Dropbox detects file changes in the folder, it will synchronize the changes between local folder and the cloud.

File compression

To test whether Dropbox client uses file compression, the following logical steps were performed:

Step 1 A file of certain size was generated using SparseFileGenerator (see 3.4.3).
Step 2 The network packet sniffer was launched to record network traffic.
Step 3 The generated file was "uploaded" to the synchronization folder.
Step 4 After the file was synchronized by the client, the network packet sniffer was stopped.
Step 5 Step 1 to Step 4 were repeated with different file sizes (1 MB, 10 MB, 100 MB, 300 MB, 500 MB).

All the above steps can be done with the help of BenchCloud by writing benchmarking configuration files. A sample configuration file for the benchmarks is A.7. The size of traffic is analysed using Wireshark. The results of experiments are shown in Figure 4.8. The traffic size (Y-axis) records the sum of sizes of all packets sent to Dropbox servers in the process of file uploading. From the result we can see the

[7]The network bandwidth is tested via speedtest.net, against a server located in Wuxi, China, which is near to the location of the testing client.

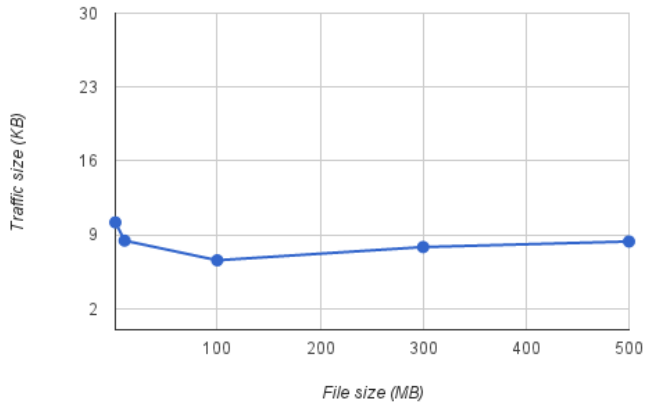


Figure 4.8: The amount of traffic generated while synchronizing sparse files

traffic size was greatly smaller than the file size, which means the files were definitely compressed before uploaded to the cloud. Besides, because the amount of information stored in the files were the same, the resulting amount of traffic was also almost the same. The difference between the maximum and the minimum traffic size is 3.6 KB. This may be caused by unstable network (packet re-transmission) and can be ignored.

Delta encoding

To test whether Dropbox client uses delta encoding, the following logical steps were performed:

- Step 1 Five files of 10 MB were generated by a DeltaFileGenerator, with a certain percentage of identical content (see 3.4.3).
- Step 2 The network packet sniffer was launched to record network traffic.
- Step 3 The files were put into synchronization folder one by one, with enough sleeping time in between to make sure a file was synchronized to the cloud before the next file was put into the folder. The files were put into the same location/folder with the same file name, so a file will be over-written by the next file.
- Step 4 After the last file was synchronized by the client, the network packet sniffer was stopped.
- Step 5 Step 1 to Step 4 were repeated with different percentage of identical part.

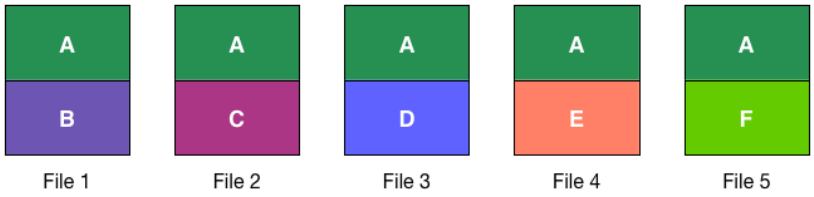


Figure 4.9: Content structure of five files with 50% of identical part

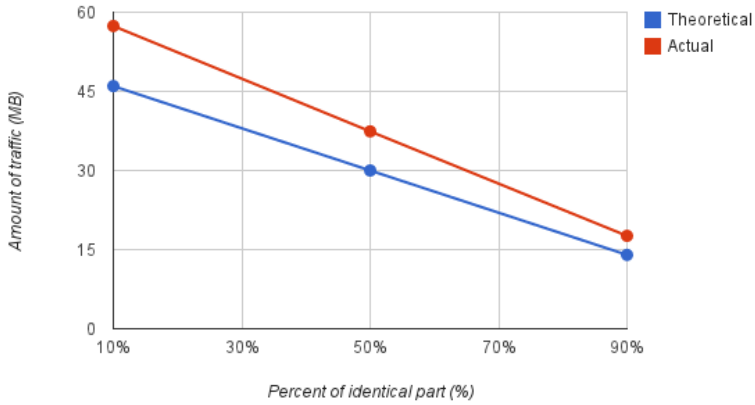


Figure 4.10: Theoretical and actual amount of traffic generated for uploading delta-encoded files with different percentage of identical part

Again, all the steps can be done with the help of BenchCloud, and the size of traffic is analysed using Wireshark. A sample configuration file for the benchmarks is A.8.

Take the group of files with 50% identical content as an example. The structure of the files are shown in Figure 4.9. Part A to F are file blocks of $10MB \times 50\% = 5MB$. All the five files have the same file block, i.e. block A, while the other parts are different from each other. Assume delta encoding is used, the amount of traffic sent to Dropbox server for all the five files should be:

$$(10MB \times 50\%) + 4 \times (10MB \times (1 - 50\%)) = 30MB$$

Table 4.7: Amount of traffic generated while uploading files with exactly the same content

File size	#Files	Traffic size (MB)
20 MB	3	25.58

File deduplication

To test whether Dropbox supports file deduplication, the following logical steps were performed:

Step 1 Three files of 20 MB with identical content were generated with an IdenticalFileGenerator (see 3.4.3).

Step 2 The network packet sniffer was launched to record network traffic.

Step 3 The three files were put to synchronization folder one by one, with enough sleeping time in between to make sure a file was synchronized to the cloud before the next file was put into the folder.

Step 4 After the last file was synchronized by the client, the network packet sniffer was stopped.

Again, all the steps can be done with the help of BenchCloud, and the size of traffic is analysed using Wireshark. A sample configuration file for the benchmarks is A.9. The amount of traffic sent to Dropbox servers was analysed with Wireshark. Table 4.7 shows the results. If file deduplication was performed, the amount of traffic should be at least $20MB \times 3 = 60MB$. However, we can see from Table 4.7 that the amount of traffic was around 26 MB, which was much smaller than 60 MB. So we can conclude that file deduplication was supported in Dropbox synchronization client.

4.2.7 Summary

In this chapter we described the basic steps to benchmark cloud storage systems with BenchCloud. Several experiments/benchmarks were conducted to analysis different features of Dropbox by testing against both the web APIs and the synchronization client. With the help of BenchCloud, most of the experiments did not require knowledge of programming and benchmarks were designed by configuration files. The results of the experiments proved that BenchCloud was capable of analysing a wide range of features for cloud storage systems.

Chapter 5
Conclusion and future work

5.1 Summary

In this thesis, we focused on benchmarking cloud storage systems, and provide a solution to make its process convenient, efficient and flexible by developing BenchCloud.

Firstly, we briefly introduced the background of cloud computing and the rise of cloud storage systems, and why benchmarking is important in choosing the right system for our requirements.

Secondly, we made a study on the features of six cloud storage systems. Security and privacy features were studied in detail due to the important role they plays in ensuring data privacy. Resiliency features were studied briefly and some of them are still to be studied in the future. Other features such as file chunking, file bundling, file compression, delta encoding, file deduplication, file sharing, are also studied.

Thirdly, we presented BenchCloud, a tool for benchmarking cloud storage systems. We first studied existing benchmarking tools for cloud systems and found that no mature tool was developed for benchmarking cloud storage systems like Dropbox, Google Drive, etc. We then studied the requirements of such a benchmarking tool and described the design goal and internal architecture of BenchCloud.

Finally, we presented how to use BenchCloud to analysis cloud storage systems and took a series of experiments on Dropbox to show how BenchCloud can be used to inspect various kinds of features. Interesting results and analysis were shown about how concurrency and file size can affect the performance of file uploading and downloading. An experiment was conducted to study the feasibility to use Dropbox as a backend storage system for an IoT system. Another series of experiments were conducted to study three internal features of the synchronization client of Dropbox, namely file compression, delta encoding, and file deduplication.

5.2 Future work

Future work can be done to give a more thorough analysis to the six cloud storage systems. Some security and resiliency features are still to be studied by experiments. Because some of the systems are black-boxes without public accessible source code, it is not an easy (or even possible) task to study some system features, like how user credentials are saved and managed inside the cloud.

Besides, in Chapter 4 the experiments were conducted for Dropbox, and it will be interesting to conduct these experiments on other cloud storage systems tool and make an analysis on the difference of the results.

Finally, BenchCloud can be extended to study a broader range of system features. So far testing for security and privacy features is not directly supported by BenchCloud (although it may be studied by analyzing captured packets) and it would be nice to have it supported. Some other functions are also considered good to have, like testing the distribution of the servers (IPs, locations, etc.) of cloud storage systems.

References

[1] Leonard M Adleman, Ronald L Rivest, and Adi Shamir. Cryptographic communications system and method, September 20 1983. US Patent 4,405,829.

[2] LaCie AG. Lacie wuala. https://www.wuala.com/. Accessed: 2014-03-08.

[3] Carsten Binnig, Donald Kossmann, Tim Kraska, and Simon Loesing. How is the weather tomorrow?: towards a benchmark for the cloud. In *Proceedings of the Second International Workshop on Testing Database Systems*, page 9. ACM, 2009.

[4] Peter Buxmann, Thomas Hess, and Sonja Lehmann. Software as a service. *Wirtschaftsinformatik*, 50(6):500–503, 2008.

[5] Eric Conrad. Advanced encryption standard. *White Paper*, 1997.

[6] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[7] Jack B Dennis and Earl C Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155, 1966.

[8] Tim Dierks. The transport layer security (tls) protocol version 1.2. 2008.

[9] Idilio Drago, Enrico Bocchi, Marco Mellia, Herman Slatman, and Aiko Pras. Benchmarking personal cloud storage. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 205–212. ACM, 2013.

[10] Dropbox. Privacy policy. https://www.dropbox.com/security#security. Accessed: 2014-03-08.

[11] Inc. Dropbox. Dropbox. https://www.dropbox.com/. Accessed: 2014-03-08.

[12] Roy Fielding. Representational state transfer. *Architectural Styles and the Design of Netowork-based Software Architecture*, pages 76–85, 2000.

[13] George H. Forman and John Zahorjan. The challenges of mobile computing. *Computer*, 27(4):38–47, 1994.

[14] Inc. Google. Google drive. https://drive.google.com/. Accessed: 2014-03-08.

[15] Van Jacobson, Craig Leres, and S McCanne. The tcpdump manual page. *Lawrence Berkeley Laboratory, Berkeley, CA*, 1989.

[16] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 1–14. ACM, 2010.

[17] GNU General Public License. Version 2. *URL http://www. gnu. org/licenses/gpl-2.0. html*, 1991.

[18] MEGA. Help centre - security & privacy. https://mega.co.nz/#help_security. Accessed: 2014-03-08.

[19] MEGA. A word on cryptography. https://mega.co.nz/#blog_3. Accessed: 2014-03-08.

[20] Microsoft. Microsoft azure. https://www.windowsazure.com/. Accessed: 2014-03-08.

[21] Angela Orebaugh, Gilbert Ramirez, and Jay Beale. *Wireshark & Ethereal network protocol analyzer toolkit*. Syngress, 2006.

[22] Sybil P Parker. Mcgraw-hill dictionary of scientific and technical terms. 1989.

[23] Ken Pepple. *Deploying OpenStack*. O'Reilly Media, Inc., 2011.

[24] Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proceedings of the VLDB Endowment*, 3(1-2):460–471, 2010.

[25] Amazon Web Services. Amazon s3. http://aws.amazon.com/s3/. Accessed: 2014-05-18.

[26] Yingjie Shi, Xiaofeng Meng, Jing Zhao, Xiangmei Hu, Bingbing Liu, and Haiping Wang. Benchmarking cloud-based data management systems. In *Proceedings of the second international workshop on Cloud data management*, pages 47–54. ACM, 2010.

[27] Will Sobel, Shanti Subramanyam, Akara Sucharitakul, Jimmy Nguyen, Hubert Wong, Arthur Klepchukov, Sheetal Patil, Armando Fox, and David Patterson. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. In *Proc. of CCA*, volume 8, 2008.

[28] Secure Hash Standard. Federal information processing standard publication 180-2. us department of commerce, national institute of standards and technology (nist), 2002. *csrc. nist. gov/publications/fips/fips180-2/fips180-2withchangenotice. pdf.*

[29] Tahoe. Tahoe-lafs: Convergence secret. https://tahoe-lafs.org/trac/tahoe-lafs/browser/trunk/docs/convergence-secret.rst. Accessed: 2014-03-08.

[30] Hakim Weatherspoon and John D Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Peer-to-Peer Systems*, pages 328–337. Springer, 2002.

[31] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe: the least-authority filesystem. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 21–26. ACM, 2008.

[32] Wuala. Privacy policy. http://www.wuala.com/%20/about/privacy. Accessed: 2014-03-08.

[33] Wuala. Recent security scandals. https://support.wuala.com/2013/08/recent-security-scandals/. Accessed: 2014-03-08.

[34] Wuala. Security: Frequently asked questions. http://support2.wuala.com/faq/security/. Accessed: 2014-03-08.

[35] Wuala. What is the local (client side) encryption? http://support2.wuala.com/faq/security/what-is-client-side-encryption/. Accessed: 2014-03-08.

[36] Qing Zheng, Haopeng Chen, Yaguang Wang, Jiangang Duan, and Zhiteng Huang. Cosbench: A benchmark tool for cloud object storage services. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 998–999. IEEE, 2012.

Appendix A

Sample configuration files and scripts

A.1 Sample configuration to upload 50 files of 100KB each from Dropbox with 25 concurrent threads

```
[test]
description = Upload random binary files to dropbox
times = 50
sleep = False
#sleep_seconds = 1

[logging]
enabled = True
log_file =
 /tmp/dropbox_upload_random_binary_50f_100KB_25threads.log

[driver]
class = benchcloud.drivers.dropbox_driver.DropboxDriver

[operator]
class = benchcloud.operators.uploader.Uploader
operation_method = upload
operation_method_params = {"local_filename":
 "<generated_file.name>",
 "remote_dir": "/benchmark-test"}

[file_generator]
class = benchcloud.file_generators.random_file_generator
 .RandomFileGenerator
#directory =
prefix = benchmarking-
#suffix = .test
```

```
28 delete = True
29 size = 102400
30
31 [concurrent]
32 threads = 25
```

A.2 Sample configuration to download files from Dropbox with 25 concurrent threads

```
1  [test]
2  description = Download all files (not recursively)
3   in a remote directory to a local directory
4  sleep = False
5  #sleep_seconds = 1
6  remote_dir = /benchmark-test
7  local_dir = /tmp/benchcloud-junk
8
9  [logging]
10 enabled = True
11 log_file = /tmp/benchmarking-dropbox-download-25threads.log
12
13 [driver]
14 class = benchcloud.drivers.dropbox_driver.DropboxDriver
15
16 [concurrent]
17 threads = 25
```

A.3 Sample configuration file to upload 10 files of 10 MB each

```
1  [test]
2  description = Upload random binary files to dropbox
3  times = 10
4  sleep = False
5  #sleep_seconds = 1
6
7  [logging]
8  enabled = True
9  log_file = /tmp/dropbox_upload_random_binary_10f_10MB.log
10
11 [driver]
12 class = benchcloud.drivers.dropbox_driver.DropboxDriver
```

```
13
14 [operator]
15 class = benchcloud.operators.uploader.Uploader
16 operation_method = upload
17 operation_method_params = {"local_filename":
18  "<generated_file.name>",
19  "remote_dir": "/benchmark-test"}
20
21 [file_generator]
22 class = benchcloud.file_generators.random_file_generator
23  .RandomFileGenerator
24 #directory =
25 prefix = benchmarking-
26 #suffix = .test
27 delete = True
28 size = 10485760
```

A.4 Sample configuration file to simulate IoT system with 50 sensors and 10 seconds of interval

```
1  [test]
2  description = IoT Benchmarking
3  times = 500
4  sleep = True
5  sleep_seconds = 10
6
7  [logging]
8  enabled = True
9  log_file = /tmp/benchmark-iot/dropbox_iot_50t_10s.log
10
11 [driver]
12 class = benchcloud.drivers.dropbox_driver.DropboxDriver
13
14 [operator]
15 class = benchcloud.operators.uploader.Uploader
16 operation_method = upload
17 operation_method_params = {"local_filename":
18  "<generated_file.name>",
19  "remote_dir": "/benchmark-test"}
20
21 [file_generator]
22 class = benchcloud.file_generators
```

```
23  .random_file_generator.RandomFileGenerator
24 prefix = benchmarking-iot-
25 delete = True
26 size = 5120
27
28 [concurrent]
29 threads = 50
```

A.5 Sample benchmark configuration file simulating 100 sensors sending 1000 files in total to Dropbox with an interval of 5 seconds

```
1  [test]
2  description = 100 sensors sending 1000 files in total
3   to Dropbox with an interval of 5 seconds
4  times = 1000
5  sleep = True
6  sleep_seconds = 5
7
8  [logging]
9  enabled = True
10 log_file = /tmp/iot-benchmark-1000f-100t-5s.log
11
12 [driver]
13 class = benchcloud.drivers.dropbox_driver.DropboxDriver
14
15 [operator]
16 class = benchcloud.operators.uploader.Uploader
17 operation_method = upload
18 operation_method_params = {
19   "local_filename": "<generated_file.name>",
20   "remote_dir": "/benchmark-test"}
21
22 [file_generator]
23 class = benchcloud.file_generators
24  .random_file_generator.RandomFileGenerator
25 prefix = iot-benchmark-
26 delete = True
27 size = 5120
28
29 [concurrent]
```

```
30 threads = 100
```

A.6 Script to try downloading a file from Dropbox repeatedly

```
1  import os
2  import argparse
3  from time import time, localtime, strftime, sleep
4
5  from benchcloud.drivers.dropbox_driver import DropboxDriver
6
7
8  def log(msg):
9      millis = int(round(time() * 1000))
10     timestamp = "[{}] {} |".format(millis,
11         strftime("%d %b %Y %H:%M:%S", localtime()))
12     whole_message = "{} {}".format(timestamp, msg)
13     print whole_message
14
15
16 if __name__ == '__main__':
17     arg_parser = argparse.ArgumentParser(
18         description='Test if a file exists in Dropbox')
19     arg_parser.add_argument('-rf', action='store',
20                             dest='remote_filename',
21                             default='/target_file',
22                             help='Local file', required=False)
23     results = arg_parser.parse_args()
24     remote_filename = results.remote_filename
25     local_filename = './downloaded_target_file'
26     # delete old target file
27     if os.path.isfile(local_filename):
28         os.remove(local_filename)
29
30     # Test if file exists
31     dropbox = DropboxDriver()
32     dropbox.connect()
33     while True:
34         log('Start to download file')
35         dropbox.download(remote_filename=remote_filename,
36             local_filename=local_filename)
37         log('Operation finished!')
```

```
38          if os.path.isfile(local_filename):
39              break
40          sleep(0.5)
41      log('File␣downloaded␣successfully!')
```

A.7 Sample configuration file to test if file compression is supported in Dropbox synchronizaiton client

```
1  [test]
2  description = Upload a sparse file to Dropbox via sync client
3   to test if client-side compression is used
4  times = 1
5  sleep = False
6  #sleep_seconds = 1
7
8  [logging]
9  enabled = True
10 log_file = /tmp/benchmarking.log
11
12 [driver]
13 class = benchcloud.drivers.localfs_driver.LocalFSDriver
14
15 [operator]
16 class = benchcloud.operators.uploader.Uploader
17 operation_method = upload
18 operation_method_params = {"local_filename":
19   "<generated_file.name>",
20   "remote_dir": "/Users/wangxing/Dropbox"}
21
22 [file_generator]
23 class = benchcloud.file_generators
24  .sparse_file_generator.SparseFileGenerator
25 #directory =
26 prefix = benchmarking-sparse-
27 suffix = .jpg
28 delete = True
29 size = 104857600
```

A.8 Sample configuration file to test if delta encoding is supported in Dropbox synchronizaiton client

```
1  [test]
```

```
description = Upload random binary files with common part
 (i.e. delta files) to dropbox
times = 5
sleep = True
# The sleep time should long enough to make sure
# the last file is fully synchronized to the cloud
sleep_seconds = 120

[logging]
enabled = True
log_file = /tmp/benchmarking.log

[driver]
class = benchcloud.drivers.localfs_driver.LocalFSDriver

[operator]
class = benchcloud.operators.uploader.Uploader
operation_method = upload
operation_method_params = {"local_filename":
  "<generated_file.name>",
  "remote_filename": "/Users/wangxing/Dropbox/delta-file"}

[file_generator]
class = benchcloud.file_generators
 .delta_file_generator.DeltaFileGenerator
#directory =
prefix = benchmarking-delta-
#suffix = .test
delete = True
size = 10485760
# the 'percent' param is unique to DeltaFileGenerator
percent = 0.9
```

A.9 Sample configuration file to test if file deduplication is supported in Dropbox synchronizaiton client

```
[test]
description = Upload binary files
 with identical content to dropbox
times = 3
sleep = True
sleep_seconds = 3
```

```
7
8  [logging]
9  enabled = True
10 log_file = /tmp/benchmarking.log
11
12 [driver]
13 class = benchcloud.drivers.localfs_driver.LocalFSDriver
14
15 [operator]
16 class = benchcloud.operators.uploader.Uploader
17 operation_method = upload
18 operation_method_params = {"local_filename":
19   "<generated_file.name>",
20   "remote_dir": "/Users/wangxing/Dropbox"}
21
22 [file_generator]
23 class = benchcloud.file_generators
24  .identical_file_generator.IdenticalFileGenerator
25 #directory =
26 prefix = benchmarking-
27 #suffix = .test
28 delete = True
29 size = 20971520
```