INSTITUTO POLITÉCNICO DE BRAGANÇA

Pedro Sernadela

D3S – A Distributed Storage Service

Escola Superior de Tecnologia e de Gestão

Novembro 2012

D3S – A Distributed Storage Service

Dissertação apresentada ao Instituto Politécnico de Bragança para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Sistemas de Informação, sob a supervisão do Prof. Doutor Rui Pedro Lopes.

Pedro Sernadela
Novembro 2012

Abstract

In recent years, computing allowed an explosion of service provisioning in the cloud. The main paradigm dictates the migration of user information, such as documents, photos and others, from the desktop to the network, allowing anytime, anywhere access through the Internet.

This paradigm provided a adequate environment to the emergence of online storage services, such as Amazon S3. This kind of service allows storing digital data in a transparent way, in a pay-as-you-go model.

On the other hand, peer-to-peer networks have been responsible for most Internet traffic, mostly derived from the BitTorrent protocol. The inherent support to resource sharing, scalability and fault-tolerance, have been making this approach popular, particularly for file-sharing applications.

This thesis presents an implementation of an S3 compatible storage service based on peer-to-peer networks, in particular, through the BitTorrent protocol. Several measurements are discussed, to assess differences in access time and throughput.

Agradecimentos

A minha sincera gratidão a todos os que me encorajaram e contribuíram neste percurso de realização da minha tese de mestrado.

Gostaria de agradecer em particular e principalmente ao meu orientador pela disponibilidade e apoio prestado e também a todos pela paciência, compreensão e ajuda prestada na etapa de realização de testes. À minha adorada família que esteve sempre do meu lado ao longo desde percurso.

Agradeço também aos meus amigos, professores e ao IPB, pelas excelentes condições de trabalho e recursos académicos que me foram proporcionados.

A todos muito obrigado!

"Innovation distinguishes between a leader and a follower."
Steve Jobs

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet growth allowed an explosion of service provision in the cloud. Among the multitude of services available in the cloud, storage services is one of the most popular. This storage paradigm dictates the users information migration from the desktop into the network allowing access everywhere, anytime.

Along with the security and privacy concerns that arise with this data shift, Service Level Agreements (SLA) also play an important role, as it is needed to increase the overall trust that the provider depends on and to guarantee the level of productivity that consumers demand. Relying on a cloud storage service has a broad set of challenges, that include security and privacy concerns but also capacity, access speed, availability and cost. Nowadays, services such as Dropbox, Amazon S3, Google Drive and others provide an easy to use, flexible way to store information and easy access.

Following a different paradigm, peer-to-peer networks offers unique characteristics that foster the adoption of alternatives to several existing client-server applications. Peer-to-peer offers a radically new way of distributing information among a broad number of symmetric nodes (or peers) instead of concentrating it at a single server. This solves many issues, related to fault tolerance, load balancing and availability.

BitTorrent is one of many P2P file-sharing systems that has attracted millions of users. In this protocol each peer is responsible for maximizing its own download rate by contacting specifically chosen peers. High upload rates is a factor that contributes to the choice of a peer to connect to, contributing to an overall high speed download.

Cloud services provisioning usually rely on a single connection, that, probably, will affect the SLA. To assess connection characteristics between this two different paradigms, we implemented an Amazon S3 cloud storage service over a peer-to-peer network. This implementation is used to measure connection speed and throughput, thus comparing both paradigms according to several scenarios.

1.1 Overview

Cloud storage services are provided by organizations that purchases storage capacity over the time. This offers the ability to scale according to the demand, providing a

lower cost for cloud based services. This type of architectures are very scalable and can easily manage huge amounts of data, whereas managing this much data with a traditional database may involve unforeseen costs.

On the other hand, having data accessible to third parties, such as a service provider, can represent security and compliance issues. And once it serves multiple customers, various issues related to multiple customers sharing the same piece of hardware can arise, e.g. if one user compromises the system with the malicious data, it can also compromise the other users data that share the same system. Above all, hosted cloud services are often remote and they can suffer from latency and bandwidth related issues associated with any remote application.

In peer-to-peer (P2P) networks the participants are at the same time both suppliers and consumers of resources (in contrast to the traditional client–server model). Without the need of the direct connection to the central data unit repository, this distributed architecture allows good resource availability and high performance proved by the file sharing environment.

In this way, our Distributed Simple Storage Service (D3S) seeks to delivery a peer-to-peer based storage service functionally compatible with Amazon S3. This system is based on BitTorrent and thus making use of features where each user makes an information repository of files available for distribution. Combined with the ability of others peers to join the network, leads to the fast growth of a network composed of distributed information, constituting an high available system with high transfers rates.

In order to achieve this peer-to-peer storage service based on the S3 interaction model many challenges can hamper the final solution. The suitability of the BitTorrent operation model to working with the S3 functionality and the peers coordination represent the main challenges discussed in this document. With the results measurement analysis we intend to assess throughput of this alternative storage service providing a study comparison with the S3 service.

1.2 Goals

This document manly focus in studying and comparing peer-to-peer to cloud file storage services. In order to get some conclusions, some tasks were defined:

- Develop of a compatible Amazon S3 REST interface.
- Build a desktop application for accessing the S3 service.
- Implement a compatible S3 server, following the same authentication model and functionality.
- Implement a compatible S3 server, backed by a BitTorrent network.
- Organize and measure transfer and access speeds for three different scenarios: Amazon S3, local, centralized server and peer-to-peer.

1.3 Document structure

This document is structured in five main chapters, as follows. Chapter 1 makes an introduction to the scope of the document. Chapter 2 describes all technologies involved on the implementation. Chapter 3 describes the overall architecture implemented to develop the alternative Amazon S3 storage system over the BitTorrent protocol. In the chapter 4 it can be found the measurement analysis and discussion of all experiments. Finally, Chapter 5 concludes the paper.

Chapter 2

State Of The Art

Cloud computing is clearly one of today's most appealing technology areas. It represents the result of evolution of different technologies that allowed more processing capacity, virtualization of resources and the ability to store data relying in existing bandwidth. This factors combined led to a new paradigm that responds to a set of current problems as scalability, cost reduction, fast operation of applications and solutions.

On the other hand, this new ecosystem leads us inevitably to new challenges of integration, architecture, organization of resources and business models change. In this way, the cloud provides more business agility and is used as a competitive system through rapid deployment, parallel batch processing, use of compute-intensive business analytics and mobile interactive applications that respond in real time to costumer requirements [Marston et al., 2011]. For that, and others reasons, the phenomenon has been widely adopted by the general public and by industry leaders such as Microsoft, Google, IBM, Amazon,...

In a parallel approach, peer-to-peer networks are based on a model in which each computer acts as a client and server to other computers, providing an infrastructure for sharing resources, such as files, computing power, peripherals and others.

Many reasons arises to invest time and effort in a P2P system. Namely, it is a vehicle to implement new concepts in networks or even economics. P2P technology influences and demonstrates to have impact in many fields of human endeavor. The bottom line is that P2P resumes itself to person-to-person (no central infrastructure necessary) where people work together for a specific and usually common goal.

Many technologies are related to P2P, that provide the infrastructure to organize peers, route requests and responses (Distributed Hash Table – DHT, ...), as well as providing a protocol, such as The BitTorrent[1]. In sum, P2P applications can be used to share any type of digital asset in the form of packed information that can be uniquely identified.

This chapter makes an overview of the main technologies involved in the final solution, introducing terminology and describing typical architectures.

[1] `http://www.bittorrent.org/beps/bep_0003.html`

2.1 Cloud Computing

According to NIST (National Institute of Standards and Technology), *Cloud computing is a model for enabling ubiquitous, 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* [Mell and Grance, 2011]. With the evolution of Cloud Computing there are many definitions, but shortly it refers to both the applications delivered as services over Internet and the hardware and systems' software in data centers that provide these services [Armbrust et al., 2009].

The term "Cloud" comes from the data center hardware and software association and usually is based on a model of payment associated with the use of resources.

An application that works on cloud can use resources on a pay-as-you-go manner, i.e., a customer can start the process on a small scale and dynamically adjust the use of resources according to their needs. In this way, it is evident that the cloud computing is regarded as an utility for businesses and is becoming prevalent in organizations.

One of the great advantages of cloud computing is its resilience. In theory, cloud computing services are built in such way that if a machine fails, the system readjusts itself so that the user will never perceives that a machine has failed. Given this characteristic, cloud computing is a promising technology to ensure a level of stability that a single server cannot provide.

There are many issues to consider when an enterprise consider to move their applications to the cloud environment. The majors benefits of this change are briefly presented below:

Shared resource pooling: The infrastructure provider offers a pool of computing resources that can be dynamically assigned to multiple resource consumers providing more flexibility and reducing providers operating costs. Examples of resources include storage, processing, memory, network bandwidth, and virtual machines.

Broad network access: Any device with Internet connectivity is able to access cloud services because the capabilities are available over the network.

Service oriented: In a cloud, each IaaS, PaaS and SaaS provider offers its service according to the Service Level Agreement negotiated with its customers [Zhang et al., 2010].

Dynamic resource provisioning: One great feature of cloud computing is that computing resources can be purchased in any quantity at any time according to consumer needs.

Utility based price: Cloud computing uses a pay-per-use pricing model that allows customers pay only for the services resources that they use. Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service [Mell and Grance, 2011].

2.1.1 Service Models

It is possible to distinguish two different architectural models for clouds [Dikaiakos et al., 2009] (Figure 2.1). One is designed to expand providing additional computing instances of demand like supply services in the form of Software-as-a-Service (SaaS) and Platform-as-a-Service (PaaS) and the other model is designed to provide scalable resources such hardware, bandwidth and storage (Infrastructure-as-a-Service – IaaS). According to this three fundamental models the IaaS is the most basic and each higher model abstracts from the details of the lower models. The self-service functionality of this cloud services allows users to obtain, configure and deploy autonomously without requiring the assistance of IT.

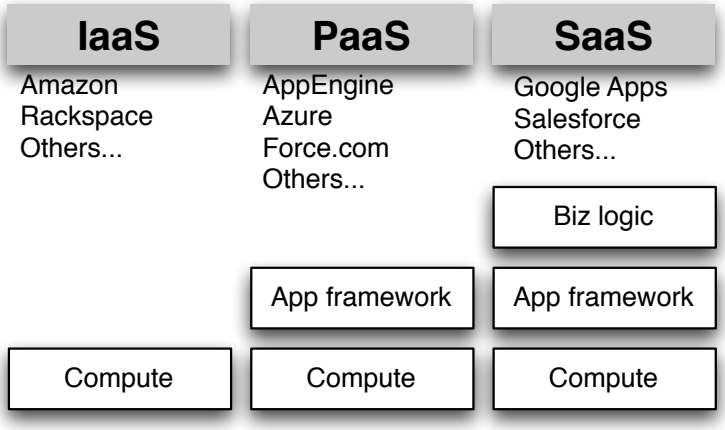

Figure 2.1: Cloud Service Model.

Software-as-a-Service (SaaS)

SaaS is a software delivery method that provides access to software and its functions remotely as a Web-based service. These applications are typically web applications that run embedded in the browser. The users can use the applications without having to install or run on their local machines, since the SaaS vendors offer software as a service on the network. Also, the users does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user specific application configuration settings [Mell and Grance, 2011].

Namely, on the customer side, it means no upfront investment in servers or software licensing and on the provider side, costs are low compared to conventional hosting [Knorr and Gruman, 2008].

Platform-as-a-Service (PaaS)

This type of platform provides a set of features that contains developer tools to develop applications, web graphical interface, databases and others. This tools with

the providers API (Application Programming Interface) allows to create and deploy new applications based on Cloud services. These help developers focus on business logic, freeing them from concerns about infrastructure (because they do not need to manage or control the underlying cloud infrastructure [Mell and Grance, 2011]) as well as getting advantages of PaaS quality of services available. But, in order to protect the stability of the whole system and to ensure the QoS for real-time interactive applications these systems restricted some functionality of the application (e.g. only comunication by http/s, limited access to the file system, etc).

Many PaaS providers exist today such as Google AppEngine, Microsoft Azure, Salesforce, Rackspace Sites, Bungee Connect, EngineYard, Heroku, Intuit, Cloudera, Aptana, VirtualGlobal, LongJump, AppJet, Wavemaker, Aprenda, and others [Boniface et al., 2010].

Infrastructure-as-a-Service (IaaS)

Infrastructure as a Service (IaaS) is the delivery of hardware (server, storage and network), and associated software (operating systems virtualization technology, file system), as a service [Bhardwaj et al., 2010].

It is an evolution of traditional hosting that does not require long commitment and allows users to provision resources on demand.

The IaaS supplies resources such as virtualized servers, network devices, and many other resources, reducing management operations burden and without compromising deployment and management of the software services, control operating systems, storage, applications, and networking components (e.g., host firewalls) [Mell and Grance, 2011].

IaaS providers compete on the performance and pricing offered on their services by delivering different hardware, bandwidth, memory and storage to their customers. So, the equipment is owned by the provider and is responsible for housing, running and maintaining it [Bhardwaj et al., 2010].

However, the key benefit of IaaS is to be flexible on the pricing, since the customer only pay for the resources that their applications require. Other important concept to take account is that PaaS/SaaS services usually work over the IaaS layer. Examples of IaaS offerings are Amazon Web Services Elastic Compute Cloud (EC2) and Secure Storage Service (S3).

2.1.2 Deployment Models

There are many considerations to be taken for migration and development of applications based on this new paradigm. There are three different type of clouds, each with complementary benefits and drawbacks: private, public and hybrid. A cloud might be restricted to a single organization or group (private clouds), available to the general public over the Internet (public clouds), or a composition of two or more clouds (hybrid clouds) [Dikaiakos et al., 2009, Mell and Grance, 2011, Ramgovind et al., 2010, Zhang et al., 2010]. When a cloud is shared by groups or organizations it can also be called community cloud [Mell and Grance, 2011].

Private Cloud

Private clouds, also known as internal clouds, are designed for exclusive use by a single organization. Data and processes are managed within the organization without the restrictions of network bandwidth, security exposures and legal requirements [Rimal et al., 2009].

This type of cloud offers the highest degree of control over performance, reliability and security [Zhang et al., 2010]. Nevertheless, they are often criticized for being similar to traditional server infrastructure.

Public Cloud

The public cloud infrastructure is made available to the general public or a large industry group in a remote location and provides flexible means without high upfront capital investment.

The public clouds key benefits is that they represent a structure much larger than a private cloud, ensuring more flexibility and scalability. Likewise, public clouds are less secure than the other models because any attacker with a credit card can establish an account on some virtual machine in the cloud, and begin hacking through the hypervisor, a risk that does not apply in conventional systems [Rosenthal et al., 2010]. This is an additional burden, ensuring that all applications and data accessed on the public cloud are not subjected to malicious attacks [Ramgovind et al., 2010]. Generally, the maintenance becomes the responsibility of the organization that selling Cloud Computing services.

Hybrid Cloud

The hybrid cloud infrastructure combines two or more clouds (private, community, or public) in the same data center, centrally managed, provisioned as a single unit, and circumscribed by a secure network [Ramgovind et al., 2010]. This association tries to address the limitations of each approach offering more flexibility than both public and private clouds.

Designing this cloud model requires carefully determining the best split between public and private cloud components. [Zhang et al., 2010] Hybrid clouds may soon associate a public cloud with a private cloud that hosts the most sensitive data [Rosenthal et al., 2010].

Community Cloud

Community clouds are shared by several organizations. Normally, support a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations) [Mell and Grance, 2011, Marinos and Briscoe, 2009].

To summarise, in the cloud services model, users can purchase services in the form of Infrastructure-as-a-service (IaaS), Platform-as-a-service (PaaS), or Software-as-a-service (SaaS), according their needs and the decision of which type of Cloud

to deploy relies basically on security considerations of the enterprise architecture [Ramgovind et al., 2010].

2.2 Cloud Services Delivery

The structure of cloud services (SaaS, PaaS and IaaS) contextualizes the providers' position in the market. The trust on the cloud provider is one factor to take in consideration. When a customer subscribes a service, usually there is a "mandatory connection" with the provider, because in most cases there is no possibility for service migration.

Recently, several groups have been appearing to focus on the interoperability and compatibility problems. One example is the Cloud Computing Interoperability Forum[2] that *"was formed in order to enable a global cloud computing ecosystem whereby organizations are able to seamlessly work together for the purposes for wider industry adoption of cloud computing technology and related services"*. With this the customers can change the provider with more flexibility because there is a framework to do that, allowing information exchange between clouds.

Nowadays, there are several service providers that offers different types of services and resources (e.g. storage, database, ...), which contributes to market competition and more and better services.

2.2.1 Commercial Services

The implementation of a cloud system requires one large investment on the infrastructure and a lot of experience in information technologies (IT). The industry leaders like Amazon and Google took advantage of the fact that they already possess large infrastructures and profit from their business.

Amazon is considered a leading provider of cloud services. The platform Amazon Web Service (AWS) offers a wide range of services. Amazon EC2 is an IaaS example of cloud computing service. EC2 provides access to different types of virtual machine images using the open-source virtualization middleware Xen [Ostermann et al., 2010].

It provides the flexibility to choose from a number of different instance types[3]. Each instance provides a certain amount of dedicated computing capacity and is charged per instance-hour consumed. Every instance of Amazon EC2 is a VM (Virtual Machine), and there is no functionality to backup the modifications on the virtual disk space. So in most cases, users need to purchase storage space.

The Simple Storage Service (S3) allows storing the modifications of customized machine images registering the disk image for later use. The S3 provides a storage service that enables persistent data and it also has mechanisms for redundancy and disaster recovery.

[2] `http://www.cloudforum.org/`
[3] `http://aws.amazon.com/ec2/instance-types/`

Some research work have pointed this service as a good start for a reasonable scientific environment and for quick experiments from the poit of view of cost performance [Ostermann et al., 2010, Akioka and Muraoka, 2010].

Google App Engine[4] (GAE) is a PaaS which provides facilities for creating Web based services and applications. This platform supports two widely known programming languages, Java and Python, and allows users to run and host their web applications on Google data centers.

Developers can use the GAE and server applications free of charge or obligation with some restrictions like to 5 million page views per month (bandwith and cpu usage) and can consume only 500mb of persistence storage. However, it is possible to scale to whatever traffic and data storage is needed [Ciurana, 2009]. Applications created are offered as SaaS, consumed directly from the end-users web browsers.

The Microsoft IaaS/PaaS platform is Windows Azure[5], which also allows developers to run applications, storage data and rent some computing power over the cloud. They offer an SDK (.NET), that allows the development of applications, and SQLAzure, providing data storage services in the cloud. This platform can be used by applications running in the cloud and by applications running on local systems [Zhang et al., 2010]. The operation model price can be as a pay-as-you-go manner or by six month plans.

Salesforce[6] is an enterprise cloud computing company providing SaaS that supplies a set of tools to model the client business data in its cloud service. The client access a configurable interface, from the site of the cloud service, as well as through Simple Object Access Protocol (SOAP). The operation model is associated with a user/monthly subscription service.

They also provide a PaaS called Force.com that allows the developer to deploy the application in the Salesforce cloud hosting using Apex programming language.

In short, the global tendency of cloud enterprises is not only offering SaaS platforms of their products but also appearing in other levels like PaaS where developers build and launch their applications to bridge the gaps in their business. SaaS replace some traditional software products but will not eliminate them anytime soon [Cusumano, 2010].

2.2.2 Business model

There are many different potential pricing models such as a flat fee (usually monthly or yearly), a pay-per-use fee (units with fixed price values), or a mix of both. Pay-per-use is what customers usually prefer and it is the most popular business model among cloud providers. Users and providers prefer simple, static models in which it is easy to predict payments [Weinhardt et al., 2009]. Thus, an application that works on cloud can use resources on a pay-as-you-go manner, i.e., a customer can start the process on a small scale and dynamically adjust the use of resources according to their needs without a long term contract with service providers.

[4] `http://code.google.com/appengine/`
[5] `https://www.windowsazure.com`
[6] `http://www.salesforce.com`

These pricing policies should be analyzed in conjunction with capacity investment decisions and QoS guarantees, to decide which of these policies are the best suited for the organization/costumer in the particular market target.

In the cloud storage market the providers can improve the revenue by offering customized forms of payment and products to achieve the customer satisfaction level. However, penalty policies may be defined in the SLA.

2.2.3 Service Level Agreement

The existence of several providers offering the broad spectrum of cloud services and resources (e.g. computing, storage, database, ...) provides and adequate competition ecosystem, that is great for more and better services, since the quality and reliability of the services constitutes important aspects at the time of the adoption.

There are many points that an organization or a customer really cares about when choosing a cloud service. One of them is the possibility of almost immediate access to hardware resources without needing significant initial capital expenditure. This possibility paves the way to the small enterprises trying to benefit from large amounts of computing power that was present only on the largest corporations [Marston et al., 2011]. Typically this compute power involves short periods of time saving extreme infrastructure that are really need. The opportunity to scale their services makes it more easy and desirable, for example, it is very comfortable to increase the server storage space at distance of a mouse click.

In a traditional computing setup, the main stakeholders are the providers and consumers and as more and more consumers empower their duties to cloud providers, Service Level Agreements (SLA) between consumers and providers become an important aspect. On the provider side it is difficult to meet all the customers expectations and the solution can be a balance preparation of their needs. At this point the social perspective can be relevant [Demirkan et al.,]. The negotiation process can start between the provider and client when the client wants to buy a service. At the end of this negotiation the provider and the consumer committed to an agreement. In Service Oriented Architecture (SOA) terms, this agreement is referred to a SLA [Patel et al., 2009]. Therefore, an SLA is a formal contract used to guarantee that consumers service quality expectation can be achieved [Wu and Buyya, 2010].

Definition 1 *A Service Level Agreement is a document that includes a description of the agreed service, service level parameters, guarantees, and actions and remedies for all cases of violations [Alhamad et al., 2010].*

In cloud computing systems, different QoS (Quality of Service) attributes (such as response time and throughput) have to be guaranteed and closely monitored by the consumer. In this type of scenario resource availability continuously vary and the assurance that service quality is kept at acceptable is one crucial factor (for example, S3 is designed to provide 99.999999999% durability and 99.99% availability of objects over a given year). A clear SLA includes both general and technical specifications (i.e. business parties, pricing policy,...). Moreover, the concise contract helps

parties to resolve conflicts more easily because they define the reward and penalty policies of the service provision. In other words, consumers need an SLA before they transfer their infrastructure to cloud data centers to ensure the desired level of productivity, and cloud providers need an SLA to define the trust, quality and reliability of services they provide.

The parties, when committed to an agreement, are bound to pay attention to their rights and obligations related to notifications of breaches in security, data transfers, creation of derivative works, change of control, and access to data by law enforcement entities [Marston et al., 2011]. The contract clauses may deserve additional review because of the cloud data storage nature: what happens if the copyright law in the country where the data is stored allows legal copying of files? – questions like this should have an answer in the SLA document.

Many languages and frameworks have been developed for SLA specification and management. The WS-Agreement and Web Service Level Agreement (WSLA) are examples of the most popular and widely languages used in research and industry [Wu and Buyya, 2010].

Much of Cloud services providers offer predefined SLA documents. One example is the Amazon EC2. The EC2 Service Level Agreement abides user compensation if the resources are not available for acquisition at least 99.95% of the time in the year [Ostermann et al., 2010]. So the customer is eligible to receive a service credit equal to 10% of their bill.

In sum the SLA assurance is a critical objective for every provider and a aspect key for the client adoption time.

Another factor that the client sometimes not given importance in case of not sensible data dealing is its security. Anyway nobody likes that their data becomes public aware.

2.2.4 Data Security

In a cloud data storage system, the users' local data is stored on the distributed cloud servers. Although the existence of encryption and access control systems some organizations feel afraid when their proprietary data is switched to a public access cloud. This fact is more relevant in the case of sensitive data. To ensure fast access data and to prevent data loss the data must be mirrored on data centers that are geographically far apart. Replication, in turn, makes it even more vulnerable [Rewatkar and Lanjewar, 2010].

Examples of security breaches in IT systems appear from time to time and cloud computing its not different. One of the key issues is to effectively detect any unauthorized data modification and corruption [Cong Wang et al., 2009]. Besides, in the distributed systems when such inconsistencies are successfully detected, in most cases, they are quickly recovered without the users perception.

Despite outsourcing data into the cloud is economically attractive for the cost and complexity (large scale data storage) it does not offer any guarantee on data integrity and availability [Cong Wang et al., 2010].

Verify the correctness of outsourced cloud data without the local copy can be a daunting task because traditional cryptographic cannot be applied. In order to save the users cloud computing resources and ensure the data security one option may be enable public auditability for the data storage [Cong Wang et al., 2010].

Summarizing, the advantages of using clouds are incontestable but as mentioned above there are many risks sharing the data in a cloud system mainly due the increased risk of loss of users privacy [Cachin et al., 2009].

2.2.5 Cloud Storage Frameworks

Many cloud providers offer a wide variety of data storage services, some relying on WebServices for implementing additional functionalities. To ensure interoperability, some SDKs have been created and many are progressively growing to cope with the features that characterizes a cloud system.

The jclouds[7] open-source library allows to use portable abstractions or cloud specific features with the opportunity to reuse java developments skills. So without dealing with REST-like APIs this framework provides drivers that allow operate in restricted environments like Google App Engine. With stub connections that simulate a cloud without creating network connections ensures the unit testability. At the moment, the jclouds structure includes two abstraction APIs: Compute and Blobstore. The Compute API helps bootstrap machines in the cloud and the Blobstore API helps to manage key-value data. All of abstractions are location-aware. This API support 30 cloud providers and cloud software stacks, including Amazon, GoGrid, Ninefold, vCloud, OpenStack, and Azure.

The AWS Software Development Kit[8] include libraries, documentation, templates, sample applications, and other developer tools prepared to build applications in different technologies like Android, iOS, Java, .NET, PHP and Ruby, that tap into the cost-effective, scalable, and reliable Amazon cloud. Using the AWS SDK, developers can build solutions for Amazon Simple Storage Service (Amazon S3), Amazon Elastic Compute Cloud (Amazon EC2), Amazon SimpleDB, etc. The library provides APIs that hide much of the lower-level plumbing, including authentication, request retries and error handling.

Apache Libcloud[9] is a standard Python library that abstracts away differences among multiple cloud provider APIs. This library currently supports more than 26 different providers. Example of Cloud Servers supported are Amazon EC2 and Rackspace CloudServers (libcloud.compute.*) and for Cloud Storage are Amazon S3 and Rackspace CloudFiles (libcloud.storage.*).

The Dasein Cloud[10] is an open-source Java API made available by enStratus Networks LLC under the terms of the Apache License 2.0. It provides an abstraction for applications that wish to be written independent of the clouds they are controlling. With this API the developers can write the calls against the Dasein Cloud API without having to learn the specifics of the web services calls from different

[7] http://www.jclouds.org
[8] https://aws.amazon.com/code/
[9] http://libcloud.apache.org
[10] http://dasein-cloud.sourceforge.net

providers. This API specifies cloud-independent interfaces for enabling an application to access resources across multiple clouds like Amazon, AT&T Synaptic Cloud (Storage), Azure (Storage), Cloud Central, CloudSigma, GoGrid, Google (Storage), Joyent, Rackspace, Tata, Terremark, etc.

JetS3t[11] is a open-source Java toolkit and application suite for Amazon Simple Storage Service (S3), Amazon CloudFront content delivery network and Google Storage. This toolkit provides to developers a powerful and simple API for interacting with storage services and managing data stored there. In the JetS3t suite there are too applications for managing the account contents (e.g. Cockpit, Synchronize, etc.) and for mediated account access (e.g. Gatekeeper, Uploader, etc.).

2.3 Peer-to-peer Networks

The limitations of the client-server architecture became evident in the Internet distributed environment. Since the development of the original application and service of Napster (the first P2P file sharing program) the use of P2P systems have increased and nowadays multiple nearly clones emerged with millions of users sharing their files. This only happened because these system proved its practical applicability. The P2P in the scientific research community is a popular topic with many projects exploring the characteristics of this type of networks. Today, this networks are used on a large scale in order to share a number of features with a large number of users and is currently wrapped in discussion due copyright and licensing issues.

The peer-to-peer term is generally assigned to a network architecture where all nodes offer the same services and follow the same behavior making files available to download as well as downloading them. Because this server and client approach, the node is common called a *servant*. In this architecture the set of servants share resources between them, which may include bandwidth, storage space, and computing power. Allowing access to its own resources with resource sharing support requires self organization, scalability properties and fault-tolerance. This last provide the system stability because if one peer on the network fails the whole network is not compromised.

In a typical client–server architecture, clients share only their demands with the system, but not their resources. In this case, as more clients join the system, fewer resources are available to each client, and if the central server fails, the entire network is taken down. The peer-to-peer technology aims to solve that.

Furthermore, it can provide an infrastructure in which the desired information can be located and downloaded preserving the anonymity of both requesters and providers. Other great feature is the lack of a system administrator on this type of networks providing a easier and faster setup running with efficiency and stability without a full staff to control it.

In recent research and derived from these features the peer-to-peer networks have accounted for roughly 43% to 70% (depending on geographical location) of all Internet traffic [Schulze and Mochalski, 2009].

[11] `http://www.jets3t.org`

2.3.1 Concepts and Definitions

In short, *a distributed network architecture may be called a Peer-to-Peer (P-to-P, P2P,...) network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers,...). These shared resources are necessary to provide the Service and content offered by the network (e.g. file sharing or shared workspaces for collaboration): They are accessible by other peers directly, without passing intermediary entities* [Schollmeier, 2001].

Peer-to-peer systems often implement an abstract overlay network, built at the application layer, on top of the native or physical network topology. Such overlays are used for indexing and peer discovery and make the P2P system independent from the physical network topology. Content is typically exchanged directly over the underlying Internet Protocol (IP) network. This peer-to-peer overlay systems go beyond services offered by client-server systems where a client may also be a server (Figure 2.2).

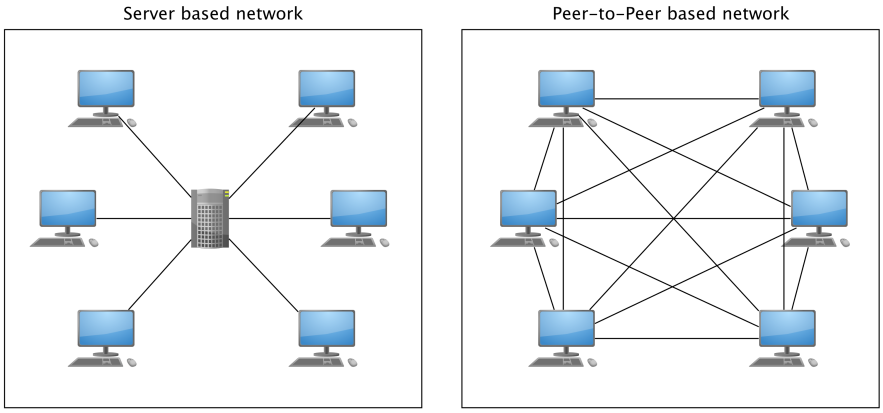

Figure 2.2: Comparison model of P2P and Server based networks.

Currently, P2P systems can be classified into two main categories: unstructured and structured [Wang et al., 2005]. The technical meaning of structured is that the P2P overlay network topology is tightly controlled and content is placed at specific locations that will make subsequent queries more efficient [Eng Keong Lua et al., 2005].

Structured P2P systems (such as Chord, CAN, Tapestry and Pastry) use a Distributed Hash Table (DHT), in which data objects' (or values) location information is placed deterministically, at the peers with identifiers corresponding to the data objects unique key. In contrast to the random overlay in unstructured systems, all peers in structured systems are organized into a clear logical overlay that allows DHTs to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

For unstructured P2P systems (such as Gnutella and KaZaA) peers are organized arbitrarily. Each peer connects with other peers randomly. An arbitrary overlay

network is formed by those connections among peers.

Although, these systems are simple, they provide friendly keywords searching models and powerful abilities to locate duplicated objects. Related with unstructured P2P architectures is normally associated the query flooding algorithm. In flooding, the node who wants the file simply broadcast its search query to its immediate neighbours. If the neighbours do not have the resource, they forward it to its neighbours until a maximum number of hops. This mechanism generates a large mount of messages per query, which difficulties the scalability when the number of peers grows.

Other typical protocol used is gossiping. In this "epidemic" mechanism each participating node chooses another random node to send the query. Then, each node combines the information it receives from other nodes with information it already has and propagate it in the next round [Zaharia and Keshav, 2008]. So each member node is in charge of forwarding each message to a set of other, randomly chosen, group members [Ganesh, 2003]. Normally, the gossip mechanism adds a bandwidth overhead to the system.

Among the non-structured P2P networks, are considered the pure, the hybrid and the centralized P2P networks (Figure 2.3).

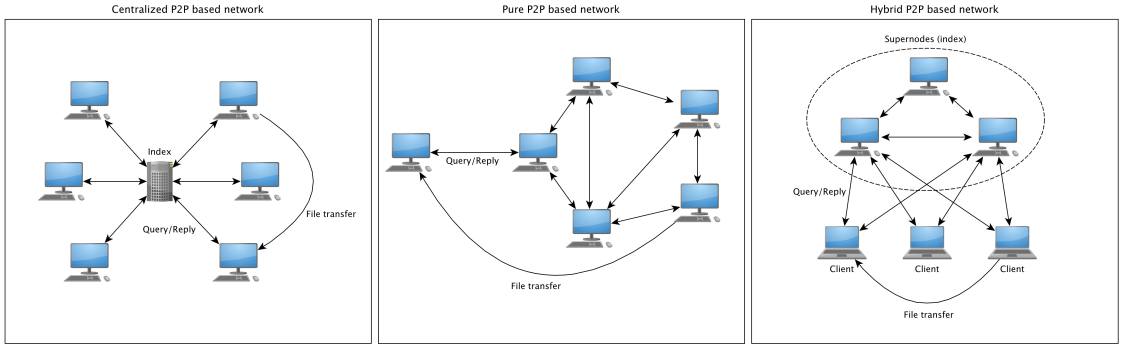

Figure 2.3: Comparison model of P2P non-structured based networks.

In pure P2P networks, each peer acts as an equal, integrating the client and server role. In such networks, there is not a central point for managing and routing in the network. The most important characteristic of a pure P2P network is the fact that any of the peers can be removed without suffering any loss in services provider.

In hybrid P2P networks nodes can assume different roles. A group of nodes or a central node usually perform control tasks and information indexing. Others are still free to communicate directly with each other.

In centralized P2P networks peers connect to a central node, where they publish information and services. The central node when receive a solicitation, forwards it to the peers of the list that can answer it. The rest of the operation is carried out between two nodes without intervention of the central node. In case of failure of the central node, the network cease to operate.

2.3.2 File Sharing

Peer-to-peer networks have become more influential since 1999 with the introduction of Napster, a file sharing program that linked people who had files with those who requested files.

In this system a connection is established from the user local machine to Napster's central servers, informing them of local files available for sharing. Thus, these servers keep a complete list of the songs (only music files supported) shared on each machine currently connected. When someone searched for a file, the server would find all of the available copies of that file and present them to the user. The files would be transferred between the two private computers.

After Napster shut down in 2001[12] because copyright infringement, Gnutella and Kazaa rose as the most popular peer-to-peer service. While Napster connected users through a centralized server, these new services connected users remotely to each other. These services also allowed users to download files other than music.

Another protocol that emerged more recently is BitTorrent. BitTorrent allows users to create an index file containing the metadata of the files they want to share, and upload them to dedicated websites to make them available to downloaders.

BitTorrent became the third generation protocol of P2P networks[13]. The main difference between BitTorrent and previous generations is that it creates a new network for every set of files instead of trying to create one big network of files using servers.

Nowadays it constitutes one of the most common protocols for transferring files, counting over 150 million active users [14].

2.4 BitTorrent

The P2P overlay networks attempt to solve a long list of features as well as problems [Eng Keong Lua et al., 2005]:

- selection of nearby peers;
- self-organization;
- load balancing;
- redundant storage;
- efficient search of data items;
- data guarantees;
- hierarchical naming;
- trust and authentication;

[12] http://www.businessweek.com/2000/00_33/b3694003.htm
[13] http://filesharingz.com/guides/filesharing-history.php
[14] http://www.bittorrent.com/intl/es/company/about/ces_2012_150m_users

- anonymity;
- massive scalability;
- fault tolerance.

So peer-to-peer computing offers a radically new way of isolating and focusing on the networking aspect. A P2P network distributes information among member nodes instead of concentrating it at a single server and this paradigm offers exciting advantages in information sharing [Parameswaran, 2001].

In a P2P model, each member node can make information available for distribution and can establish direct connections with any other member node to download information.

The P2P networks widely known today engage primarily in sharing music, video or gaming software not directly contributing to electronic commerce. This networks are mostly using the BitTorrent protocol. However, the P2P paradigm, more particularly the BitTorrent Specification[15] can be extended and enhanced productivity in other areas.

2.4.1 Operation Model

BitTorrent[16], a peer-to-peer file sharing protocol, is used for distributing large amounts of data over the Internet. With BitTorrent, the task of distributing the file is shared by those who want it, i.e. it is entirely possible for the peer to send only a single copy of the file itself and eventually distribute it to an unlimited number of peers. It is fundamentally different from all previous peer-to-peer applications (like Napster and Gnutella) because it does not rely on a peer-to-peer network federating content sharing users. Instead, it creates a new transfer session, called a torrent, for each file.

Clients involved in a torrent, cooperate to replicate the file among each other using swarming techniques. A user who wants to upload a file first creates a torrent descriptor file (with the extension '.torrent') that they distribute by conventional means (web server, email, etc.) starting acting as a *seeder*.

Peers that want to download the file must first obtain a torrent file for it and connect to the specified tracker, which tells them from which other peers to download the pieces of the file (typically 256 KB but other piece sizes are possible too).

BitTorrent distinguishes between two kinds of peers depending on their download status: clients that have already a complete copy of the file and continue to serve other peers are called *seeds*; clients that are still downloading the file are called *leechers*.

The tracker is not involved in the distribution of the file. It only coordinates the file distribution by keeping track of the peers currently active in the torrent. The tracker is the only centralized component of BitTorrent.

Pieces are typically downloaded non-sequentially and are rearranged into the correct order by the BitTorrent client. Peers download each piece by connecting to

[15] `http://wiki.theory.org/BitTorrentSpecification`
[16] `http://www.bittorrent.com`

the seed and/or other peers that has the file, thus acting a a *leecher*. Peers that previously downloaded some of the pieces act as seeders too.

Due to the nature of this approach, the download of any file can be halted at any time and be resumed later, without the loss of previously downloaded information. This makes BitTorrent particularly useful for exchanging large files.

Active clients report their state to the tracker. When joining a torrent, a new client obtains from the tracker a list of IP addresses of active peers to connect to and cooperate with (typically 50 peers chosen at random in the list of active peers).

Interactions between clients are primarily guided by two principles. First, a peer preferentially sends data to peers that reciprocally sent data to him. This "tit-for-tat" strategy is used to encourage cooperation and ban "free-riding". Altruism is enforced during the download phase by the tit-for-tat policy, as a selfish client will be served with a very low priority.

Each time a client obtains a new piece, it informs all the peers it connects to. Interactions between clients are based on two principles [Legout et al., 2005]: first, the *choke algorithm* that encourages cooperation among peers and limits the number of peers a client is sending simultaneously data to; second, the *rarest first algorithm* that controls the pieces a client will actually request in the set of pieces currently available for download.

As more peers join the swarm, the complete successful and rate download by any particular node increases resulting in more efficient content delivery.

2.4.2 Main Components

To work properly the BitTorrent network depends on several components. It depends on a torrent file and a tracker and on the peers connectivity.

Every torrent file must contain two fields, or keys: the `info` (information) and `announce`. The `info` ensures that any modification of the piece can be reliably detected, and thus prevents both accidental and malicious modifications of any of the pieces. On the other side, the `announce` stores the location of the tracker. The others keys are optional (Table 2.1).

Keys	Description
info	A dictionary which describes the file(s) of the torrent including the number of pieces and the SHA-1 hash values of each piece
announce	The announce URL of the tracker
announce-list	List backup trackers
creation date	The creation time of the torrent
comment	Any comments by the author
created by	Author and/or Name and Version of the programme
encoding	String encoding format used to generate the pieces part

Table 2.1: Torrent metadata file structure.

Instead of transmitting the keys in plain text format, the keys contained in the torrent file are encoded before they are sent. Encoding is done using BitTorrent

specific method known as "Bencoding"[17].

A tracker is used to manage peers that participate in a torrent (Figure 2.4). It stores statistics about the torrent, but its main role is to allow peers to find each other and to start the communication.

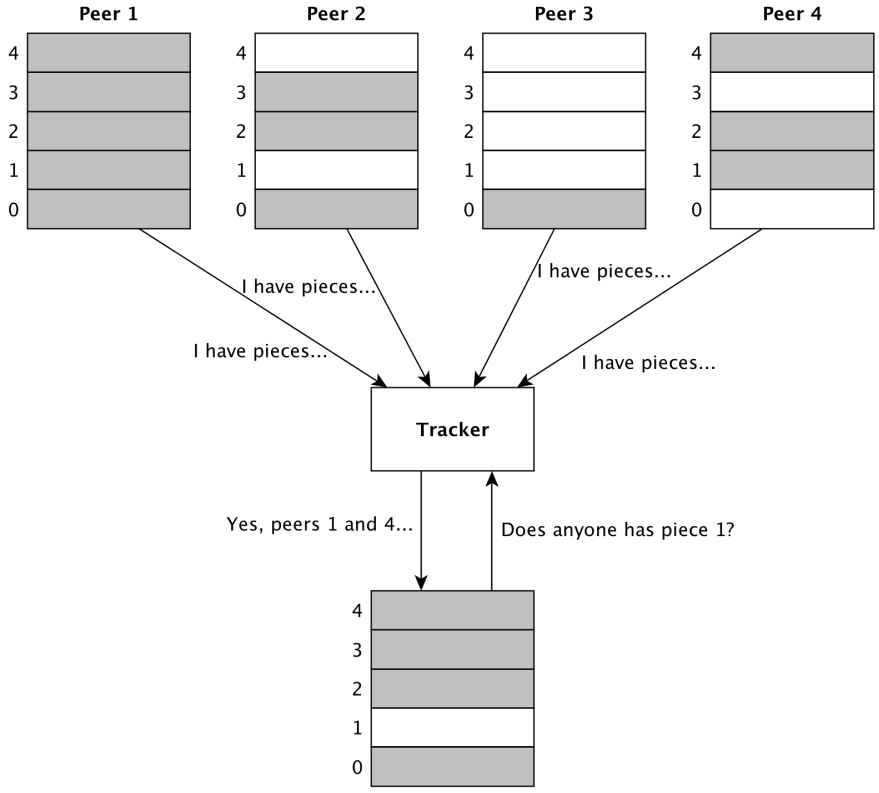

Figure 2.4: A tracker providing a list of peers with the required data.

The tracker is one HTTP/HTTPS service that typically works on port 6969 and one tracker can manage multiple torrents. BitTorrent clients communicate with the tracker using HTTP GET requests. This consists of appending a parameters to the URL. The parameters accepted are:

- **info_hash**: 20-byte SHA1 hash of the info key from the metainfo file.
- **peer_id**: 20-byte string used as a unique ID for the client.
- **port**: The port number the client is listed on.
- **uploaded**: The total amount uploaded since the client sent the 'started' event to the tracker in base ten ASCII.
- **downloaded**: The total amount downloaded since the client sent the 'started' event to the tracker in base ten ASCII.

[17] `http://wiki.theory.org/BitTorrentSpecification`

- **left**: The number of bytes the client till has to download, in base ten ASCII.
- **compact**: Indicates that the client accepts compacted responses. The peer list can then be replaced by a 6 bytes per peer. The first 4 bytes are the host, and the last 2 bytes are port.
- **event**: If specified, must be one of the following: started, stopped, completed.
- **ip**: (optional) The IP address of the client machine, in dotted format.
- **numwant**: (optional) The number of peers the client wishes to receive from the tracker.
- **key**: (optional) Allows a client to identify itself if their IP address changes.
- **trackerid**: (optional) If previous announce contained a tracker id, it should be set here.

The tracker then responds with a "text/plain" document with the following keys:

- **failure message**: If present, then no other keys are included. The value is a human readable error message as to why the request failed.
- **warning message**: Similar to failure message, but response still gets processed.
- **interval**: The number of seconds a client should wait between sending regular requests to the tracker.
- **min interval**: Minimum announce interval.
- **tracker id**: A string that the client should send back with its next announce.
- **complete**: Number of peers with the complete file.
- **incomplete**: number of non-seeding peers (leechers)
- **peers**: A list of dictionaries including: peer id, IP and ports of all the peers.

Peers

Peers communicate using TCP (Transport Control Protocol) that guarantees reliable and in-order delivery of data from sender to receiver. It uses ports 6881-6889 to send messages (that are made up of a handshake) and data between peers.

When a peer receives a request for a piece from another peer, it can opt to refuse to transmit that piece. If this happens, mostly because it is busy, the peer is said to be choked.

The tracker allows peers to query which peers have what data, and allows them to begin communication. Peers communicate with the tracker via the plain text via HTTP (Hypertext Transfer Protocol).

BitTorrent does not have restrictions on file size or type. Data is only split into smaller pieces to be sent between peers using the BitTorrent protocol. These pieces

have a fixed size (the most common piece sizes are 256kb, 512kb and 1mb). Pieces with large size can cause download inefficiency. Every piece is of equal length except for the final piece, which is irregular. The number of pieces is thus $ceil(\frac{length}{piecesize})$.

2.5 Summary

Nowadays, sharing resources on the Internet rely, essentially, on a couple of approaches. Cloud computing has become a commodity for service provisioning, such as processing power, virtualized hardware, storage space and others. Organizations take advantage of their hardware and knowledge resources to make them available, through a business model, to customers.

On the other hand, peer-to-peer has been evolving independently from enterprises, providing a familiar and reliable infrastructure for resource sharing. Despite several copyright issues, it managed to evolve into a robust, resilient, service.

Chapter 3

Distributed Storage Service

The low-level usage pattern of Amazon S3 is, essentially, storing, updating and retrieving sequence of bytes through SOAP, REST WS or BitTorrent, having in mind the durability and availability defined in the SLA. Off course, the SLA parameters reflects on the client budget. In this pattern, the access speed depends on the end-to-end throughput to where the buckets are stored (in Amazon's data centers). As stated above, S3 has also support the BitTorrent protocol. This allows users to further save on bandwidth costs for popular pieces of data by letting users download from Amazon and other users simultaneously, in addition to the default client/server delivery mechanism. With this system Amazon thrive to reduce costs.

Peer-to-peer networks do not depend on a specific companies to provide a service. Their distributed nature makes them democratic, in the sense that the clients that require a specific service also have to give back to the network. The importance of P2P is evident on the traffic it generates. In other words, clients can use and access resources, e.g. storage space, if they are also available to share some resources to the P2P community. Privacy and security is also an issue although possible to overcome [Cong Wang et al., 2010,Rewatkar and Lanjewar, 2010]. Due to its nature, P2P networks do not provide SLAs to ensure access speed, redundancy, scalability, capacity, and others. Many of these characteristics, if not all, are dependent on the number of peers the network have and on the probability of peer availability.

This section describes the overall architecture, going through three scenarios. We started to develop a REST SDK to support connection to the Amazon S3 server. This SDK was specifically build in order to assess compatibility with our own implementations. The SDK was further used to develop two client applications: a java desktop application, with visual controls, and a FUSE[1] (File System in User-Space) driver. This app allows the user to upload, list, delete and retrieve regular files through Amazon's REST service. With the clients in place, we proceeded to build a local, centralized server.

This was used to ensure a compatible server-side REST service as well as to compare access speed between a networked and local servers (see chapter 4). By changing the server address and the user credentials, it is possible to configure the above mentioned clients to choose to connect to Amazon's S3 or to the local

[1] `http://fuse.sourceforge.net`

server in order to retrieve, delete or make others operations on the service. Finally, a BitTorrent based, distributed "server" was developed. In this implementation, buckets and objects are stored in BT nodes.

The overall architecture is depicted in Figure 3.1 where it is possible to verify that the clients connects to different service implementations by the SDK.

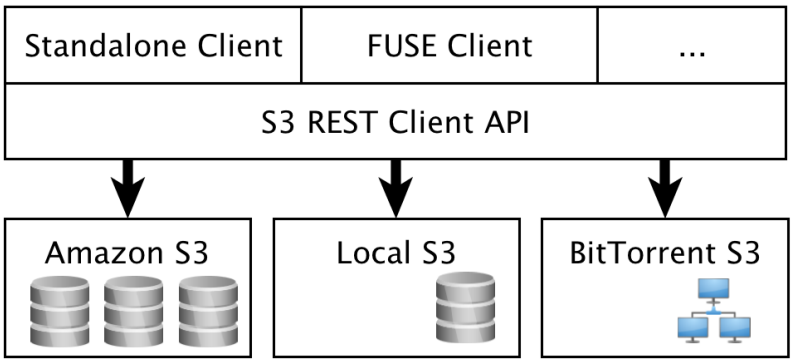

Figure 3.1: Overall architecture implemented.

3.1 Amazon Simple Storage Service

The Amazon S3 is a web service that enables storing data in the cloud. It gives access to the same data storage infrastructure that Amazon uses to run its own global network of web sites.

At the end of the first quarter of this year, there were 905 billion objects in Amazon S3 and routinely handle 650,000 requests per second[2] for those objects with occasional peaks substantially above that number. That numbers showed that S3 has been constantly growing along the users satisfaction. This service is designed to provide 99.999999999% durability and 99.99% availability of objects over a given year[3]. In sum, the S3 design aims to provide scalability, high availability, and low latency at commodity costs.

3.1.1 Concepts and Architecture

The data stored in Amazon S3 is organized in two levels. At the top level resides the **bucket**, similar to folders which have a unique global name. Buckets are needed to organize the Amazon S3 namespace at the highest level and to identify the account responsible for storage and data transfer charges. They can be created in a specific *region* with a specific role in access control. They can serve as the unit of aggregation for usage reporting too.

Each bucket can store an unlimited number of data **objects**. Objects are the fundamental entities stored in Amazon S3. Objects consist of data (blob) and metadata.

[2] http://aws.typepad.com/aws/2012/04/amazon-s3-905-billion-objects-and-650000-requestssecond.html
[3] http://aws.amazon.com/s3/

The data portion is opaque to Amazon S3. The metadata is a set of name-value pairs that describe the object (Content-Type, Date,..). An object is uniquely identified within a bucket by a **key** (name) and a version ID. A key is the unique identifier for an object within a bucket. Every object in a bucket has exactly one key. Users can create, modify and read objects, subject to access control restrictions.

In the creation of a bucket the choice of the geographical region can be important, in order to optimize latency, minimize costs, or address regulatory requirements. By default is used the US-Standard region.

Other functionality is that every object contained in a bucket is accessible by URL. For example, if the object named "2006-03-01/AmazonS3.wsdl" is stored in the "doc" bucket, then it is addressable using the URL "`http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl`".

S3 stores arbitrary objects up to 5 terabytes in size, each accompanied by up to 2 kilobytes of metadata. Thus, objects are organized into buckets (each owned by an Amazon Web Services or AWS account), and identified within each bucket by a unique, user-assigned key.

This service is well characterized by the flexibility it presents. This cloud storage system allows incrementing or decrementing available space according to the business needs, even for short periods of time.

S3 also has a more affordable service, with lower levels of redundancy. The Reduced Redundancy Storage (RRS) is geared towards storing non-critical information, for example organizations' centrally managed data backups.

Amazon S3 adopts the pay-per-use model[4]: the prices are based on the data location (region) and the amount of requests (GET, PUT, COPY, POST, and LIST), as well as the data transfers into and out of an Amazon S3 region. The amount of storage is accounted as well and relies on the number and the size of objects stored in buckets.

For example, 1,000 PUT, COPY, POST or LIST requests cost $0.01; 10,000 GET requests cost $0.01. Fairly, data transfer prices can vary from $0.120 per GB (up to 10 TB/month) at $0.050 per GB (next 350 TB / month). The first 1 GB/month is not charged. The storage price lies between $0.125 ($0.093 for RSS) per GB for the first 1 TB/month up to $0.055 ($0.037 for RSS) per GB for over 5000 TB/month. This values match with the US-Standard region.

3.1.2 Data Access Protocols

Many cloud providers offer a wide variety of flexible data storage services and some of them offer their services through WebServices. For example, DropBox allows users to synchronize data between computers and all data is persisted in Amazon S3 storage service[5]. The S3 architecture is designed to be programming language-neutral providing REST and a SOAP interfaces to store and retrieve objects (Figure 3.2).

REST web services were developed largely as an alternative to some of the perceived drawbacks of SOAP-based web services [Hamad, 2010]. With REST, standard

[4] `http://aws.amazon.com/s3/pricing/`
[5] `https://www.dropbox.com/help/7/en`

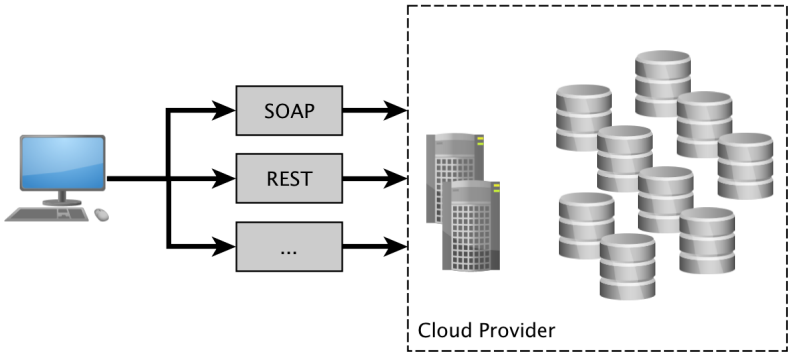

Figure 3.2: S3 architectural model.

HTTP requests are used to create, fetch, and delete buckets and objects (Table 3.1). So, this interface work with standard HTTP headers and status codes to allow the communication in S3[6].

Operations	Description
GET /	get a list of all buckets
PUT /{bucket}	create a new bucket
GET /{bucket}	list the contents of the bucket
DELETE /{bucket}	delete the bucket
PUT /{bucket}/{object}	create/update an object
GET /{bucket}/{object}	get the contents of the object
DELETE /{bucket}/{object}	delete the object

Table 3.1: HTTP operations with URI pattern that can be performed in S3.

Thus, the REST architecture is fundamentally client-server architecture and aims to be simple by limiting the types of operations that can perform on a resource.

Currently, S3 also supports the BitTorrent data access protocol. This popular file-sharing protocol enables efficient cooperative data distribution: data is initially distributed at one or more seed sites that are pointed to by a tracker. As clients begin to download a BitTorrent file, those clients register themselves with the tracker and make portions that they have downloaded available to other clients. S3 can provide both tracker and seed functionality, allowing for substantial bandwidth savings if multiple concurrent clients demand the same set of object.

3.1.3 Authentication Model

When users register an Amazon Web Services (AWS) account, AWS assigns one Access Key ID (a 20-character, alphanumeric string) and one Secret Access Key (a 40-character string). The Access Key ID uniquely identifies an AWS Account. AWS uses a typical implementation that provides both confidentiality and integrity

[6] `http://www.oreillynet.com/pub/wlg/3005`

(through server authentication and encryption).

The S3 REST API[7] uses a custom HTTP scheme based on a keyed-HMAC (Hash Message Authentication Code) for authentication. The standard HTTPAuthorization header is used to pass authentication information.

To authenticate a request, selected elements of the request are concatenated to form a string with the following form: "AWS AWSAccessKeyId:Signature". In the request authentication, the first "AWSAccessKeyId" element identifies the user that originated the request and the secret key that was used to compute the "Signature". The "Signature" element is the SHA1 hash of the request selected elements [Krawczyk et al., 1997]. In the Amazon S3 Request authentication this algorithm takes as input two byte-strings: a key and a message. The key correspond to the AWS Secret Access Key and the message is the UTF-8 encoding of one string that represents the request. This string includes parameters like the HTTP verb, content MD5, content type, date that will vary from request to request. The output of HMAC-SHA1 is also a byte string, called the digest. The final "Signature" (Figure 3.3) request parameter is constructed by Base64 encoding this digest.

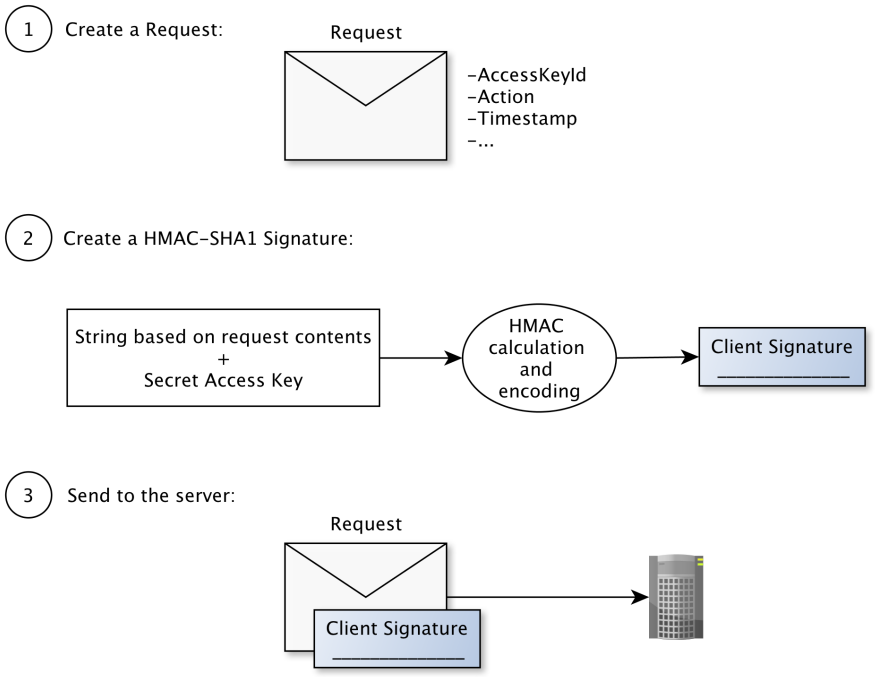

Figure 3.3: Client authentication.

When the system receives an authenticated request, it fetches the AWS Secret Access Key and uses it in the same way to compute a "Signature" for the message it received. It then matches signatures to authenticate the message (Figure 3.4). If they are equal the user is authenticated by the system.

To summarize, this REST architecture implemented by the AWS offers to the

[7] http://docs.amazonwebservices.com/AmazonS3/latest/dev/RESTAuthentication.html

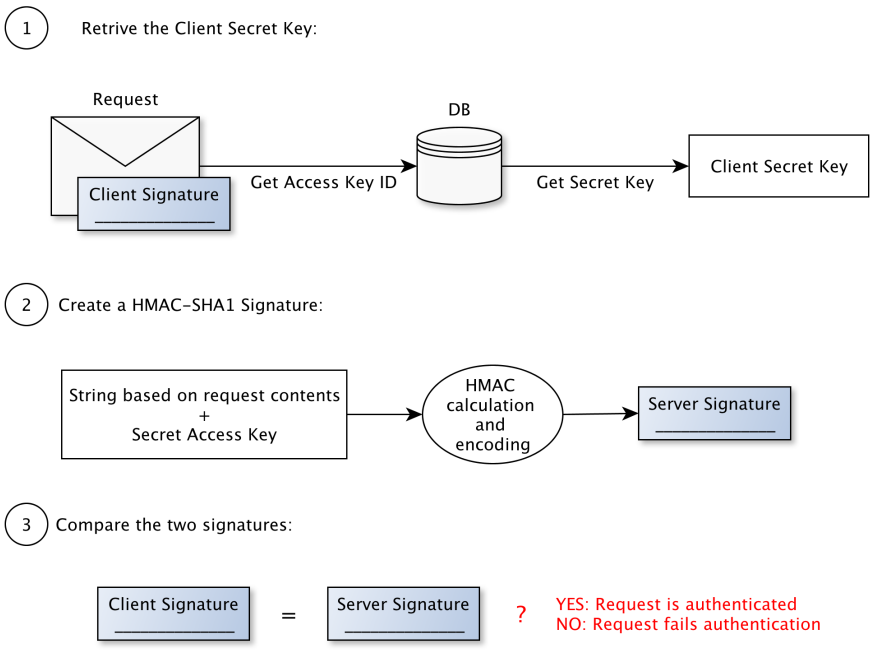

Figure 3.4: Server authentication.

register users access to theirs files. Essentially it also allows developers to create new applications to interact with.

This S3 survey provided an overview of the system functionality to better understand the sections below.

3.2 Amazon S3 Client

Amazon made available a Java API that can be used by client applications to connect to its REST service. However, we decided to develop our own REST client to better understand the server interfaces and to be able to use them in the role of the client as well as in the role of a server. It was validated through a client applications and, after that, used to build an S3 compatible server.

Our implementation is based on the `IStorage` interface, which encapsulates the methods that can be called in order to interact with the service (Figure 3.5).

The `StorageRestAccess` class implements the interface and is responsible to make the requests and consequent connection to the service. To make the connection a host and a port is needed, in the case of the S3 service is located on *s3.amazonaws.com* at port 80 (or at port 443 with SSL support).

For the authentication process an Access Key Id and a Secret Key must be provided as explained on section 3.1.3. With all of this parameters the class is able to process the requests.

The next code represents an example of a GET request sent to the S3 REST service:

Figure 3.5: Storage REST service class diagram.

```
GET

Sun, 14 Oct 2012 13:27:21 GMT
/BUCKET_AKIAIRJLQ2KQRXLQM3OQ/16MB.zip
Authorization: AWS AKIAIRJLQ2KQRXLQM3OQ:2w15rVHKA6WCFUZ4DcRPkQzy6rU=
http://s3.amazonaws.com:80/BUCKET_AKIAIRJLQ2KQRXLQM3OQ/16MB.zip
```

The first line represents the action operation. The next represents the timestamp of the request. Following is represented the object path (i.e., the bucket location plus the object key name) and the string included in the authorization header. Finally, is showed the accessible url, composed with the host and port and object path, to connect with.

Based in this class, a desktop client application was used to invoke the operations developed (Figure 3.6). This generic client receives the file through the user action (drag-and-drop). It then reads the file contents and uploads the file content to the bucket. The bucket is created by the application if not existing. The desktop application provides a GUI to the user, which allows listing, retrieving, removing or uploading files. This application is configured, in a preferences panel, with the user credentials and connection parameters, for a correctly authentication and connection to the S3 server.

Another application, a FUSE[8] driver, was also developed to better understand the influence of different connection scenarios. This driver mounts an S3 bucket in

[8] `http://fuse.sourceforge.net`

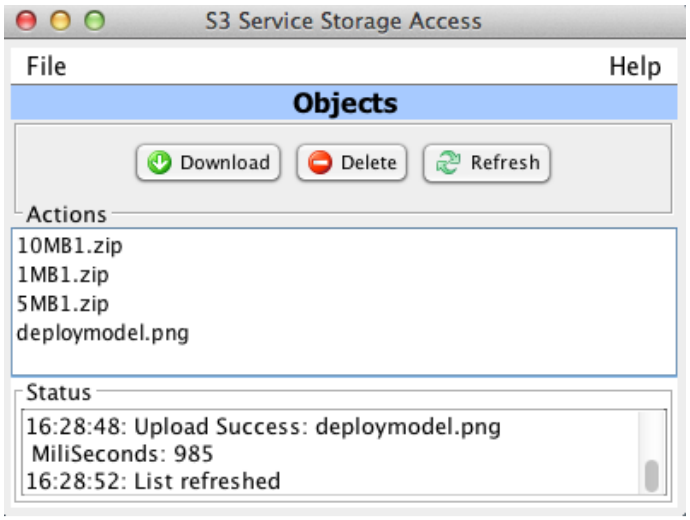

Figure 3.6: Desktop S3 client.

the local file system, along regular files, thus providing an unified view to the local and remote files (Figure 3.7).

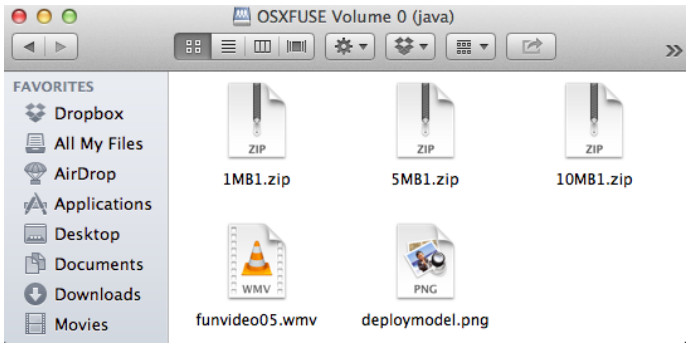

Figure 3.7: FUSE Amazon S3 client.

Amazon built the S3 service without much care about the GUI applications provision that help users to organize S3 files more easily. It particular worries with the storage service provision leaving the creation GUI task to the developers. Indeed the only way to access S3 resources official is by the AWS Management Console[9], where all of Amazon Web Services (EC2, EBS, S3, ..) are administered. Also caring about the service provision the next implementation was the replication of the S3 REST server locally for also working with the client application. This implementation are discussed in the section below.

[9] `http://aws.amazon.com/console/`

3.3 Centralized S3 server

After having the client application working with Amazon S3, a local, S3 compatible, server was built. This server will be used as a reference for an access speed test. It is network accessible, based on Jetty[10], and implements the operations through REST WS. Naturally, due to the centralized nature, it does not cope with the durability and availability users expect for the Amazon S3 service. However, access speed is limited only by the local host processing speed and operating system, thus having the best throughput possible.

The developed WS implement basic service operations:

```
public interface IService {

@GET
@Path("/")
@Produces("application/xml")
public ListAllMyBucketsResult getAllMyBuckets();

@GET
@Path("/{bucket}/")
@Produces("application/xml")
public ListBucketResult getBucketObjects(@PathParam("bucket")
                String bucket);

@GET
@Path("/{bucket}/{object}")
public Response getObject(@PathParam("bucket") String bucket,
                @PathParam("object") String object);

@DELETE
@Path("/{bucket}/{object}")
public Response deleteObject(@PathParam("bucket") String bucket,
                @PathParam("object") String object);

@PUT
@Path("/{bucket}/{object}")
public Response putObject(@PathParam("bucket") String bucket,
                @PathParam("object") String object,InputStream inputStream);

@HEAD
@Path("/{bucket}/")
public Response checkBucketExists(@PathParam("bucket") String bucket);

@PUT
@Path("/{bucket}/")
public Response createBucket(@PathParam("bucket") String bucket);
}
```

These operations also provide a good starting point for a simple storage service implementation. The first operation `getAllMyBuckets` retrieve all the buckets created by the user and in this implementation the buckets are in simple terms local user folders. When the operation `getBucketObjects` is called it returns the object

[10] `http://www.eclipse.org/jetty/`

list in that bucket and with the `getObject` call the object is returned. As expected, `deleteObject` removes the object. When is necessary to upload some object, the `putObject` operation is called and if the object already exists, it is replaced by the new one. The operations on buckets `createBucket` and `checkIfBucketExists` are only for the bucket creation but if already exist the operation fails for not running the risk of any information loss in the that bucket.

In order to know what buckets belong to a certain user, it is recorded in the server's database a bucket-user registry. It is also recorded the creation date of the bucket for S3 compatibility purpose. All about the repository file and their DB transaction is managed in the code by the `RepositoryManager` class.

The main challenge at this point was to develop a secure RESTful web API compatible with the S3 REST authentication model:

1. The server and the client know a public and private key but only them know the private key. The public key identifies the user information (User ID) and anyone can know about the public key but never the private key.

2. Before making the REST API call, the client creates a unique hash (HMAC-SHA1), hashing request data (all request parameters and values) with the private key assigned by the system, representing the request to the server (as shown in Figure 3.3).

3. The server receives the request and using the user-identifying data sent along with the request (User ID) look the user up in the DB and load their private key. After, it generates the unique hash based on the submitted values using the same methods that the client used (as shown in Figure 3.4). To hinder replay attacks the current server timestamp is compared to the timestamp of the client making sure that the difference between them matches an acceptable time range (e.g. 15min).

4. It then compares the two hash and if they are equal the server trusts the client and runs the request. If not it drops the request and submit a forbidden response to the client.

The authentication call on local Jetty server has been implemented by a security request handler. This security handler is always called on every request sent to the server. So, before the request processing is always attended first the security handler to verify if the user has access to the service, i.e. if he is a registered user.

In this implementation, buckets correspond to directories in the local server filesystem. Objects, on the other hand, are stored as files, with the metadata being stored in a local database.

We then proceeded to building an S3 compatible server backed by a BitTorrent network.

3.4 D3S – The BitTorrent S3 server

Peer-to-peer protocols, such BitTorrent, provide a storage service by sharing and replicating data over several peers. Storage systems depend on redundancy to achieve availability and resilience, natural in P2P networks [Pamies-Juarez, 2011]. On the other hand, storage space is proportional to the number of peers.

Thus, the REST S3 server interface is local although files are stored in a BT network where each bucket objects are segmented, replicated and scattered by several nodes. In this proof of concept, files are replicated only to the peers that belong to a given user, thus requiring that at least one owned peer has to be online for the file to be accessible in other peers. This drawback is easily solved by scattering buckets objects through several "random" peers.

There are several challenges to this implementation:

- How to authenticate users;
- How to provide a .torrent file to relevant peers;
- How to remove the file from the replicas.

The sections below aims to clarify the D3S functionality model and the challenges of this implementation.

3.4.1 The Functionality

Taking advantage of the S3 functionality, i.e., the interaction model with the objects, it was used the local REST server referred in section 3.3. With this implementation is achieved one structure capable to support the interaction model similar to the S3 storage system. Instead of storing objects in a local server only, they are sent to a BT network, taking advantage of replication and different throughput patterns.

After the user complete the preferences windows in the client app with the correct credentials he is able to make the connection with the system. This connection is only for testing the user credentials because in every request this authentication is repeated. So when an user uploads a file to the D3S service he invokes a PUT object operation. If he is authenticated with the system he continues with the process of upload. This process includes the insertion of a bucket if not existing in order to save the file.

For every object, a .torrent is generated. During this .torrent file creation the system also sends this metafile to the tracker and finishes the operation launching a peer for seeding the file (Figure 3.8). On the right side of the figure, the GET operation is represented. In this operation the system must firstly check if the file is locally available. If not, the system retrieves the .torrent file from the tracker and starts downloading it until completed.

The last user scenario is when the user wants to delete a file. At a DELETE operation call the system remove the files (i.e., the .torrent and real file) from the local repository. During this process the .torrent file is also removed from the tracker repository.

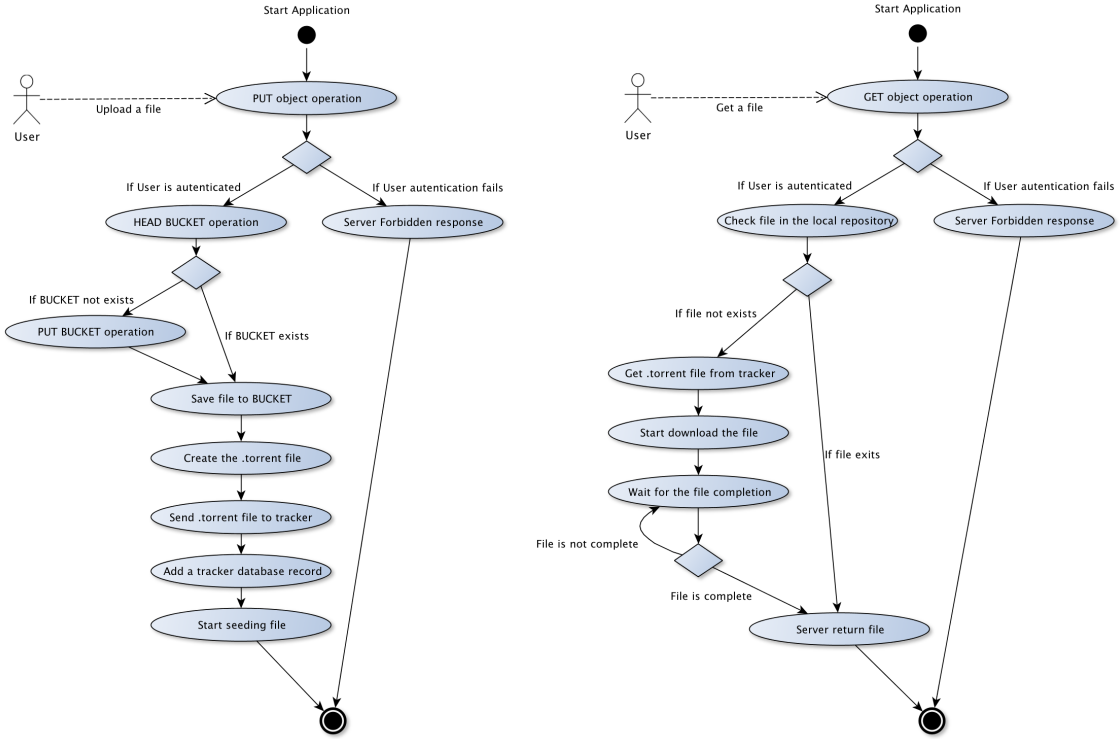

Figure 3.8: Upload (left) and download (right) activity diagram of the D3S.

To allow all of this specific transactions it was necessary to implement additional features that are discussed in the section below. So the next section provide a more information of the implemented architecture represented in the Figure 3.9. As shown, the tracker is the only centralized entity with the responsibility to coordinate the system operation. The other modules are located on the user side.

3.4.2 Implementation

Instead of extending the BitTorrent protocol we chose to provide a specific service to authenticate users and another to install the .torrent file in the tracker (Figure 3.10). We opt for using RMI (Remote Method Invocation) to implement these mechanisms.

RMI is Java mechanism for supporting distributed object based computing [Maassen et al., 1999]. It allows client/server based distributed applications to communicate by invoking remote objects the same way as local objects. So the use of RMI provided a mechanism by which the tracker and the client communicate and pass information.

In particular, the client running at a node can access a remote service in the tracker by invoking a method of the object that implements the service. Thus, the RMI framework enables applications to exploit distributed object technology rather than low level message passing (e.g., sockets) to meet their communication needs.

The RMI distributed application uses the RMI registry to obtain a reference to

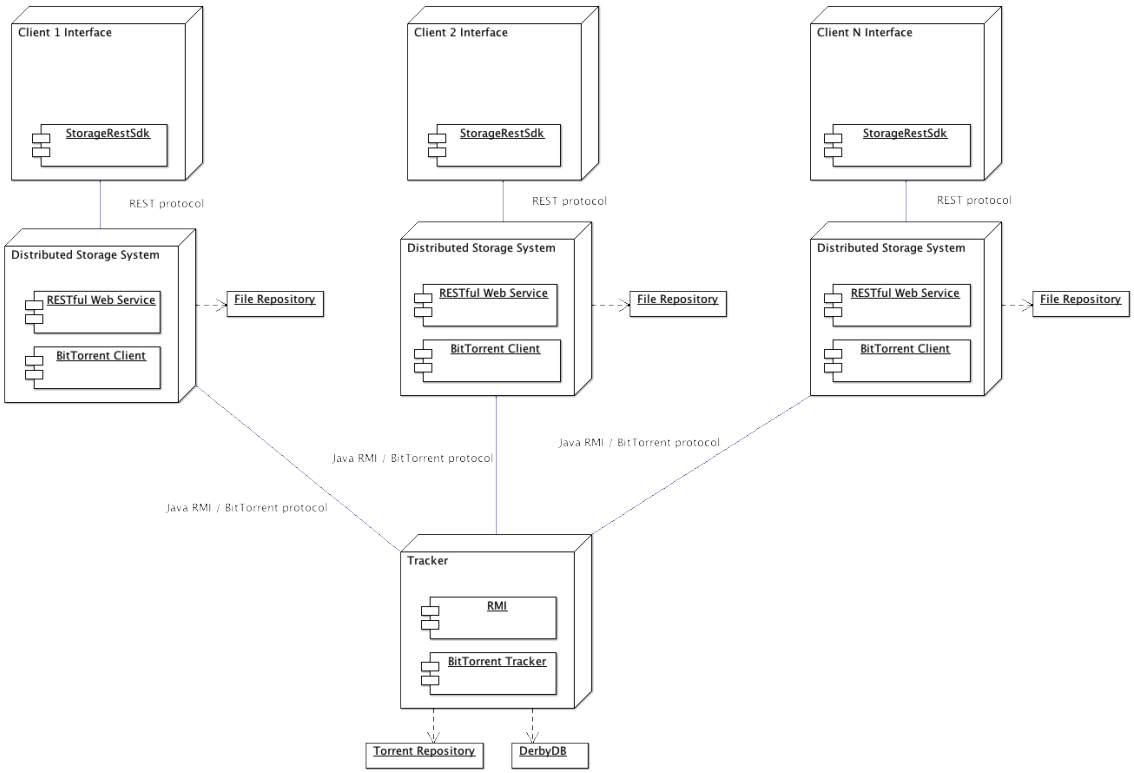

Figure 3.9: D3S deploy model diagram.

a remote object. The server in the tracker calls the registry to associate (or bind) a name with a remote object. The client looks up the remote object by its name in the server's registry and then invokes a method on it.

The following code shows the remote interface methods that can be invoked by the User Manager and the Torrent Manager to interact with the tracker:

```
public interface Compute extends Remote {

        // UserManager RMI invocations:
        public boolean addUser(UserService n);
        public boolean updateUser(String userId, UserService n);
        public void removeUser(UserService u);
        public List<UserService> getUsers();
        public UserService getUser(String id);
        public List<ListEntry> getUserObjects(String userId);

        // TorrentManager RMI invocations:
        public boolean putTorrent(String name,byte[] data,String userId);
        public byte[] getTorrent(String name);
        public List<String> getTorrents();
        public boolean isTorrentUser(String userId,String torrent);
        public boolean removeTorrent(String torrent);

}
```

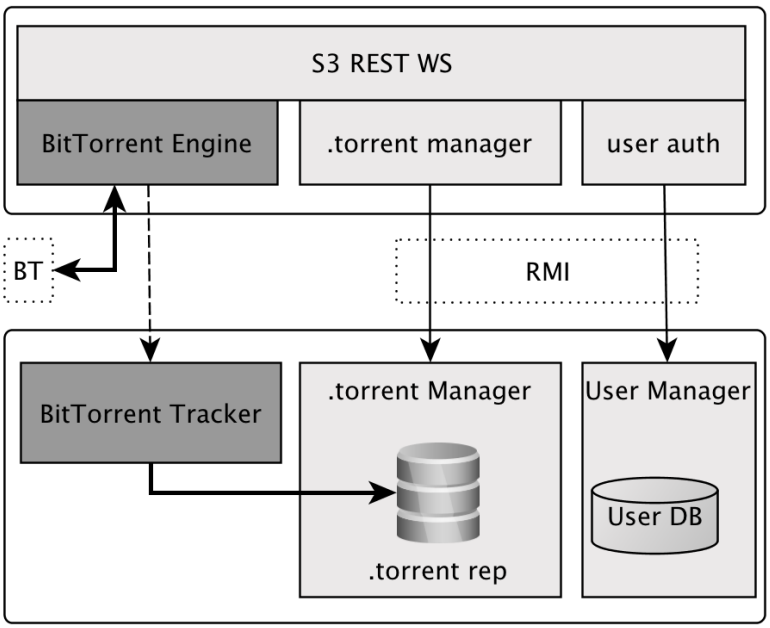

Figure 3.10: BitTorrent S3 implementation.

About the overall system functionality, BT peers contact the tracker when needed, to manage the overlay network. In this implementation, each peer will also contact the User Manager, which maintains a database of user credentials, to authenticate each user's peers. The User Manager also maintain a relation of each user's .torrent files, residing in the .torrent repository.

The .torrent Manager receives the .torrent files from the peers and associates them with the user that originated the request. The .torrent files are generated when the user uploads a file to the service through the client application. So when a user uploads a file to the service the `putObject` operation is called and the object file is locally saved by the Repository Manager (Figure 3.11). The BT Manager deals with the creation of the .torrent file through the Torrent Manager. This last is responsible to send the file to the tracker repository. Finally, the BT Manager only create a peer client (BTClient) that share the object file to others peers.

In the GET operation case can be found 2 different scenarios. The first is when the file is locally available (Figure 3.12). In this case the Repository Manager just returns the file available on the File Repository. The other case is when the system needs to contact the BT Manager to download the file through the BitTorrent network (Figure 3.13). The BT Manager is responsible for the download of the *.torrent* file through the Torrent Manager in order to download the file (BTClient). When the file is complete its returned to the Service but the BT Client stills sharing the file with the BT network.

Peers will periodically pool the .torrent Manager to check for new additions, so that each one of them can replicate the file, thus making it available to the other peers as well as locally. Anytime that the user requests a file, it is already available

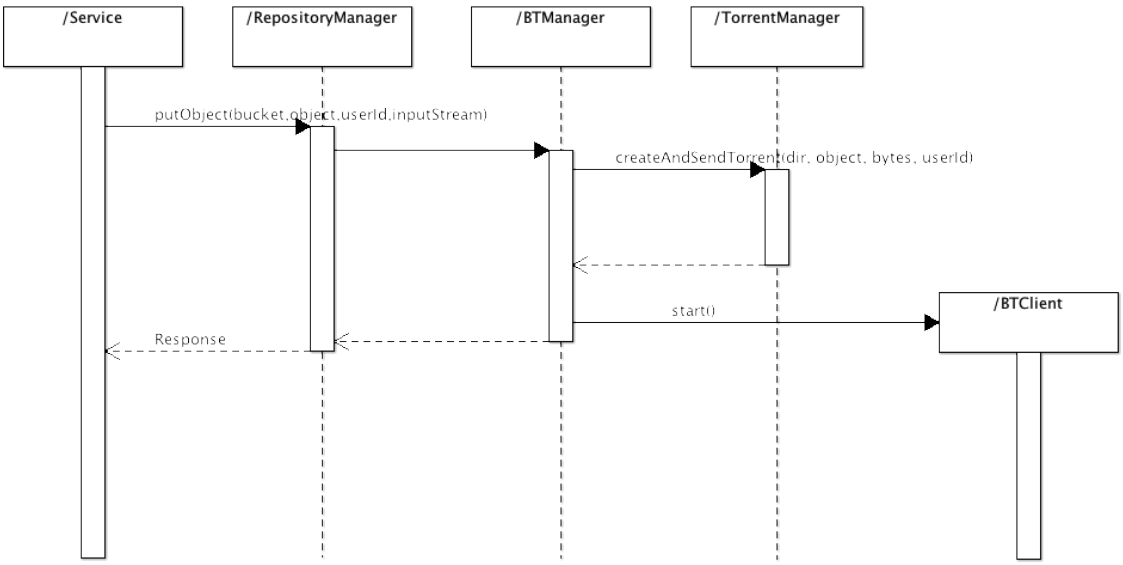

Figure 3.11: Sequence diagram of the PUT object operation.

locally. If the file is not available locally, it is downloaded from several replicas simultaneously, increasing the access speed and only becoming available to the user when the download has finished.

In sum, BT networks natively supports resource sharing, which requires self organization, load balancing, redundant storage, efficient search of data items, data guarantees, trust and authentication, massive scalability properties and fault-tolerance (i.e., if one peer on the network fails the whole network is not compromised) [Eng Keong Lua et al., 2005]. These characteristics are thus imported to this implementation, meeting many of the requirements of cloud storage services. Moreover, it intrinsically copes with scalability, redundancy and availability: the more replicas there are, the higher the redundancy, access speed and availability.

3.4.3 The NAT Traversal problem

Network Address Translation (NAT) [Egevang and Francis, 1994] allows a single device, such as a router, to act as an agent between the Internet (public network) and a local (private) network. This means that only a single, unique IP address is required to represent an entire group of computers what is very useful to build a small private network. However, NAT creates a private IP address realm behind NAT translators and according to common firewall and NAT rules, hosts in private address realm cannot be reached directly from public Internet. This built-in privacy and security benefits of NAT represent a trouble because it is hard to locate and communicate with the private hosts behind a NAT gateway.

Thus, NAT causes well-known difficulties for peer-to-peer communication, since the peers involved may not be reachable at any globally valid IP address. Several NAT traversal techniques are known [Hu, 2005], but are not standardized and their

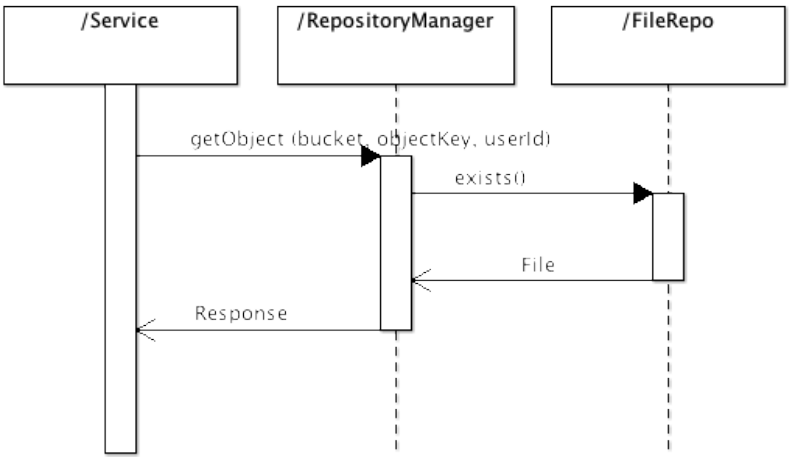

Figure 3.12: Sequence diagram of the GET object operation when file is locally available.

robustness or relative merits are slim. So, in peer-to-peer networks, hosts behind NAT gateway have to be reached directly by some way in order to be reachable by other peers.

UPnP is an open standard for flexible connectivity of intelligent both wired and wireless devices. Currently, almost all Internet gateways are UPnP NAT devices and the automatic service discovery, addressing, without configuration are suitable for P2P applications. UPnP specification is based on TCP/IP and when a new host needs a connection, the UPnP device can automatically configure network address translating, announce its presence on a local network, and permit the exchange of device and service descriptions.

We decided to add UPnP functionality to the D3S peer, to foster wider adoption and to facilitate installing peers in domestic networks. Each peer detects if it is is behind an UPnP-enabled NAT device and then learn the shared, globally IP address, and configure UDP and TCP port mappings to forward packets from the external ports of the NAT to the internal ports of the client application.

This allowed a flexible connectivity where the application traverse a NAT gateway by dynamically opening and closings ports for communication with other peers.

3.5 Summary

This section described the implementation of the infrastructure required to assess and compare throughput in several scenarios. Starting with the Amazon S3 cloud storage service, two different clients where developed. An additional server, based on Jetty, was developed, allowing local and LAN access. Moreover, a BT based implementation of the server interface was also implemented, allowing taking advantage of peer-to-peer networks characteristics, namely, replication, resilience and availability.

Next chapter describe several experiments that where performed, to compare access speed and latency of several scenarios.

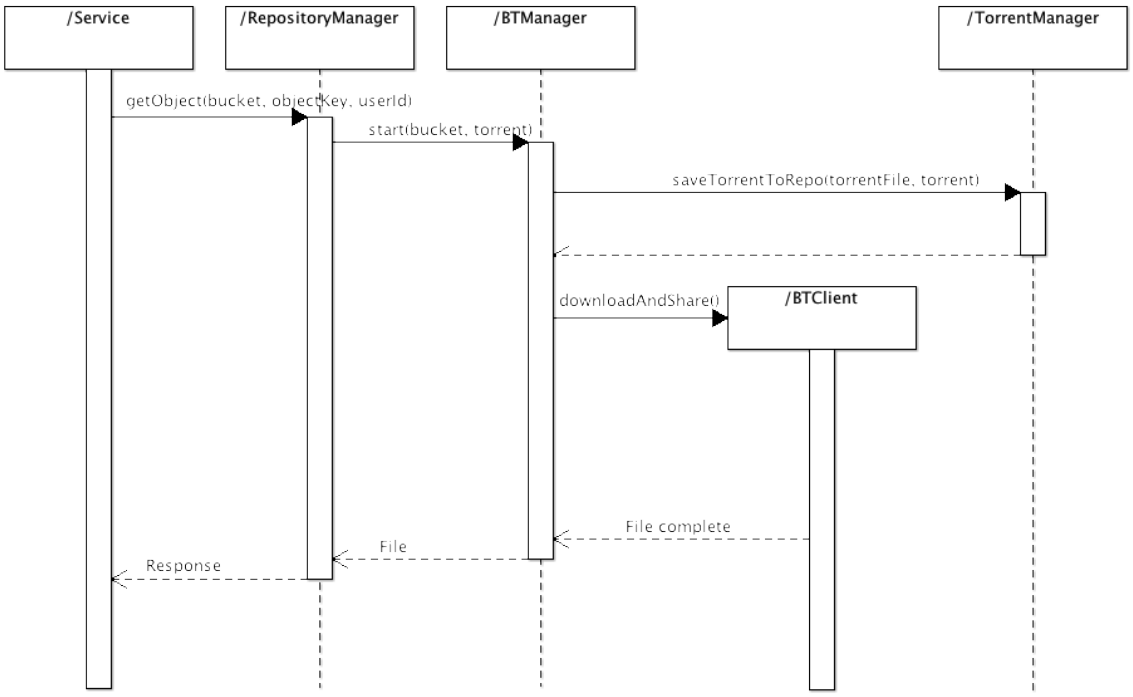

Figure 3.13: Sequence diagram of the GET object operation by the BT network.

Chapter 4

Results and Discussion

BitTorrent has become a standard for scalable content distribution over the Internet with the ability to efficiently achieve high scalability during peak demands. At the other side of the spectrum, cloud storage services like Amazon Simple Storage Service (S3) has been growing providing "low-cost", highly available, storage services. Similarly, at opposite sides of the spectrum, the former depends on the openness of the protocol and the users' volunteerism in sharing, and the latter depends on commercial rules. In the end, both provide a means to store information and make it available anytime, anywhere.

In terms of availability, Amazon's S3 is bound to an SLA, typically in the order of the 99.99%. Availability in a BitTorrent network will depend on the number of seeds. On the other hand, transfer speed depends on the local connection speed, remote connection speed and overall network traffic flows. BitTorrent, besides local connection speed, depends heavily on the number of seeds.

This chapter describe some experiments to try to assess the differences between both scenarios as well as to infer the relation between the number of seeds and the throughput.

4.1 Experimental Setup

As stated above, we expect to measure the throughput in four different scenarios: Amazon S3, local S3, local network S3 and D3S. Access speed depends on the data size as well as network conditions. We defined 5 objects with different sizes that will be used in all experiments: 1 Byte, 1 KByte, 1 MByte, 16 MBytes and 100 MBytes. The 1 Byte object is intended to measure the service latency, or the minimum time the server takes to respond. In contrast, the 100 MByte object represents the maximum data throughput.

To automate the experiments, we developed a script which measures the elapsed time to upload these different sized objects sequentially to the server in each scenario using the `currentTimeMillis` java system call. In other words, the program uploads the 1B, 1K, 1M, 16M and 100M in sequence to the server. Next, the program downloads the previously uploaded objects, in the same sequence, again measuring

the elapsed time. After finishing this interaction, the program waits for 15 minutes and repeats the experiment. We expect that this approach will clear cache influence (Figure 4.1). It should be highlighted that the script requires that each transference be successfully resolved before the next was initiated.

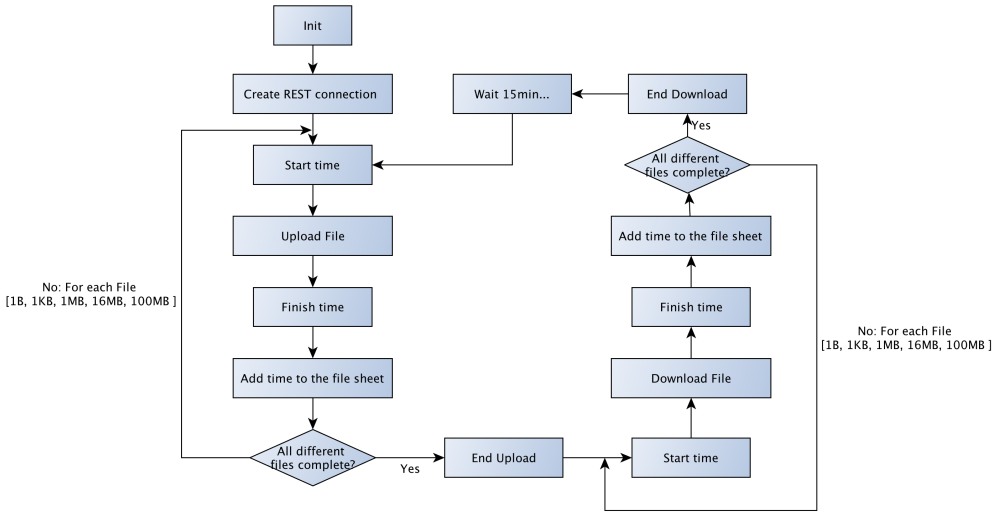

Figure 4.1: Experiment script flowchart.

All the presented results were made with a wired connection to the Foundation for National Scientific Computing[1] (FCCN).

We started with the Amazon S3 REST Web Service experiment on the 21 of September and finished on the 26, in a total of 342 samples (for each file).

The S3 local Web Service experiment was performed on 24 of September during 3h30, in a total of 1532 samples (for each file). We also tested the same server but through a local network connection. It ran for 24 hours and gathered a total of 708 samples (for each file). With the WS running locally and with the same program test it was tested the WS performance. In this scenario, was measured the iterations with no delay between them. Without delay allows a performance "stress test" to this implementation. The next experiment was to test the REST WS implementation in a network system, i.e., the service run on a machine that is network accessible. With this experiment is mainly intended to show the variations of the service on a network system.

To assess the relative influence, the D3S service requires some variability in the number of seeds, so we conducted 4 experiments for D3S alone with 1 (58 samples), 2 (65 samples), 4 (62 samples) and 16 (48 samples) peers. These tests were performed from 14 to 21 of October.

There are several approaches to measure BitTorrent networks' performance. Some of them use log traces from trackers, while others rely on crawling techniques to retrieve the information from the system [Kaune et al., 2009]. We intend to make

[1] `www.fccn.pt`

D3S transparent to the user, acting as Amazon S3. We then decided to take the measures on the user side. In this way, the measurement software consists is the same program that was used above. The script essentially recorded the times of user experience depending on the number of seeds and the file size. This user experience consisted in aggregating the time necessary to:

- download the .torrent file by RMI invocation;
- download all the pieces from the BT network;
- "transcoding" pieces into the real file;
- and finally the download time of the GET operation on the local server.

Another point to consider is that, because of polling that the D3S performs, some times the file will be locally available when the user requests it. Other times it does not exist locally and has to be downloaded. In the latter, the access speed will be considerably lower.

The list of samples that were obtained in each experiment where statistically processed. First, the outliers where identify and removed. Outliers can be a major source of skewness in any data set so its important to exclude them. The method used is commonly referred as the Quartile[2] method. Essentially, boundaries are identified for each of the quartiles in the data set, measure the fourth-spread that is the distance between the lower quartile (lowest 25% of data) and upper quartile (highest 25% of data), and set the upper and lower outlier boundaries as a function of fourth-spread.

After excluding outliers, standard deviation and mean values where calculated and the normality test was performed. We concluded that in almost every experiment, values could not be modeled by a normal distribution and we decided to proceed with histogram analysis.

Histograms where built around 8 classes, which where afterwards accumulated to build a cumulative chart, similar to a CDF (Cumulative Distribution Function). The different experiments scenarios are analysed and discussed in the following section.

4.2 Amazon S3

The Amazon Simple Storage Service (S3) service makes REST and SOAP access available. This allows storing data through "objects", that are grouped in "buckets", as stated before. The REST API essentially supports PUT, GET, HEAD and DELETE operations, having no way to copy or rename an object or move an object to a different bucket.

Now the focus is on the evaluation speed of the Amazon S3 service, one of the most well known storage services. The S3 requires a high-speed network to provide connection between the end user and the service infrastructure to achieve an satisfactory user experience. The network traffic latency and the physical distance (from user to infrastructure) are factors that can influence throughput.

[2] http://en.wikipedia.org/wiki/Quartile

This experiment conducted sequential transference for the upload and the download operations for each object test file. The test was made with a bucket in the US-Standard Region.

From the user perspective, the upload speed of the S3 depicts the speed of saving a file to the Amazon data centers. The download represents a connection to Amazon data centers in order to save the file on the local disk.

For each object size, we measured the upload (Table 4.1) and download (Table 4.2) speed.

File size	1 Byte (Bps)	1 KB (KBps)	1 MB (MBps)	16 MB (MBps)	100 MB (MBps)
Average	1,31	5,55	0,70	2,62	2,71
Stdev	0,07	0,39	0,03	0,46	0,41
Max	1,43	6,25	0,74	3,89	3,38
Min	1,09	4,55	0,62	1,35	1,53
Sample size	297	319	296	285	278

Table 4.1: Statistical metrics results at upload operation on S3.

File size	1 Byte (Bps)	1 KB (KBps)	1 MB (MBps)	16 MB (MBps)	100 MB (MBps)
Average	6,53	6,24	0,44	0,84	0,92
Stdev	0,23	0,24	0,01	0,01	0,01
Max	7,04	6,80	0,45	0,85	0,93
Min	5,99	5,59	0,42	0,82	0,88
Sample size	295	292	243	267	261

Table 4.2: Statistical metrics results at download operation on S3.

The tables rates values comply with the file size, i.e., for 1 Byte file the rates are in Bytes per second, for the 1 KB file are in Kilobyte per second and for the 1, 16 and 100 MB files are in Megabytes per second. Excluding the last line that represents the number of transactions (sample size) done all values represent throughput values. Both tables show that the average upload transfer rate increases with the file size.

The observed throughput for the 1 byte scenario represent the direct measurement of S3 speed [Simson L. Garfinkel,]. So, in average terms, the observed upload throughput of 1,31 bytes per seconds and the 6,53 bytes per seconds on download, shows that a client could execute a maximum of 1,31 transactions-per-second (TPS) on the server in the upload situation and a maximum of 6,53 transactions-per-second (TPS) when download.

The data was processed and charted in histogram charts (Figure 4.2).

Each chart represents data from a single file, namely, 1B, 1K, 1M, 16M and 100M. These histograms represents the range rates of PUT requests. Figure 4.3 represent the cumulative distribution function for the previous histograms.

We are using logarithmic scale to visually enhance the CDF. Likewise, GET requests are plotted in histograms (Figure 4.4) and the CDF chart is built (Figure 4.5).

In the PUT request (upload) for the small file (1 Byte) the higher probability

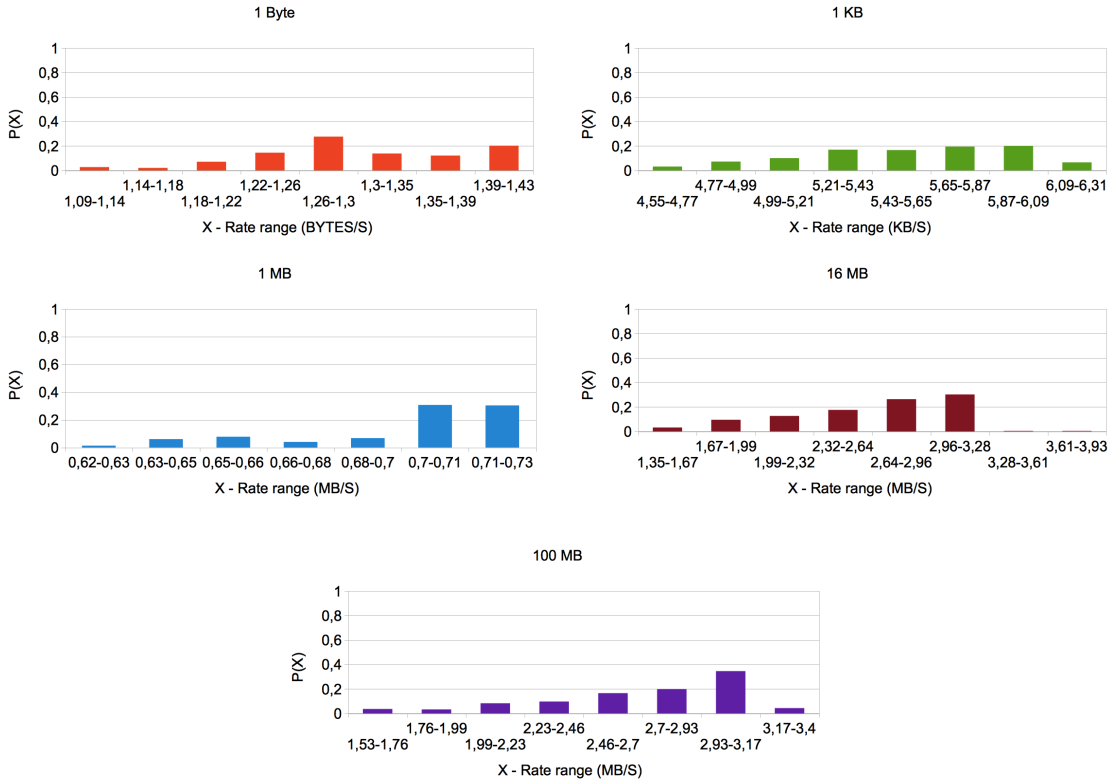

Figure 4.2: Rate range probabilities of different files at upload operation on S3.

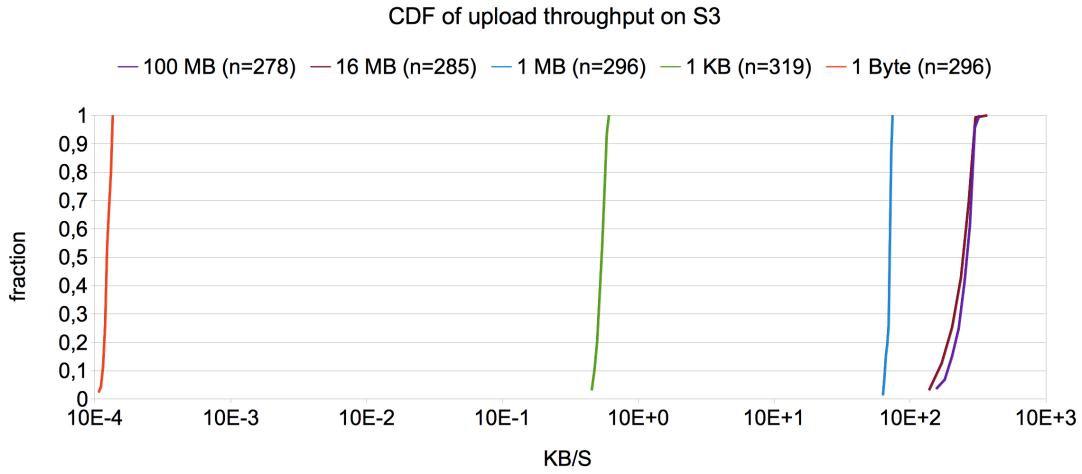

Figure 4.3: CDF of file upload on S3.

(around 30%) lies between the range of 1,26-1,3 bytes per second. For the greater file (100 MB) about 40% is located between the 2,93-3,17 (MB/S) range. In the GET request (download) the higher probability of the 1 Bytes file is located at the 6,52-6,65 (B/S) range rate and for the 100 MB file in the 0,92-0,93 (MB/S)

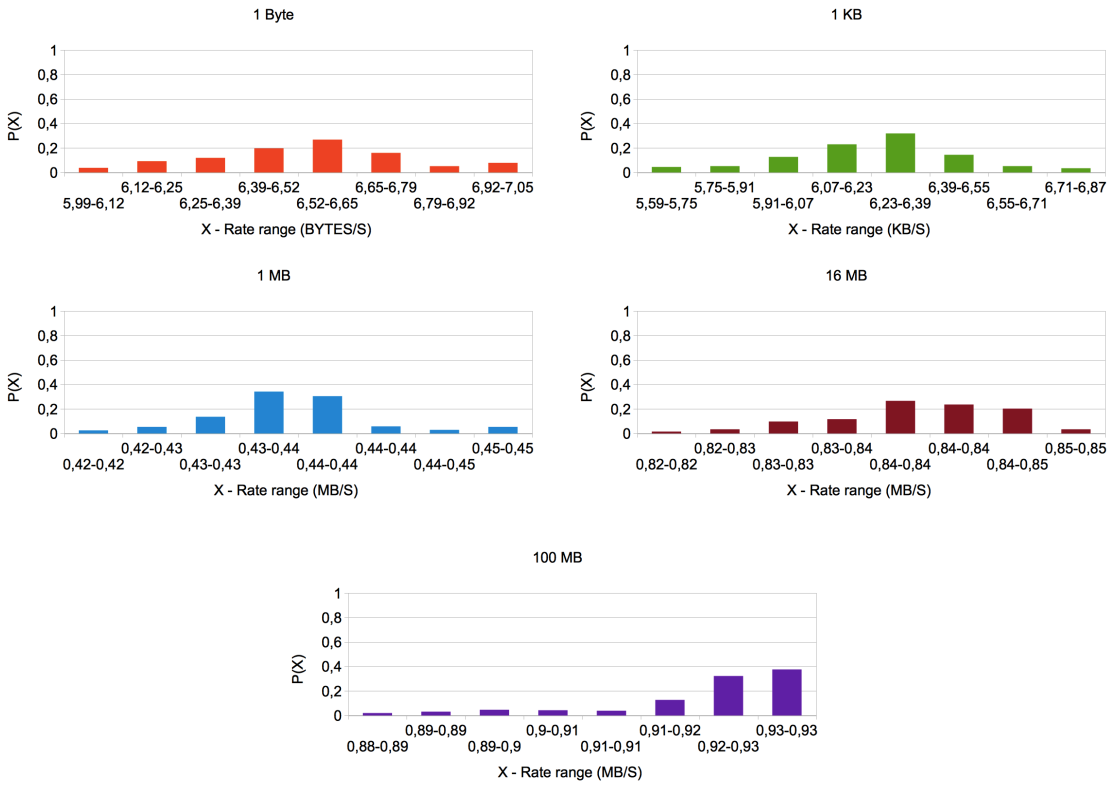

Figure 4.4: Rate range probabilities of different files at download operation on S3.

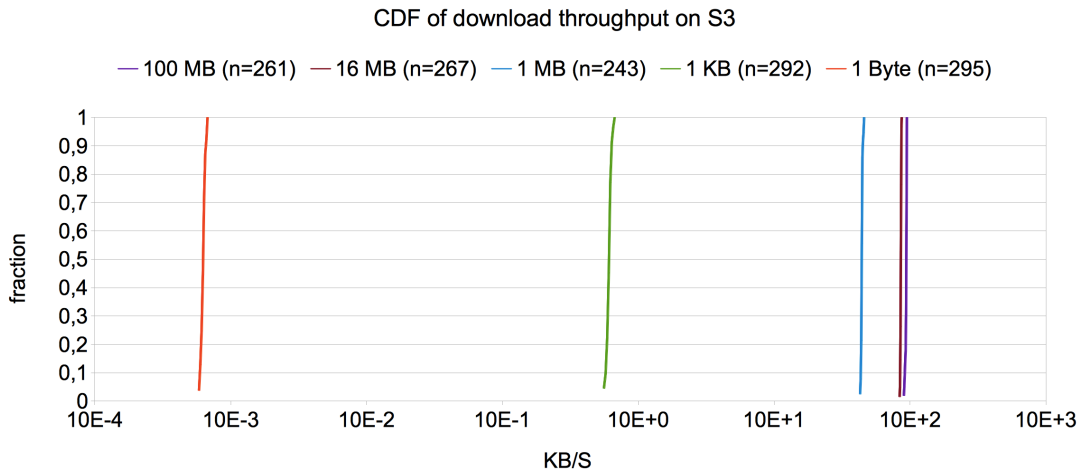

Figure 4.5: CDF of file download on S3.

range. Looking at the ranges rate in the two figures of histograms it can be stated that the speeds of PUT requests are lower than the GET requests for the small files (1 B and 1 KB) but higher for the big files (1, 16 and 100 MB). Thus, in this test, can be asserted that the upload operation achieved higher rates that the

download only for MB sized files. Finally, its important to recall that the results can be influenced by the provider Internet connection speed. This experiment results provides a comparison point for the following tests.

After the S3 measures, the focus migrates for the speed analysis of the REST server built.

4.3 Local server connection

In the Representational State Transfer (REST) architecture, clients and servers exchange representations of resources using a standardized interface and protocol and these principles encourage REST applications to be simple, lightweight, and have high performance. In this way, it was been implemented a RESTful web service, a web application built upon the REST architecture, that provide compatibility with the Amazon S3 REST WS structure. In other words, this Web Service implements the same architecture model to interact with objects and buckets. In virtue of testing the performance speed of this Web Service (WS), we installed this server locally and recorded the throughput for each transaction. This probe expressed the best performance of all tests, as expected, since network influence was not an issue – both WS and the client app were in the same machine.

As the previous experiment we measured the upload (Table 4.3) and download (Table 4.4) speed for each object size.

File size	1 Byte (Bps)	1 KB (KBps)	1 MB (MBps)	16 MB (MBps)	100 MB (MBps)
Average	86,78	112,75	39,92	60,98	37,28
Stdev	13,36	20,78	4,90	9,19	19,18
Max	111,11	142,86	50,00	80,81	68,54
Min	50,00	62,50	27,03	36,36	2,68
Sample size	1337	1435	1436	1439	1532

Table 4.3: Statistical metrics results at upload operation on local WS.

File size	1 Byte (Bps)	1 KB (KBps)	1 MB (MBps)	16 MB (MBps)	100 MB (MBps)
Average	169,95	167,15	59,32	113,31	45,63
Stdev	20,63	17,67	4,03	10,71	6,33
Max	200,00	200,00	66,67	135,59	59,88
Min	125,00	142,86	50,00	83,33	29,26
Sample size	1455	1359	1320	1344	1382

Table 4.4: Statistical metrics results at download operation on local WS.

In average terms, the observed upload throughput is about 87 bytes per seconds (or TPS) and 170 bytes per seconds (or TPS) on download case with 1 Byte data object. Obviously, this values are more higher than the values of S3.

The data was processed and charted in histogram charts (Figure 4.6). Each chart represents data from a single file, namely, 1B, 1K, 1M, 16M and 100M. These

histograms represents the range rates of PUT requests.

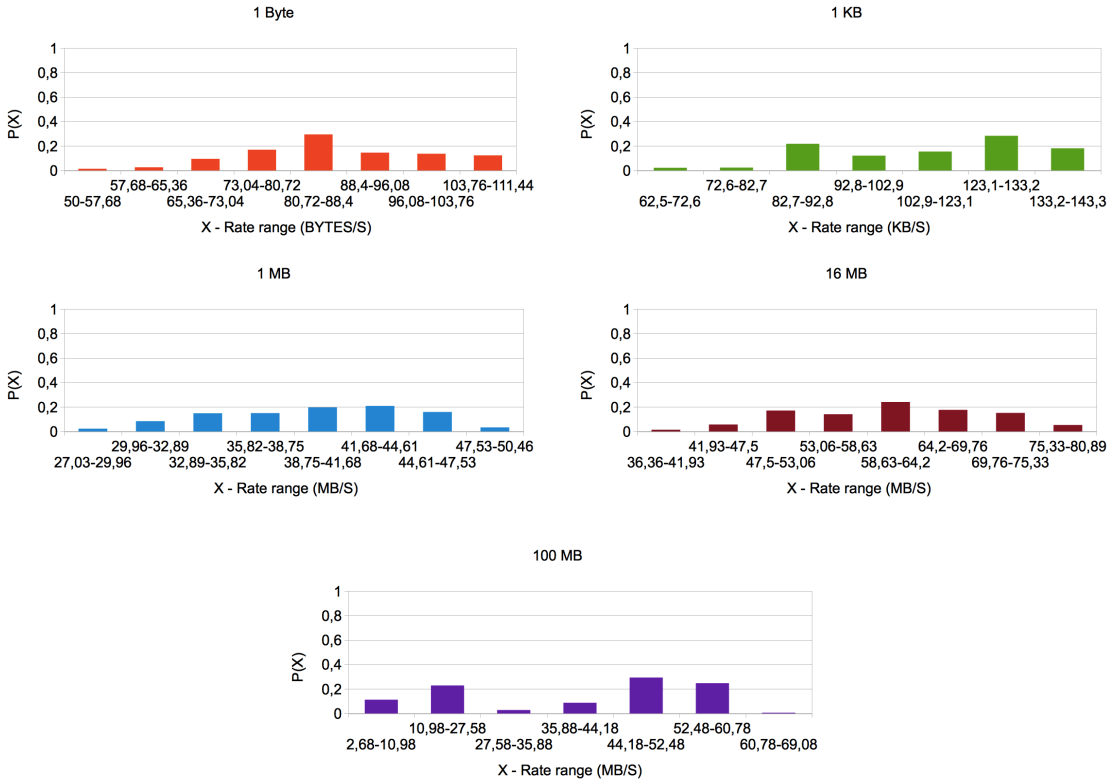

Figure 4.6: Rate range probabilities of different files at upload operation on the local WS.

Figure 4.7 represent the cumulative distribution function for the previous histograms.

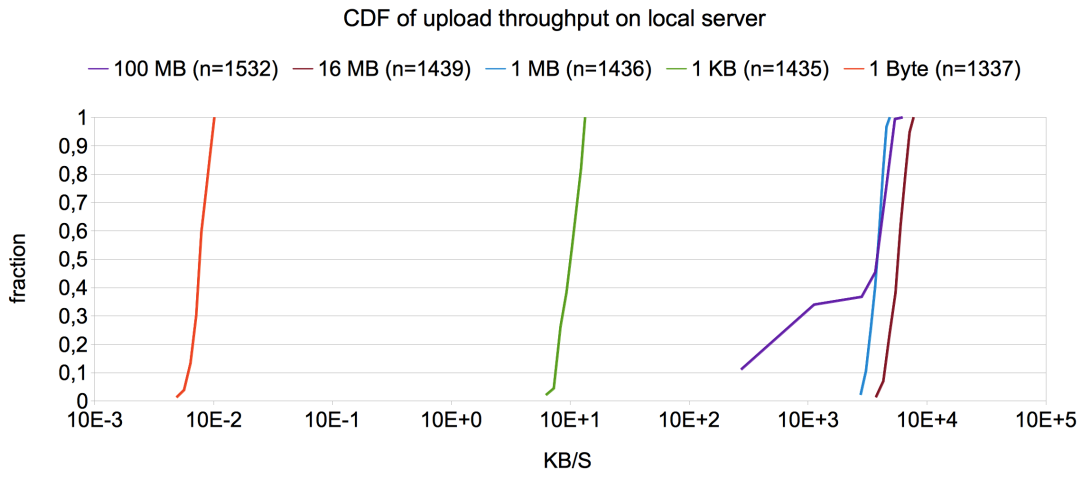

Figure 4.7: CDF of file upload on local REST WS.

In the same way, GET requests are plotted in histograms (Figure 4.8) and the CDF chart is built (Figure 4.9).

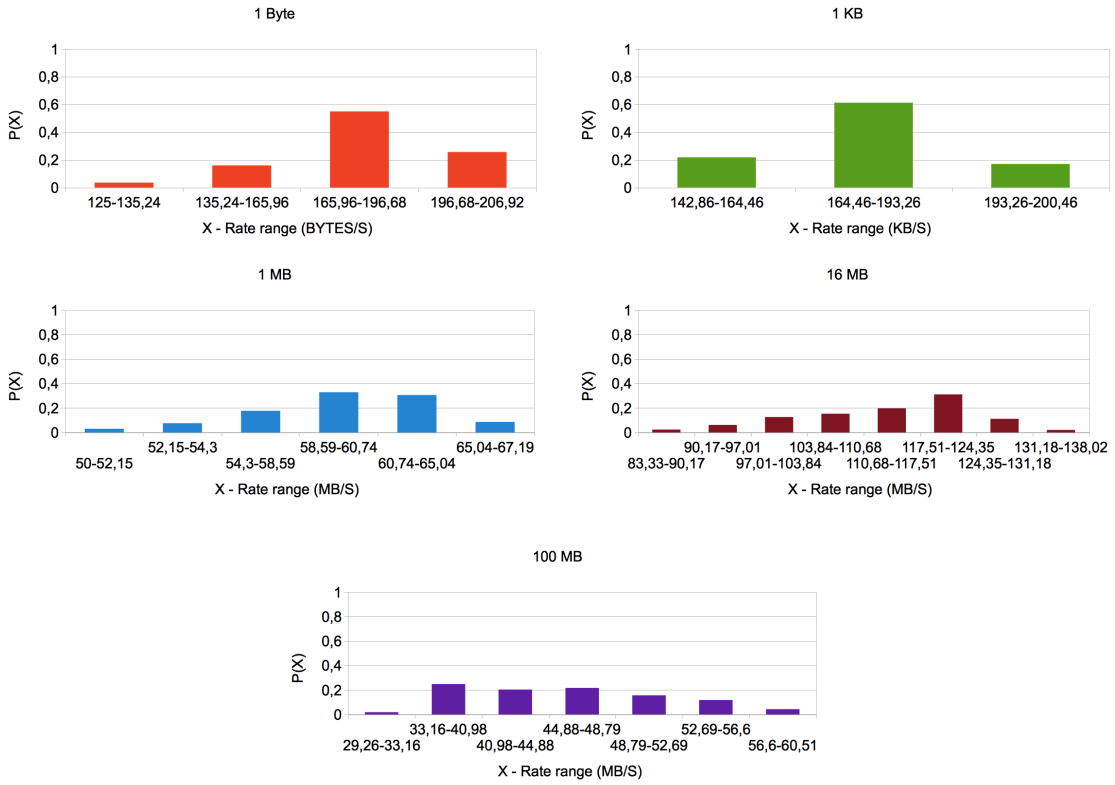

Figure 4.8: Rate range probabilities of different files at download operation on the local WS.

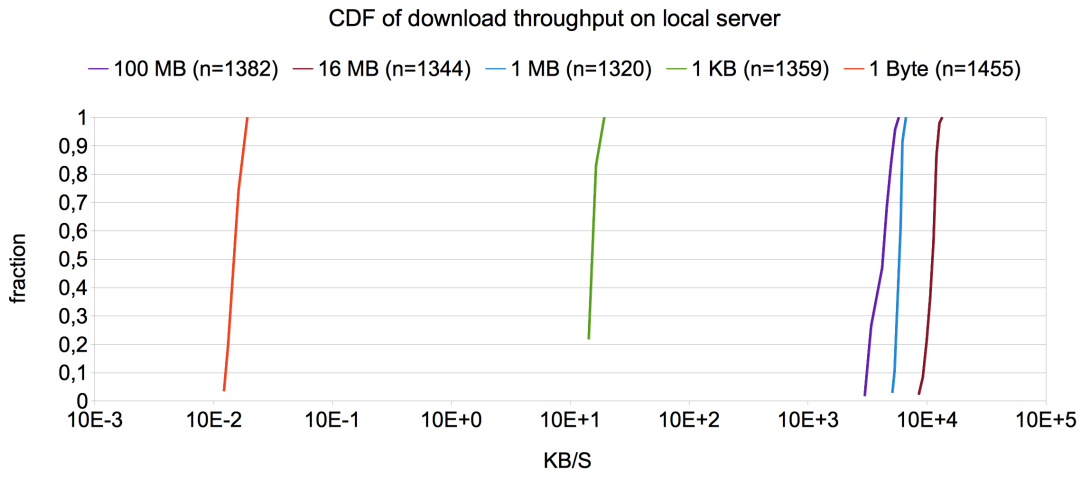

Figure 4.9: CDF of file download on local REST WS.

Each CDF graph contains five traces showing the distribution of observed bandwidths for transactions of 1 Byte, 1 KByte, 1 MByte, 16 MBytes and 100 MBytes.

Not surprisingly, the local WS performs better with larger transaction sizes. However, there is little discernible difference in the 100 MBytes transactions on download, indicating that the 1 and 16 MBytes objects gained face to 100 MBytes data object. In upload this difference is present too.

As expected, the speed rates are approximately the same in both CDF plots that are in the same scale, but a more variation can be found looking at the range rates probabilities histograms. So, comparing the Figure 4.6 and Figure 4.8 is denoted that the download operation (GET) gains face to the upload operation (PUT). In this scenario was expected that both operation on the service had more smaller variations because both operations had the same processing tasks in the machine (read file from the disk and write to the same disk – copy). This variations can be justified by the influence of the machine disk – read/write speeds and caches.

In sum, it is evident that the speeds reached in this implementation do not affect the user experience because it can achieve high transfers data much better than S3. In order to provide more tests about this implementation we change the connection scenario discussed in the section below.

4.4 Network server connection

The same server we used in the previous section was installed in a local network connection, to provide a networked best case scenario. This section depicts the experience with the same previous implementation but it changes the user local access to the network. It is expected lower throughput than in the previous test because the nature of the connection in this environment.

About the experiment method its the same as the previous. For each object size, we measured the upload (Table 4.5) and download (Table 4.6) speed.

File size	1 Byte (Bps)	1 KB (KBps)	1 MB (MBps)	16 MB (MBps)	100 MB (MBps)
Average	35,77	46,24	1,86	1,95	1,96
Stdev	14,03	14,35	0,31	0,28	0,27
Max	71,43	83,33	2,51	2,43	2,42
Min	5,95	12,82	1,14	1,37	1,43
Sample size	689	618	688	683	677

Table 4.5: Statistical metrics results at upload operation on network by the WS.

Both tables show that the average upload transfer rate increases with the file size as the S3 experiment.

In average values, the observed bandwidth for 1 byte transaction in the upload case is about 35 TPS and almost 60 TPS in the download case. Values much better than S3 (1,31 for upload and 6,53 for download). With this values it can be assert that the GET operation is more faster than the PUT operation.

File size	1 Byte (Bps)	1 KB (KBps)	1 MB (MBps)	16 MB (MBps)	100 MB (MBps)
Average	59,95	63,37	2,08	2,14	2,14
Stdev	4,60	5,16	0,23	0,24	0,24
Max	66,67	71,43	2,51	2,45	2,40
Min	50,00	52,63	1,50	1,56	1,60
Sample size	565	585	658	671	669

Table 4.6: Statistical metrics results at download operation on network by the WS.

After the data was processed and depicted in histogram charts (Figure 4.10). These set of histograms represents the range rates of PUT requests for each single file, namely, 1B, 1K, 1M, 16M and 100M.

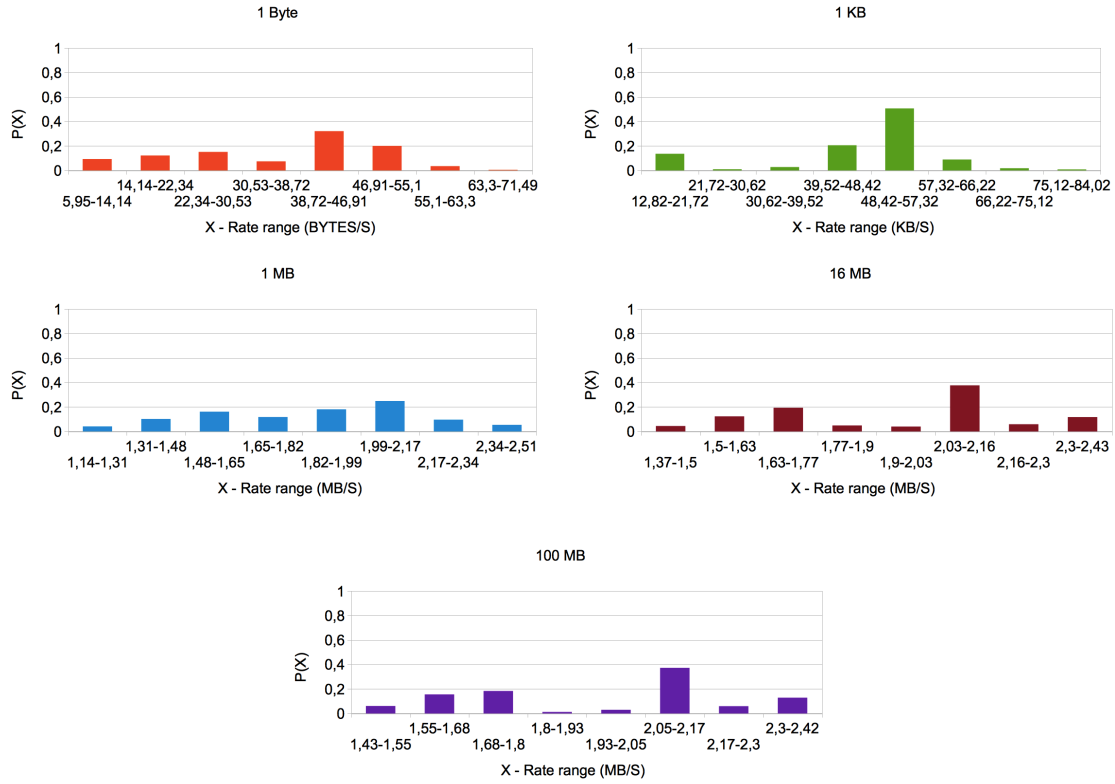

Figure 4.10: Rate range probabilities of different files at upload operation on the network by the WS.

The next graph (Figure 4.11) shows the CDF of the previous histograms.

Equally, GET requests are plotted in histograms (Figure 4.12) and the CDF chart is built (Figure 4.13).

Both CDF plots are on the same scale, making it easy to compare the performance differences between them. In the same plots can be seen that the representation of the function in the files of 1, 16 and 100 MB are practically identical.

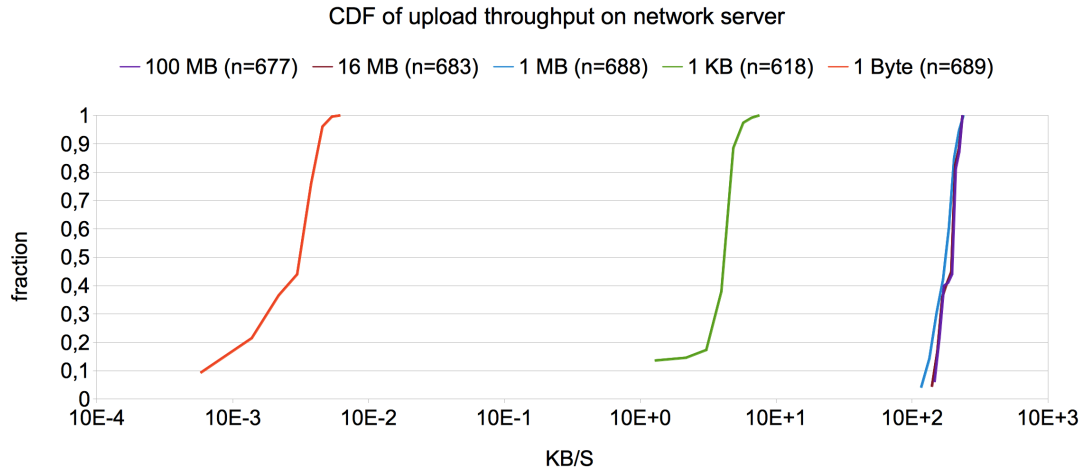

Figure 4.11: CDF of file upload on the network REST WS.

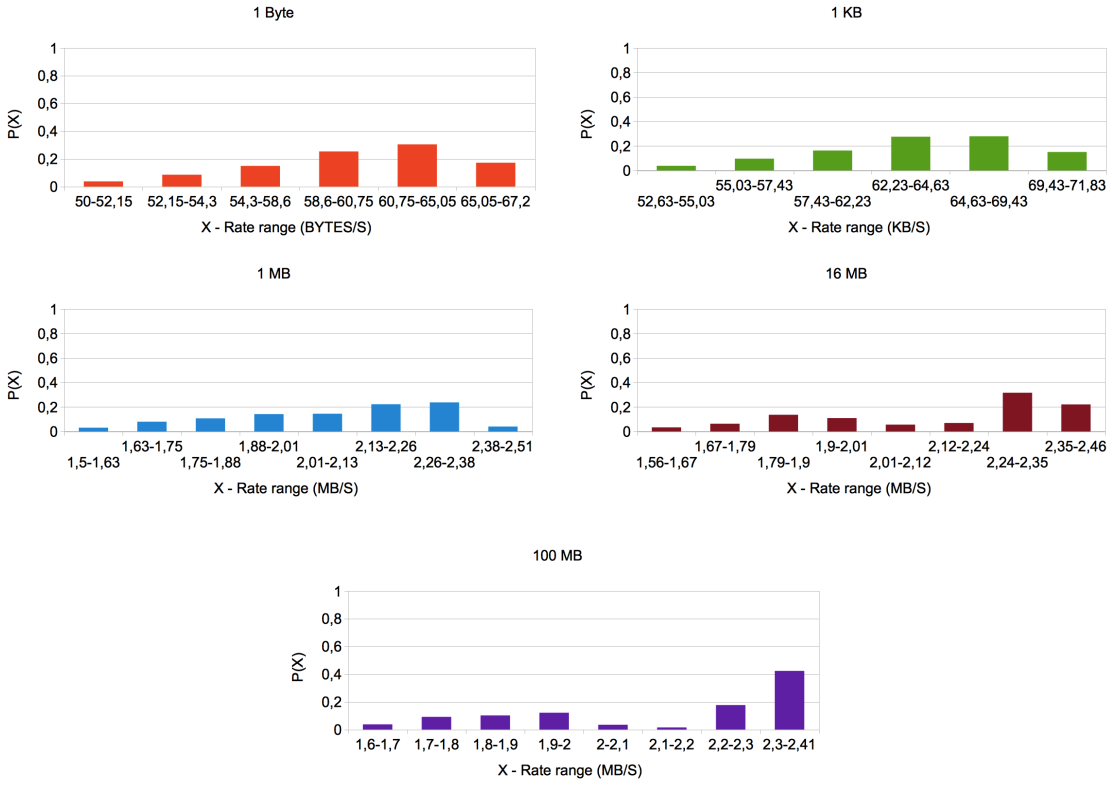

Figure 4.12: Rate range probabilities of different files at download operation on the network by the WS.

Looking at Figures 4.10 and 4.12 its showed the probabilities histogram of different files where is represent the ranges rates. For the upload case, if the range rates probabilities are compared with the S3 case it is visible that this server has more

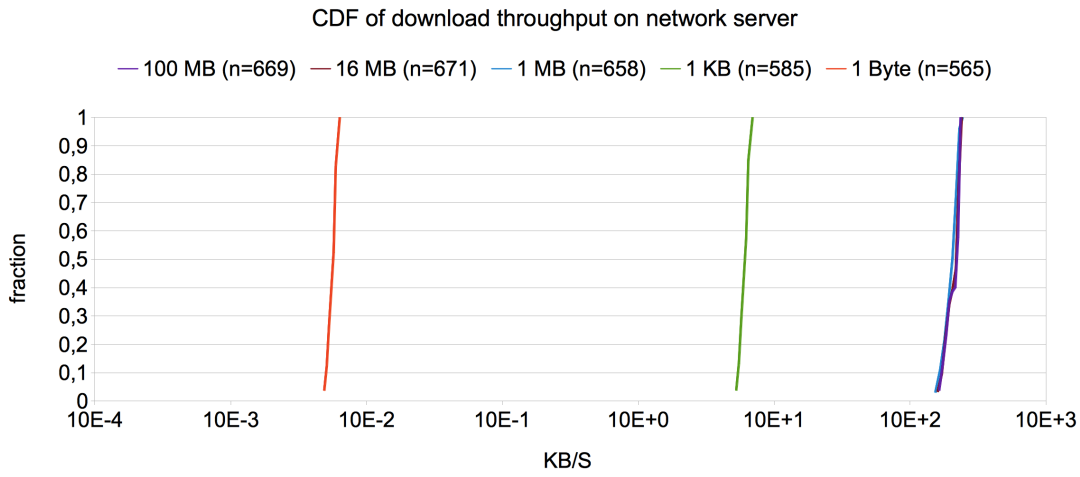

Figure 4.13: CDF of file download on the network REST WS.

high chances and high speed values only for the 16 an 100 MB files. In the download case this server has always much high values than the S3 service.

This experiment intended to show some performance aspects about the WS developed on the network connection environment. Next section focus on the D3S implementation with the analysis of the BitTorrent network environment.

4.5 D3S - BitTorrent Network

This last experiment aims to get some insight on the throughput in the BitTorrent scenario. Mainly, it is investigated the influence that the number of seeders have. We expect that with increasing number of seeds, throughput will be higher, because of the nature of BitTorrent protocol.

BitTorrent was designed to provide better results with large files. Peers are constantly joining the swarm with time and as more peers are available better performance is achieved. Other important aspect is the piece length used. The piece length specifies the nominal piece size, and is usually a power of 2. The piece size is typically chosen based on the total amount of file data in the torrent, and is constrained by the fact that too-large piece sizes cause inefficiency, and too-small piece sizes cause large .torrent metadata file. So current best-practice is to keep the piece size to 512 KB or less between the common sizes (256, 512, and 1024 KB) and with this size is achieved one intermediate solution.

In this scenario, all peers were installed behind a NAT gateway and limited by the Internet provider connection (upload rates). This can influence the final results providing a worst case scenario overview.

Starting with one seeder and up to 16 seeders we measured the elapsed time of the download action on D3S, using the same methodology from the previous experiments.

This download action includes:

- getting the .torrent file through RMI to make it available in the local peer;
- authenticating the peer;
- downloading the file through the BT network;
- returning the file to the S3 client.

This download action represents the user experience time when the user download his own file that are replicated by the several peers. In this experiment the file upload time is ignored because it was already tested in the local connection scenario. So when a user uploads a file in the D3S it is the same as uploading it to the local server. The main difference is that in the D3S system a BT client its launch to seed the file for the others peers.

The first measure attempt was to provide only one seed. With one replica of each file the throughput was low (about 0,2 TPS) reaching the maximum average value of the 0,05 MBps in the 100 MB file (Table 4.7).

File size	1 Byte (Bps)	1 KB (KBps)	1 MB (MBps)	16 MB (MBps)	100 MB (MBps)
Average	0,203	0,274	0,023	0,045	0,049
Stdev	0,187	0,173	0,018	0,006	0,002
Max	0,677	0,664	0,048	0,054	0,053
Min	0,001	0,003	0,003	0,032	0,044
Sample size	59	47	59	53	52

Table 4.7: Statistical metrics results at download operation on D3S with 1 seed.

We advanced to a new test with 2 seeders. In this situation, the values for the small files (1 Byte and 1 KB) remain very low relatively to the one seeder case (Table 4.8). But the average values are growing along the seeders size if it is compared the two tables.

File size	1 Byte (Bps)	1 KB (KBps)	1 MB (MBps)	16 MB (MBps)	100 MB (MBps)
Average	0,276	0,574	0,043	0,062	0,073
Stdev	0,298	0,462	0,014	0,024	0,023
Max	0,944	1,783	0,078	0,098	0,099
Min	0,0004	0,0001	0,017	0,002	0,033
Sample size	62	64	45	62	62

Table 4.8: Statistical metrics results at download operation on D3S with 2 seeds.

For both cases histograms charts were built (Figure 4.14 and 4.15). Each chart represents data from a single file, namely, 1B, 1K, 1M, 16M and 100M. These histograms represents the range rates of GET requests.

From the histograms of Figure 4.14 and 4.15 we represented the CDF of both situations in the Figure 4.16.

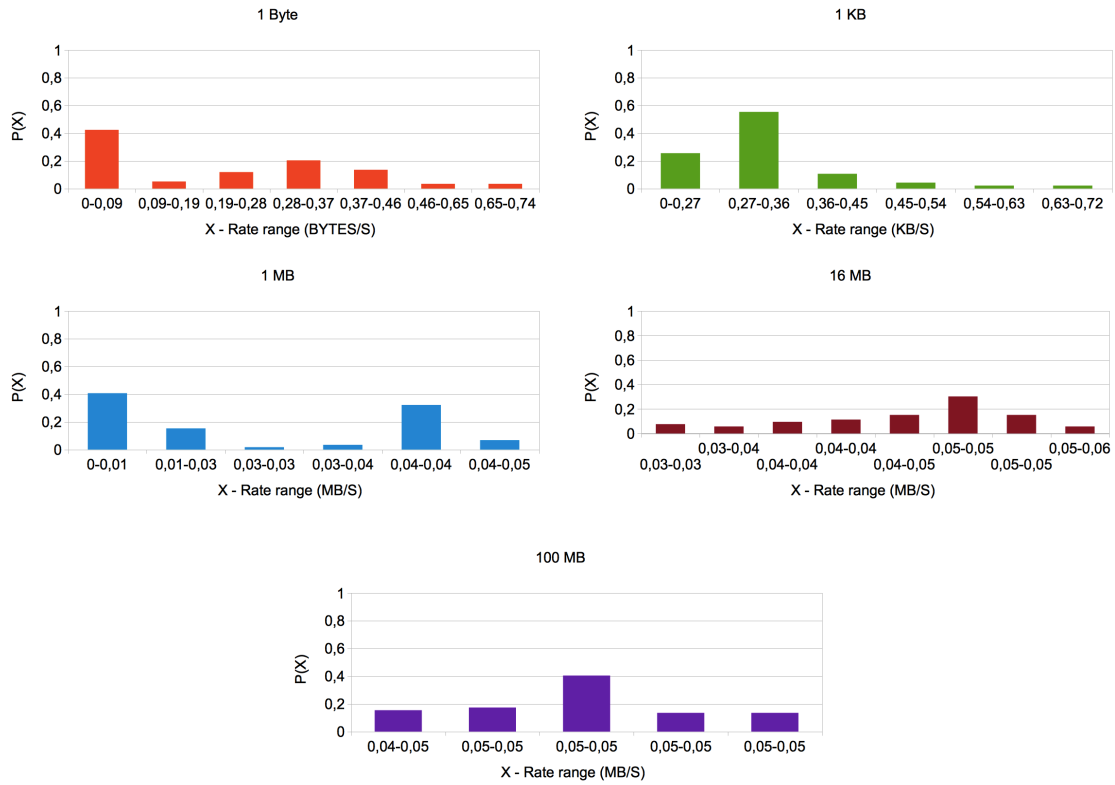

Figure 4.14: Rate range probabilities of different files at download operation on D3S with 1 seed.

Looking at a more seeders injection scenario with 4 seeds the the small sized files (1 Byte and 1 KB) remain at very lows values (Table 4.9).

File size	1 Byte (Bps)	1 KB (KBps)	1 MB (MBps)	16 MB (MBps)	100 MB (MBps)
Average	0,021	0,295	0,071	0,107	0,195
Stdev	0,023	0,327	0,030	0,055	0,076
Max	0,086	1,181	0,142	0,231	0,291
Min	0,002	0,003	0,012	0,032	0,044
Sample size	51	55	55	57	61

Table 4.9: Statistical metrics results at download operation on D3S with 4 seeds.

At this time, it can be assert that this system is not appropriated for this files (1 Byte and 1 KB) due to the nature of the BitTorrent protocol. So the time of the download process in this case not offset the size file hindering the high transfer rates and showing, such others studies [Wei et al., 2005], that BitTorrent suffers from a high overhead when transmitting small files . Relatively to the others files (1, 16 and 100 MB) is visible that the transfer rate increase substantially with the file size.

Figure 4.17 represents the histograms charts for the 4 seeds. Each chart represents the range rates of GET request from a single file, namely, 1B, 1K, 1M, 16M and 100M.

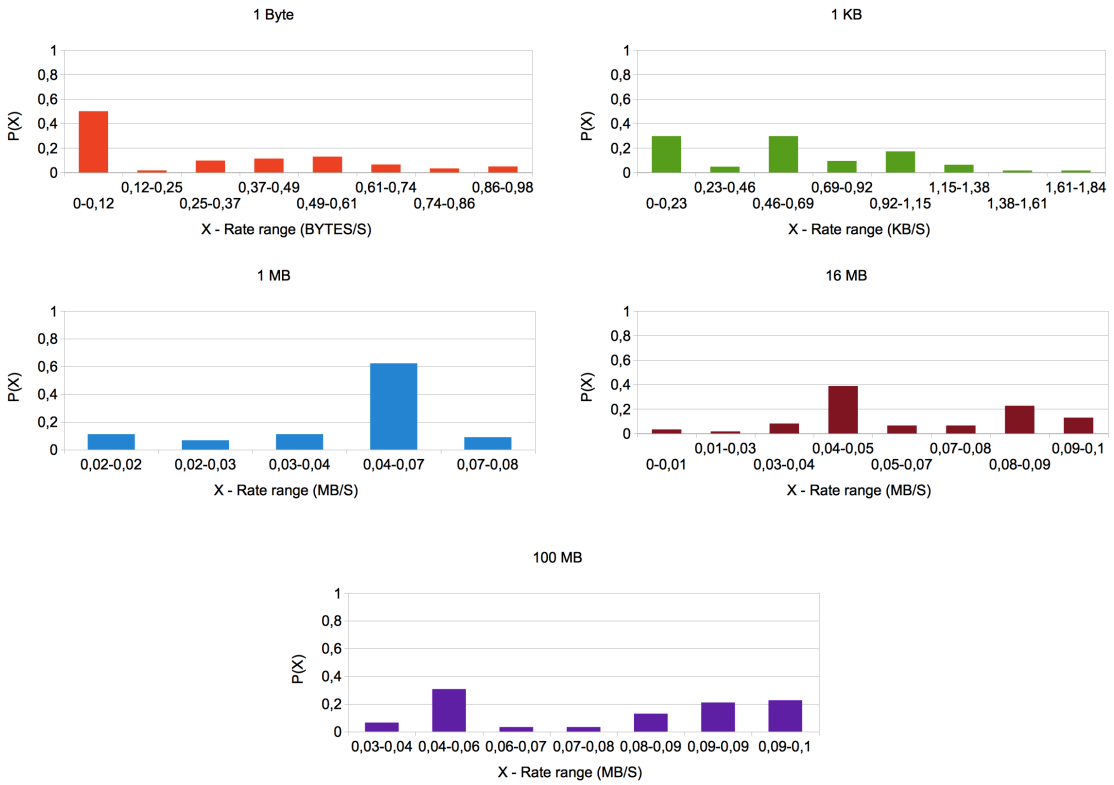

Figure 4.15: Rate range probabilities of different files at download operation on D3S with 2 seeds.

With the 16 seeds last experiment is notable that the service sometimes can achieve high transfers rates up to 0,782 MB/S in the 100 MB file case (Table 4.10). In average terms, the performance grew along with the file size hitting 0,02 TPS with the 1 Byte file, a very low value comparatively with the others tests (S3 for example).

File size	1 Byte (Bps)	1 KB (KBps)	1 MB (MBps)	16 MB (MBps)	100 MB (MBps)
Average	0,021	0,038	0,061	0,122	0,248
Stdev	0,029	0,044	0,070	0,101	0,212
Max	0,149	0,171	0,218	0,395	0,782
Min	0,001	0,002	0,003	0,021	0,049
Sample size	36	37	41	38	39

Table 4.10: Statistical metrics results at download operation on D3S with 16 seeds.

In the Figure 4.18 are represented the range rate probabilities for each file and is perceptible that most of the results are at low ranges.

These probability histograms charts of GET request of both scenarios (Figure 4.17 and 4.18) supported the creation of the CDF plot (Figure 4.19) that represents the differences of the speed rates in the 4 vs 16 seeds case. This both CDF provided an better overview of the different files throughput and the last scenario of 16 seeds

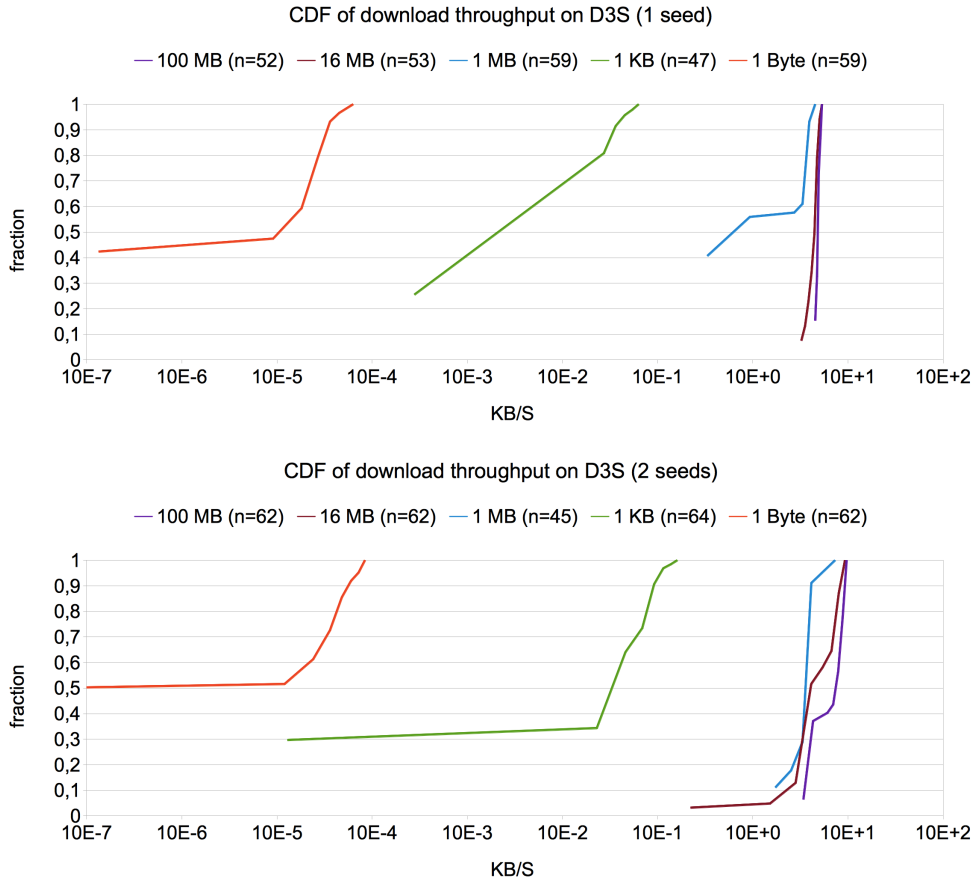

Figure 4.16: CDF of download throughput of the D3S service – 1 seed VS 2 seeds.

(bottom CDF) represent the biggest seeds case that we analyse. Comparatively with the S3 service this last scenario not provided enough transfers rates to confront with. Sometimes achieved similar transfer rates if we compared the maximum values but still not be enough in constant values face to the cloud service. In the section below, we discussed all of this results presenting the main reasons that might be concerned to obtain such values.

4.6 Discussion

Previously, we described the performance results of all the scenarios implemented. In this section we intend to discuss all of this results with a especial focus in our final solution (D3S) and the S3 service.

Making an overall overview to all different files in the GET experiment it is visible that the D3S system is not suitable for the small files (1 Byte and 1 KB). As mentioned, the main reason of this values is the nature of BitTorrent protocol that causes inefficiency with small files but other reason that hampers this performance results is the huge time (for this files) of the download process structure. As stated,

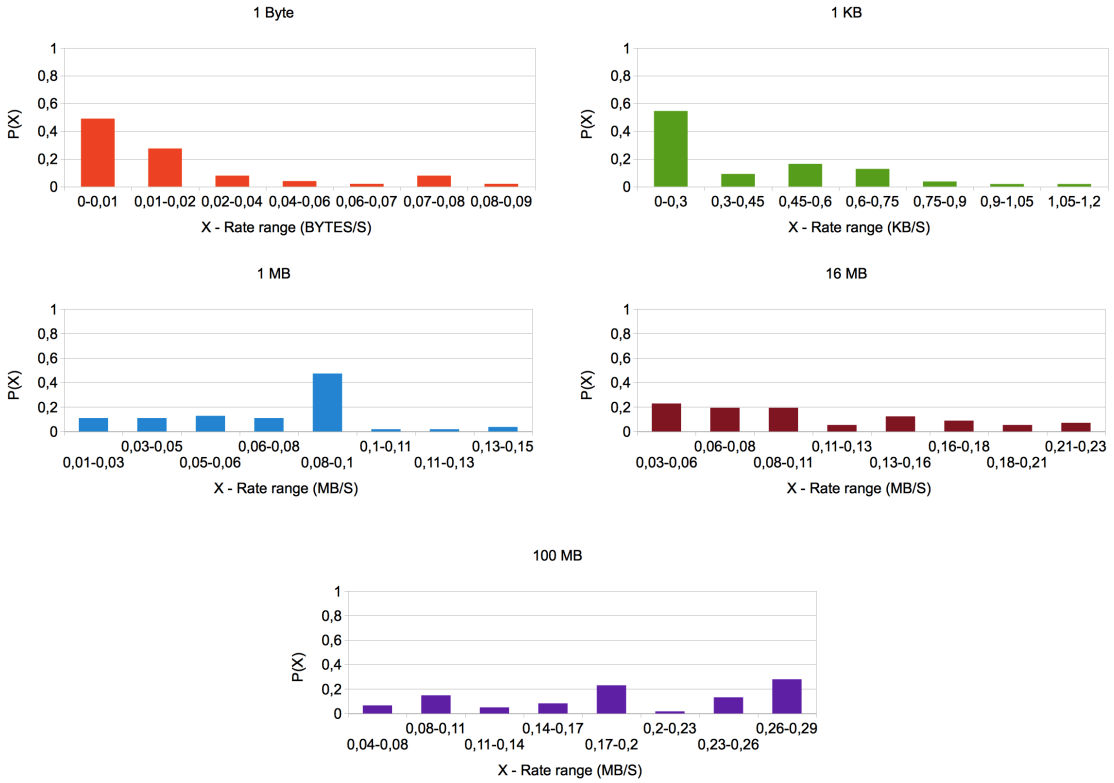

Figure 4.17: Rate range probabilities of different files at download operation on D3S with 4 seeds.

the download process consists in a variety of tasks. This tasks includes the time of the *.torrent* file download and the peers authentication and connection (peer handshake). We point this last as the main reason of this low values. In all experiments the 1 Byte file tested the direct measurement of the server speed rate. But in the BitTorrent network case this file tested the REST server measure plus the whole process that the our BitTorrent structure requires. So the problem is that the file is not sufficiently large to address the time download process and to exploit the high transfers of the BitTorrent protocol. For a better visualization of this huge differences between the different sized files we represented, with a logarithmic scale, a bar plot of the D3S case in contrast with a S3 line plot (Figure 4.20).

In Figure 4.20 it can be shown the difference in scale between D3S and S3, particularly for large files. It is evident that the number of seeders influence the performance of the D3S system – more seeds represents more performance. As can been seen, the S3 throughput line is always higher than any combination of seeds in the D3S which was unexpected. Through such figure and looking at the best score of the D3S (16 seeds with the 100 MB file) we can affirm that the S3 service is about three times faster.

The first attempt is to state that the seeders must grow beyond to the sixteen to achieve the S3 results. This is not so evident. This only happens because all of peers are beyond a NAT gateway (firewall), like general home networks. This

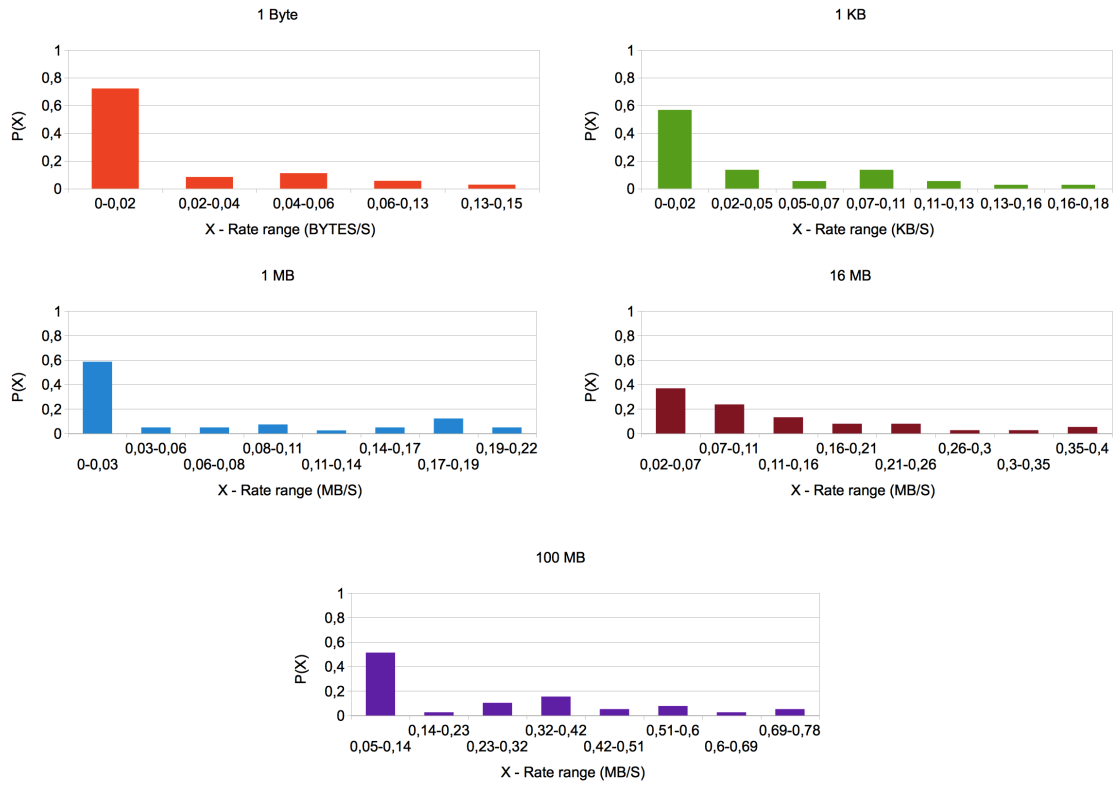

Figure 4.18: Rate range probabilities of different files at download operation on D3S with 16 seeds.

NAT environment hampers the peers connection being responsible in most cases for the connection timeout warning displayed in the system logs. This warn was only displayed when the leecher cannot contact some of the seeders. In reality only a few part of the seeders contributing to the system throughput despite the sixteen seeds. This results in a evident loss of performance.

This was not happened in a monitored private/corporate network connection. This scenario is not reported in this document because the aim is to provide only a real scenario overview. Although, this monitored network connection scenario is reported in the appendix section A.1. With them we intended to show the initial performance results that overcome the S3 service changing only the seeders location, i.e., instead of using the seeders geographical apart beyond a NAT gateway we used Amazon EC2 instances to install the D3S seeders. However, this not represent a real user experience scenario once the EC2 machines (peers) are fully available (without firewall) to the leecher peer (located at the FCCN) with high upload transfers rates.

Returning to the previous scenario the upload rates of each peer connected are, in almost cases, very low because are limited by the Internet service provider which might have influenced the leecher performance. So if the peers connected could increase the upload rate will provide better results of course. This dependency of the upload rate represent a critical element for whole system throughput.

Other important observation which complicates the system performance is the

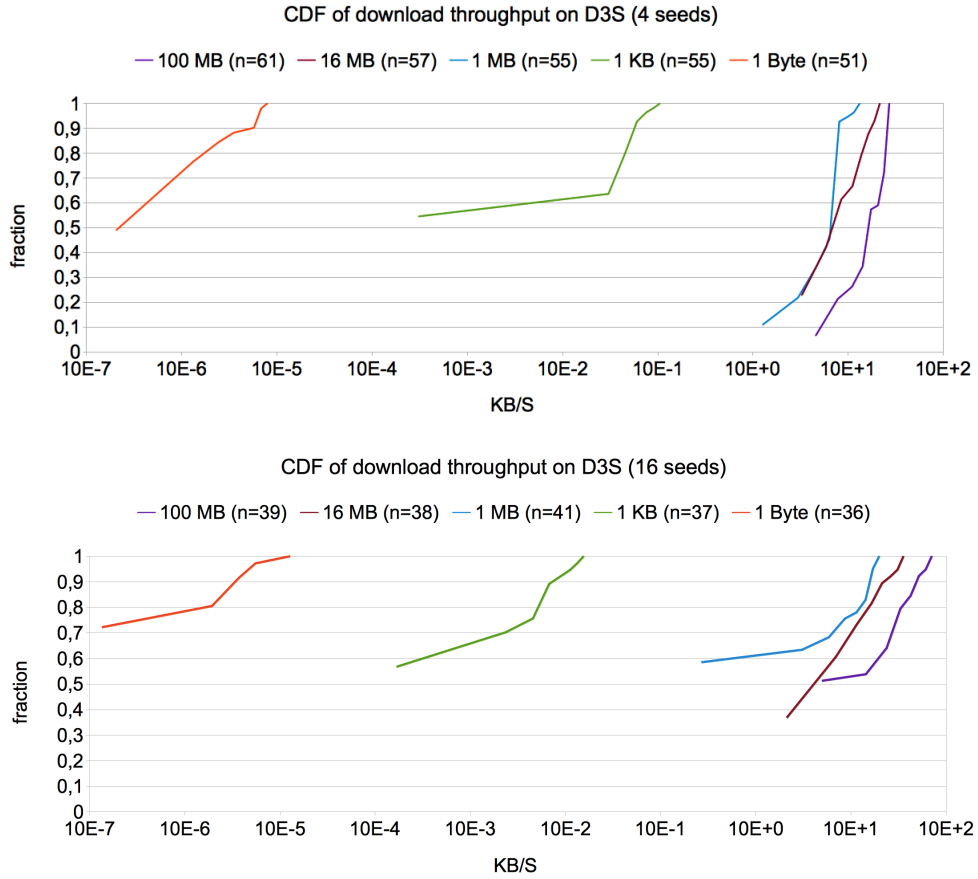

Figure 4.19: CDF of download throughput of the D3S service – 4 seed VS 16 seeds.

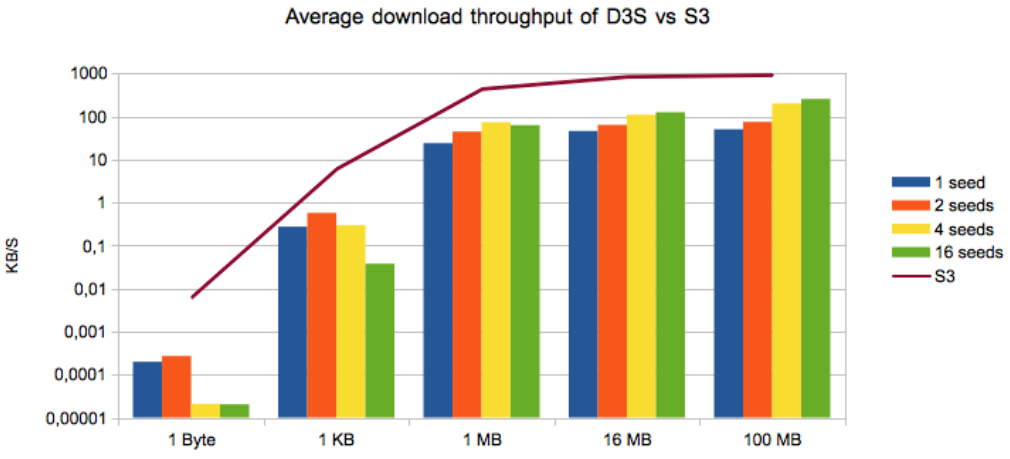

Figure 4.20: D3S vs S3 – Average download throughput (logarithm scale) with different files.

BitTorrent choking algorithm. In the BitTorrent environment peers continuously download pieces from all peers which they can attempting to maximize its own

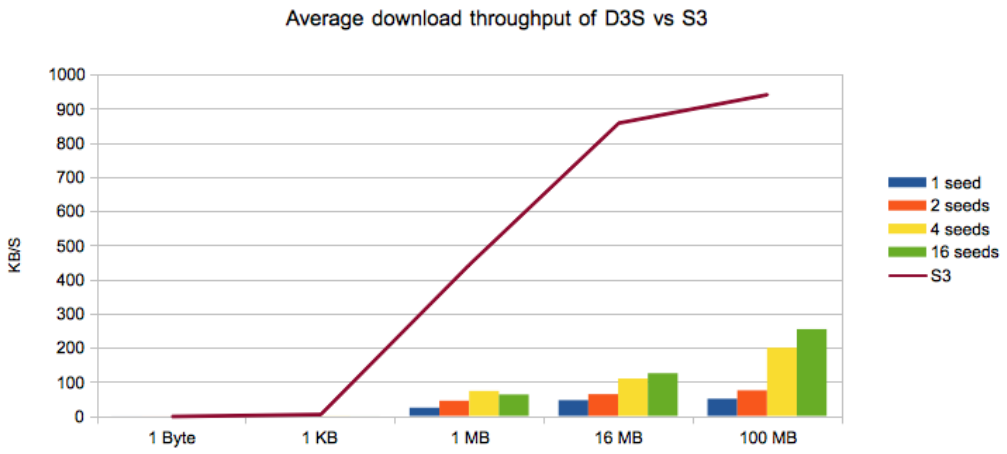

Figure 4.21: D3S vs S3 – Average download throughput with different files.

download rate. They of course cannot download from peers they are not connected to. To cooperate the peers upload and when do not cooperate they "choke". This choking algorithm, known by temporary refusal to upload, uses all available resources, providing reasonably and consistent download rates for everyone. This consisted download rates influence the the peers that only download and not upload. And in this experiment the leecher is affected by this algorithm since it only downloads the content.

Chapter 5

Conclusion and future work

The emergence of cloud computing made several cloud storage services available to users. The pay-per-use economic model is proving to be an interesting alternative to traditional storage (e.g. RAID) solutions. Cloud storage services allow storing data with high level of availability and scalability, with controlled costs and anytime, anywhere access. Because it is a client-server architecture the main power is located on the server side, with the corresponding high installation and maintenance costs. However, several companies already possess such infrastructure to power their main business, such as Microsoft, Amazon or Google. Little effort is then required to make this infrastructure available to users, through cloud computing services.

On the other side of spectrum, peer-to-peer networks also provide interesting file-sharing services, although following a completely different commercial and technological model. In particular, the BitTorrent protocol has made a big impact on the file sharing community, and nowadays, it is one of the most used protocols in the Internet. The advantages of this protocol for file sharing are mainly related to the ability to easily deal with flash-crowd situations as well as speedy transfer of large files.

The work described in this document focus on cloud storage services, in particular, Amazon Simple Storage Service (S3). The study of such service allowed a survey of how it works in the marker as well as technologically.

To assess this service, two client applications were developed: a desktop based GUI file repository in the cloud and a FUSE driver, where the files stored in the cloud are mounted side-by-side with regular files. Two S3 compatible servers where also developed. The first is a standalone instance, that can be accessed locally and through a network connection. Finally, a Distributed Simple Storage Service (D3S) was also developed, which allowed storing files in a cloud backed by BitTorrent peer-sets. In other words, we implemented a cloud storage service compatible with S3 over a peer-to-peer BitTorrent network.

The most complicated part of a BitTorrent storage scheme implementation lies in the distribution of the .torrent file. We approached this problem through Java RMI, which allows each peer to distribute, download and eliminate .torrent files.

To achieve the required level of data availability, the simplest scheme is to store

data redundantly – where the same object is stored in different locations. This redundancy is also used generally to protected data on unreliable servers. In D3S, the redundancy was generated by replicating data between the peers. This also contributes to increase throughput, because BitTorremt allows simultaneous download of different file pieces.

With the developed clients and servers we structured several experiments, aiming at measuring the throughput and latency of these storage services. We started by measuring Amazon S3, a local installed S3 server and a LAN installed S3 server. Among these, the results were as expected, increasing throughput with the file size, although uploading to Amazon S3 revealed to be higher than downloading.

We also performed several experiments to the D3S, varying the number of peers. As expected, the number of peers influenced the throughput. However, we did expected to have more throughput for 16 peers than we were able to measure. NATed peers contributed to connectivity difficulties, that led to lower than expected throughput.

About the file size variation it was clear that larger files presented higher throughput, until a specific limit.

The bottom line is that D3S did not managed to overcome the Amazon S3 in throughput. On the other hand, D3S achieves reasonably good performance because it typically accesses data items directly (i.e. it only uses replication for both maintenance and data access) and achieves very high data availability at a reasonable level.

In short, the results suggest a distributed storage system with few maintenance costs, good access time, desired high availability, and reasonable storage overhead.

In future efforts, we intend to improve the solution in order to obtain higher throughput, closing on or exceeding that of Amazon S3.

We also expect to replace the Java RMI based indexer with a Distributed Hash Table (DHT), to further eliminate centralized modules.

The other functionally that is intended to add is the file encryption. With this the files can travel through the BitTorrent network with privacy despite it will result in a addiction of time because the time of the client encryption file during uploads and downloads calls.

Bibliography

[Akioka and Muraoka, 2010] Akioka, S. and Muraoka, Y. (2010). HPC Benchmarks on Amazon EC2. pages 1029–1034. IEEE.

[Alhamad et al., 2010] Alhamad, M., Dillon, T., and Chang, E. (2010). Conceptual SLA framework for cloud computing. In *4th IEEE International Conference on Digital Ecosystems and Technologies*, pages 606–610. IEEE.

[Armbrust et al., 2009] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Others (2009). Above the clouds: A berkeley view of cloud computing. Technical report, Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

[Bhardwaj et al., 2010] Bhardwaj, S., Jain, L., and Jain, S. (2010). Cloud computing: A study of infrastructure as a service (IAAS). *International Journal of engineering and information Technology*, 2(1):60–63.

[Boniface et al., 2010] Boniface, M., Nasser, B., Papay, J., Phillips, S. C., Servin, A., Yang, X., Zlatev, Z., Gogouvitis, S. V., Katsaros, G., Konstanteli, K., Kousiouris, G., Menychtas, A., and Kyriazis, D. (2010). Platform-as-a-Service Architecture for Real-Time Quality of Service Management in Clouds. *2010 Fifth International Conference on Internet and Web Applications and Services*, pages 155–160.

[Cachin et al., 2009] Cachin, C., Keidar, I., and Shraer, A. (2009). Trusting the cloud. *SIGACT News*, 40(2):81–86.

[Ciurana, 2009] Ciurana, E. (2009). *Developing with Google App Engine*. Springer.

[Cong Wang et al., 2009] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou (2009). Ensuring data storage security in Cloud Computing. pages 1–9. IEEE.

[Cong Wang et al., 2010] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou (2010). Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing. pages 1–9. IEEE.

[Cusumano, 2010] Cusumano, M. (2010). Cloud computing and SaaS as new computing platforms. *Commun. ACM*, 53(4):27–29.

[Demirkan et al.,] Demirkan, H., Goul, M., and Soper, D. Service Level Agreement Negotiation: A Theory-based Exploratory Study as a Starting Point for Identifying Negotiation Support System Requirements. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, pages 37b–37b. IEEE.

[Dikaiakos et al., 2009] Dikaiakos, M. D., Katsaros, D., Mehra, P., Pallis, G., and Vakali, A. (2009). Cloud Computing: Distributed Internet Computing for IT and Scientific Research. *IEEE Internet Computing*, 13(5):10–13.

[Egevang and Francis, 1994] Egevang, K. and Francis, P. (1994). The IP Network Address Translator (NAT). RFC 1631 (Informational). Obsoleted by RFC 3022.

[Eng Keong Lua et al., 2005] Eng Keong Lua, Crowcroft, J., Pias, M., Sharma, R., and Lim, S. (2005). A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials*, 7(2):72– 93.

[Ganesh, 2003] Ganesh, A. (2003). Peer-to-peer membership management for gossip-based protocols. *..., IEEE Transactions on*.

[Hamad, 2010] Hamad, H. (2010). Performance evaluation of restful web services for mobile devices. *International Arab Journal of e-Technology*.

[Hu, 2005] Hu, Z. (2005). NAT traversal techniques and peer-to-peer applications. *... and Multimedia Laboratory, Helsinki University of ...*.

[Kaune et al., 2009] Kaune, S., Rumin, R. C., Tyson, G., Mauthe, A., Guerrero, C., and Steinmetz, R. (2009). Unraveling BitTorrent's File Unavailability: Measurements, Analysis and Solution Exploration. *arXiv:0912.0625*.

[Knorr and Gruman, 2008] Knorr, E. and Gruman, G. (2008). What cloud computing really means. *InfoWorld*, 7.

[Krawczyk et al., 1997] Krawczyk, H., Bellare, M., and Canetti, R. (1997). HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational). Updated by RFC 6151.

[Legout et al., 2005] Legout, A., Urvoy-Keller, G., and Michiardi, P. (2005). Understanding bittorrent: An experimental perspective. *INRIA Sophia Antipolis/INRIA ...*.

[Maassen et al., 1999] Maassen, J., van Nieuwpoort, R., and Veldema, R. (1999). An efficient implementation of Java's remote method invocation. *ACM Sigplan ...*.

[Marinos and Briscoe, 2009] Marinos, A. and Briscoe, G. (2009). Community cloud computing. *Cloud Computing*, pages 472–484.

[Marston et al., 2011] Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., and Ghalsasi, A. (2011). Cloud computing — The business perspective. *Decision Support Systems*, 51(1):176–189.

[Mell and Grance, 2011] Mell, P. and Grance, T. (2011). The NIST definition of cloud computing (draft). *NIST special publication*, 800:145.

[Ostermann et al., 2010] Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., and Epema, D. (2010). A performance analysis of EC2 cloud computing services for scientific computing. *Cloud Computing*, pages 115–131.

[Pamies-Juarez, 2011] Pamies-Juarez, L. (2011). Towards the design of optimal data redundancy schemes for heterogeneous cloud storage infrastructures. *Computer Networks*.

[Parameswaran, 2001] Parameswaran, M. (2001). P2P networking: an information sharing alternative. *Computer*.

[Patel et al., 2009] Patel, P., Ranabahu, A., and Sheth, A. (2009). Service Level Agreement in cloud computing.

[Ramgovind et al., 2010] Ramgovind, S., Mm, E., and Smith, E. (2010). The Management of Security in Cloud Computing. *Security*.

[Rewatkar and Lanjewar, 2010] Rewatkar, L. and Lanjewar, U. (2010). Implementation of Cloud Computing on Web Application. *International Journal of Computer Applications IJCA*, 2(8):28–32.

[Rimal et al., 2009] Rimal, B. P., Eunmi Choi, and Lumb, I. (2009). A Taxonomy and Survey of Cloud Computing Systems. pages 44–51. IEEE.

[Rosenthal et al., 2010] Rosenthal, A., Mork, P., Li, M. H., Stanford, J., Koester, D., and Reynolds, P. (2010). Cloud computing: A new business paradigm for biomedical information sharing. *Journal of Biomedical Informatics*, 43(2):342–353.

[Schollmeier, 2001] Schollmeier, R. (2001). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. *Peer-to-Peer Computing, 2001. Proceedings. ...*.

[Schulze and Mochalski, 2009] Schulze, H. and Mochalski, K. (2009). Internet Study 2008/2009. *IPOQUE Report*.

[Simson L. Garfinkel,] Simson L. Garfinkel, S. L. G. An Evaluation of Amazon's Grid Computing Services: EC2, S3, and SQS.

[Wang et al., 2005] Wang, H., Zhu, Y., and Hu, Y. (2005). To unify structured and unstructured P2P systems. *Parallel and Distributed Processing ...*.

[Wei et al., 2005] Wei, B., Fedak, G., and Cappello, F. (2005). Scheduling independent tasks sharing large data distributed with bittorrent. *Grid Computing, 2005. The 6th ...*.

[Weinhardt et al., 2009] Weinhardt, C., Anandasivam, A., Blau, B., and Stosser, J. (2009). Business Models in the Service World. *IT Professional*, 11(2):28–33.

[Wu and Buyya, 2010] Wu, L. and Buyya, R. (2010). Service Level Agreement (SLA) in Utility Computing Systems. *arXiv:1010.2881*.

[Zaharia and Keshav, 2008] Zaharia, M. and Keshav, S. (2008). Gossip-based search selection in hybrid peer-to-peer networks. *Concurrency and Computation: Practice ...*.

[Zhang et al., 2010] Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.

Appendix A

Annex

A.1 Monitored network connection results

This scenario represents the private/corporate network connection experience with the D3S system. With this monitored scenario we intended to show the performance results modifying only the seeders side location. So instead of using the seeders beyond a NAT gateway we use Amazon EC2 instances to run the D3S seeders. This experiment aims to demonstrate the throughput performance of the D3S without a firewall in a controlled system with high upload transfers rates.

To not provide an exhaustive overview of the seeders variations we only focus on the 8 seeders download case because with this seeds number we already achieve better results that S3 with some files. However, it was proved that the seeders play an important role in the performance aspect.

File size	1 Byte (Bps)	1 KB (KBps)	1 MB (MBps)	16 MB (MBps)	100 MB (MBps)
Average	0,101	0,029	0,095	0,652	1,226
Stdev	0,170	0,035	0,125	0,756	0,917
Max	0,543	0,135	0,389	2,021	2,659
Min	0,001	0,001	0,001	0,039	0,100
Sample size	61	62	66	69	68

Table A.1: Statistical metrics results at download operation on D3S with 8 seeds on a monitored network.

Looking at Table A.1 the observed download throughput is 0,1 bytes per seconds (TPS) in the 1 Byte file size. As can be seen, the small files remain at low values. But if we look at the big files, in particular the 100 MB file, it can achieve about 1,23 MB/S a value much higher than S3 (0,92 MB/S).

The data was processed and charted in histogram charts which represents the range rates of GET requests (Figure A.1) like the others experiments reported in the document.

The Figure A.2 represent the cumulative distribution function for the previous histograms. Observing the figure is denoted that the performance grow with the file size.

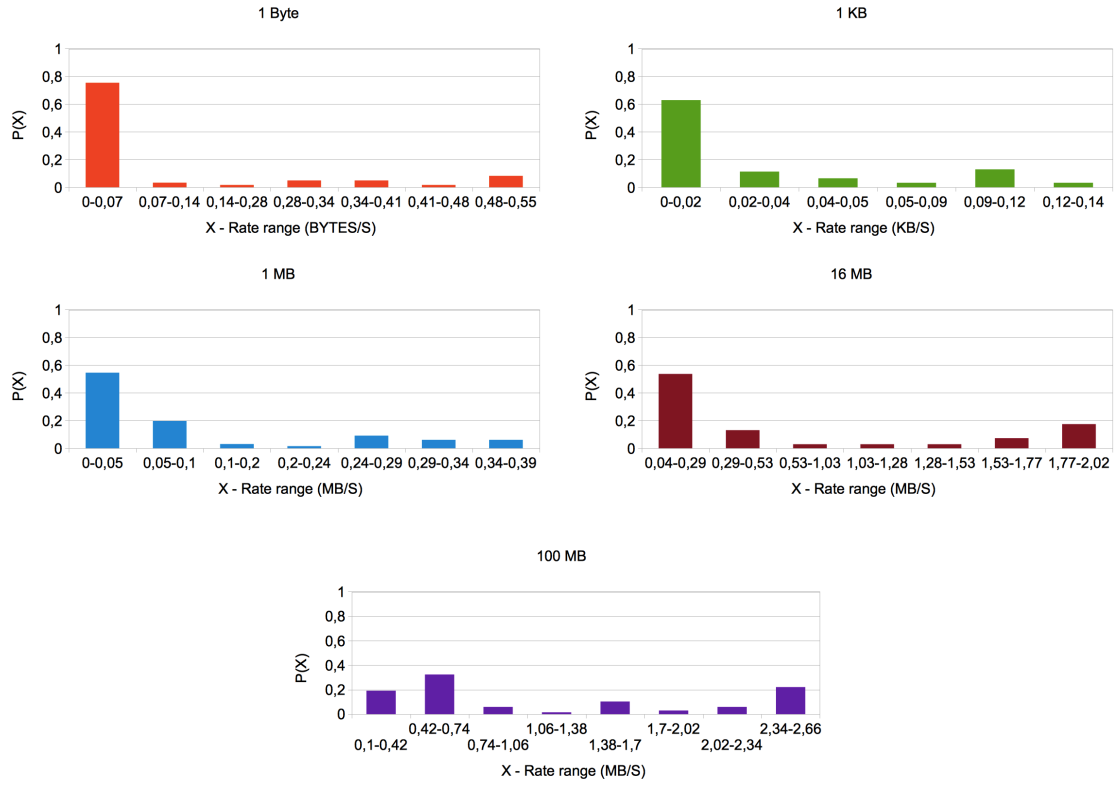

Figure A.1: Rate range probabilities of different files at download operation on D3S with 8 seeds on a monitored network.

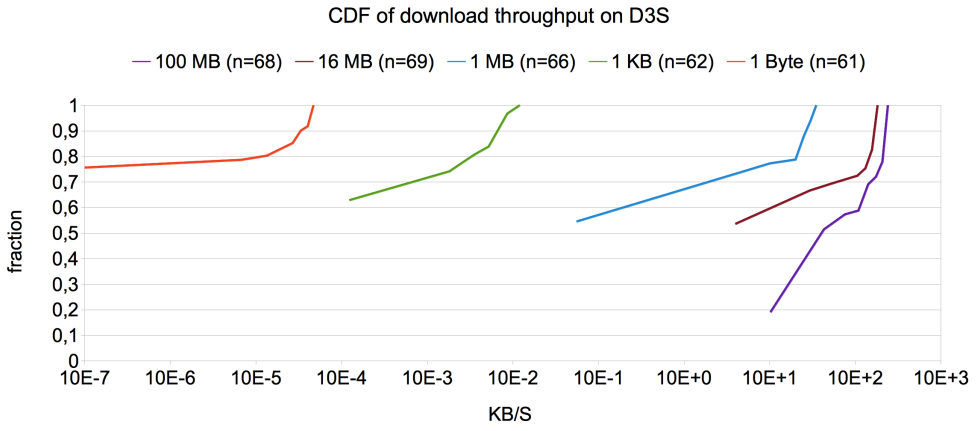

Figure A.2: CDF of download throughput of the D3S service with 8 seeds on a monitored network.

In order to compare the D3S in a monitored network to the S3 service was plotted the Figure A.3. This figure has a logarithm scale for enable the small files speed visualization. The S3 gains face to the D3S in the small files but loses when the file size grow.

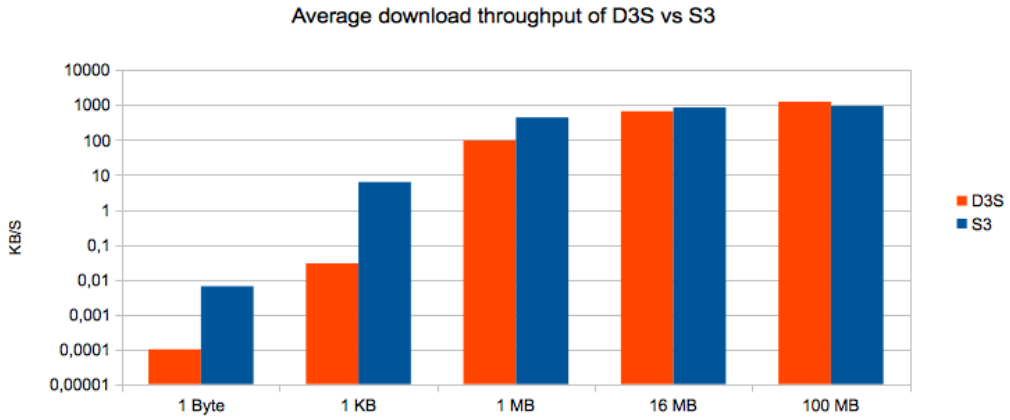

Figure A.3: D3S vs S3 – Average download throughput (logarithm scale) with different files on a monitored network.

The Figure A.4 shows better that difference where the D3S system gains in the huge sized file case. The 8 seeders are the minimal seeds number in order to achieve better results with the 100 MB file.

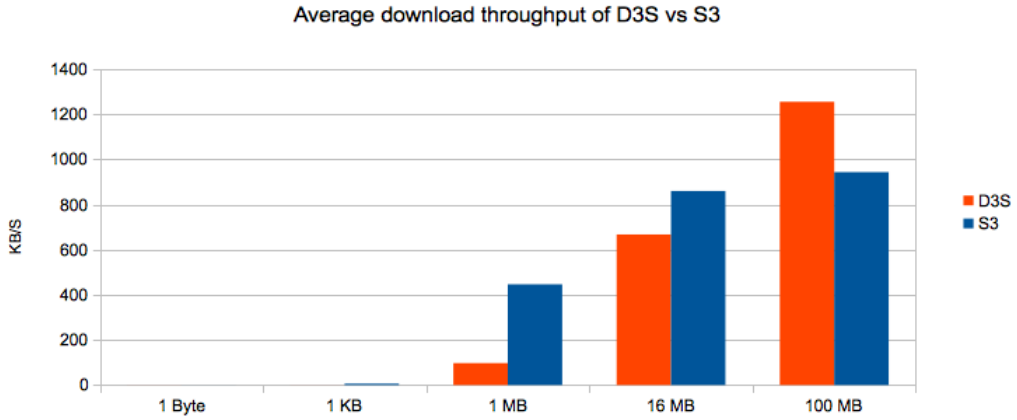

Figure A.4: D3S vs S3 – Average download throughput with different files on a monitored network.