# D3S – A Distributed Storage Service

Rui Pedro Lopes and Pedro Sernadela

**Abstract** The Internet growth allowed an explosion of service provision in the cloud. The cloud paradigm dictates the users' information migration from the desktop into the network allowing access everywhere, anytime. 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. This paper describes an implementation of an S3 compatible cloud storage service based on peer-to-peer networks, in particular, through the BitTorrent protocol. This approach allows taking advantage of the intrinsic features of this kind of networks, in particular the possibility for simultaneous downloading of pieces from different locations and the fault tolerance.

**Key words:** Storage, BitTorrent, Amazon S3, Cloud Computing, P2P

## 1 Introduction

Cloud computing has emerged as an important paradigm for managing and delivering services over the Internet. The term "Cloud" comes from the data center hardware and software association and is usually based on a model of payment associated with the use of resources. This on-demand, pay-as-you-go model creates a flexible and cost-effective means for using and accessing distributed computing resources [12, 1, 10].

Several providers have appeared, such as Amazon, Hewlett-Packard, IBM, Google, Microsoft, Rackspace, Salesforce, on the promise to increase revenue by optimizing their existing IT infrastructure and personal. According to the abstraction layer

Rui Pedro Lopes
Polytechnic Institute of Bragana, Bragana - Portugal, e-mail: `rlopes@ipb.pt`

Pedro Sernadela
Polytechnic Institute of Bragana, Bragana - Portugal, e-mail: `psernadela@ipb.pt`

as well as to the type of service, users' are allowed to access applications on the cloud (SaaS), to deploy applications to the cloud without the added complexity of software and hardware management (PaaS), and access to "low level" storage and computing capabilities (IaaS) [9].

This paradigm suffers, essentially, from two drawbacks. First, the centralized nature is vulnerable to single point of failure, originated from several causes, such as fire, lightning storms, hardware failure, and others [3, 17, 11]. Second, it is necessary a considerable investment on the infrastructure and a lot of experience in Information Technologies (IT). Industry leaders like Amazon and Google take advantage on the fact that they already possess large infrastructures and knowledge to fuel their business (Amazon S3 and Google Drive respectively). However, guarantying 99.99% availability requires a huge investment that usually companies resist to make.

This paper describes a first approach to address centralization related problems in cloud services, specifically on storage IaaS clouds. We will use Amazon S3 as base, instantiating an S3 service over a BitTorrent network. In this model, we store objects in BitTorrent swarms, as a means to achieve throughput, redundancy, scalability, capacity, and others. Many of these requirements are naturally available in peer-to-peer networks, although dependent on the number of peers that belong to a specific swarm, as well as on node availability. This implementation will be used in the future to assess availability and throughput.

## 2 Storage on the Cloud

Cloud computing is the result of the evolution of different technologies that allowed more processing capacity, virtualization of resources and the ability for data storage in the network. The combination of these factors led to a new business paradigm that responds to a set of problems, such as scalability, cost reduction, fast operation of applications and solutions [15].

According to the degree of service abstraction, it is possible to distinguish two different architectural models for clouds [4]. One is designed to allow the expansion of computing instances as a result of the increase on the demand (Software-as-a-Service and Platform-as-a-Service). The other model is designed to provide data and compute-intensive applications via scaling capacity – Infrastructure-as-a-Service. Regarding the deployment model, a cloud may 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) [4, 10, 14, 18].

Among the multitude of services available on the cloud, storage services are rather popular. Services such as Dropbox[1], Amazon S3[2], Google Drive[3] and others,

[1] `http://www.dropbox.com`
[2] `http://aws.amazon.com/s3`
[3] `https://drive.google.com`

provide an easy to use, flexible way to store information and easy access anytime, anywhere.

## *2.1 Amazon S3*

The Amazon Simple Storage Service (Amazon S3) is an online service that enables storing data in the cloud. This service is designed to provide 99.999999999% durability and 99.99% availability of objects over an year. Amazon S3 adopts the pay-per-use model: the prices are based on the data location (region) and the request count (GET, PUT, COPY, POST, and LIST) and the data transfers into and out of an Amazon S3 region.

Data is organized in two levels. At the top level, there is the concept of buckets, similar to folders, identified by unique global name. Buckets are needed to organize the Amazon S3 namespace and to identify the account responsible for storage and data transfer charges.

Each bucket can store a large number of objects, each comprising data (blob) and metadata. The data portion is opaque to Amazon S3 and has a limit of 5 terabytes. The metadata is a set of name-value pairs that describe the object (Content-Type, Date, ...), with a limit of 2 kilobytes. An object is uniquely identified within a bucket by a *key* (name) and a *version ID*. The key is the unique identifier for an object within a bucket. Users can create, modify and read objects in buckets, subject to access control restrictions.

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 (through server authentication and encryption).

These kind of service, typically provide a specific, SOAP or REST based, interface. The S3 REST API[4] uses a custom HTTP scheme based on a keyed-HMAC (Hash Message Authentication Code) for authentication. 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 make 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 [7]. In the Amazon S3 Request authentication this algorithm takes as input two byte-strings: a key and a message. The key corresponds 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, etc.. that will vary from request to request. The output of HMAC-SHA1 is also a byte string,

4 `http://docs.amazonwebservices.com/AmazonS3/latest/dev/RESTAuthentication.html`

called the digest. The final “Signature” request parameter is constructed by Base64 encoding this digest. 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.

The S3 architecture is designed to be programming language-neutral providing REST and a SOAP interfaces to store and retrieve objects. REST web services were developed largely as an alternative to some of the perceived drawbacks of SOAP-based web services [6]. With REST, standard HTTP requests are used to create, fetch, and delete buckets and objects. So, this interface works with standard HTTP headers and status codes (Table 1).

**Table 1** HTTP operations with URI pattern that can be performed in S3.

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

## 3 Distributed S3 Storage Service

The low-level usage patterns of Amazon S3 are, essentially, storing, updating and retrieving sequence of bytes through SOAP or REST WS, having in mind the durability and availability defined in the SLA. In this pattern, the access speed depends on the end-to-end throughput to where buckets and objects are stored (Amazon’s data centers).

We implemented the S3 REST services as a loop-back to a BitTorrent swarm, providing locally the same functionality as Amazon’s interface – the Distributed S3 Storage Service (D3S). From now on, we will identify the locally available gateway between S3 server-side interfaces and the BitTorrent swarm as a *D3S node*.

According to the usage patterns, the D3S node will receive requests through the REST WS and translate them to BitTorrent operations. Storing data will create a .torrent file and distribute it among a set of peers, that will define the swarm associated to these pieces of data. Updating data will require that the corresponding piece on the file to be updated. Retrieving data will require that the peer will download and make available the corresponding file by contacting a set of peers from the same swarm.

In a typical, centralized, client-server architecture, clients share a single server. In this case, as more clients join the system, fewer resources are available to each client, and if the server fails, the entire network fail as well.

Peer-to-peer networks are intrinsically distributed, where each peer behaves as a client and as a server simultaneously. In other words, "*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 share workspaces for collaboration): They are accessible by other peers directly, without passing intermediary entities*" [16].

## *3.1 Overall system architecture*

BitTorrent became the third generation protocol of P2P networks, following Napster and Gnutella, where the main difference between the previous generations is the creation of a new network for every set of files instead of trying to create one big network of files using servers. Nowadays, BitTorrent is actually one of the most popular protocols for transferring large files, with over 150 million active users[5].

Storing and organizing objects in D3S requires the creation and distribution of files in a set of BitTorrent peers. Each object, in the context of S3, will correspond, in our architecture, to a single file, which is registered in the swarm through a shared `.torrent`.

The overall architecture is depicted in Figure 1. The file that corresponds to an object is created in the local host, through the interface provided by a D3S node. The file is sliced up in pieces by the BitTorrent peer, to get small amounts of verifiable data from several peers at a time. Each time a piece is fully downloaded, it will be checked (using the SHA1 algorithm). The pieces are then spread in the swarm, following the "tit-for-tat" characteristic of BitTorrent [2].

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 has to be made available to the other peers – this operation is called seeding.

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

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. Those peers download it by connecting to the seed and/or other peers that has the file – acting as leechers.

---

[5] `http://www.bittorrent.com/intl/es/company/about/ces_2012_150m_users`

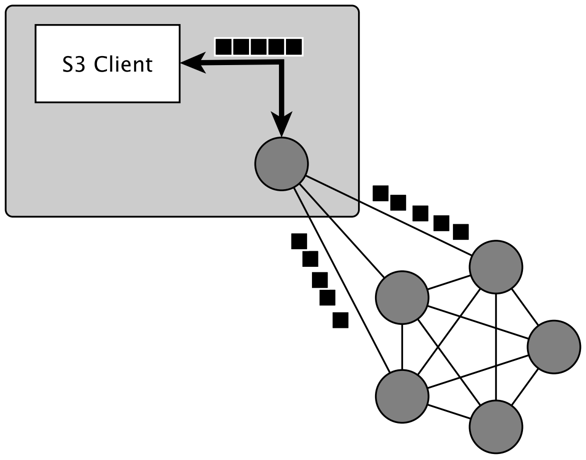

**Fig. 1** Overall architecture implemented.

Active peers report their state to the tracker and each time a peer obtains a new piece, it informs all the peers it is connected to. Interactions between peers are based on two principles [8]. First, the *choke algorithm* that encourages cooperation among peers and limits the number of peers a client is simultaneously sending data. Second, the *rarest first algorithm* that controls the pieces a client will actually request in the set of pieces currently available for download. The more peers join the swarm, the access speed and availability increases, contributing to the overall user experience [13].

## *3.2 Bucket and Object management*

The role of a bucket is to organize objects in a namespace. In our implementation, this translates to a file name prefix. In turn, each object is associated to a single file, that will be uploaded to the BitTorrent swarm. Adding a bucket corresponds to defining a prefix that will be used in all files belonging to that bucket. Adding and object corresponds to creating a file and making a `.torrent` available to other peers.

Removing a file corresponds to removing the file from all the peers and eliminating the `.torrent`. This is performed through the User Manager and, when signaled, the peer will eliminate them consequently.

D3S nodes contact the tracker when needed, to manage the overlay network. In our 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 `.torrent` files from the peers and associates them with the user that originated the request. `.torrent` files are generated when the user uploads a file to the S3 service. 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 locally. If the file is not available locally, it is downloaded from several replicas simultaneously, increasing the access speed.

As stated, we took advantage of RMI that provides the mechanism by which the server and the client communicate and pass information. Then we show the remote interface methods that can be invoked by the User Manager and the Torrent Manager to interact with the tracker.

This approach constitutes a bottleneck and a single point of failure. We intend, in the future, to move the User Manager to a Distributed Hash Table (DHT), to provide scalability and fault tolerance.

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) [5]. These characteristics are thus imported to our 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.

## 4 Conclusions and Future work

With the emergence of cloud computing, many cloud storage services have been offered and provisioned to customers. The pay-per-use model is, nowadays, an attractive solution to the traditional storage solutions. One well known example is Amazon S3, providing a transparent, fault tolerant and scalable storage service.

On the other hand, the BitTorrent protocol has made a big agitation on the file sharing community, and is nowadays one of the most used protocol in Internet. The advantages of this protocol are well know, particularly in terms of supporting distributed storage and efficient transfer of large files. The possibility to download (and upload) a file from several locations simultaneously allows excellent transfer speeds and the replicas allow for good availability.

Integrating an S3 compatible REST interface to a BitTorrent network allows using the scalability, openness and transfer speed in S3 compatible applications, without the need to rely on a single network connection to the provider.

In future work, we will investigate the relative performance of this approach to the cloud based service.

## References

1. Akioka, S., Muraoka, Y.: HPC Benchmarks on Amazon EC2. pp. 1029–1034. IEEE. DOI 10.1109/WAINA.2010.166
2. Cohen, B.: The bittorrent protocol specification. `http://www.bittorrent.org/beps/bep_0003.html`. Accessed on 14 january 2013
3. Cohen, R.: Cloud computing forecast: Cloudy with a chance of fail. `http://www.forbes.com/sites/reuvencohen/2012/07/02/cloud-computing-forecast-cloudy-with-a-chance-of-fail/`. Accessed on 2 january 2013
4. Dikaiakos, M.D., Katsaros, D., Mehra, P., Pallis, G., Vakali, A.: Cloud Computing: Distributed Internet Computing for IT and Scientific Research. IEEE Internet Computing (5), 10–13. DOI 10.1109/MIC.2009.103
5. Eng Keong Lua, Crowcroft, J., Pias, M., Sharma, R., Lim, S.: A survey and comparison of peer-to-peer overlay network schemes. IEEE Communications Surveys & Tutorials (2), 72–93. DOI 10.1109/COMST.2005.1610546
6. Hamad, H.: Performance evaluation of restful web services for mobile devices. International Arab Journal of e-Technology
7. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational) (1997). URL `http://www.ietf.org/rfc/rfc2104.txt`. Updated by RFC 6151
8. Legout, A., Urvoy-Keller, G., Michiardi, P.: Understanding bittorrent: An experimental perspective. INRIA Sophia Antipolis/INRIA ...
9. Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., Ghalsasi, A.: Cloud computing - The business perspective. Decision Support Systems (1), 176–189. DOI 10.1016/j.dss.2010.12.006
10. Mell, P., Grance, T.: The NIST definition of cloud computing (draft). NIST special publication p. 145
11. Ngak, C.: Gmail, google drive, chrome experience outages. `http://www.cbsnews.com/8301-205_162-57558242/gmail-google-drive-chrome-experience-outages/`. Accessed on 2 january 2013
12. Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., Epema, D.: A performance analysis of EC2 cloud computing services for scientific computing. Cloud Computing pp. 115–131
13. Parameswaran, M.: P2P networking: an information sharing alternative. Computer
14. Ramgovind, S., Mm, E., Smith, E.: The Management of Security in Cloud Computing. Security (2010)
15. Rimal, B.P., Eunmi Choi, Lumb, I.: A Taxonomy and Survey of Cloud Computing Systems. pp. 44–51. IEEE. DOI 10.1109/NCM.2009.218
16. Schollmeier, R.: A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. Peer-to-Peer Computing, 2001. Proceedings. ...
17. Weinstein, N.: Netflix outage mars christmas eve. `http://news.cnet.com/8301-1023_3-57560784-93/netflix-outage-mars-christmas-eve/`. Accessed on 2 january 2013
18. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications **1**(1), 7–18 (2010)