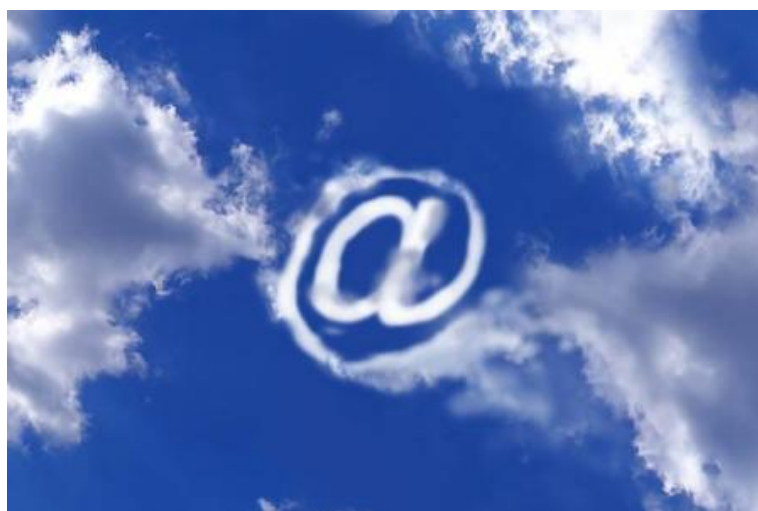


Secure Use of Public Cloud Services

Project Assignment



Trondheim, December, 2011

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Telematics

Christian Askeland, Anders Emil Salvesen

Abstract

Cloud computing is a fast growing area within information technology. Hosting web applications, data computation and storing of almost unlimited amounts of information is just some of the things that can be done in the cloud.

More and more companies see the advantages with utilizing the cloud, but they also have some issues regarding privacy and security when it comes to handing over their data to a third-party. A company wants to know that their data is secure and that that it can not be accessed by anyone, not even the cloud providers.

We solve these issues by introducing the Rain C&C which is a prototype of a system that provides secure data storage in the cloud. Our implementation splits up the data and distribute it to multiple cloud storage providers which make it impossible for both attackers and cloud providers to access or read the original data owned by the company.

Preface

This report serves as a specialization project in Information Security in the 9th semester of the Master's Programme in Communication Technology at The Norwegian University of Science and Technology, NTNU. The assignment was given by SINTEF ICT, and the project has been carried out at a laboratory located at their premises in S. P. Andersens vei, Trondheim.

This project has challenged us to implement forward thinking ideas in the up and coming area which Cloud Computing is, giving us the ability to try and influence the way government agencies and companies with a need for high degree security mechanisms when using the cloud for storage.

We would like to thank our supervisor Martin Gilje Jaatun for his valuable input and weekly feedback. We appreciate his enthusiasm and that he has been a driving force throughout the project. His contribution to the scientific paper based on our project is also appreciated.

Finally we would like to thank our professor, Danilo Gligoroski.

Trondheim, December 2011

Christian Askeland

Anders Emil Salvesen

Abbreviations

API Application Programming Interface

C&C Command & Control Center

CRM Customer Relationship Management

DoS Denial of Service

IaaS Infrastructure as a Service

IDA Information Dispersal Algorithm

NIST National Institute of Standards and Technology

NTNU Norwegian University of Science and Technology

PaaS Platform as a Service

RAIN Redundant Array of Independent Net-storages

RDBMS Relational Database Management System

RDP Remote Desktop Protocol

SaaS Software as a Service

SSH Secure Shell

TCG Trusted Computing Group

TDM Trusted Data Module

UI User Interface

Contents

Abstract	i
new abstract	i
new attempt	i
Preface	iii
Abbreviations	v
Contents	vii
List of Figures	xi
List of Tables	xiii
Listings	xv
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Limitations in scope	2
1.4 Methodology	2
1.5 Document Structure	3
2 Theoretical Background	5
2.1 Cloud Computing	5
2.2 Challenges with Cloud Computing	6
2.2.1 Availability	7
2.2.2 Data Security	7
2.2.3 Third-Party Control	7
2.2.4 Legal Issues	8
2.3 RAIN	8
2.4 Alternatives to RAIN	8
2.4.1 Encryption	8
2.4.2 Trusted Computing	9

2.4.3	Homomorphic Encryption	9
2.4.4	Secure Multiparty Computation	10
2.4.5	IDA	10
2.4.6	Shamir Secret Sharing Scheme	10
3	Lab Environment and Tools	11
3.1	Software Choice	11
3.2	OpenStack	11
3.2.1	Nova	12
3.2.2	Swift	12
3.2.3	Glance	12
3.3	Installation and Configuration	12
3.4	Benchmarking computer and Web server with C&C	13
3.4.1	Benchmarking computer	13
3.4.2	Web server with C&C	14
4	Implementation	15
4.1	Development Environment	15
4.1.1	Python	15
4.1.2	User Interface to the C&C	15
4.1.3	Packages	15
4.2	Architecture	16
4.3	The C&C	16
4.3.1	Persistence	17
4.3.2	The Controller	18
4.3.3	The Splitter	18
4.3.4	The Storage Provider interfaces	20
4.4	Process Description	21
4.4.1	Uploading files	21
4.4.2	Downloading files	22
4.5	Improvements done	23
4.5.1	Splitting and sending	23
4.5.2	Merging	24
4.6	Use Case	24
5	Results	27
5.1	Measurements	27
5.2	Procedure	27
5.3	Accuracy of Results	28
5.4	Other Measurements	28
5.5	Results	28
5.5.1	Uploading	28
5.5.2	Downloading	31
5.5.3	Total	31

6	Discussion	37
6.1	Discussion of Results	37
6.1.1	Overhead	37
6.1.2	Importance of Performance	38
6.1.3	Redundancy	38
6.2	How RAIN Solves Privacy Issues	38
6.2.1	Cloud Provider is Compromised	38
6.2.2	Curious Cloud Providers	38
6.2.3	Government Claims	39
6.2.4	Drawbacks	39
6.3	Limitations	39
6.3.1	Data Processing in the Cloud	39
6.3.2	Corrupted or Lost Chunks	39
6.3.3	The Splitting Algorithm	40
6.4	Possible Attacks	40
6.4.1	Eavesdropping the Network	40
6.4.2	Attacking the C&C	40
6.4.3	Attacking the Cloud Provider	41
6.4.4	Curious or malicious cloud providers	41
6.5	Commercial Use of RAIN	41
7	Conclusion and Further Work	43
7.1	Further work	43
7.1.1	Performance	43
7.1.2	Security	43
7.1.3	Redundancy	43
7.2	Conclusion	44
	Bibliography	45
A	Example of a a file and its chunks	49

List of Figures

2.1	Challenges with cloud computing adaption	6
3.1	Overview of the environment. http://cssoss.wordpress.com/ Creative Commons. Client 1 can be any machine connected to the internet.	13
4.1	The C&C is within the company, communicating via an encrypted line to the different storage providers.	17
4.2	The two database tables.	18
4.3	A file is split into $n=8$ chunks. All parts with the same color goes in the same chunk.	20
4.4	The process from the user uploads the file until he receives a notification of the finished upload.	22
4.5	The process from the user sends a request to download a file, until the file is returned.	23
4.6	Downloading via the RAIN Dashboard.	24
4.7	Bob is about to upload <code>alice.txt</code>	25
4.8	Cloudfile in the database, not uploaded.	25
4.9	Cloud chunks in the database after being uploaded.	25
4.10	Cloudfile in the database, uploaded. Notice that the uploaded-value is changed from 0 to 1.	25
4.11	<code>Alice.txt</code> is successfully uploaded.	26
5.1	The duration of uploading files of different sizes and different number of chunks to the provider.	29
5.2	The bitrate when uploading files of different sizes and different number of chunks to the provider.	30
5.3	The duration from starting download to all chunks are merged.	32
5.4	The bitrate from starting download to all chunks are merged.	33
5.5	Total average duration of upload and download.	34
5.6	Total average bitrate of upload and download.	35

List of Tables

3.1	Configuration of OpenStack environment	13
3.2	Specifications of benchmarking computer and web server	14
5.1	The overhead in percent when uploading files.	31
5.2	The overhead in percent when downloading files.	31

Listings

4.1	The code for splitting files into n chunks.	19
4.2	The code for merging n chunks into a file.	21
A.1	Full text file.	50
A.2	Chunk no. 1.	50
A.3	Chunk no. 2.	50
A.4	Chunk no. 3.	51
A.5	Chunk no. 4.	51
A.6	Chunk no. 5.	51
A.7	Chunk no. 6.	51
A.8	Chunk no. 7.	51
A.9	Chunk no. 8.	51

Chapter 1

Introduction

Cloud computing is still an evolving paradigm which makes it hard to give one precise definition of the term "cloud computing". A definition of the term was given in a draft [23] from the National Institute of Standard and Technology (NIST):

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

As we see from the definition, cloud computing covers a lot of the information technology infrastructure, and in this project we will dive deeper into the storage part of the cloud. We will give an extended description of cloud computing in the background chapter.

1.1 Motivation

Using the cloud for file storage is an ideal solution for a lot of organizations, from small one person businesses to big enterprises. A challenge with data storage in the cloud is the issues regarding privacy and security. There will always exist a risk of someone compromising the security and stealing the data, but you also have the case where "curious" or malicious cloud providers want to peek into your data.

Our motivation is to overcome these issues by providing a secure way for organizations to store their sensitive data in the cloud. We want to provide a solution that guarantees that their data is secure and stored in a way that makes it inaccessible for curious or malicious cloud providers.

1.2 Objectives

Our work is based on the work of Jaatun et al. in [19] and [41]. We will implement a prototype of their proposed solution which introduces the concept of Redundant Array of Independent Net-storage (RAIN). We will also design and implement a user interface on top of this prototype to make testing and demonstration of the functionality easier. The back-end will be implemented in Python and the UI will be implemented in Django, a Python web framework.

The performance of the prototype will be benchmarked to compute the overhead imposed by splitting and merging.

1.3 Limitations in scope

Since we only have a limited time available, we had to put some limitations on the scope of this project.

Jaatun et al. [19] proposes a solution with splitting of chunks and the use of a two layer distribution model. They use a Command & Control Center (C&C) unit and an intermediate processing layer. In this project we are not implementing the intermediate processing layer, but we instead let the C&C do this part as well.

They also describe requirements for the splitting algorithm, but a specific implementation is not proposed. Implementing this algorithm was not our main focus, thus a temporary, simple algorithm designed by us is used instead. More advanced and mathematically proven secure alternatives are discussed in the discussion chapter and in further work.

RAIN also introduces redundancy of the split chunks. Since the number of data clouds available to us have been very limited, we have not prioritized this part.

1.4 Methodology

The methodology we will use is design science [17]. The objective in design science is to start developing with an initial design. When facing issues during development, you change the design to fix these issues.

Our initial design will be the prototype based on the design given in the papers written by Jaatun et al.[19, 41] During the development we will dig deeper into the problems regarding splitting, merging and distribution of the files.

The prototyping will first be a simple algorithm which divides a file and send it to two different servers in the cloud. We then show that the prototype can be expanded to support interfaces to multiple cloud services.

1.5 Document Structure

Chapter 2 - Theoretical Background gives a wide description of cloud computing, and presents alternative methods for storing data securely in the cloud.

Chapter 3 - Lab Environment and Tools presents our laboratory at Sintef ICT, and the computer on which we perform the benchmarking.

Chapter 4 - Implementation describes how we implemented the prototype, also referred to as the C&C.

Chapter 5 - Results presents results from benchmarking the prototype.

Chapter 6 - Discussion discusses the results from chapter 5, and we also discuss the limitations of the project.

Chapter 7 - Conclusion and further work concludes the report and we provide a list of work to be done based on the discussion in chapter 6.

Chapter 2

Theoretical Background

In this chapter we will give a presentation of cloud computing and introduce challenges with cloud computing, some of which we try to solve with the project. RAIN, the solution we are implementing is introduced, and we also present other solutions for secure storage in the cloud.

2.1 Cloud Computing

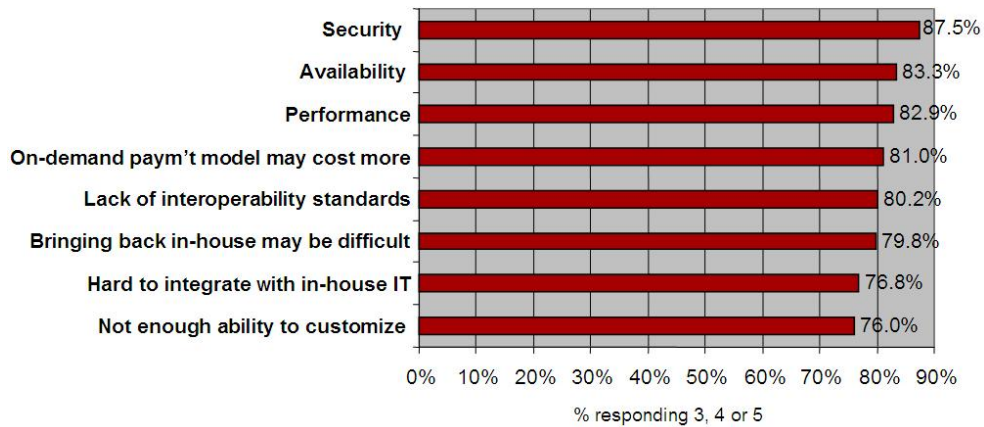
Cloud Computing is, and have been for a while, a driving force in the software business. Big vendors like Amazon[2], Microsoft[24] and Google[14] are leading the way.

Cloud computing can be divided into three categories, Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

- **Software as a Service:** SaaS is complete software delivered in the cloud and is often more commonly referred to as web applications. Some good examples are Google Apps which can be used for document editing and Salesforce which is an online customer relationship management (CRM) application
- **Platform as a Service:** PaaS delivers a computing service or development platform. This is best illustrated through examples like Microsoft Azure and Google App Engine which delivers a complete platform with a pre-defined runtime environment, like .Net, Java or Python. The platform usually provides a set of interfaces to easily deploy applications.
- **Infrastructure as a Service:** IaaS provides the same service as if the customer had its own servers. The cloud provider handles the hardware maintenance and network infrastructure. The customer basically rents the amount of servers he needs and then he configures them as he wants, usually through root access via SSH or Remote Desktop (RDP).

Q: Rate the *challenges/issues* of the 'cloud'/on-demand model

(Scale: 1 = Not at all concerned 5 = Very concerned)



Source: IDC Enterprise Panel, 3Q09, n = 263

Figure 2.1: Challenges with cloud computing adaption

The biggest difference between the three categories is the amount of configuration needed by the customer. In SaaS the customer gets access to an application which needs minimal configuration and is almost ready to use "out of the box". Less options to configure also means less possibilities to put a custom touch on the product, and that's why the cloud providers also have other opportunities. IaaS gives the customer all the freedom he wants, but that also requires a lot more technical knowledge and a lot more configuration by the customer. PaaS is somewhere in between. If the customer already knows which development environment he needs and don't need low-level services, PaaS can be a good choice.

In this project we are working somewhere in the middle of the IaaS and PaaS. We are utilizing IaaS for the hosting of the C&C and PaaS is used for the storage of the files in Amazon S3, which is a platform for file storage.

2.2 Challenges with Cloud Computing

There are a number of security benefits in moving your data to the cloud. Redundancy is done with automatic replication, and cloud companies invest in security infrastructure, and also dedicated security teams [22].

While acknowledging these benefits, it is interesting to see that security issues and availability challenges are one of the main reasons for companies to not use cloud computing services [4, 18]. This can be seen in figure 2.1.

Armbrust et al.[4] and Atayero et al. [5] mention availability, data security, third-party control and legal issues as important issues with the cloud from client's point of interest.

2.2.1 Availability

The term availability means that a user should be able to access their data at any given time. Availability is critical for many companies, as they rely on web applications in many aspects of their day-to-day operations. Data should be available when needed, therefore outages is important to avoid.

Examples of well-known outages are presented in [8], of which the 4 day long Amazon AWS outage in April 2011 may be the worst [6]. It affected major internet websites like Quora[33] and Foursquare[13].

The possibility to migrate¹ your data is also a part of the term availability. If users risk not being able to change providers when they want to, this can be looked at as lock-in, and be a major reason for not using a service.

2.2.2 Data Security

There are many data security issues in cloud computing, among which some are shared with in-house hosting, and others are unique issues caused by the architecture used.

Unique issues are virtualization vulnerabilities[26] and issues with the multi-tenancy and resource sharing models used[10]. Phishing scams against users of the particular cloud service has been seen [21].

Also, as pointed out by Abadi [1], maintaining ACID² properties when doing data replication over large geographic zones may be difficult.

Security issues shared with in-house hosting include denial of service-attacks, data interception, direct intrusion attacks on the server and on the client communicating with the server.

2.2.3 Third-Party Control

Leaving data in the care of a third-party means risk of loss of intellectual property and trade secrets. Providers can be seen as honest-but-curious, they will deliver what they have promised, but may read your data for own gain. Examples of this is the Google using aggregated information from for example your email correspondence for directed marketing [15]. Due to the patriot act, US companies may have to provide the US government with any data that

¹Migrating, moving your data to another solution

²Atomicity, consistency, isolation and durability, a set of properties that guarantee database transactions are processed reliably[16].

is requested, even if this data is stored in a data center on another continent [39].

Malicious insiders and corporate espionage are also fears to take into consideration.

2.2.4 Legal Issues

As many cloud data centers are not located in the country of the customer, legal issues arise, especially when it comes to sensitive information e.g. patient records, insurance data and financial information. The legality of storing this data abroad is often questioned, and not complying with government policies might lead to legal liabilities.

2.3 RAIN

The Redundant Array of Independent Net-storages (RAIN) [19] tries to solve many of the challenges described above, especially the issue of third-party control. RAIN is designed like a bot network with a Command & Control unit (C&C) which splits up data into small chunks, and a underlying network of processing and storage cloud providers which are used for secure distribution of the file chunks.

The main advantages with RAIN is that a single chunk does not contain any valuable information by itself. RAIN also has an unique distribution model which makes it very hard to associate the transferring chunks with the file owner. By distributing the files to several different cloud providers, the risks of curious or malicious cloud providers are eliminated. Even if a single provider manage to break the security of the splitting algorithm, it still needs all the other chunks of the file to reconstruct the original data.

2.4 Alternatives to RAIN

There are other proposals on how to keep your data secure in the cloud. Here we describe some of them.

2.4.1 Encryption

Data encryption is probably the most widely used method when it comes to securing your data. Encryption is an old and reliable technique, but it has some drawbacks concerning data storage in the cloud.

One of the main advantages with the cloud is the possibility to easily scale the data power (hardware) and process your data in the cloud. When your data is encrypted, you will not be able to do any computations on it and you will

not be able to use this advantage unless you decrypt your data first. While the data is decrypted, even if it is only in the memory, it is exposed to the cloud provider or anyone else with access to the hardware.

In a scenario where you can't trust your cloud provider, you still make all your data accessible to the provider or a possible attacker. Even if it is encrypted, the provider is still in possession of the ciphertext and with enough time and computational power they may be able to compromise the key. If this happens or the key ever gets compromised in another way, all the data that have been in the cloud provider's possession will be exposed since he also will be able to decrypt your old data.

2.4.2 Trusted Computing

Trusted computing is a way to modify computers such that they behave in an expected manner. With use of modification to software and hardware it is possible to restrict access to the machine and enable the ability to execute signed code only.

Santos et al. propose a use of trusted computing in the cloud [36] which ensures that the cloud provider can't access the data. They introduce a Trusted Data Module (TDM) which is a small data chip that is part of the hardware and verify that the hardware is trusted. This will in second turn verify the software running on the machine. The software starts a trusted virtual machine that will run in its own sandbox that won't be accessible for administrators working for the cloud provider.

Trusted computing moves the trust from the cloud provider to the hardware manufacturer. If the manufacturer has tampered with the hardware, customer data is still insecure.

2.4.3 Homomorphic Encryption

Homomorphic encryption is a way to encrypt data and still be able to do arithmetic operations on the ciphertext. This enables the possibility to encrypt data, send it to the cloud, store it or process it, without anyone knowing what the data is. The advantage with this is that you are able to use the cloud to process the data, but this will still have some of the same disadvantages as with normal encryption since the cloud provider still have access to all of the ciphertext.

Another problem is that homomorphic encryption schemes are malleable by design. That means that from an encrypted plaintext m it is possible to alter the ciphertext and generate another ciphertext which decrypts to $f(m)$, for a known function f , without necessarily knowing or learning m . This means

that it can be possible for a cloud provider or attacker to tamper with the integrity of the data in the cloud [11].

2.4.4 Secure Multiparty Computation

To explain what multiparty computation is, we begin by providing the standard explanation for secure two-party computation, the millionaire problem [40]:

Two millionaires want to find out who is the richest of the two, but they do not want to disclose how much they are actually worth.

This task was first solved by Yao himself [40]. Secure two-party computation is a subset of secure multiparty computation, with the multiparty one introducing more players. The first practical and large-scale implementation of multiparty computation was done by Bogetoft et al. [7] to implement a double auction, to let a player bid without letting the other players know his bid.

2.4.5 IDA

Rabin[34] proposes an algorithm, Information Dispersal Algorithm (IDA), for partitioning files into unrecognizable blocks of which a subset is needed to reconstruct the file. To reconstruct the data file, a subset of the blocks are needed.

Lu et al.[20] proposes a solution for using IDA for massive data distributing and parallel access in the Scientific Data Grid.

This scheme does for the Scientific Data Grid what [19] and [41] intend to do for cloud computing. Many of the security properties are shared with the properties from these two papers, thus this is a reasonable scheme to compare RAIN to.

2.4.6 Shamir Secret Sharing Scheme

Shamir's secret-sharing scheme[37] proposes a way of splitting a secret multiple parts that are shared with different parties. Shamir's idea is that if you split your secret into n parts, then $t \leq n$ parts should be needed to read the data. An example implementation would split data into 10 parts, of which at least 7 of the chunks are needed to merge the data again.

Chapter 3

Lab Environment and Tools

This chapter will present the open source cloud which we installed on servers at Sintef ICT's SISLab for testing purposes. We also mention briefly the computer used for benchmarking, and the web server used to host the RAIN Dashboard.

3.1 Software Choice

Our primary need was for an open source IaaS-platform which could provide us a storage solution similar to Amazon S3, Windows Azure Storage and Rackspace Cloud Files[3, 25, 35].

After searching for alternatives we were left with a handful, notably OpenStack [32], OpenNebula[27] and Eucalyptus[12].

Based on the supervisor's former knowledge and seeing as a lot of the momentum of the Open Source Cloud Computing has undoubtedly shifted towards OpenStack. Ubuntu has changed its Ubuntu cloud foundation technology to use OpenStack instead of Eucalyptus[38], and we consider OpenStack to be a safe and powerful option with a lot of possibilities for custom configuration.

3.2 OpenStack

OpenStack is a project started by NASA and Rackspace to offer infrastructure as a service.

The two main components of OpenStack are Nova and Swift, with Nova being the computing component, and Swift the storage component. There's also Glance, which Nova uses for management of virtual machine images.

As our focus is on storage, we will concentrate mostly on Swift throughout this chapter. However, our lab environment implements all the components.

This allows for experimenting with for example putting the C&C in the cloud.

3.2.1 Nova

Nova is the computing part of OpenStack, in effect controlling the Infrastructure as a Service (IaaS) cloud computing platform. In OpenStack it's the equivalent to for example Amazon EC2 and Rackspace Cloud Servers [29].

3.2.2 Swift

Swift is the storage part of OpenStack, near equivalent of Amazon's S3, and based on Rackspace Cloud Files.

Swift can be used in conjunction with Nova, or it can be used separately for plain storage. Swift comes with its own API, identical to Rackspace's Cloud Files API, but it is possible to add a module to use Amazon's S3 API instead [28]. This means that all libraries that can be used to interact with S3 or Cloud Files, also can be used to interact with Swift.

3.2.3 Glance

Glance provides services for discovering, registering, and retrieving virtual machine images [30], and is used by Nova for these purposes.

3.3 Installation and Configuration

The installation of OpenStack was done on two servers at the Sintef ICT SisLab.

It was easiest to base the install upon the OpenStack Compute Starter Guide [31], which is also based on two servers. Server 1 is the main machine in the setup. It has Nova, Swift and Glance installed, and acts as the entry point for interaction with the OpenStack setup. Server 2 is installed with Nova, and is managed through Server 1. See figure 3.1.

Installing OpenStack should be straight forward, and only involve installing packages from Ubuntu repositories [9], and following the beforementioned Starter Guide.

Even if the packages were easily installed from the repositories, several issues were encountered before a working solution was up and running. The working solution uses the latest stable versions available for both the OS and OpenStack, which is Ubuntu 11.10 Oneiric Ocelot and the Diablo release of OpenStack.

Also, we've reserved the IP addresses from 78.91.98.249-78.91.98.254 for Virtual Machines running in the cloud with Nova Compute.

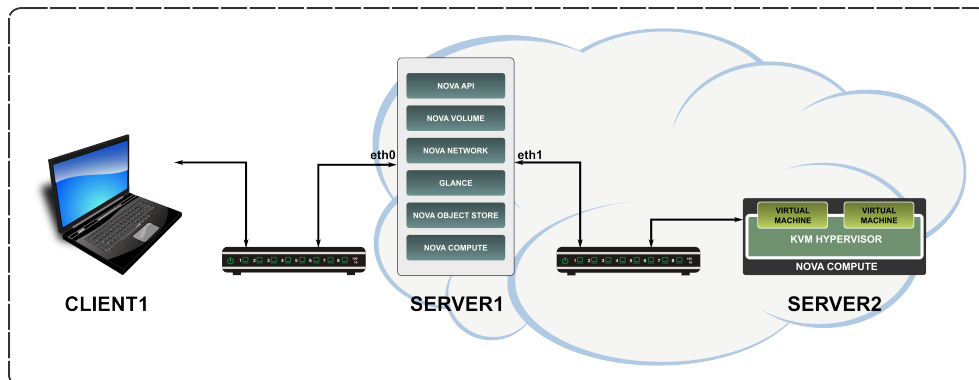


Figure 3.1: Overview of the environment. <http://cssoss.wordpress.com/> Creative Commons. Client 1 can be any machine connected to the internet.

	Server 1	Server 2
Components	All of OpenStack	nova-compute
Interfaces	eth0 - Public eth1 - Private	eth0 - Public eth1 - Private
IP addresses	eth0 - 78.91.98.245 eth1 - 192.168.3.1	eth0 - 78.91.98.246 eth1 - 192.168.3.2
Hostname	rain-1.sislab.no	rain-2.sislab.no
DNS server	4.2.2.1	4.2.2.1
Gateway	78.91.98.225	78.91.98.225
Memory	4GB	4GB

Table 3.1: Configuration of OpenStack environment

3.4 Benchmarking computer and Web server with C&C

With the cloud installed at Sintef ICT, we have been mostly working from our office at NTNU. Here we have two machines, one for benchmarking and one set up to act as the C&C and web server for the RAIN Dashboard.

The specs are as seen in Table 3.2.

3.4.1 Benchmarking computer

The computer where the benchmarks are performed is a regular workstation with a wired connection to the NTNU network.

	Benchmarking computer	Web server
IP address	129.241.164.127	129.241.208.189
CPU	Intel Core i7 860 @ 2.80GHz	Intel Core 2 Duo E8300 @ 2.83GHz
Memory	4GB	4GB
Storage	4GB	4GB
Software	IDE	Apache with mod_wsgi

Table 3.2: Specifications of benchmarking computer and web server

3.4.2 Web server with C&C

A computer is set up with the C&C and an Apache web server to host the RAIN Dashboard. This setup is mostly used for testing the interaction between the C&C and dashboard on a running server solution, in order to catch issues that might only be noticeable in this setting.

Chapter 4

Implementation

This chapter will present the implementation of the prototype.

4.1 Development Environment

The prototype, from now on referred to as the C&C, is developed in Python, using Eclipse with the Pydev-extension. Mercurial is used for version control, hosted at Bitbucket ¹.

4.1.1 Python

We chose Python because of its capabilities as a scripting language, in our opinion making it much easier and faster to write and change code, while also being fully object-oriented.

4.1.2 User Interface to the C&C

To keep the C&C as lightweight and versatile as possible, we have separated it into modules. This means that there is no default user interface for the C&C, instead an API is defined so everyone who wants can build their own interface. For our use, we have developed a web interface in Django which we call the RAIN Dashboard.

4.1.3 Packages

This is a list of the most important packages we have used with Python:

- Python 2.7.1
- SQLAlchemy 0.7.3

¹<https://bitbucket.org/andersem/raincc>

- SQLite3
- python-cloudfiles 1.7.9.1
- boto 2.1.0
- Django 1.3.1

4.1.3.1 SQLite and SQLAlchemy

SQLite is a lightweight, embedded relational database management system. SQLAlchemy is an Object Relational Mapper for Python, providing easier interaction with both SQLite, but also making it very easy to change RDBMS to for example PostgreSQL or MySQL.

4.1.3.2 Python-cloudfiles and boto

Python-cloudfiles and boto are the Python interfaces to respectively Openstack Swift and Amazon Web Services, providing easy saving and loading to the remote server.

4.1.3.3 Django

Django is a web application framework written in Python. The RAIN Dashboard, a web client developed for easy file handling, is implemented in Django for compatibility with the C&C.

4.2 Architecture

We envision an architecture very much like the one seen in Figure 2b in [19], except that we have focused solely on the storage part. The resulting architecture is seen in Figure 4.1.

For our solution, the C&C and the RAIN Dashboard are both installed on a server within an organisation, maintaining the only database of where in the cloud the file parts are stored. This server runs on Apache with SSL for secure uploading of files within the organisation.

We refer to the use case in section 4.6 for a more detailed description of which files are saved where, and what is saved in the database.

4.3 The C&C

The C&C is the single point of entry for users of RAIN. Users interact with the C&C to split and merge, upload and download files to and from the cloud. It maintains a database for persistence, storing all information on files uploaded

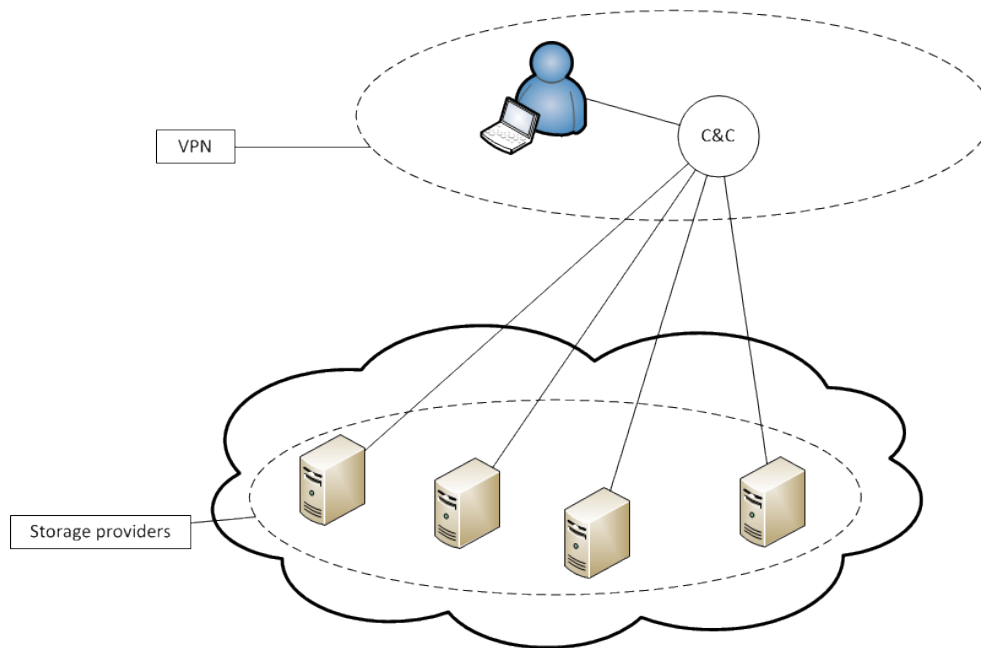


Figure 4.1: The C&C is within the company, communicating via an encrypted line to the different storage providers.

through it, mainly *What files where uploaded by who and when?* and *To what providers are the different chunks uploaded?*

The C&C consists of different components:

- The Controller, providing the API for different User Interfaces.
- The CloudFile- and Chunk-objects which are mapped to the database.
- A splitter-class, providing the splitting and merging of the files.
- Interfaces to interact with the different storage providers.

4.3.1 Persistence

Persistence is taken care of by a Relational Database Management System (RDBMS), used in the C&C via SQLAlchemy. The database consists of two tables, `cloud_file` and `cloud_chunk` as seen in 4.2. Each cloud file is saved to the database when the upload process begins, and updated with `uploaded=True` if and only if it is successfully uploaded. Each cloud chunk is saved when successfully uploaded, with its cloud file as a foreign key.

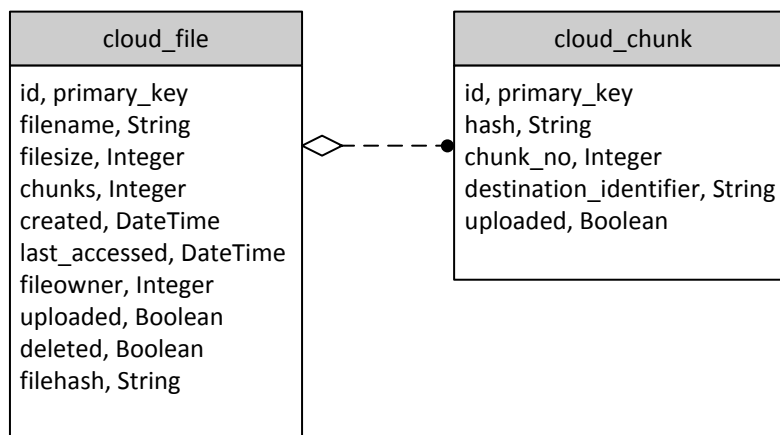


Figure 4.2: The two database tables.

4.3.2 The Controller

The controller provides an API for the user interfaces to interact with the C&C.

The main functions are:

- showAllFiles - returns a list with all the CloudFile-objects in the database
- sendFile - takes a Python file object as argument, splits the file object into n chunks and uploads them to the providers.
- getFile - takes the filename as argument, initiates the download of all the chunks belonging to the CloudFile
- deleteFile - takes the filename as argument, deletes all chunks belonging to the specified file from the storage provider, and the the file is set to deleted=True in the database.

When any action is taken on a new file, a reference to this file is saved in the local database.

The upload and download process is further explained in 4.4.1 and 4.4.2.

4.3.3 The Splitter

The splitter splits and merges a CloudFile.

The split-process takes in a file, and splits it into a configurable number of chunks, default is 8, which are stored temporarily on the disk. The chunks are returned to the CloudFile-object directly after they are saved to disk, for uploading to the storage provider.

Listing 4.1: The code for splitting files into n chunks.

```
def split_file(self, filename, noOfChunks):
    # read the file
    data = self.read_file(filename)

    # create hash-object to calculate
    # the MD5 hash of the chunk
    hash = hashlib.md5()

    # for each chunk...
    for i in range(0, noOfChunks):

        # Create chunk output file
        # (could also be array in memory)
        out = open("chunk" + i + ".tmp", 'wb')

        # Loop through file and sample every
        # n-th byte into the chunk
        for j in range(i, len(data), noOfChunks):

            # Get byte form data array
            b = data[j]

            # Write byte to chunk
            out.write(b)

            # Update hash
            hash.update(b)

        yield(out)
```

The merge-process does the exact opposite, takes in all chunks of a CloudFile, puts it together and stores to disk. It then returns the path to the file.

4.3.3.1 The splitting process

The splitting process is a critical part of the system. To split the data we are using a simple algorithm which loops through the files and sample every n-th byte into a new file (chunk) where n is the number of chunks. A simplified version of the code is shown in listing 4.1

We store the chunks in temporary files in the C&C before we send them to the cloud storage providers.

After the chunks are uploaded to the storage providers, the info on that chunk is stored in the database with a foreign key to the file it belongs to.

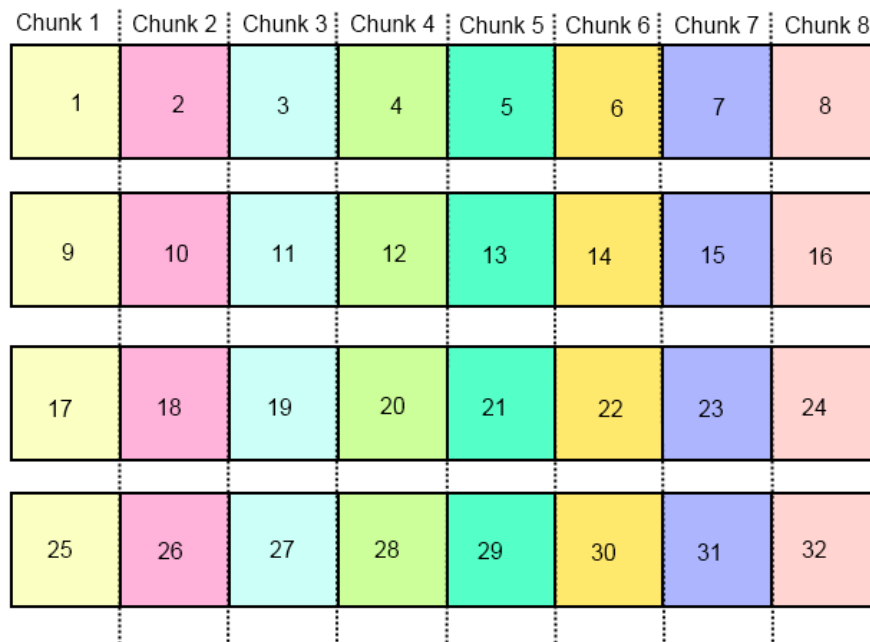


Figure 4.3: A file is split into $n=8$ chunks. All parts with the same color goes in the same chunk.

4.3.3.2 The merging process

The info on each chunk belonging to a file is retrieved from the database so we know where to get the chunks from, and in which order to merge them.

The chunks are iterated over, writing every n th-byte to the out-file. The code for it can be seen in listing 4.2.

4.3.4 The Storage Provider interfaces

Most cloud storage providers offer a solution to interact with the platform in different programming languages. Currently we've implemented interfaces for Swift, the storage solution in OpenStack, and Amazon S3. These implement the following functions:

- `sendChunksInList`
- `getChunksInList`

Listing 4.2: The code for merging n chunks into a file.

```
def merge(self, chunklist, filename, filesize):
    path = settings.completeFilePath
    out = open("%s/%s" % (path, filename), 'wb')
    cache = 1000000 # About 10 MB
    chunks = []
    for chunk in chunklist:
        chunks.append( open(chunk.f.name, 'rb') )

    mergeStart = datetime.utcnow()

    byteWritten = 0
    tempdata=[]
    while byteWritten < filesize:
        for i, chunk in enumerate(chunks):
            tempdata.append(chunk.read(1))
            byteWritten += 1
            if (len(tempdata)>cache and i==len(chunks)-1)
                or byteWritten==filesize:

                out.write(''.join(tempdata))
                tempdata=[]
        for chunk in chunks:
            chunk.close()
    out.close()

    return out
```

4.3.4.1 Extending with Other Storage Providers

Extending the C&C with other cloud storage providers should be just a matter of implementing APIs from other providers.

4.4 Process Description

To get a better overview over the processes, we've put down this step-by-step guide of uploading and downloading.

4.4.1 Uploading files

1. A user uploads a file.
2. The dashboard sends the file to the C&C-controller.
3. The C&C sendFile-method serializes the file to a CloudFile-object and it's stored in the database.

4. The desired splitting algorithm is used to split the file into the desired number of chunks, putting new items in the queue of its cloud storage provider.
5. When all chunks are transferred successfully, the CloudFile-object gets a signal, whereupon it updates the database with info that the upload is successful.
6. A message is sent to the dashboard and displayed to the user real time.

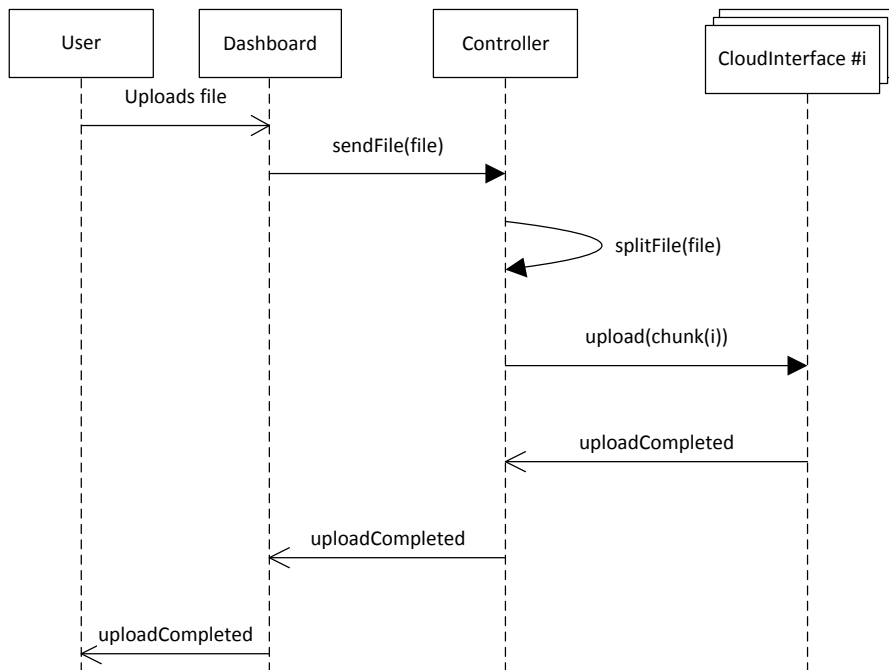


Figure 4.4: The process from the user uploads the file until he receives a notification of the finished upload.

4.4.2 Downloading files

1. A user views a file in the dashboard, and clicks the download button. See figure 4.6.
2. The dashboard sends the name of the file to the C&C's Controller-class.
3. The CloudFile-object of the file with that name is grabbed from the database.
4. All Chunk-objects belonging to that CloudFile are grabbed from the database.

5. The chunks are downloaded from the different providers.
6. The chunks are merged, and saved to a file object.
7. The path of this file object is returned to the dashboard, so that the user can grab it from there.

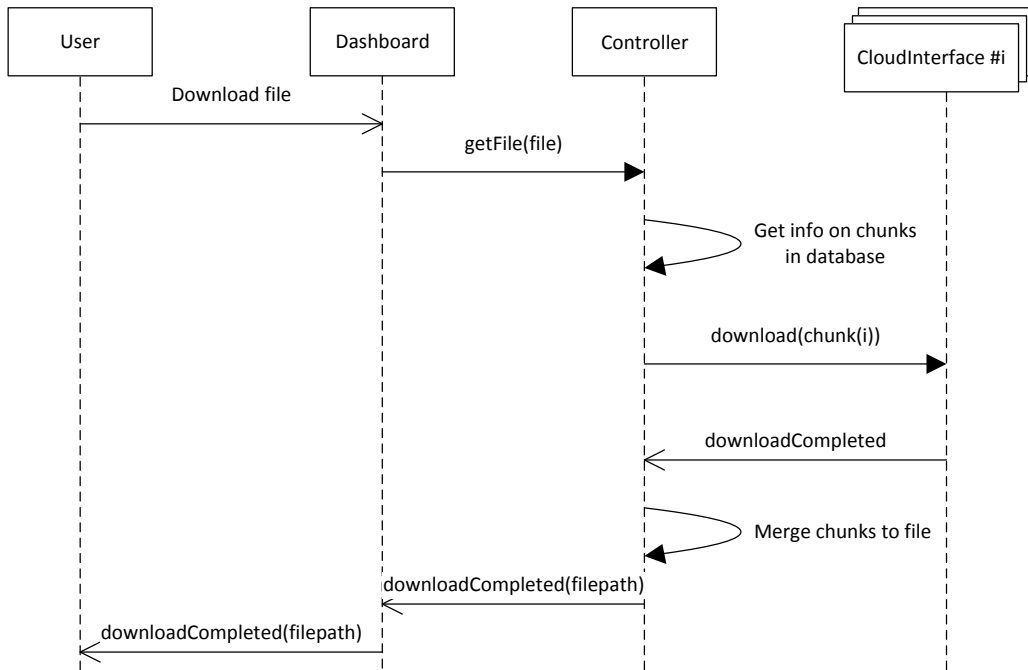


Figure 4.5: The process from the user sends a request to download a file, until the file is returned.

4.5 Improvements done

4.5.1 Splitting and sending

In the first versions of the prototype, all chunks of a file were gathered before starting to upload it. This forced uploading of files to wait for the splitting to finish, thus representing a big performance drag.

This was fixed by:

- Yielding chunks at the moment they are finished writing to disk.
- Putting the chunks from the splitter in a queue-object on the provider-plugin.

Together, this ensures that splitting and uploading is done asynchronously.

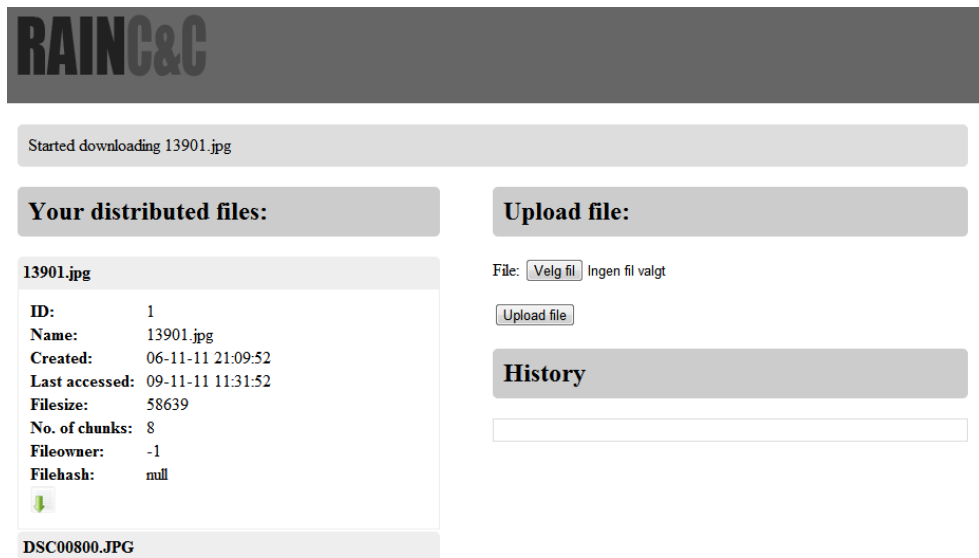


Figure 4.6: Downloading via the RAIN Dashboard.

4.5.2 Merging

In the first version of the prototype, merging was done in-memory. When operating with files of a certain size, this led to the memory being filled, giving a memory error.

To fix this, we had to rewrite the merging algorithm to cache a certain amount of data in the memory before writing to disk.

4.6 Use Case

Bob works in ACME. ACME is utilising RAIN for storage in the cloud, and has accounts at 8 storage providers, provider A through H.

Bob wants to securely store `alice.txt` in the cloud, `alice.txt` is shown in listing A.1. He logs onto the RAIN Dashboard, figure 4.7, and uploads it. Now, the file exists on the C&C in complete form, and is saved to the database, as seen in figure 4.8.

After saving the file in the database, it's split into chunks. The chunks of `alice.txt` can be seen in listing A.2 through A.9. Each chunk gets its own provider, and is uploaded to that provider. When it's finished uploading, the chunks are saved in the database to know where to get them later, see figure 4.9.

The entry for `alice.txt` is updated in the database, figure 4.10, and `alice.txt` is deleted from RAIN C&C so that it now only exists as the chunks at the eight



Figure 4.7: Bob is about to upload alice.txt

	id	filename	filesize	chunks	created	last acce	fileowner	uploaded	deleted	filehash
1	1	alice.txt	1329	8	2011-12-	2011-12-1	2343	0	0	

Figure 4.8: Cloudfile in the database, not uploaded.

	id	hash	file id	chunk no	destination	uploader
1	1	36825e86d204556e12b6f9247c2b9723	1	0	providerA	1
2	2	1eae87369a3e8ffc16f43cc1641f6395	1	1	providerB	1
3	3	e47302e39a0f9fd0910c52afb589b2f8	1	2	providerC	1
4	4	79b3d96d9289dcec55be891695f00b1e	1	3	providerD	1
5	5	b6204145098a282fcb73b0959ee9d81d	1	4	providerE	1
6	6	552e7509842baeee6cb1539aba827eb0	1	5	providerF	1
7	7	f54f756fed7bbf51bfb373bcad894a0	1	6	providerG	1
8	8	4902823804f90ff2133251eb18ea394c	1	7	providerH	1

Figure 4.9: Cloud chunks in the database after being uploaded.



cloud providers.

	id	filename	filesize	chunks	created	last acc	fileowner	uploaded	deleted	filehash
1	1	alice.txt	1329	8	2011-12-	2011-12-	2343	1	0	

Figure 4.10: Cloudfile in the database, uploaded. Notice that the uploaded-value is changed from 0 to 1.

So, alice.txt is safely stored in the cloud, and what is seen in figure 4.11 is what Bob can see in the dashboard.

The screenshot shows the RAINC&C dashboard interface. At the top is a dark grey header with the logo 'RAINC&C'. Below the header are two main sections: 'Your distributed files:' and 'Upload file:'. The 'Your distributed files:' section contains a table for the file 'alice.txt' with the following details: ID: 1, Name: alice.txt, Uploaded: true, Created: 18-12-11 22:03:47, Last accessed: 18-12-11 22:03:47, Filesize: 1329, No. of chunks: 8, Fileowner: 2343, Deleted: false. Below the table are two icons: a green download arrow and a red X. The 'Upload file:' section includes a file selection button labeled 'Velg fil' with the text 'Ingen fil valgt', a 'Number of Chunks' input field with a value of 8 and a description 'Optional value to define the number of chunks you want to split the file into. Default is 8.', and an 'Upload file' button. Below these sections is a 'History' section with a single entry: '18-12-11 22:03:49: alice.txt was successfully uploaded'.

Your distributed files:	
alice.txt	
ID:	1
Name:	alice.txt
Uploaded:	true
Created:	18-12-11 22:03:47
Last accessed:	18-12-11 22:03:47
Filesize:	1329
No. of chunks:	8
Fileowner:	2343
Deleted:	false
 	

Upload file:

File: Ingen fil valgt

Number of Chunks: Optional value to define the number of chunks you want to split the file into. Default is 8.

History

18-12-11 22:03:49: alice.txt was successfully uploaded

Figure 4.11: Alice.txt is successfully uploaded.

Chapter 5

Results

In this chapter we try to present measurements of the C&C's performance. These results will be discussed in section 6.1.

5.1 Measurements

The C&C is doing IO operations on the file when splitting and merging. Therefore we want to measure the IO overhead when uploading and uploading. We will do measurements with different file sizes and vary the number of chunks to find the best conditions for using the C&C.

All benchmarks are done from the benchmarking computer, shown in subsection 3.4.1, to the OpenStack Storage provider at Sintef, shown in table ??.

The measurements where done with version 1.6 of the C&C, downloadable from the repository.

5.2 Procedure

We first upload files with different sizes without splitting them, followed by downloading these files again. The uploading and downloading without splitting is the reference for which to measure the overhead against.

Then we start the same procedure with split files. We start with splitting into chunks, then downloading and merging these chunks into a file. Th by uploading the same files split into different numbers of chunks.

1. Upload all files in the set to the cloud providers, without splitting.
2. Download these files.

3. Repeat steps 1-2, with splitting and merging the files in a varying number of chunks.
4. Repeat steps 1-3 until the sample space is large enough.

Each upload and download is recorded. After the benchmarking is finished, the average upload and download duration and upload and download bitrate is calculated. The results are shown in the graphs below.

Also, when splitting and merging, the time taken to split and merge is reported. Since splitting is done asynchronously with uploading, the split time is not of much value. The merge time, however, is of great value seeing as we need to download all chunks before merging them.

5.3 Accuracy of Results

Every value presented is a result of a set of minimum ten uploads or downloads, where each value is within 90% of the median value in the result set. When analysing the data, the values seemed to be so consistent that the result set obtained was deemed large enough to draw conclusions.

5.4 Other Measurements

Every significant improvement in code is tested before approval to be certain that the change is in reality an improvement. These measurements are not presented here as they are of little use to the reader.

5.5 Results

5.5.1 Uploading

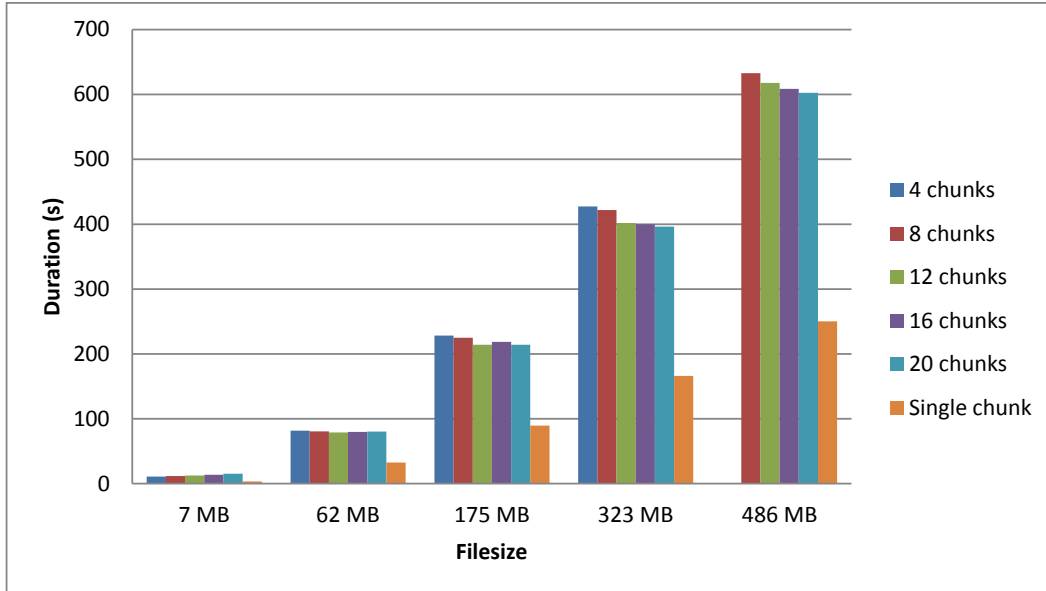
Figure 5.1a and 5.1b shows the duration of uploads in seconds.

From the graphs we see that more chunks leads to faster uploads. Control measurements have been performed, with 30 and 50 chunks, without including these in the graphs, and they show that the tendency continues.

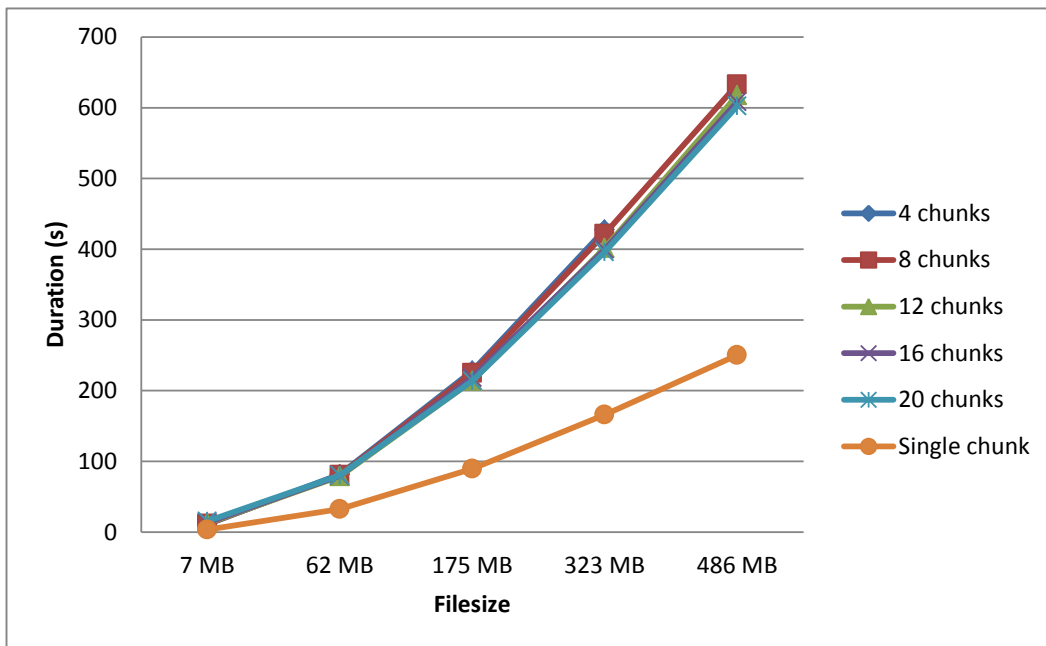
When looking at the overhead in percent, see table 5.1, we see that the overhead is off the charts when regarding the 7MB file. If we exclude that file, we notice that the overhead is almost consistent per the number of chunks.

We conclude that the average overhead is a little above 140% when the number of chunks is between 16 and 20.

Figure 5.2 shows the bitrate achieved, and the same tendencies as discussed earlier are clearly visible here also.

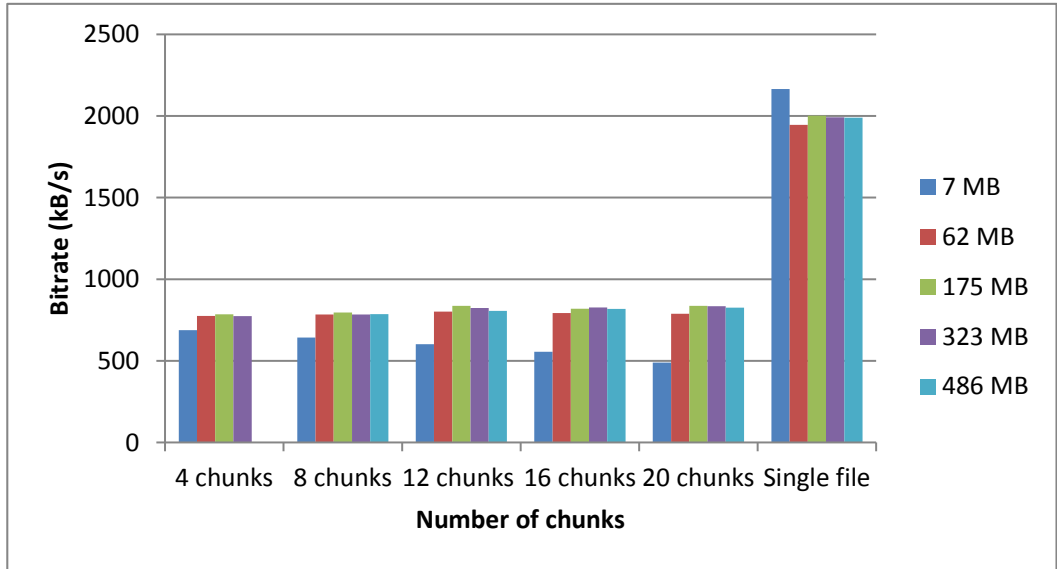


(a)

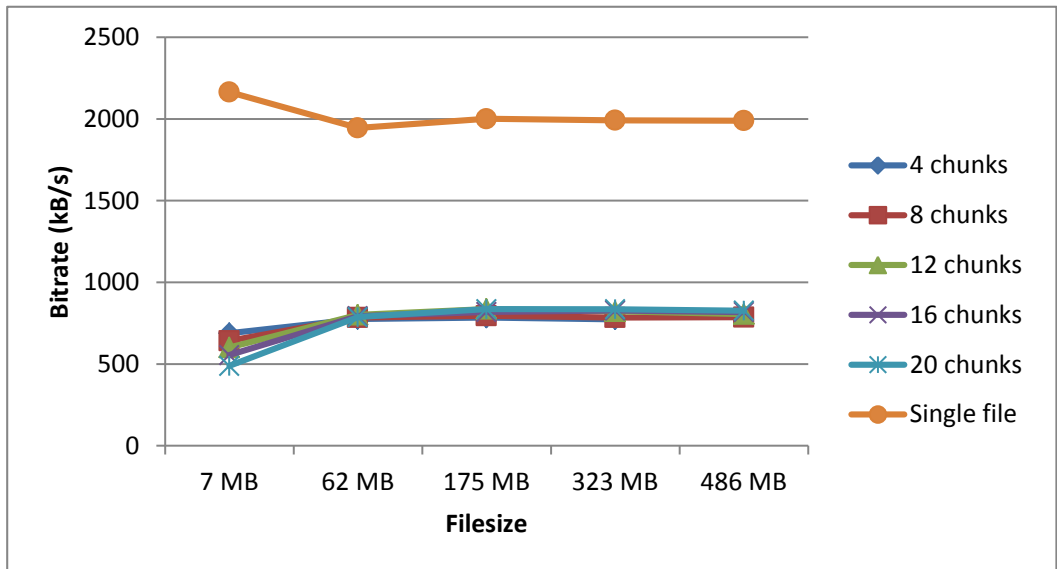


(b)

Figure 5.1: The duration of uploading files of different sizes and different number of chunks to the provider.



(a)



(b)

Figure 5.2: The bitrate when uploading files of different sizes and different number of chunks to the provider.

	4 chunks	8 chunks	12 chunks	16 chunks	20 chunks
7 MB	214%	237%	260%	289%	342%
62 MB	151%	148%	143%	145%	147%
175 MB	155%	151%	139%	144%	139%
323 MB	157%	154%	142%	141%	139%
486 MB	-	153%	147%	143%	141%
Average	169%	169%	166%	173%	182%
Average wo 7MB	154%	152%	143%	143%	141%

Table 5.1: The overhead in percent when uploading files.

5.5.2 Downloading

Figure 5.3a and 5.3b shows the duration of downloads in seconds.

From the graphs we see that optimum number of chunks is between 12 and 20 chunks, represented by 16 chunks here. From table 5.2 we see that the average overhead is 73% when using 16 chunks, so we will operate with this number as the overhead when downloading.

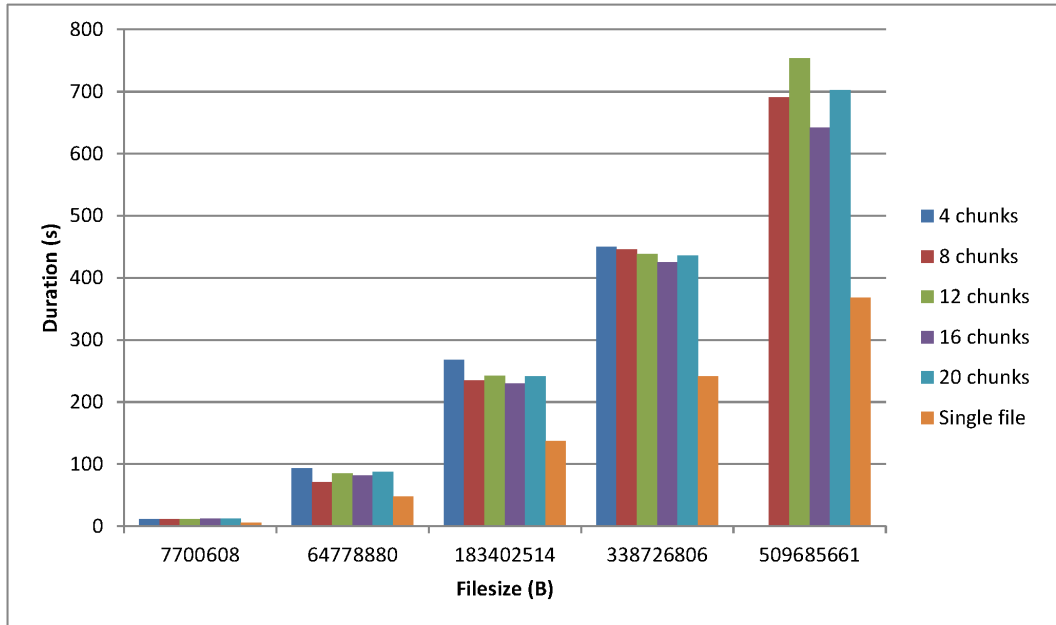
Figure 5.4 shows the bitrate when downloading, obviously just representing the same values as in figure 5.3 in a different way.

	4 chunks	8 chunks	12 chunks	16 chunks	20 chunks
7 MB	103%	106%	102%	122%	130%
62 MB	96%	73%	78%	72%	84%
175 MB	95%	71%	76%	68%	76%
323 MB	87%	85%	82%	76%	81%
486 MB	-	88%	105%	75%	91%
Average	95%	84%	89%	83%	92%
Average wo 7MB	93%	79%	85%	73%	83%

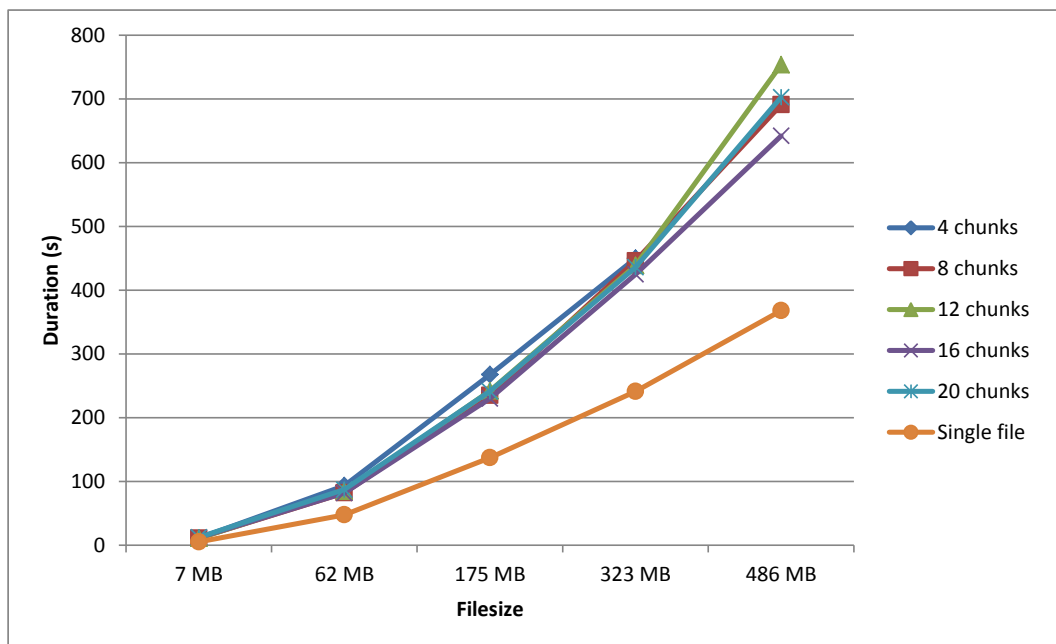
Table 5.2: The overhead in percent when downloading files.

5.5.3 Total

Figures 5.5 and 5.6 shows that the total duration, therefore also the total lowest overhead is when using 16 chunks.

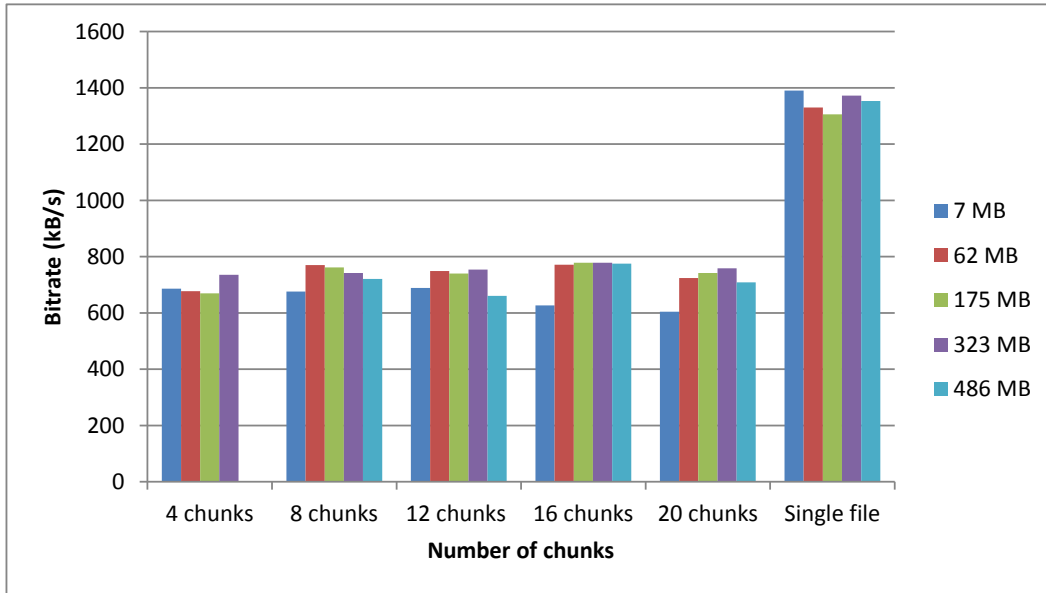


(a)

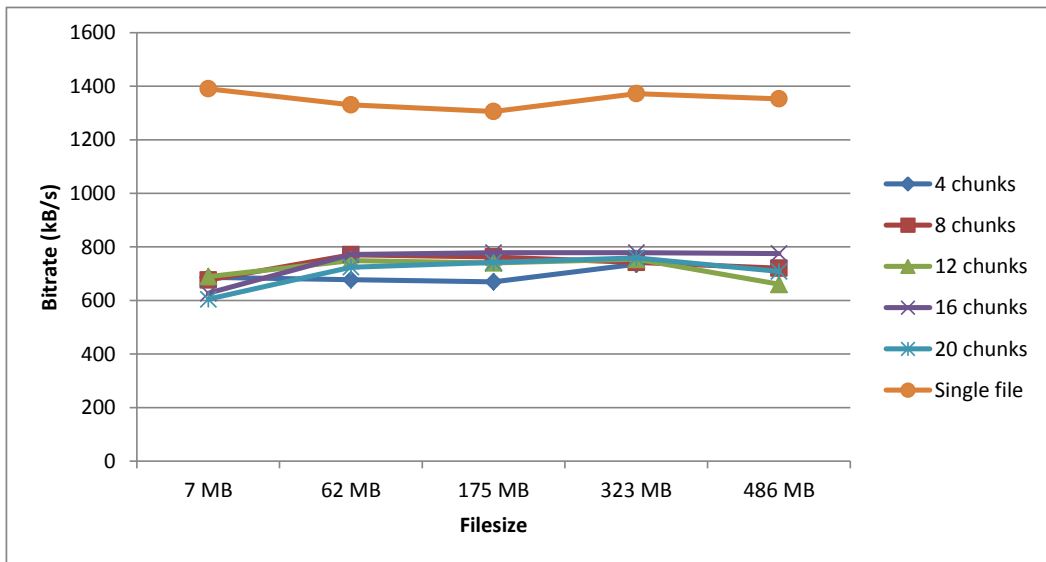


(b)

Figure 5.3: The duration from starting download to all chunks are merged.

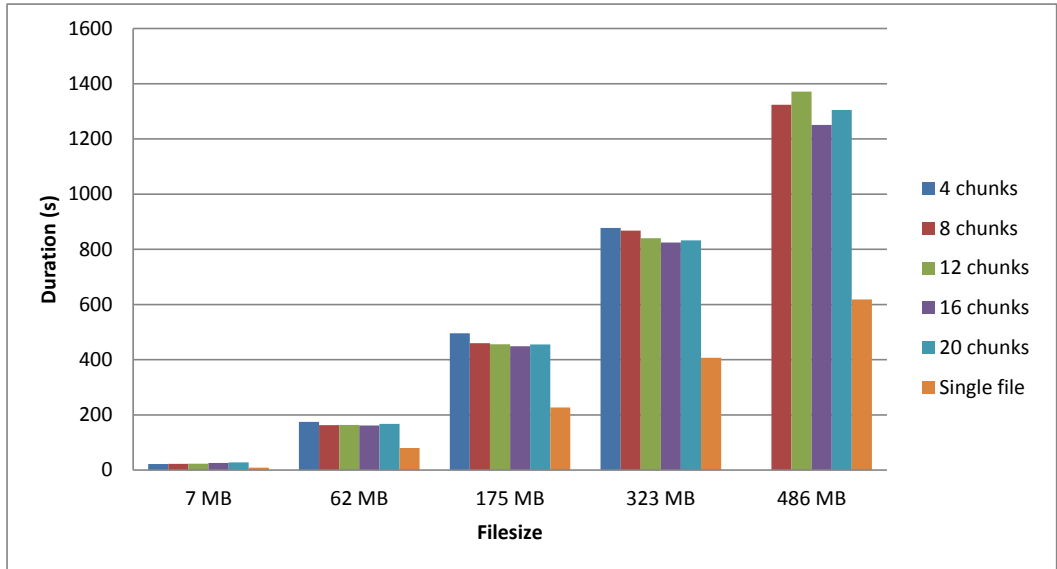


(a)

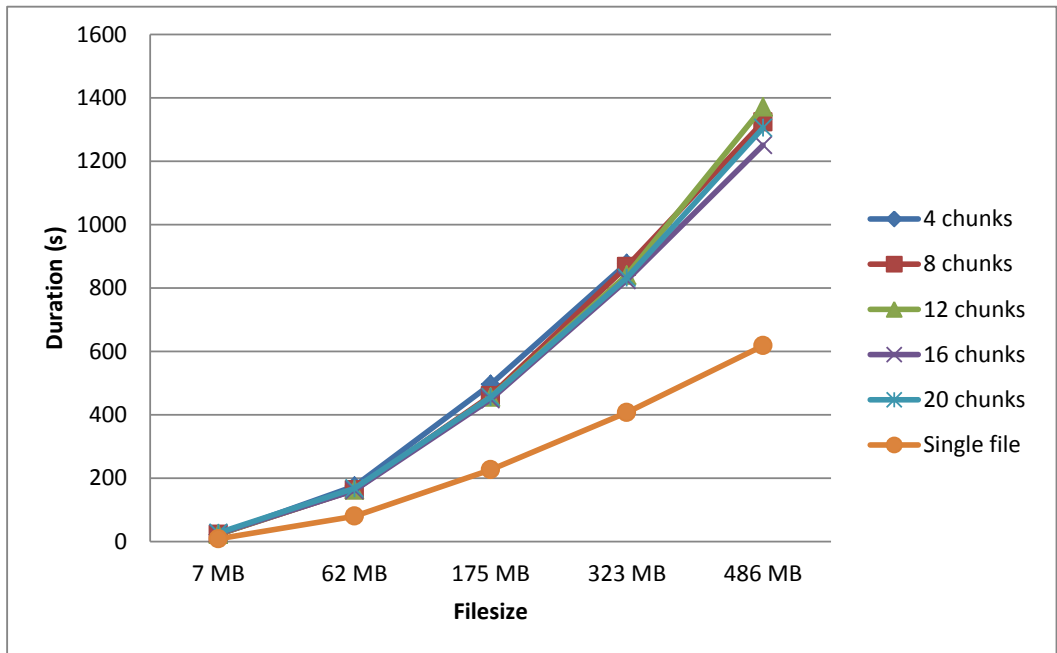


(b)

Figure 5.4: The bitrate from starting download to all chunks are merged.

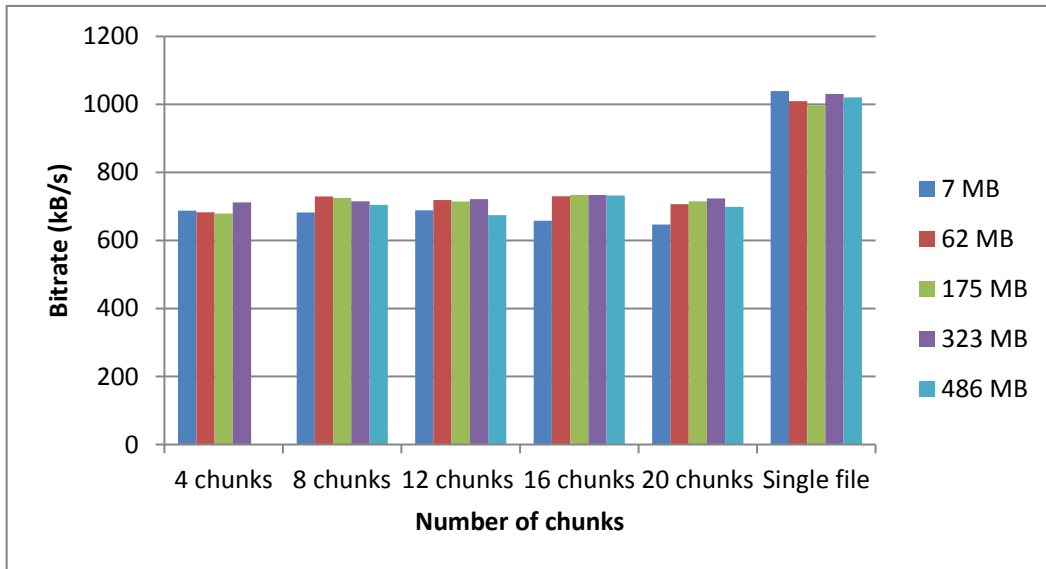


(a)

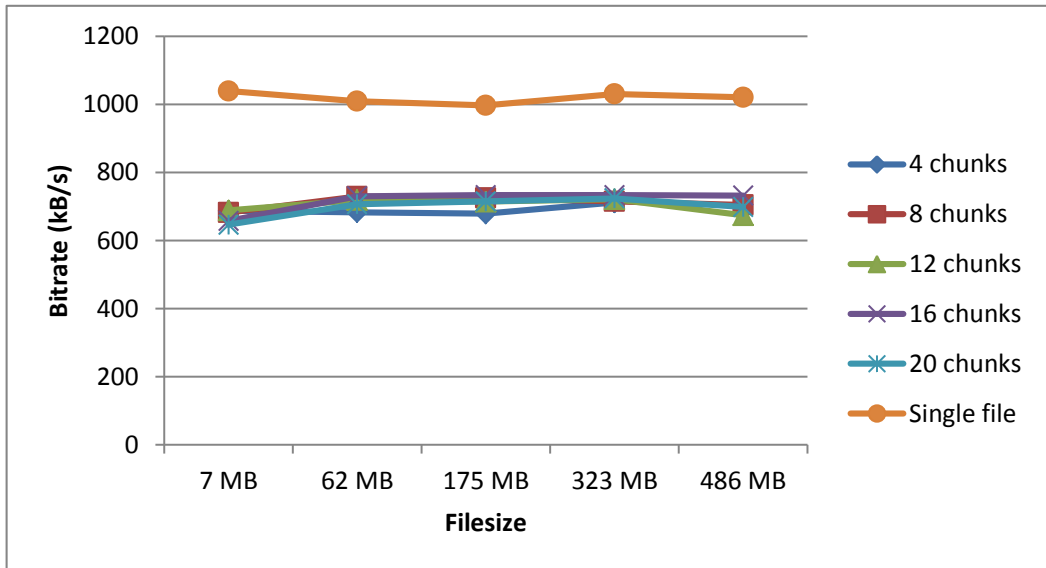


(b)

Figure 5.5: Total average duration of upload and download.



(a)



(b)

Figure 5.6: Total average bitrate of upload and download.

Chapter 6

Discussion

6.1 Discussion of Results

In this section we will look at the results from chapter 5, give an explanation to the numbers and discuss how the performance could be improved.

6.1.1 Overhead

6.1.1.1 Upload

Looking at the overhead on uploads, we see that writing chunks takes a considerable amount of time. When discarding all but the smallest file, the overhead percentage remains consistent when increasing the file size. The duration of each split is smaller than the duration of each upload, which means that the system needs to wait for the next chunk to be finished writing to disk.

6.1.1.2 Download

As with uploads, when discarding the smallest files, the overhead remains consistent when increasing the file size. The overhead on downloads comes from having to wait for all chunks to download before merging can start.

6.1.1.3 Improvements on Overhead

Tweaking the algorithms to improve on the overhead might result in much better results.

Suggestions include:

- Do adjustments so individual tuning of the storage provider interfaces have a big impact.
- Find out if splitting can be done in threads, and if this will be an improvement.

- Use better suited storage solutions with faster writing, e.g. SSD, RAID.
- Find out if it is possible to combine splitting and uploading in a way so that the chunks could be uploaded directly from memory.

6.1.2 Importance of Performance

Even if it is unquestionably a matter of importance, it can be argued that the performance issues might be negligible when we consider the cases we foresee for RAIN in section 6.5. In these cases, the typical use pattern allows the user to start the uploading of a file, for then not to think about the remainder of the process.

6.1.3 Redundancy

An important point with data security is redundancy. This is also pointed out by Jaatun et al. [19, 41] when introducing RAIN, and we have dedicated a portion of our background chapter to introduce other schemes for splitting and distributing files.

The main reason for not implementing redundancy in this prototype of the C&C is the abundance of existing solutions and lack of time.

Proposed solutions for implementing redundancy is to write a new splitting class with redundancy in mind, and use IDA, proposed by Lu et al. [20], or Shamir's secret-sharing scheme [37] in the development of this splitting class.

6.2 How RAIN Solves Privacy Issues

Since RAIN splits up the data and spreads them out to multiple cloud providers, it won't be possible for a single provider to get any valuable information out of the data. This can be a good thing in several scenarios.

6.2.1 Cloud Provider is Compromised

If you store your whole file unencrypted in the cloud and the provider gets compromised by an attacker, you will lose your data. With RAIN the attacker will only get parts of the file that will most likely look like garbage. This gives extra security to both the customer and the cloud provider, but of course it does not mean that preventive security measures should be omitted.

6.2.2 Curious Cloud Providers

As mentioned in the theoretical background chapter some cloud providers aggregate data from their customers to make customers profiles that can be used to i.e. target advertisements. RAIN makes sure that the customers

do not have to worry about this, since their data won't be accessible for the cloud provider. This is of course a disadvantage for the cloud provider which probably earn money on the targeted advertising.

6.2.3 Government Claims

Due to the patriot act, US companies may have to provide the US government with any data that is requested, even if this data is stored in a data center on another continent [39]. If a customer uses RAIN, the cloud provider will not be able to give the government any data, even if it wanted to. If the government wants the data it has to go directly to the customer that controls the C&C.

6.2.4 Drawbacks

RAIN makes it harder to access data if you don't have access to the C&C. This can be misused by criminals to hide or try to hide data from governments or law enforcement. Another issue you have to be aware of is that the distributed nature of RAIN can make the system harder to manage. If the customers have special needs or requirements, you have to make sure all parts of the system that are used by this customer meet these needs. If i.e. all the data has to be stored within Europe, all the cloud providers that store chunks for this customer has to be located within Europe.

6.3 Limitations

Unfortunately our solution is not perfect. Here we discuss some of the limitations to it.

6.3.1 Data Processing in the Cloud

One big advantage with cloud computing is the possibility to use the cloud for scalable processing of data. Since RAIN split up the data and send it to different cloud providers, we also lose this possibility. The data has to be merged together in the C&C before it can be processed, but we can't send the merged files back into the cloud since then we give away all our data by letting the cloud providers access it.

6.3.2 Corrupted or Lost Chunks

Our version of RAIN does not have any redundancy or error correction code, which makes it vulnerable if one of the cloud providers loses a chunk or if a chunk gets corrupted or lost during file transfer. If the chunk can't be recovered, the data in the original file will be lost.

6.3.3 The Splitting Algorithm

The splitting algorithm is designed by us and it is pretty simple. Other more advanced splitting methods with more mathematical proofs should be implemented. Proposed alternatives for splitting are the same as in section 6.1.3, using either IDA [20] or Shamir's secret-sharing scheme[37].

6.4 Possible Attacks

6.4.1 Eavesdropping the Network

If an attacker eavesdrop the network connection out from the C&C it will be possible for him to collect all the data that are transferred. Even if a secure transfer protocols like SSL is used, it will be possible to sniff the key or perform a man-in-the-middle attack to obtain the plaintext data.

Collecting the chunks from one file can be done by assuming that chunks sent at approximately the same time, belong together. If the attacker does this, he still has to put the chunks together. If the attacker knows the splitting algorithm and have collected all the chunks from the file, he also needs to know the order of the chunks. If the number of chunks is low, this can be done easily with some trying and failing.

To make it a little harder for the attacker to reassemble the file, there are some countermeasures that can be used. First the C&C can randomize the number of chunks for each file. The randomization will make it harder for attackers to know the exact number of files to reassemble since they don't know how many chunks each file consist of.

Another method is to send file chunks in batches, i.e. let the C&C send chunks from two different files at the same time. If the attacker can't distinguish the chunks from each other it will be a lot harder to reassemble the original files.

A third method is to introduce garbage chunks that are transferred together with the real chunks. Even if the attacker knows that the data contains garbage chunks he needs to sort out which chunks it is to be able to obtain the original file.

6.4.2 Attacking the C&C

The C&C is definitely the most valuable in the system from an attackers point of view. If the C&C is exposed, either physically or digitally, the attacker will be able to access the data or perform a denial of service (DoS) attack.

In our system design we propose that the C&C should be within the company's own infrastructure. This means that they have to secure the C&C physically by making sure it is located in an locked room where only authorized people

can access it. They also have to make sure that the machine is not remotely accessible through the network. This means that they have to have a proper firewall, authentication, updated software and disabled unused services.

If the security of the C&C fails, the attacker will have access to the main database. This means that he can find out who owns the files, where they are located and how to reassemble them. To make the system more secure, all the data in the database can be encrypted with a user specific password that must be applied for every upload or download.

6.4.3 Attacking the Cloud Provider

An attack on a single cloud provider will not exploit any of the original data since the chunks do not contain any information by themselves. An attacker has to attack all the cloud providers and find the chunks that belongs together by i.e. look at the file created timestamps. This attack have the same countermeasures as in eavesdropping mentioned above.

Even if an attacker can't obtain the original data, he can still do a lot of damage if he compromises a single provider. If he changes or deletes the data in any of the chunks, the C&C will not be able to reassemble the original file.

6.4.4 Curious or malicious cloud providers

Curious or malicious cloud providers will not be able to obtain any of the original data since they only have some parts of it. If all the cloud providers decide to collaborate, we will have to introduce the same countermeasures as in eavesdropping mentioned above. They will still be able to corrupt or delete chunks and thus destroy the original data.

6.5 Commercial Use of RAIN

RAIN is well suited for secure data storage in the cloud. A commercial version of RAIN could be implemented like the popular data storage service, Dropbox [?]. User interfaces for different type of clients can be implemented on top of RAIN. This will make your data accessible from i.e. your desktop or mobile device. Files can be uploaded to the cloud asynchronously in the background, so even if RAIN introduces some overhead during file transfer, the user wont see much to it. File stored here will much likely be 0-50 MB, like images and documents.

Another use of RAIN can be for data backup. Backup can take a lot of space and the cloud is perfect for storage of the backup files. You can use a application like Zamanda Cloud Backup [?] and then use RAIN to store the

data in the cloud. Backup files are usually large and it can be uploaded in batches at night.

RAIN can also be implemented all places where secure data storage in the cloud will be needed and where some overhead in the transfer time is allowed. A great advantage with RAIN is that it supports interfaces against multiple cloud providers and it is based on open source software which is an great advantage to prevent vendor lock-in.

Chapter 7

Conclusion and Further Work

7.1 Further work

During our implementation of the prototype we have discovered some areas that need further work.

7.1.1 Performance

As shown in the results, the overhead of splitting and merging the files is pretty high and measures needs to be taken to lessen this overhead. Section 6.1.1.3 suggests solutions for improvement.

7.1.2 Security

Security is the main priority of this application and as we pointed out in the discussion section 6.4, there are some more security measures that can be implemented. To make the system more secure against eavesdropping garbage chunks, batch data transfer and more randomization of the number of chunks should be implemented.

A lot of the security in this system also relies on the splitting method which is pretty basic in our prototype. More advanced splitting methods are mentioned in ??, and should be utilized.

Considerations about how well the C&C should be secured should be taken. You can choose to only secure the server that the C&C is running on, or you can add more security by encrypting the database like mentioned in 6.4.2.

7.1.3 Redundancy

Our system as it is today is very vulnerable to data loss or data corruption. To make the system more robust, redundancy must be introduced. Solutions on how to introduce redundancy are proposed in section 6.1.3. The solution used

here will be closely connected with the implementation of a more advanced splitting algorithm, see section 7.1.2, as redundancy is taken care of when splitting and merging.

7.2 Conclusion

In this project we have implemented a prototype called Rain C&C based on previous work done by Jaatun et al. about Redundant Array of Independent Net-storages (RAIN).

We have shown that implementing a simplified version of their design is feasible and working. We have also shown that our solution integrates well with open source cloud solutions like OpenStack as well as a commercial version like Amazon's S3. This is a great advantage to prevent vendor lock-in and makes migration from one cloud provider to another easier.

We have tested and measured the prototype and we have pointed out some improvements that can be applied to our prototype when it comes to performance and security, in addition to lack of protection against data loss and corruption. These issues are addressed and we have proposed solutions that can be implemented and explored in further work.

Bibliography

- [1] D.J. Abadi. Data management in the cloud: Limitations and opportunities. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 32(1):3–12, 2009.
- [2] Amazon. Amazon AWS. <http://aws.amazon.com/>. [Website; last visited 12-18-2011].
- [3] Amazon. Amazon Simple Storage Service. <http://aws.amazon.com/s3/>. [Website; last visited 11-24-2011].
- [4] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. Technical report, Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.
- [5] A.A. Atayero and O. Feyisetan. Security issues in cloud computing: The potentials of homomorphic encryption. *Journal of Emerging Trends in Computing and Information Sciences*, 2(10), 2011.
- [6] Mat Bettinson. Amazon AWS downtime shakes faith in the cloud. <http://www.pcr-online.biz/news/read/amazon-aws-downtime-shakes-faith-in-the-cloud/020014>. [Website; last visited 12-15-2011].
- [7] Peter Bogetoft, Dan Lund Christensen, Ivan Damgard, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Multiparty computation goes live. Cryptology ePrint Archive, Report 2008/068, 2008. <http://eprint.iacr.org/>.
- [8] Cloud business review. The 10 Worst Cloud Outages: Lessons Learned. <http://www.cloudbusinessreview.com/2011/06/30/the-10-worst-cloud-outages-lessons-learned.html>. [Website; last visited 12-15-2011].

- [9] Thierry Carrez. Delivery channels for OpenStack on Ubuntu. <http://fnords.wordpress.com/2011/06/15/delivery-channels/>. [Website; last visited 12-18-2011].
- [10] D. Catteddu. Cloud computing: benefits, risks and recommendations for information security. *Web Application Security*, pages 17–17, 2010.
- [11] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 542–552. ACM, 1991.
- [12] Eucalyptus. Eucalyptus Website. <http://www.eucalyptus.com/>. [Website; last visited 11-17-2011].
- [13] Foursquare. foursquare.com. <https://foursquare.com/>. [Website; last visited 11-24-2011].
- [14] Google. Build your business with Google’s cloud services. <http://www.google.com/enterprise/cloud/>. [Website; last visited 12-18-2011].
- [15] Google. Gmail Privacy Policy. <http://www.google.com/intl/en/privacy/privacy-policy.html>. [Website; last visited 12-15-2011].
- [16] J. Gray et al. The transaction concept: Virtues and limitations. In *Proceedings of the Very Large Database Conference*, pages 144–154, 1981.
- [17] A.R. Hevner, S.T. March, and J. Park. Design science in information systems resarch. *MIS Quarterly*, 28(1):75–105, 2004.
- [18] IDC. New IDC IT Cloud Services Survey: Top Benefits and Challenges. <http://blogs.idc.com/ie/?p=730>. [Website; last visited 12-15-2011].
- [19] M.G. Jaatun, A.A. Nyre, S. Alapnes, and G. Zhao. A farewell to trust: An approach to confidentiality control in the cloud. In *Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference on*, pages 1 –5, 28 2011-march 3 2011.
- [20] W. Lu, Y. Zhou, L. Liu, and B. Yan. An ida-based parallel storage scheme in the scientific data grid. *Data Science Journal*, (0):1005150227, 2010.
- [21] Robert McMillan. Salesforce.com Warns Customers of Phishing Scam. http://www.pcworld.com/businesscenter/article/139353/salesforcecom_warns_customers_of_phishing_scam.html. [Website; last visited 12-18-2011].

- [22] P. Mell and T. Grance. Effectively and securely using the cloud computing paradigm. *NIST, Information Technology Lab*, 2009.
- [23] P. Mell and T. Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800:145, 2011.
- [24] Microsoft. Windows Azure. <http://www.windowsazure.com/en-us/>. [Website; last visited 12-18-2011].
- [25] Microsoft. Windows Azure Storage. <http://www.microsoft.com/windowsazure/features/storage/>. [Website; last visited 11-24-2011].
- [26] J. Oberheide, E. Cooke, and F. Jahanian. Empirical exploitation of live virtual machine migration. In *Proc. of BlackHat DC convention*, 2008.
- [27] OpenNebula. OpenNebula Website. <http://opennebula.org/>. [Website; last visited 11-17-2011].
- [28] OpenStack. Configuring Object Storage with the S3 API. http://docs.openstack.org/trunk/openstack-object-storage/admin/content/configuring-openstack-object-storage-with-s3_api.html. [Website; last visited 12-18-2011].
- [29] OpenStack. Nova Concepts and Introduction. <http://nova.openstack.org/nova.concepts.html>. [Website; last visited 12-17-2011].
- [30] OpenStack. OpenStack Image Service. <http://openstack.org/projects/image-service/>. [Website; last visited 12-18-2011].
- [31] OpenStack. Openstack Starter Guide. <http://docs.openstack.org/diablo/openstack-compute/starter/content/>. [Website; last visited 11-16-2011].
- [32] OpenStack. Openstack Website. <http://openstack.org/>. [Website; last visited 11-17-2011].
- [33] Quora. Quora.com. <http://www.quora.com/>. [Website; last visited 12-15-2011].
- [34] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36:335–348, April 1989.
- [35] Rackspace. Windows Azure Storage. http://www.rackspace.com/cloud/cloud_hosting_products/files/. [Website; last visited 11-24-2011].

- [36] N. Santos, K.P. Gummadi, and R. Rodrigues. Towards trusted cloud computing. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, pages 3–3. USENIX Association, 2009.
- [37] Adi Shamir. How to share a secret. *Commun. ACM*, 22:612–613, November 1979.
- [38] Steven J. Vaughan-Nichols. Canonical switches to OpenStack for Ubuntu Linux cloud. <http://www.zdnet.com/blog/open-source/canonical-switches-to-openstack-for-ubuntu-linux-cloud/8875>. [Website; last visited 11-17-2011].
- [39] Zack Whittaker. Google admits Patriot Act requests; Handed over European data to U.S. authorities. <http://www.zdnet.com/blog/igeneration/google-admits-patriot-act-requests-handed-over-european-data-cite{c}-to-us-authorities/12191>. [Website; last visited 12-15-2011].
- [40] A.C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.
- [41] G. Zhao, M.G. Jaatun, A. Vasilakos, A.A. Nyre, S. Alapnesy, Q. Yue, and Y. Tang. Deliverance from trust through a redundant array of independent net-storages in cloud computing. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 625–630. IEEE, 2011.

Appendix A

Example of a a file and its chunks

Listing A.1: Full text file.

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

So she was considering in her own mind (as well as she could, for the hot day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

There was nothing so very remarkable in that; nor did Alice think it so very much out of the way to hear the Rabbit say to itself, 'Oh dear! Oh dear! I shall be late!' (when she thought it over afterwards, it occurred to her that she ought to have wondered at this, but at the time it all seemed quite natural); but when the Rabbit actually took a watch out of its waistcoat-pocket, and looked at it, and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across the field after it, and fortunately was just in time to see it pop down a large rabbit-hole under the hedge.

Listing A.2: Chunk no. 1.

```
Asiet snogg hetosanioenotwtokgeuroiose ssu
eesndhp ns etbepcess wka
et m ;dt t h stO !leno t,u tgaett melw a wt ato
r,ste hsi eeb aoe ou,rtser f t te dab h.
```

Listing A.3: Chunk no. 2.

```
l ntisbi kf oteeoisgt svn,hhf,h teno rhm
ltd lld)elogyw hlt k ,uaRi nb
hvai hsm weRashO ! uoe rh hvrh tae )hRcta
wt,oitr toeiesnsnfnie atw t nhi oftauw ioriue
```

Listing A.4: Chunk no. 3.

```
ibg riyst , ntnw p ks , p es ae 't' svnscieiwsdhah  
e ,ref -oweetaid d ate y  
wiernnAiouoaaaye hIb'  
sgvriresteeiaildn ;eatotoa- kthiAa ttd dheo tiwt ,  
at ai trsiennaittwgtn
```

Listing A.5: Chunk no. 4.

```
ce vet th hoocihet tr hior 't a wp e?horrneh , eyeves  
a acuo innawdWbhyc  
anrk oln cfyrb ld e hhewterh dstml a nbuocfipae , eelrh ,  
h evra ta- taonncyaserdasno ne-dh
```

Listing A.6: Chunk no. 5.

```
egtedtheeaat ecadhheebacrsia u tAior 'eng dle  
reptwtsm hlrtondgihehb elhTsgyatrikvh  
btfeds (etra d etw , e qtb bakh sond ndite faet  
ee whipockfdgu , n l t p hee
```

Listing A.7: Chunk no. 6.

```
ior ier nvhd ed eeraudt annisbhltrs  
s o l fhm ryuhhuadadrfg seniipsoeh bh c e ttio , aehlw  
ro t ooa t suuutil itcd a cerflcrhhr riesorhe r  
tdifejtsoarord
```

Listing A.8: Chunk no. 7.

```
wn yonr bdiioo i r dt uct dseooiht a  
wiiw( cooafy peerkai ho ptinltti srensladeirohoht  
raahtiadctho ntbhieirhtlaotck anhoed oar aa satrtc  
ibwisah toluiep al g
```

Listing A.9: Chunk no. 8.

```
ang fg oa mn:rspnb wi nroii oucouctSadnnaaortde ait eiinb  
uguihe ye nre.rooeeti tyue e i '!rltehtfscoauhd ueteta e y  
usoeltdun frsomtdebh ckatotuiohcea , rysme lbete
```