USTO.RE: A Private Cloud Storage Software System

Frederico Durão[1], Rodrigo Assad[2], Anderson Fonseca[3], José Fernando[3,4], Vinícius Garcia[3], and Fernando Trinta[5]

[1] Federal University of Bahia, Computer Science Department
Av. Adhemar de Barros, Salvador - Bahia, Brazil
freddurao@dcc.ufba.br
[2] Federal Rural University of Pernambuco
Recife - Pernambuco, Brazil
assad@deinfo.br,assad@usto.re
[3] Federal University of Pernambuco
Recife - Pernambuco Brazil
{afs8,jfsc,vcg}@cin.ufpe.br
[4] University Center of João Pessoa - UNIPÊ
João Pessoa - Paraiba, Brazil
gentio@gmail.com
[5] Federal University of Ceará
Fortaleza - Ceará, Brazil
fernando.trinta@lia.ufc.br

Abstract. Cloud computing is a computing model where hardware, platforms and software are seen as services; viz. Infrastructure as a Service, Platform as a Service, and Software as a Service, respectively. Data as a Service (DaaS) is based on the concept that the product, data in this case, can be provided on demand to the user, regardless of geographic or organizational separation between provider and consumer. DaaS applications are for the most part based on excessive data replication in order to guarantee data availability, which means excessive costs in hardware investments. This white paper presents the specification, implementation and evaluation of a system called **USTO.RE** which aims to be an effective and low-cost alternative for storing data, thereby mitigating the problem of excessive data replication and thus allows itself to be considered a reliable platform from the perspective of data availability. Evaluation scenarios and the results achieved in our experiments to evaluate the system as well as possible lines for future development will be presented.

1 Introduction

Recent CISCO report[1] on predictions about mobile data traffic, show that they currently achieve a flow of approximately 0.6 *exabytes* of mobile data per month.

[1] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2011 – 2016, URL: http://bit.ly/x5V50B, last access on 05/03/2012.

F. Daniel, P. Dolog, and Q. Li (Eds.): ICWE 2013, LNCS 7977, pp. 452–466, 2013.

This report states that the flow of mobile data will be multiplied by a factor larger than 10 by 2016, based on the fact that the speeds will be multiplied by similar values. The considerable increase in the amount of information produced by users associated with the need to access data in a ubiquitous manner led to the emergence of data storage systems in the cloud [8].

Cloud Storage Systems appear quite attractive to home users, by providing them with the ability to access, retrieve and store their files from anywhere at anytime. Studies show that in 2010, the amount of data produced was greater than the storage capacity [2]. Thus, we can infer that the tendency is for this disparity to grow increasingly, corroborating the report submitted by CISCO. The applications of cloud storage based on the principles of Cloud Computing, in which users pay for what they consume and can access data from anywhere with growth in storage resources, are done on demand. Because of this relationship, a service model commonly used for the applications of cloud storage is the cloud storage of Data as a Service (DaaS).

There are a number of platforms in the cloud data storage [8], most of which have common features: i) the need to assemble a dedicated infrastructure to ensure the availability of data when the user requests access; or ii) the lack of reliability from the standpoint of ensuring the availability of data, which can be clearly identified from the analysis of contracts of service providers.

In the context of these issues, the project **USTO.RE** proposes the reduction of costs associated with the need for a dedicated infrastructure of cloud storage systems through an approach based on the Peer to Peer (P2P) network. This approach significantly reduces the *CAPEX*, which encompasses capital expenditure or investments on acquisition for companies to build their own data storage environment. The main advantage of **USTO.RE** is that it opportunistically benefits from opportunistic storage resources and idle disks of computers in the existing infrastructure of the company.

The remainder of this paper describes the architecture and results of tests conducted with **USTO.RE**. Section 2 reviews other works related to our proposal, and Section 3 introduces the system and provides an overview of its architecture. Section 4 describes the project use case, testing scenarios and their achievements. Finally, Section 5 draws the main conclusions and outlines future lines of investigation.

2 Related Work

There are several existing solutions for data storage in the cloud DaaS model. In general, these solutions are proprietary and closed, which therefore prevents a more detailed technical analysis on each proposal. Looking at some of the major existing solutions, such as *Amazon S3*, *Megastore*, *MSFSS* and *Hadoop Distributed File System* (HDFS), we note that they invariably utilize all a common strategy based on replication of data across multiple servers, where the number of servers involved in replication, ranges from 3 to 7 depending on

the solution. This forced replication is done because there is no automation in the processes of replication that can guarantee 99.9999999% availability [8].

Amazon S3 (*Simple Storage Service*) [2] is the storage system behind many of the services of Amazon.com known as Dynamo [8], which recently had its SQL-database launched under the name DynamoDB [8]. The DynamoDB has a backup policy where at least three copies of the same data are made, and two are made in one and the same zone on a third outer zone, so as to increase the availability of the information. According to 2009 data, the system stored about 40 billion files of 400,000 customers. Their challenges include ensuring availability and fault management.

Megastore, is a system developed with a focus on online interactive services [3]. It was developed and is used by Google for a long time and handles more than 3 billion written and 20 billion read transactions daily. It also stores a value close to a petabytes of raw data in their datacenters spread globally.

MSFSS is a distributed file system highly scalable and flexible, designed to store a large amount of small files [18]. Its architecture is divided into three main components: *single master*, *storage nodes* and *metadata servers*. Every file stored in the system receives a 128-bit identifier (*FID - File ID*) generated after its storage. These files are stored in the local file system of each node storage. Further, MSFSS supports data replication, which ensures consistency between replicas and performs rapid synchronization to identify obsolete ones. The FSI (*File System Interface*) library provides access to the system for external clients and allocates a large amount of FIDs processes in *batch* and leaves stored in local memory for later use [18]. By default, MSFSS replicates the files twice although this value can be configured to ensure greater availability and reliability. During the reading process, any replica can be considered, on the other hand, in the process of writing, all replicas must be updated automatically. To minimize latency, the data is stored near the user and the replicas close to each other. It separates groups by regions and creates 3 to 5 replicas for data centers.

Wuala[2] is an online cloud file storage that unlike the majority of online file storage services like Dropbox and Box.com, which offer paid plans only as a means to expand storage. Wuala offers the option to gain additional online storage in exchange for some of the user's unused local disk space to commit to the network. Similarly to our approach, uploads are encrypted automatically, anything that is sent to backup is given a high-grade file encryption to prevent hackers and other nefarious evildoers from messing with the cache of files. Also similarly as our approach, Wuala offers a simple, yet capable, mobile app for Android and iOS devices, making file access easy and accessible from anywhere. An interesting feature about Wuala is the possibility of adjusting the bandwidth so that one can easily cap the amount of upstream and downstream net traffic that available to Wuala. Unlike our approach, after installing Wuala, it will mount WualaDrive on the computer as a network drive, where one can directly access all files and even save files directly there. A key feature is the possibility of building groups and share files with distinct collections of users.

[2] `www.wuala.com/en`

Worth mentioning that every uploaded files in a group will be counted against storage quota for every group members. Tagging mechanism is allowed. Unlike our approach Wuala doesn't keep files versioned with date. Instead they will keep your files for up to 10 versions.

Symform[3] is a also an online cloud backup service that encrypts, shreds, and globally distributes data. Similarly as Wuala, Symform's customers join the Symform network by contributing excess local drive space and, in exchange, receive free cloud backup. Symform is basically a folder synchronization product. One can only select folders that wants to synchronize with the cloud. There is not option of selecting individual files or file types for backup. Once a folder is selected, all sub-folders in that tree are included. Similarly as our approach, Symform encrypts the data and chops it up into fragments. The ideal size for a folder is 64 MB and that block of data is broken into 64 1 MB fragments. To that data, 32 parity fragments are added and all 96 fragments are distributed to 96 member devices (contribution nodes) out in the cloud. The algorithm for adding parity is actually a RAID algorithm. Every piece of data sent to backup is monitored by a Cloud Control metadata where Symform keeps track of where each fragment is sent. Symform constantly monitors the performance and availability of contribution nodes. If a member's device becomes unavailable, Symform sends an email notifying them, and can recreate the data from the unavailable device (from other data and parity) to send to another contribution node. Data is geographically dispersed, which increases data security. According to Symform, member contribution nodes are located in 150 countries.

Freenet[4] is a free P2P software designed for anonymous file sharing. Indeed, the Freednet is targeted at those who want to exercise free speech without fear of censorship or retribution. Unlike most of file sharing discussed in this paper is the single one which allows one to publish websites and take part in online bulletin boards obviously only accessible to those who use the software. Unlike our approach, the Freenet network is decentralized without any central hub. The shared files are stored encrypted in different computers around the world. The size of the default folder where shared files are stored is determined by the user during installation and it can go from a few Megabytes up to dozens of Gigabytes. Because all the stored data is encrypted Freenet users do not know what they are sharing and have no saying on what is being shared, this allows for denial of knowledge. A particular feature is that files in Freenet are kept or deleted depending on popularity, if something isn't downloaded for some days, Freednet wil delete them.

AeroFS[5] is on software for file sharing on the Web and is nearly as seamless and simple Dropbox, but with the added security that comes from keeping data (and data transfers) confined to local computers rather than on someone else's server. The level of OS integration present and its ease of configuration make it a very promising "personal cloud" solution. As to the security, data is fully

[3] `http://www.symform.com/`
[4] `https://freenetproject.org/`
[5] `https://aerofs.com/`

encrypted before being transferred to other computers, preventing it from being intercepted and decoded in transit. Installing the AeroFS software creates a folder in the user profile, much like Dropbox does, and this can be changed from the default location as desired. Aside from the security advantages, AeroFS frees users from the limited amount of storage space provided by other cloud service vendors. The size of a "personal cloud" is limited only by the amount of disk space on your various computers. Limiting download and upload bandwidth is also possible, which is especially desirable if the syncing is occurring between multiple clients. The versioning system in AeroFS keep older versions stored as long as there is sufficient free local disk space; if one's disk gets too full, it will begin silently deleting the very oldest items to make more room. AeroFS' versioning system keeps old versions until it runs out of space, and then begins deleting the oldest copies of files to make more room. Our approach does not implement this feature, rather we let the users take care of this own files. Conflict management is the a concern, file conflicts are not solved properly if problems occurs. And finally, the AeroFS's biggest shortcoming, which is lack of access from Web browsers or mobile devices. Dropbox (and, for that matter, Google Drive, SugarSync, Box.com, and most of the other major players in this field) offers apps for iOS and Android as well as a robust Web client that can be used to access your files while on public computers.

The HDFS is a file system used by Hadoop [14] and its related projects. Hadoop is a framework for analysis and processing of large amounts of data using *MapReduce* [7]. One of the main features of Hadoop is the partitioning of data and computation thereof using thousands of hosts. For instance, the *cluster* Hadoop at Yahoo! reached 25,000 servers (with cluster up to 3,500 servers) and stored 25 *Petabytes* of data [14].

The abovementioned works have in common excessive replication of information. This fact has to do with the need to ensure high data availability, increase reliability, and performance of information retrieval. However, the use of replicas does not only bring benefits. It creates an extra data traffic on the network, which can be so excessive to the point of becoming the main bottleneck for an application, and may generate a cost so high that the band can make the application unfeasible [17].

These solutions infer the need to purchase infrastructure in order to provide a dedicated service and to guarantee a replica of the data. In order to improve this scenario, the goal of **USTO.RE** is to allow for the creation of a cloud data storage using P2P technology that is based on the availability of each peer to dynamically create federations and set the amount of replications for the *chunk*[6] of each file. This approach is allowed in environments where peers have increased availability (low failure rate) with one minor replication, and in the case of a more dynamic environment as company Intranet, it has further replication. In the next section, our proposal will be explained in detail.

[6] Chunk is a piece of information, i.e. a "small" part of a file stored. Files are divided into chunks, which in turn are replicated storage points in cloud infrastructure.

3 The USTO.RE System

The architecture of **USTO.RE** was specified with the target of achieving a set of quality attributes peculiar to distributed storage systems that met the main benefits offered by P2P applications, namely

Scalability: Given the possibility of exploring hardware resources of a large number of (hosts) machines connected to the network, this is done mainly through the rational use of idle resources in large corporations..

Optimization Iterations (messaging): The distance between the peers interacting in the system has an impact on overall performance in the latency of individual interactions, so the load network traffic also suffers from a negative impact on this latency. In this context, the choice of using the strategy is aimed at federation grouping related peers and to reduce this latency.

Availability: P2P systems are based on free computers that join the system at any time. Furthermore, the connections are not managed by the system or some authority that ensures connectivity and quality of service. In this context, a strategy for ensuring service availability should be implemented considering the basic characteristics of a storage system with high availability and the restrictions imposed on a P2P network.

Data Security: When it comes to data storage, protection and security policies should be effectively adopted to ensure user privacy and system data consistency.

Figure 1 overviews the **USTO.RE** architecture, its major components, dependencies and relationships. The architecture comprises a set of five components structured in three layers. They are: i) *Super Peers Rendezvous Relay* or simply *Super Peers*, ii) *Servers* iii) *proxies*, iv) Relational SQL and Non-Sql Databases and v) Simple Peers. These components have different functions and interact as a decentralized distributed and hybrid system, similar to a P2P network, where each node performs both functions as a server and clients. The components are grouped dynamically as associations of data, where the groups are assembled so as to minimize the messaging system.

The organization of this system architecture enables a multi-layered distribution of processing, since the components are physically distributed. However, because it is a hybrid P2P architecture that is structured and multi-layered, the system has a horizontal distribution. In this horizontal distribution in a P2P network, a client or a server can be physically divided into pieces that are logically equivalent, where each operates on its own portion of the data that provides a balanced load. In the following, we present the components of the **USTO.RE** architecture.

3.1 Super Peers

The *Super Peers* act as reference points for other components of the architecture, being the gateway to the participation of servers, proxies and simple peers. The role of a *Super Peer* is to set up federations when each data peer requests the

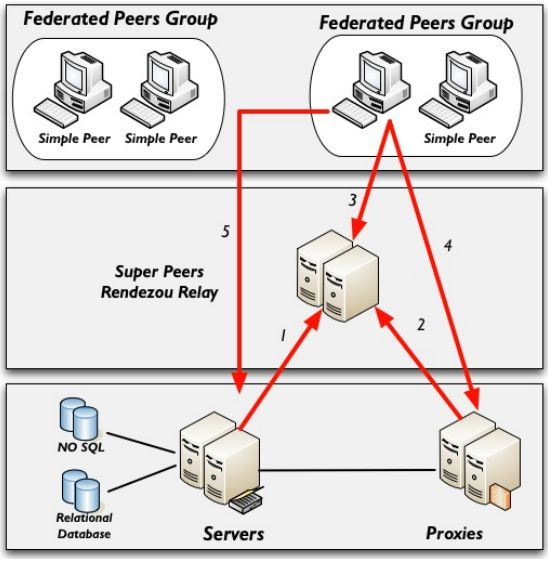

Fig. 1. USTO.RE Architecture

network connection. For this, *Super Peers* must have its location previously known by all other peers through a pre-configuration. Consequently, they are the first components to be initialized to the correct functioning of **USTO.RE**. Also as a result, a *Super Peer* keeps information on all servers, *Proxies* and *Simple Peers*, grouping them dynamically according to the profile of each peer, to be explained below.

It is also the role of this type of *peer*, to dynamically choose peers and federations of servers based on a proximity algorithm [9]. The federation grouping allows elastic growth and ensures system scalability, because there is a limit to the number of associations that can be created. By definition, elastic growth is the characteristic of the system grow or decrease, in terms of capacity and resource consumption of dynamic and non-intrusive way. The peers communicate with the network through the P2P JXTA protocol [16] and can optionally offer a service interface REST [16] to allow for interoperability with other applications.

3.2 Servers

The *Server Peers* offer a list of services and must be executed immediately after the startup of *Super Peers* (Step 1 in Figure 1). *Super Peers* establish a synchronization scheme, thus ensuring that the list of servers in each of them would be updated as the input or output of a *Sever Peer*.

The definition of *peers* with specific functionality in the network differs from some proposals for P2P systems, where which each *peer* should be able to play all the roles, thus promoting the idea of a DHT (*Distributed Hash Table*) [13,11]. However, the implementation using a DHT in its essence is quite costly and

difficult to scale [12]. Therefore, **USTO.RE** adopted the creation of hierarchical levels that implement well-defined services and that can grow horizontally. Among the available services via the servers, we can mention.

1. **Authentication:** used for each peer to get authenticated;
2. **Availability:** allows one to check the availability of each peer;
3. **Chunk:** used to monitor the chunks storage;
4. **Error:** allows servers to monitor eventual errors;
5. **Output Control:** controls the voluntary withdrawal of a peer when it disconnects from the network voluntarily;
6. **Management Directories:** used for storage and retrieval of entire directories;
7. **Manages File:** used for file storage and retrieval;
8. **Search Peers:** search for a set of peers that comply with Service-Level Agreement (SLA) for file storage;
9. **tree** Directories: used to preview entire directories;
10. **Access Security:** controls the access permission to the chunks;
11. **Trace:** maintains a list of users and files being accessed when a file is requested to be recovered.

As shown in Figure 1, the *Server Peers* access two types of database to maintain consistency of the system. A traditional relational database contains data from users of the system, and a non-SQL database [5] which allows horizontal growth and faster recovery of information related to the files and *chunks* saved. The choice of this separation is given by the issues related to system performance, since with increasing volume and file *chunks* saved the management system database try to become a bottleneck point, and using a distributed system enables natively become a viable and scalable solution [5].

All information regarding authentication and the SLA peers are saved in a relational database because of the relational integrity assurance provided by this type of database. Already, the service of *FileTracker*, which allows for the identification of which peer has pieces of the file to be recovered, is used in a SQL database that allows for its horizontal growth. Instances of banks, whether they be relational or non-sql, can be shared between more than one server. A *Server Peer* can provide one or more network services; therefore, the same as creating federations of data, one can start peer and server proxies with increasing demand scalability and resilience of the system.

3.3 Proxy

After initialization of *Super Peers* and *Server Peers*, the third component that needs to be executed is the *Proxy*. Each proxy acts as a catalog, a location service for services running on different **USTO.RE** servers. A *Proxy* when announcing a *Super Peer* (Step 2 in Figure 1), receives the list of registered servers. In addition, a *Proxy* gets the information of what services are available on each server. Thus, when a peer requests information about a particular service, a *Proxy* provide a

reference to a server that meets this requirement. Thus, a *Proxy* establishes a bridge between service consumers, typically the *Simple Peers*, and providers of a service, in this case, the *Server Peers*.

3.4 Simple Peers

Simple Peers are responsible for storing the files chunks. In fact, these machines provide storage space to be shared among multiple users. Each *Simple Peer* has a profile that defines its availability in the network. This availability is related to the time period in which the peer is available to share data. As an example, a peer that is in a corporate Intranet can have on one's profile: availability assigned at "8:00 to 12:00 and 14:00 to 18:00". Thus, when a *Simple Peer* connects to the network, it receives from the *Super Peer* the list of proxies available in the network (Step 3 in Figure 1). From this list, the proxy searches for a specific service and gets the reference over which servers proffer a particular service (Step 4 in Figure 1). A *Server Peer* is chosen randomly, and *Simple Peers* request the desired service (Step 5 in Figure 1). If a service is not met for any reason, such as a timeout, the proxy can provide a new *Server Peer* to the *Simple Peer*.

Each peer has a REST service interface [16] that allows user authentication, storage, retrieval and deletion of data saved. This feature presents a key advantage in terms of the possibility of harmonizing the system with other existing interfaces, such as Amazon's S3. In the current architecture, data storage service can be modified to work with other alternatives (i.e. Megastore, MSFSS or S3).

To ensure each balanced chunk is scattered in the network, each peer must periodically report their current state in order to maintain the SLA to date. Figure 2 **(a)** shows the *peer* (PL $_1$) workflow from the moment he announces his profile to the established communication with other peers through the server (PS $_1$). Periodically every peer sends a message to servers as *"keep-alive"* stating that he is online. This way, the server knows that the *peer* is complying with the agreed profile (SLA) and becomes eligible for receiving *chunks* at the specified time. Still, every *peer* checks with other existing peers whether its own chunks are replicated in the minimum amount of *peers* obeying the criterion of availability required [9]. Otherwise, it replicates the chunk(s) in other available peer. When the *peer* owner of this *chunk* re-connects to the network, it will be notified that there is *chunk* excess, thereby excluding it.

Figure 2 **(b)** shows the workflow between peers and servers from the login of the peer in the network until the file is stored. After the *peer* (PL_1) is connected to the P2P network$_1$, the Super Peer indicates a Server Peer to authenticate it. Once authentication succeeds, the process of identifying pairs to form federations takes place. With federation (group of *peers*) established, the system is able to receive files. By receiving a file (*"arq1.zip"*), the *peer* PL_1 informs the need to store it in the system, it is made to a segmentation of the file (in *chunks*) and these segments are sent to be saved in the P2P network$_1$. Then, to make the save, there is a routine analysis to measure the reliability of the state peers and hence the availability of segments of the network file, and if there is a combination of peers that meets the SLA for storage, the segments of the file are sent to these

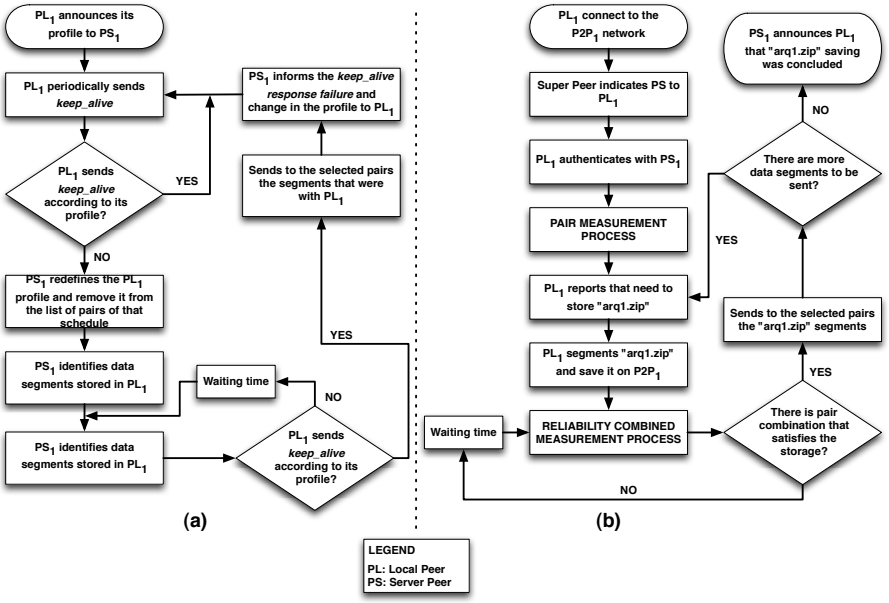

Fig. 2. a) Peer Workflow b) Server Proxy Workflow

peers. If there are no more segments of files to be saved, the PS_1 peer server communicates to the PL_1 *peer*, which requested the file saving, that it was saved successfully.

4 Experimental Evaluation

For assessing the project with regard to performance and scalability, a pilot project was planned and carried out addressing three different scenarios. The methodology *Goal - Question - Metric* (GQM) [4] guided the evaluation by establishing the purpose of the study, the questions to be answered, and the metrics used to interpret the answers.

Three scenarios were assessed using **USTO.RE**. The first two were designed to test the feasibility of peer selection algorithm and performance of the proposed solution, thus validating the system in a corporate Intranet. The third scenario validation was performed with the system implemented. In the latter case, there is the scalability of the solution by adding peers gradually until it reaches 40 P2P clients with maximum 5Gb storage, and two servers providing interfaces for accessing REST nodes.

4.1 Scenario 1

As described earlier, this scenario aims at analyzing the algorithm of peer selection in order to assess its performance within a corporate Intranet. The question

associated with this goal is: *How efficient is the algorithm for the selection of peers?* In particular, we consider efficiency as the latency of the algorithm to select peers. In the following, we list the configurations for the algorithm of availability as well as the respective metrics to be collected:

- A file was divided into 100 chunks to be distributed along 376 possible machines;
- As to the number of executions, 100 was chosen because it is large enough to display the runtime of the algorithm in practice. This number is not required to be determined by any empirical method since the asymptotic behavior of the algorithm is known, at least according to the parameters of Cormen [6].
- The failure rates were measured using the empirical model Garden [1] in a 296-day experiment of continuous monitoring software by Squid [15];
- The minimum number of machines on which the configurable piece would necessarily have to be placed was 5;
- We used the following profile of availability according to the equation shown below, where `t` represents the time:

$$f(t) = \begin{cases} \text{Reliability} \\ 0, se\ t = \{0,1,2,3,4,5,6,7,18,19,20,21,22,23\} \\ 0.50, \text{ if } t = \{12,13\} \\ 0.99 \text{ else} \end{cases}$$

The result of the executions was arranged in three separate sheets, measuring: i) runtime; ii) mean number of machines where the same chunk was sent, and iii) average chunks per machine. As a result, the algorithm has proven to be quite satisfactory in terms of execution time, achieving the arithmetic mean time of 1.92 ms with a maximum time of 34 ms, and a minimum of less than 1 ms. As will be seen in the results of Scenario 3, this time is the same as the time spent to access files on the *Windows Netbios* network.

4.2 Scenario 2

Following the GQM methodology, the goal for this scenario is to analyze the **USTO.RE** efficiency in terms of user within a corporate Intranet and the related question is: *How fast is the USTO.RE in use?*. We consider speed-related performance of **USTO.RE** as the basic system features.

As a natural behavior, the **USTO.RE** performance depends on the amount of messages exchanged by its internal components. This amount of messages in turn is directly related to the size of chunks and the size of queues of chunks that form these messages. In this context, the performance was evaluated in terms of variation in the size of chunks and the queue, thereby examining the impact of these variables on time data transmission. The objective of this test is therefore to identify what is needed to perform specific settings when the system is running

in LAN environments where throughput is high and WAN environments where throughput is lower and more varied.

For this particular test execution, 20 machines were utilized, and all had the same configuration, Pentium IV with 2GB of RAM and a 100Mbps network card, i.e. old desktops that are compliant with cloud computing features and efficient use of computing resources. Fifteen (15) out of 20 were utilized to send data whereas five (5) were used for storage with 10TB of storage space. The machines sent 1Gb of data, as follows:

- 3 machines sending 2000 files of 500 Kb;
- 6 machines sending 500 files of 5 Mb;
- 3 machines sending 50 files of 50 Mb;
- 3 machines sending 5 files of 200 Mb.

Table 1 presents the results obtained for the data transmission system in terms of Chunks and Queue. In the leftmost column are the values of the chunk size and the amount of the queue in terms of chunks. That is, for the first case, each system message has one queue comprising 10 chunks of 128Kb.

Table 1. Average Delivery Time of Chunks per Queue

Chunks/Queue	**Time**	**Chunks/Queue**	**Time**
128/10	00:03:20	32/10	00:12:03
128/8	00:03:08	32/8	00:14:53
128/6	00:04:54	32/6	00:18:50
128/4	00:06:22	32/4	00:22:59
128/2	00:13:15	32/2	00:59:19
64/10	00:05:46	16/10	00:23:17
64/8	00:07:32	16/8	00:30:00
64/6	00:10:15	16/6	00:32:15
64/4	00:12:02	16/4	00:47:49
64/2	00:30:31	16/2	02:03:10

As can be seen, there is a significant impact on system performance when there are variations in the parameters analyzed. This impact is higher when there is variation in the size of the chunk. This occurs because there is an augment in the number of messages to be sent in order for a file to be saved. This same rationale applies to the variation of the queue, but is not as significant for the final performance of the system.

Considering that the queue size was not a significant factor in the performance of the system, we opted to use chunks of 128Kb and queues in size 10 for LAN and WAN environments.

In this scenario, we also examined the issues related to the performance of the machines. It was observed that the machines receiving data reached their full storage capacity, i.e. 100% CPU usage, 2GB of memory consumption and

20Mbps of traffic in this situation. Comparing the transmission rate obtained with the results reported by *Google File System (GFS)* in the Session *Micro Test*[10], we can observe that **USTO.RE** has reached about 40% of the processing capacity of the GFS. Taking into account the difference between the real-world and test environments, the **USTO.RE** used an *HUB* and not a *switch*, as well as cheaper network cards that limit the transmission capacity. In addition, the chunks were 128KB instead of 4MB as in SFM. However, these results indicate that it is possible to overcome the results of GFS. With regard to the machines that were spreading data, they consumed little recourse, i.e. 128MB of RAM, nearly half the CPU consumption.

4.3 Scenario 3

For the latter scenario, the scalability of the system is tested. The goal therefore is to evaluate **USTO.RE** potential from the user-centric standpoint within a corporate Intranet, and the related question is: *How functional is the USTO.RE to the user?* We consider the functional potential through *USTO.RE* scalability and performance in relation to the environment that exists today on corporate Intranets for storing and sharing data.

In the latter assessment, we utilized a performatic access control tool to validate the **USTO.RE** scalability and performance of file transfer. In the test scenario the goal was to assess the system's growth capacity, both vertically and horizontally. For this, we used two servers providing REST service interfaces that connected to the P2P network, performed the download the file and stored in the P2P network. During the test, as a) interfaces services began to become bottleneck point, others were added as needed; and b) as more nodes were needed for the P2P network, they also were added thus allowing the growth of the system.

In this test scenario, all peers were 100% available, i.e. 24 hour, making each *chunk* replicated by 2-3 machines according to the algorithm described in [9].

As a result, the tests for reading files (with an average size of 11Mb), the waiting time to get it was on average 28 seconds (to measure this average, each file was restored three times and the average time obtained corresponds to this result). As to downloading files via the REST service interface, the average time was 31 seconds, thus demonstrating the efficiency of the proposed architecture with the service Filetracker (measurement performed with the same procedure as above). These test results showed that in comparison to other network storage solutions, the system can produce an acceptable performance. Figure displays the screen with some system files saved.

5 Conclusion

This paper presented **USTO.RE**, a cloud storage system at a low cost with high reliability. This tool consists of peers in a P2P network and an algorithm that allows to dynamically calculate how many nodes a *chunk* should be replicated so

that when files are requested to be restored, the system ensures its availability. Evaluations were performed in order to validate the algorithm that supports the proposal, as well as the feasibility of the solution, achieving satisfactory results. In addition, new test scenarios are being validated to investigate the solution's degree of scalability. Future works include the calculation of optimal storage size per peer in order to increase storage efficiency. Further, we also aim at researching the utilization or adaptation of an algorithm that allows to group more efficiently the peers in federations, thus improving the storage efficiency as well as its replication.

Acknowledgments. Authors would like to thank the University Center of João Pessoa - UNIPÊ for financing the publication of this article.

References

1. Abd-El-barr, M.: Design and Analysis of Reliable and Fault-tolerant Computer Systems. Imperial College Press, London (2006)
2. Amazon. Amazon Simple Storage Service (Amazon S3) (March 2012), `http://aws.amazon.com/pt/s3/` (last access March 5, 2012)
3. Baker, J., Bond, C., Corbett, J., Furman, J.J., Khorlin, A., Larson, J., Leon, J.-M., Li, Y., Lloyd, A., Yushprakh, V.: Megastore: Providing scalable, highly available storage for interactive services. In: CIDR 2011, pp. 223–234 (2011)
4. Basili, V.R., Caldiera, G., Rombach, D.: The Goal Question Metrics Approach, vol. I, pp. 528–532. John Wiley & Sons (February 1994)
5. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: a distributed storage system for structured data. In: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation, vol. 7, p. 15. USENIX Association (2006)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press (2009)
7. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM 51, 107–113 (2008)
8. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. SIGOPS Oper. Syst. Rev. 41, 205–220 (2007)
9. Duarte, M.: Um algoritmo de disponibilidade em sistemas de backup distribuído seguro usando a plataforma peer-to-peer. Dissertação de mestrado, Centro de Informática, Universidade Federal de Pernambuco, Recife-PE, Brazil (2010)
10. Ghemawat, S., Gobioff, H., Leung, S.-T.: The google file system. SIGOPS Oper. Syst. Rev. 37(5), 29–43 (2003)
11. Loest, S.R., Madruga, M.C., Maziero, C.A., Lung, L.C.: Backupit: An intrusion-tolerant cooperative backup system. In: Proceedings of the 2009 Eigth IEEE/ACIS International Conference on Computer and Information Science, pp. 724–729. IEEE Computer Society (2009)
12. Nocentini, C., Crescenzi, P., Lanzi, L.: Performance evaluation of a chord-based jxta implementation. In: Proceedings of the 2009 First International Conference on Advances in P2P Systems, pp. 7–12. IEEE Computer Society (2009)

13. Oliveira, M.: Ourbackup: A p2p backup solution based on social networks. M.sc. dissertation, Universidade Federal de Campina Grande, Campina Grande – PB, Brazil (2007)
14. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–10. IEEE Computer Society (2010)
15. Squid. Squid: Optimising web delivery (2012), `http://www.squid-cache.org/` (last access March 5, 2012)
16. Webber, J., Parastatidis, S., Robinson, I.: REST in Practice: Hypermedia and Systems Architecture. O'Reilly Media (2010)
17. Yang, Q., Xiao, W., Ren, J.: Prins: Optimizing performance of reliable internet storages. In: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, p. 32. IEEE Computer Society (2006)
18. Yu, L., Chen, G., Wang, W., Dong, J.: Msfss: A storage system for mass small files. In: Shen, W., Yang, Y., Yong, J., Hawryszkiewycz, I., Lin, Z., Barthes, J.-P.A., Maher, M.L., Hao, Q., Tran, M.H. (eds.) 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Los Alamitos, CA, USA, pp. 1087–1092. IEEE Computer Society Press (April 2007)