



Proceedings of the First International Workshop on Sustainable
Ultrascale Computing Systems (NESUS 2014)
Porto, Portugal

Jesus Carretero, Javier Garcia Blas
Jorge Barbosa, Ricardo Morla
(Editors)

August 27-28, 2014

Content Delivery and Sharing in Federated Cloud Storage

J.L.GONZALEZ¹, VICTOR J. SOSA-SOSA¹, JESUS CARRETERO², LUIS MIGUEL SANCHEZ²

Cinvestav-Tamps, Mexico

¹joseluig@ac.upc.es,¹vjsosa@tamps.cinvestav.mx

University Carlos III, Spain

²luismiguel.sanchez,jesus.carretero@uc3m.es

Abstract

Cloud-based storage is becoming a cost-effective solution for agencies, hospitals, government instances and scientific centers to deliver and share contents to/with a set of end-users. However, reliability, privacy and lack of control are the main problems that arise when contracting content delivery services with a single cloud storage provider. This paper presents the implementation of a storage system for content delivery and sharing in federated cloud storage networks. This system virtualizes the storage resources of a set of organizations as a single federated system, which is in charge of the content storage. The architecture includes a metadata management layer to keep the content delivery control in-house and a storage synchronization worker/monitor to keep the state of storage resources in the federation as well as to send contents near to the end-users. It also includes a redundancy layer based on a multi-threaded engine that enables the system to withstand failures in the federated network. We developed a prototype based on this scheme as a proof of concept. The experimental evaluation shows the benefits of building content delivery systems in federated cloud environments, in terms of performance, reliability and profitability of the storage space.

Keywords Content delivery, Virtualization, Storage federation, fault-tolerant

I. INTRODUCTION

Space agencies, hospitals, government instances, news agencies and scientific centers not only are producing huge amount of contents but also they need to distribute them to different communities or population segments through the Internet [1].

Cloud storage approach has becoming a cost-effective solution for building dynamic and elastic online content delivery(CD) systems[2]. Organizations can contract a CD service with a provider through self-service and self-organizing web applications and their users can retrieve the contents in any time from anywhere by using almost any device.

However, organizations and users still have concerns about un-available service[3], lost data risks [4] and lack of controls over data management [5] when using cloud storage. Organizations and users are quite justified in expressing their concerns about data storage and management in the cloud because a user rejects her legitimate expectation to privacy when she voluntarily relegates private content to a *third-party* [6].

Vendor dependence lock-in is another problem that has caused a big concern among organizations. This problem arises when one organization contracts with a single cloud provider the storage and management of all the data. This becomes a real problem when the organization decides to change the cloud provider or the cloud provider pulls out of the market[7]. In the first scenario it is not clear that other provider could handle the content delivery in its current state because of software dependencies. In the latter, the organization depends on the window offered by the provider to the clients for migrating their data to another provider. In both scenarios, the more data stored in the cloud the more economic impact on the costs suffered by the organizations.

The federated cloud model enables organizations to create a shared cloud storage based on strategic partnership policies[8]. In this model, a set of organizations cooperates for building a shared storage space to serve requests of other members that are in either failure or saturation circumstances. This model improves the reliability of the CD services as a federation member can withstand failures of their site by using the resources of partners for serving their user requests. Federation also enables the organizations to preserve autonomy and privacy even when a portion of their infrastructure has been used by the partners experimenting site failures.

This paper presents the implementation of a system for content delivery and sharing in Federated Cloud Storage. This system virtualizes the storage resources of a set of organizations as a single federated system, which is in charge of the content storage. The architecture includes a metadata management layer to keep the content delivery control in-house and a storage synchronization worker/monitor to keep the state of storage resources in the federation as well as to send contents near to the end-users. It also includes a redundancy layer based on a multi-threaded engine that enables the system to withstand failures in the federated network.

We developed a prototype based on this system as a proof of concept and its performance was compared with a fault-tolerant distributed web storage as well as public and private File Hosting Services.

A case study based on data obtained from the European Space Astronomy Center (ESAC) for the Soil Moisture Ocean Salinity [9] is presented as experimental evaluation. We distribute satellite images to a set of organizations, from two countries spanning two continents by using a federated Storage System (FSS). The end-users retrieved images by using a FSS client.

The experimental evaluation shows the benefits of building con-

tent delivery systems in federated cloud environments, in terms of performance, reliability and profitability of the storage space.

II. RELATED WORK

Content Delivery Networks (CDN) such as Akamai [10], Coral [11] or Globule [12] cache small pieces of information and distribute them to locations near who requests them. The final end-user observes a reduction of the latency and overhead in the content delivery process.

The cloud-based storage services enable organizations to create catalogs of contents based on URLs. The end-users can access the catalog without simultaneous download restrictions by using any of the web browsers, synchronizer based on HTTP streams or (S)FTP applications. However, studies show that users prefer local storage solutions than public solution when managing sensitive data [13]. Moreover, cloud-based storage services are based on a pay-as-you-go pricing model, which apply rates based on the monthly stored contents plus the penalizations stated at the service level agreement (SLA) contracted. These conditions could lead to long-term costs. For instance, EUMETSAT and EOSDIS transfer around 1 TB of meteorological images per day, which might press organizations to consider an alternative service.

In addition, Vendor lock-in problem could arise when the organization decides to change the CDN provider or the cloud provider pulls out of the market[7](Nirvanix provides cautionary tale for cloud storage).

In order to face up system failures, in this kind of approaches the servers split the contents into chunks by applying a given codification[13] and they distributes them according a given fault-tolerant strategy. Nevertheless, the users and the organizations send the whole contents to the storage services, which could produce concerns about the way in which the privacy data is managed. In addition, the encoding/decoding data and its distribution both represent an extra work to be performed by the organizations servers.

The distribution of data on several providers have been proposed to avoid this type of problems [14]. Nevertheless, this kind of solutions are only available for public providers and solutions taking both the user and organization concerns into account are currently required by organizations.

This system can configure federations by using either private or public storage resources for organizations to deliver contents to end-users. Moreover, the codification applied to content dispersion by our system takes advantage of continuous flows and the multiple cores available in current computers.

III. A STORAGE SYSTEM FOR CONTENT DELIVERY AND SHARING IN FEDERATED CLOUD

A traditional content delivery system commonly includes stages such as source, formatting, deliver and acquisition of contents. In the first stage providers send a set of contents in raw format to the formatting stage, in which a set of users performs a set of annotation tasks for achieving manufactured contents, which are sent to a storage system. In the phase of acquisition, the end users retrieves the contents from the content delivery service by using a web application.

Our federated storage system (FSS) takes advantage of three types of technologies to improve the effectiveness of the aforementioned

content delivery process. The first is a cloud storage federation technology [9] applied to a metadata manager for keeping the control of content delivery in-house and enabling the organization to preserve autonomy in failure scenarios. The second are publish-subscribe patterns applied to a storage synchronization monitor, which enables FSS to inform the user about new available contents and to register the consume of each storage resource. The last one is the multi-core technology applied to a redundancy layer based on a multi-threaded engine for taking advantage of multiple cores of end-users and providers computers to improve the performance of the codification and dispersion of the federated fault-tolerant schemes used by FSS in the CD service.

Figure 1 shows an example of content delivery based on FSS. Raw data is sent to providers, which perform a set of annotation tasks for achieving manufactured contents. These contents are sent in the form of a catalog to the metadata manager by using an agent of our system. The metadata performs a push operation to split the $|F|$ into $n = C1..Cn$ chunks, which are distributed to a set of storage synchronization monitors that are members of the storage federation. When the provider shares that content by publishing the catalog, the end-users are notified and can retrieve the contents by using a FSS client APP. This APP obtains the locations to retrieve the chunks once the metadata manager verifies that the credentials of the client are valid. The APP retrieves the chunks from the federated storage and reconstructs the file in-house. As a result, the members of the federation has no possibility to reconstruct any file without either the authorization of the metadata manager or the collaboration of other members of the federation.

III.1 A metadata management layer

This layer is a cloud image in charge of the metadata flows, which includes the following modules for establishing management rules for the content flows:

- *Catalog manager.* This module enables the organization to create and manage catalog of contents. This module includes an attribute-based policy for managing the access to the contents listed in the catalog. It also enables authorized users, called providers, to add new contents to the catalog.
- *Multi-tenant module.* It manages end-users and providers accounts and is also in charge of the controls of the catalogs property. In this module, the contents of a given user account are isolated and remains invisible to other accounts.
- *Publish and Subscribe server.* It is in charge of the catalog publication and the subscriptions of contents. It serves the contents orders sent by end-users and controls the location of each catalog in the federation. This module includes an alert system that notifies to providers who is subscribing their contents and delivers the links to get access to published contents. It also includes push and pull RESTful functions for storing and retrieving contents form the federation.

This critical component is installed in-house by using cloud instance placed at private cloud of the organization.

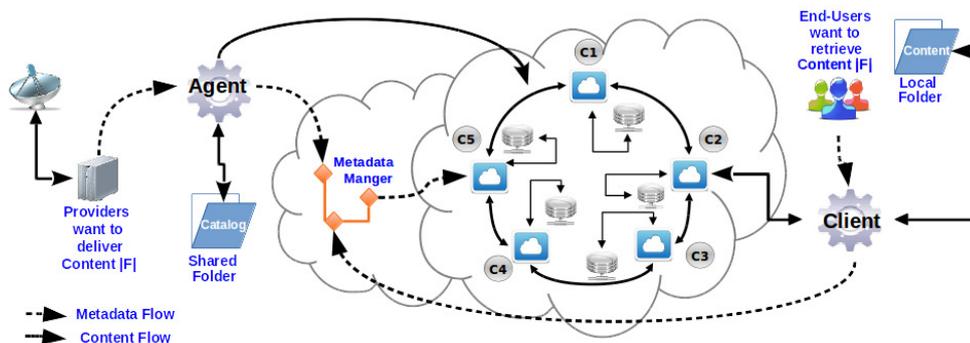


Figure 1: Content Allocation/Location in SkyStorage System.

III.1.1 A storage synchronization monitor

This module creates a database for managing the paths and access methods of each storage resource assigned by an organization to the federated cloud in the content delivery service. In order to observe the conditions of the agreement of the federation, this database also includes the consume metrics of each storage location. For instance, the storage quotas of received contents from each partner and the agreement characteristics for determining when a partner is ready to receive redundancy.

This module registers the operations performed by FSS, providers and end-users with each storage resource while an operational and safety monitor keeps updating the statistics in the database.

An agent in this module manages the metadata of the storage locations stores/retrieves contents by using the the redundancy layer.

III.2 A redundancy layer based on a multi-threaded engine

We add redundancy to the original content by using a dispersion algorithm called IDA [15]. This algorithm basically splits a given file $|F|$ into n redundant chunks, which must be distributed to n different storage locations. In scenarios where some original locations are unavailable and the user retrieves $|F|$, the algorithm recovers any of m number of chunks from the available storage locations with which the engine can reconstruct $|F|$. This means, it is granted that $|F|$ can be reconstructed when $n > m$ and the unavailable locations are $n - m$.

This algorithm can be implemented with different combinations of m and n parameters. This combination determines the codification costs in terms of storage space and computation time. The size of each resultant chunk is $|F|/m$, which results in a percentage excess of redundancy equal to $(n - m)/m$. Let us consider an IDA implementation with parameters ($n = 5; m = 3$), in this case $|F|$ is transformed into five chunks and it can be reconstructed by retrieving at least three chunks from any three different locations; as a result, the system produces 66.7% of redundancy overhead, which is less than one replica.

Table 1 shows the amount of extra capacity spent for n servers when requiring m chunks (at least) to support fault-tolerance.

Table 1: IDA Parameters Combination m (chunks required for recovering contents) and n (servers)

n(servers)	m=1	m=2	m=3	m=4	m=5	m=6
n=2	100%					
n=3	200%	50%				
n=4	300%	100%	33.3%			
n=5	400%	150%	66.7%	25%		
n=6	500%	200%	100%	50%	20%	
n=7	600%	250%	133.3%	75%	40%	16.7%

III.2.1 Parallelism and Continuous Workflows

In order to save storage space, we use this algorithm in the content delivery process instead of using several replicas. In terms of latency, the client is retrieving $(|F|/m)$ chunks of data, which is similar to retrieve the whole file.

Nevertheless, the codification of the redundant chunks produces computation overhead while the distribution of chunks produces latency, which could be a problem depending on the network characteristics.

In order to reduce the effects of overhead and latency on the encoding/decoding procedures, we proposed and implemented an IDA codification technique based on parallel and continuous flows called Continuous Workflows.

We defined a *distribution workflow* based on the IDA encoding procedure. This workflow has been designed for providers to deliver contents to the federated storage by using push operations. This workflow includes an *acquisition stage* that receives a *service key* as input parameter. This key reports the construction of this workflow to metadata manager and enables the engine to obtain n relative URLs mapping n different containers. This engine stores each chunk that will produce this workflow by using a relative URL, which includes an anonymized name that will be the identifier of that chunk in the assigned container. This means that this chunk is managed as a file by the container/provider. This stage starts when reading the content that will be distributed by the workflow and ends up when sending both the content and the URLs to the next stage.

The *transformation stage* creates as many process as cores in the computer where the engine has been launched to split the content into n redundant chunks. This stage adds redundancy to each chunk and sends the obtained results to the transport stage. This stage

Table 2: The characteristics of Agents Clients

	PCs and Cloud Instances	Cores	RAM
Agents			
UC3M-Cloud	5 Instances	2	4GB
UC3M-Colme	2 PCs	4 (i7) and 2 (i5)	4GB
Cinvestav	5 Instances	4	8GB
Clients			
UC3M-Cloud	2 Instances	4	4GB
UC3M-Colme	2 PCs	7 (i7) and 4 (i5)	4GB
Cinvestav	5 Instances	2(3) and 3(2)	2(1GB) and 3(4GB)

writes the results, sent by the previous stage, in n streams created by using the relative URLs. This stage closes the streams when the encoding of each chunk is done and reports to the engine the found errors if any.

We also defined a *retrieving workflow* based on the IDA decoding for end-users to retrieve contents from the storage federation.

In this workflow, the *acquisition stage* receives as input parameter the name of the content that will be retrieved by the workflow, creates one file $|F|$ with this name by using the local file system and sends m relative URLs to the pipeline. The *transformation stage* creates as many process as cores available and sends the URLs to the transport stage which creates m streams by using the URLs, reads the chunks and sends the results to the pipeline. The *transformation stage* receives data, starts the reconstruction of the content and sends the results to the pipeline. The *acquisition stage* writes the received data in the file $|F|$.

The goal of this technique is to use all the processing power available in the computer where the engine is placed for enhancing the performance of the content workflow as it represents the highest costs in a CD service.

This multi-threading version improves the performance of encoding and decoding tasks by taking advantage of multiple cores commonly found in current devices. This technique allows the engine to reduce the codification overhead making feasible to introduce a fault-tolerant scheme in the CD process. The implementation of CD as continuous flow allows the engine to avoid writing chunks in the local disks.

We implemented the transformation stage by using TBB technology [16] and the transportation stage by using Curl libraries [17].

IV. THE PROTOTYPE

We consider a scenario where a set of organizations build a CD service in a cloud federated network by using our FSS. The federation includes three members that are represented by the following acronyms *UC3M-Colme*, *UC3M-Cloud* and *Cinvestav*. *UC3M-Cloud* located in Leganes, *UC3M-Colme* located in Colmenarejo (both cities near to Madrid, Spain). *Cinvestav* is located in Northeastern Mexico.

The *UC3M-Cloud* is in charge of the metadata manager and all the members have installed a image of the storage monitor and including a set of agents of the multi-threaded engine. We have launched a set of client images in the infrastructure of all the members with which the end-users retrieve the contents published by a source.

Table 2 shows the features of the storage infrastructure shared by each organization as well as the agents and clients included in this prototype.

V. EXPERIMENTAL EVALUATION

We defined two evaluation scenarios: In the first we evaluated the performance of the federated storage system(FSS) with a synthetic workload while in the second we conducted a study case in which apply our FSS to the content deliver by using a set of real images.

We developed an *IO_Launcher* for producing publish, subscribe, pull (Download) and push (Upload) operations. The API sends these operations to the metadata manager. It assumes this artificial load comes from real and valid users and captures the response time, which is the only metric evaluated so far.

We captured the response time per each performed operation, which helps us to determine the degree of satisfaction or end-users. This time is measured from the publication/subscription moment of a given content until the time point in which the storage service retrieves that content, meaning that the request has been successfully dispatched. This time includes the streaming time, the network round trip latency and the write/read time in the temporal paths. This time also includes the time spent in subscription synchronization. Finally, this time also could include codification time when using redundancy.

VI. EXPERIMENTS PER EVALUATION SCENARIOS AND RESULTS

The preliminary results included in this section consider the evaluation of the two scenarios previously defined.

VI.1 Performance evaluation of Federated storage system

In this scenario, we evaluated a storage network created with *UC3M-Colme* and *UC3M-Cloud* for testing the performance of FSS when delivering and retrieving contents with different size file.

We have designed a synthetic workload scenario to test the redundancy engine in which *IO_Launcher* sends an incremental load by duplicating the file size from 512KB to 1GB.

We defined the following configurations in this scenario:

- *Phoenix*: In this configuration, users store and retrieve contents by using an Online Distributed Web Storage System called Phoenix[18]. We implemented *Phoenix* in the *UC3M-Colme* and *UC3M-Cloud* organizations and it has been configured to apply a fault-tolerant strategy to the contents based on dispersion of information by using the IDA algorithm.
- *Private FHS*: In this configuration, users store and retrieve contents by using a Private Web File Hosting System. We have implemented a *Private FHS* in *UC3M-Colme* organization in which the contents are stored without producing and distributing redundancy.
- *Amazon*: In this configuration, users store and retrieve contents by using an AWS Amazon instance with the standard redundancy associated to a free account.
- *FSS*: We implemented our file distributor and acquirer agent in the *UC3M-Colme* and *UC3M-Cloud* organizations. This configuration allows us to measure all the stages of the workflow produced by our storage system.

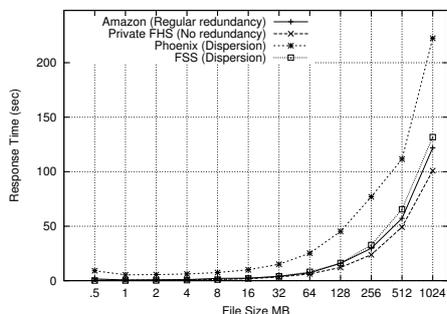


Figure 2: Response Time of Uploads

Fig. 2 shows, in vertical axis, the mean response time obtained when a user sends contents of different sizes (horizontal axis) to the online storage system.

Fig. 2 also shows that *Private FHS* yields the best performance because it returns the control to the user when the file arrives to the cloud. This means *Private FHS* does not generate redundancy overhead, which improves the response time observed by the user. Moreover, *Private FHS* is situated in Colmenarejo city while the common users of this service are from both Madrid and Comenarejo. This reduces latency because both cities are geographically close to each other. *Amazon* configuration performs the same procedure as *Private FHS* but it produces more delay because its servers are located at Ireland.

Phoenix configuration produces the worst delay because it solves the vulnerability window in which the user could loss data by immediately splitting contents into chunks and distributing them to cloud storage locations of the two organizations. Once this has been performed, *Phoenix* returns the control to the synchronizer. As a result, the user obtains data assurance and she can retrieve that file even when the site of her organization is down.

The *FSS* performance is better than *Amazon* configuration (44% in mean) when the file size <4MB because the locality compensates the overhead of codification and, when the file size >4MB, *Amazon* configuration is better than *FSS* (9.14% in mean) because *FSS* distributes five chunks per I/O request (66.7% more data than *Amazon*). Nevertheless, this is a small increment because of *FSS* optimizes the storage stages on the client-side. In addition, that increment can be reduced or even eliminated by reducing the amount of distributed chunks when applying the information assurance policies.

As we can see, *FSS* offer the same reliability as *Phoenix* at reasonable cost. Moreover, *FSS* preserves the original file in-house, which means the contents can not be reconstructed by the administrators of a given organization without obtaining some chunks from either other organizations or the user device. When the users retrieving contents by using *FSS*, we observed the same behavior that the upload operations.

VI.2 Case study: Satellite Images delivery

This case study is based on data from the European Space Astronomy Center (ESAC), located at Villafranca del Castillo (Spain), which is in charge of the Soil Moisture Ocean Salinity or SMOS mission from

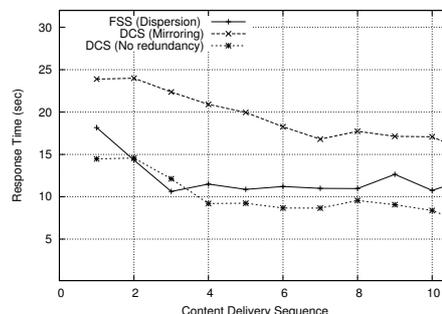


Figure 3: The response times of push operations for FSS and Distributed Cloud Storage configurations

2009. This mission has generated a vast amount of contents, that will be used to create the first Earth global salinity map. Our system distributes images of the Sun of SMOS mission in FITS format to end-users through the Internet with a mean size of 44MB. A set of organizations and end-users, from two countries spanning two continents, get the contents by using a client APP of our system.

The following three configurations were defined in this case study:

- *DCS*: In this configuration, the end-user can access the contents by using an online distributed cloud storage system or *DCS* that does not include any redundant information.
- *DCS-Mirroring*: We implemented *DCS* in the UC3M-Colme and UC3M-Cloud organizations that includes a fault-tolerant strategy based on simple mirroring. This means two replicas are sent to the federated storage. The organization can withstand the failure of one cloud storage site by using this configuration.
- *FSS*: In this configuration, the organization delivers contents to interested users by using a *FSS* system, which was configured with an IDA combination ($n = 5, m = 3$). This means the *FSS* client retrieves three chunks each time the end-users subscribe a published content. This IDA configuration allows the system to withstand 2 failures.

FHS and *Phoenix* configurations are not considered in this study as the first exposes security issues and the second produces more overhead than its version multi-threaded version *FSS*.

In this scenario we configured a storage network federation including UC3M-Cloud, UC3M-Colme and Cinvestav organizations. UC3M-Cloud was in charge of metadata manager and master of the content distribution role. In this scenario, a source sends images to the content delivery service by using publish and subscribe patterns. In this evaluation we define a push pattern in which the source sends the contents to its near members (UC3M-Cloud, UC3M-Colme). We defined two pull patterns. The first invoked by member that are not near to the source (Cinvestav). This pattern has been configured as a hot standby scheme in which each storage monitor worker of the Cinvestav member has a partner in the UC3M-Cloud and UC3M-Colme. Each worker of Cinvestav retrieves a chunk from its partner when the master receives a publish pattern. The service is accessed by the users of the members as single domain by using subscribe

patterns. Based on this unified interface, all users of all members are supported even when the servers of their organization are not available.

Figure 3 shows the mean response times observed by organizations when they distribute contents to end-users by using the mentioned configurations. Each point in this graph represents the mean response time of ten push operations. This means that 120 contents, with a mean size of 44MB, were sent to the cloud storage by the content delivery configurations. Figure 3 also shows that the response times of DCS configuration are better than the FSS configuration. Nevertheless, FSS configuration allows organization to withstand the failure of one cloud site and the source, while DCS is a single backup that can tolerate the failure of the source.

DCS-Mirroring improves the reliability of DCS configurations but it introduces a considerable overhead in the response times. Moreover, this configuration sends the whole file, so privacy and legal problems could arise. The FSS system distributes anonymized and encoded chunks, which means that the FSS agent and client know the locations of the chunks and are the only ones that can reconstruct the contents. In addition, FSS only produces up 66.7% of redundancy overhead while this overhead for DCS mirroring is of 100%.

VII. CONCLUSIONS

This paper presented the implementation of FSS: a system for content delivery and sharing in Federated Cloud Storage. This system virtualizes the storage resources of a set of organizations as a single federated system, which is in charge of the content storage. The architecture of this system includes a metadata management layer to keep the content delivery control in-house and a storage synchronization worker/monitor to keep the state of storage resources in the federation as well as to send contents near to the end-users. It also includes a redundancy layer based on a multi-threaded engine that enables the system to withstand failures in the federated network. The experimental evaluation shows the benefits of building content delivery systems in federated cloud environments, in terms of performance, reliability and profitability of the storage space.

Acknowledgment

The work presented in this paper has been partially supported by EU under the COST programme Action IC1305, Network for Sustainable Ultrascale Computing (NESUS) Nesus-members mailing list Nesus-members@arcos.inf.uc3m.es <https://www.arcos.inf.uc3m.es/cgi-bin/mailman/listinfo/nesus-members>

REFERENCES

- [1] T. J. Bittman and L. Leong. Worldwide archival storage solutions 2011-2015 forecast: Archiving needs thrive in an information-thirsty world. pages 1-21, *IDC. Market Analysis*. October 2011
- [2] <http://aws.amazon.com/es/cloudfront/> Access 29 Sep 2014
- [3] Google: Software bug caused gmail deletions (2011). <http://www.pcmag.com/article2/0,2817,2381168,00.asp>, Access 09 Sep 2014.
- [4] Storm clouds ahead. <http://www.networkworld.com/news/2009/030209-soa-cloud.html?page=1>. Access 09 Sep 2014.
- [5] <http://www.itproportal.com/2012/07/04/power-outage-generator-failure-responsible-for-instagram-netflix-blackout/>, Access 09 Jul 2013
- [6] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. "Controlling data in the cloud: outsourcing computation without outsourcing control". In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 85-90, 2009.
- [7] Report: Nirvanix customers have two weeks to get data out of cloud: <http://www.networkworld.com/article/2170916/cloud-computing/report-nirvanix-customers-have-two-weeks-to-get-data-out-of-cloud.html>
- [8] J. L. Gonzalez, J. C. Perez, V. Sosa-Sosa, J. F. Rodriguez Cardoso, and R. Marcelin-Jimenez. "An approach for constructing private storage services as a unified fault-tolerant system". *J. Syst. Softw.*, 86(7):1907-1922, July, 2013.
- [9] smos: Soil moisture and ocean salinit. <http://www.esa.int/esalp/lpsmos.html>. Access date: 23 jul 2013.
- [10] J. Dille, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. "Globally distributed content delivery". *Internet Computing, IEEE*, 6(5):50-58, 2002.
- [11] M. J. Freedman, E. Freudenthal, and D. Mazieres. "Democratizing content publication with coral". In *NSDI*, volume 4, pages 18-18, 2004.
- [12] G. Pierre and M. Van Steen. "Globule: a platform for self-replicating web documents". In *Protocols for Multimedia Systems*, pages 1-11. 2001.
- [13] R. Seiger, S. G. and A. Schill, Seccsie: A secure cloud storage integrator for enterprises, in *Proceedings of the 2011 IEEE 13th CEC2011*, pp .252-255.
- [14] J. Spillner, G. Bombach, S. Matthischke, J. Muller, R. Tzschichholz, and A. Schill, "Information dispersion over redundant arrays of optimal cloud storage for desktop users," in *Fourth IEEE International Conference on Utility and Cloud Computing*, 2011, pp.1-8.
- [15] M. O. Rabin. "Efficient dispersal of information for security, load balancing, and fault tolerance". *J. ACM*, 36(2):335-348, 1989.
- [16] A. Robison, M. Voss, and A. Kukanov, "Optimization via reflection on work stealing in tbb," in *IPDPS. IEEE*, 2008, pp. 1-8.
- [17] libcurl, <http://curl.haxx.se/libcurl/> Access date: 23 Sep 2014.
- [18] J. L. Gonzalez and R. Marcelin-Jimenez, "Phoenix: Fault tolerant distributed web storage." in *Journal of Convergence*, Section C: Web and Multimedia, Vol.2, No.1, 2011.