Data integration in Cloud Computing environment

Fernando De la Prieta[1], Sara Rodríguez[1], Javier Bajo[1], Vivian F. Lopez Batista[1]

[1]University of Salamanca, Salamanca, Spain
[2]Technical Univesity of Madrid, Madrid, Spain
{fer, srg, vivian}@usal.es, jbajo@fi.upm.es

Abstract

Information processed by applications is usually stored in databases or filing systems, allowing each system to use its own interface. As the files are not stored transparently, it is necessary to define data models to efficiently manage the information. This study proposes a process for storing information that follows the cloud paradigm defined in the +Cloud platform, which facilitates the transparent integration of different sources for the applications without requiring a description of relational database models.

Keywords: Cloud Computing, Data Integration

1. Introduction

A Information storage is not performed in the same way today as it was in the past. During the incipient stages of computer sciences, information was stored and accessed locally in computers. The storage process was performed in different ways: in data files, or through the use of database management systems that simplified the storage, retrieval and organization of information, and were able to create a relationship among the data. Subsequently, data began to be stored remotely, requiring the applications to access the data in order to distribute system functions; database system managers facilitated this task since they could access data remotely through a computer network. Nevertheless, this method had some drawbacks, notably that the users had to be aware of where the data were stored, and how they were organized. Consequently, there arose a need to create systems to facilitate information access and management without knowing the place or manner in which the information was stored, in order to best integrate information provided by different systems

The concept of computing and software has begun to vary with the appearance of Cloud computing [2][3][5]. Traditional applications are executed and installed locally, and data can be stored either locally or remotely. Cloud computing currently allows applications to be executed in a cloud without requiring a local installation to access them [1][12]. Furthermore, clouds offer storage services such as those provide by Amazon in the Amazon Simple Storage Service (Amazon S3) [4]. However, these systems do not offer as many possibilities for information management as were offered by relational database management systems. As with Amazon S3, one of the applications of these systems is Storage for data analysis.

This study proposes a Cloud architecture developed in the +Cloud system to manage information. +Cloud is a Cloud platform that was developed and makes it possible to easily develop applications in a cloud. Information access is achieved through the use of REST services [17], which is completely transparent for the installed infrastructure applications that support the data storage. In order to describe the stored information and facilitate searches, APIs are used to describe information, making it possible to search and interact with different sources of information very simply without knowing the relational database structure and without losing the functionality that they provide. Using a Cloud architecture and document manager facilitates the integration of information from different sources that is used by applications, thus facilitating the exchange of information among the different services offered by the Cloud.

This article is divided as follows: section two describes the state of the art for cloud computing; sections three and four presents the proposed model and section five provides conclusions.

2. Cloud computing

There are currently two computing models that dominate information technology: the centralized computing model, typical in mainframe systems, powerful, and connected to multiple terminals; and the distributed computing model, with the client-server model being its most widespread example, and a highly proven efficiency. A new model has recently appeared. Known as Cloud Computing, it was developed in response to the explosive increase in the number of devices connected to the internet, and to complement the ever-increasing presence of technology in our daily lives and, in particular, in the workplace.

Cloud Computing is an alternative model for acquiring and providing services that is changing the way a company and its clients do business, and the way that suppliers provide services or products. Cloud Computing refers to any services or resources that are accessed through a network by multiple devices without the user needing to install any type of software on their local terminal. The information is also stored remotely without needing to be tied to the terminals from which it is accessed. Cloud Computing incorporates various distribution models:

1. Application (SaaS) [7][11]: software that is found in the internet, eliminating the need for the client

to install anything. One example could be the Dropbox service.

2. Platform (PaaS) [8]: subsystems that can host and run applications, providing a special access API. Some examples include Google App Engine or Windows Azure [14].
3. Infrastructure (IaaS) [9][10]: server-type resources, storage and other lower level systems. Examples include Eucalyptus or Amazon Web Services (EC2 [13] and S3 [3]).

Cloud-based applications or services are changing how software is being used and files are being stored. Currently, companies such as Google prefer Chromebook for use with portable devices that do not have a hard drive; these devices are oriented towards the use of online application and storage systems. Among other things, they are facilitating the possibility of sharing documents, editing them online through a collaborative effort, and efficiently managing the different versions of the files. These tools will undoubtedly play an increasingly important role in the near future.

3. Cloud architecture for managing information / information management?

The system has a layered structure that covers the main layers of cloud computing[5]. Figure 1 displays the architectural structure for the system.

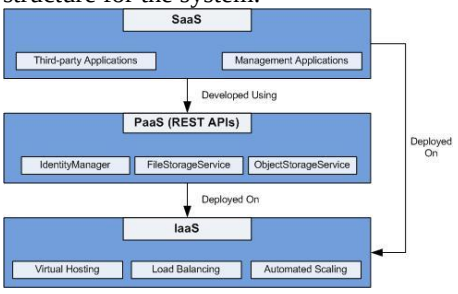

Figure 1. Cloud architecture storage system

The SaaS (Software as a Service) layer is composed of the management applications for the environment (control of users, installed applications, etc.), and other more general third party applications that use the services from the PaaS (Platform as a Service) layer. At this level, each user has a personalized virtual desktop from which they have access to their applications in the Cloud environment, and to a personally configured area as well.

The PaaS layer provides services through REST web services in API format. One of the more notable services among the APIs is the identification of users and applications, a simple non-relational database service and a file storage area that controls versions and simulates a directory structure.

All management and general purpose applications, and all services at the platform layer are deployed using the IaaS (Infrastructure as a Service) layer, which provides a virtual hosting service with automatic scaling and functions for balancing workload.

3.1. PaaS Layer: REST APIs

The services anticipated for the platform layer are presented in the form of stateless web services. The data coding format used is JSON (JavaScript Object Notation), which is more easily readable than XML and includes enough expression capability for the present case.

JSON is a widely accepted format that contains numerous libraries for different programming languages, which allows instances of classes to be converted to the JSON format (e.g., Google-Gson for Java).

3.2. File Storage Service

The file storage service provides an interface for a container of files, emulating a directory structure, in which the files are stored with a set of metadata, thus facilitating retrieval, indexing, search, etc.

The most relevant characteristics are the simulation of a directory structure, with which the software developer can interact as they would with a physical filing system, and a simple mechanism for version control.

When overwriting a file, which is to say that the space occupied by one file is made available to another, the content of the original is not actually erased, a new version is created instead. Something similar occurs with elimination: instead of erasing the content of the file, it is marked as eliminated. At any moment it is possible to access the available versions of a file by using the API functions.

In addition to being organized hierarchically, files can be organized with taxonomies, using text tags, which facilitates the semantic search for information by making it more efficient.

In addition to its context, the following information is stored for each file present in the system:

- Its virtual path (parent directory and complete name).
- Its length or size.
- An array of tags to organize the information semantically.
- A set of metadata.
- Its md5 sum to confirm correct transfers.
- Its previous versions.

The incorporation of semantic searches can be improved by including content searches, using Apache Solr [4], Lucene[3], etc.

REST Methods

The methods provided by the services to manage the files stored in a cloud are:

- PutFile: creates a new file (or a versión of an existing file) in response to a request contained in the file and basic metadata (path, name and tags) in JSON, structured in a standard multipart request.
- MoveFile: changes the path of a file.
- DeleteFile: eliminates a file.
- GetFolderContents: returns a JSON array with the content of a specific directory.

- GetMetadata: returns the metadata of a file or directory according to its identifier (path, name, size, md5, etc.).
- GetAvailableVersions: returns the available versions of a file.
- DownloadFile: returns the file content (previous versions can be specified).

Supporting architecture:

Web services are implemented through the use of the web application framework *Tornado*, which relies on some functions in the reverse proxy *Nginx*[16]. Figure 2 displays the configuration of the architecture that supports the file management system in Cloud.

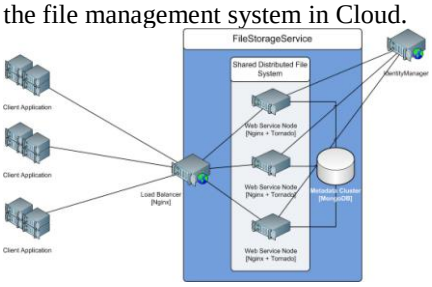

Figure 2. File storage system

File content is saved in the distributed file system so that the service can be scalable, and the multiple entry points used can be managed by a load balancer; it also allows information to be replicated. Several tests were performed with NFS and GlusterFS; a transparent migration was used for the rest of the service since the interface with those systems is limited to an assembly point in the Unix VFS (Virtual File System).

File metadata and the path indicators for accessing data content are both stored in the MongoDB B[14][15] distributed database, which has high speed and scalability and provides adequate guarantees of integrity as required by this application.

Web service nodes deploy Tornado and Nginx as well as the distributed file systems GlusterFS or NFS, and access a MongoDB cluster, which can be located either within or exterior to the nodes.

Optimizing file uploads and downloads:

In order to efficiently manage file uploads, the Nginx proxy for each basic node intercepts the web request, extracts the file travelling inside, writes it on the hard drive using the nginx_upload_module (very fast and efficient code written in C), and transfers the request, without the file, to the real web service, indicating only the path where the file is stored. This path will be the definitive path in the shared file system, to avoid the unnecessary movement of files, which negatively impacts the server's performance and the time that it would take for the information to be available for another request.

This optimization in uploading files is necessary to make the service viable: web requests in Tornado use up all of the main memory (RAM) when they are processed. These requests include uploading files, and it is easy to see how some files can be big enough to easily overload the service.

Nginx is also used to mange file downloads efficiently. Web server responses do not directly contain the file to be downloaded; instead they have a special header that indicates the proxy where the file is stored (in the shared file system) and add it to the response in an efficient manner.

These interactions between the real server and the proxy are produced in a way that is transparent to the user, who believes to be working with a single homogeneous server.

The system's design makes is possible to manage multiple and simultaneous uploads of large files without forgoing the flexibility of the Python environment (an interpreted language with great semantic capability) for the logic in the service. The highly optimized and tested code C from Nginx manages file transfers, and the Python code manages everything else: confirming identities, managing metadata, etc.

Processing a request to upload a file

1. When the request arrives, the load balancer reroutes it to one of the web service nodes.
2. The Nginx instance of the node intercepts the request and saves the file content provisionally in the distributed file system.
3. After saving the file, it sends the modified request (which no longer contains the file, but a temporary path, its length and md5 sum, and the metadata provided by the client) to the web service implemented in Python.
4. The web service processes the request and confirms authentication using an asynchronous REST request to the identity manager service (IdentityManager).
5. When the identity service responds, the processing of the request is renewed.
6. If the authentication of the application is correct, the metadata present in the request are confirmed.
7. If the metadata are correct, they are stored in the MongoDB cluster with the path for the file content.
8. The metadata for the saved file is returned to the user.

If an error occurs during any one of the steps, the user is informed with an HTTP error code, and the file content is erased from the storage.

Processing a request to download a file:

1. When the request arrives, the load balancer reroutes it to one of the web service nodes.
2. The web service processes the request and confirms the authentication using an asynchronous

REST request to the identity manager service (IdentityManager).

3. When the identity service responds, the processing of the request is renewed.
4. If the authentication of the application is correct, the file identifier present in the request is confirmed.
5. If the identifier is valid, the path for the file content is searched for in the MongoDB cluster.
6. The server responds with a header that indicates the file path to the local proxy.
7. The local proxy initiates the file download.

If an error occurs, the user is informed with an HTTP error code.

3.3. Object Storage Service

The object storage service provides a simple and flexible schemaless data base service oriented towards documents. In this context, a document is a set of keyword-value pairs where the values can be documents (in an anitable model) or references to other documents (with a very weak integrity model).

Schemaless database

The objects or documents are automatically assigned a unique alphanumeric identifier and are sorted into groups (similar to relational database tables). It is not necessary for the objects from a group to share a set of attributes, although they normally have one subset in common since they tend to represent entities from the applications. This common subset is what will be used to perform searches and filter objects (although it is not necessary that they all share it).

The permissible types of data are limited to the basic types from JSON: alphanumeric, numeric, maps (embedded documents) and arrays for any of these types.

By not needing to previously define the set of attributes for the objects in each group, the migration between different versions of the applications and the definition of the relationship among the data become much easier. Adding an extra field to a group is as easy as sending dictionary objects with an extra keyword. A search on that keyword would only look in objects that contain it.

JSON queries:

The query language is base don JSON searches: dictionary objects are created and the search returns objects from the group that satisfies the criteria of equality with the dictionary object search. In other words, if we want to search for objects with the name "Robert" from the contact group, our query would be {first_name": "Robert"}.

In the future, different methods will be incorporated to increase the search criteria beyond equality, possibly adopting a query language similar to MongoDB.

REST Methods:

The service provides the basic services expected from a database manager:

- Create: creates a new object according to the data provided. Returns this object, adding the newly generated identifier.
- Retrieve: retrieves an object according to a JSON query.
- Update: updates an object according to the data provided (the identifier for locating the object to be updated must be provided).
- Delete: deletes an object according to the JSON query.

Supporting architecture:

As with the object storing service, the web services are implemented with Python and the Tornado framework. By not managing files, there is no need to use the inverse proxy that manages them in every node; therefore only Nginx is used to balance the workload at the entry point for the service.

Figure 3 shows the architecture that permits data storage in the Cloud system.

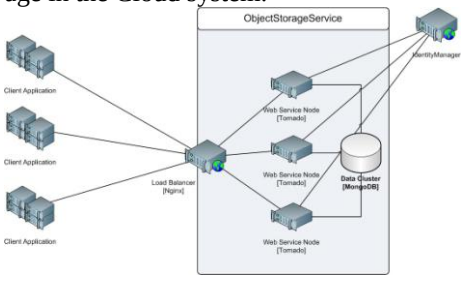

Figure 3. File Storage System

3.4. Identity Manager

The IdentityManager is the module from +Cloud in charge of offering authentication services to clients and applications. Among the functionalities that it includes are access control to data and files stored in the Cloud through user authentication and validation.

Notable functionalities provided by the identification module are:

1. A web authentication mechanism that permits integrated applications in +Cloud to know which users have access privileges to their application without the application itself implementing the authentication system.
2. REST calls to authenticate applications/users and assign/obtain their roles in applications within +Cloud, following the single sign on model.
3. New user registration.

The authentication process is shown in figure 4; the steps taken for identification as are follows:

1. A user accesses an application from +Cloud.

2. The application detects that the user is not authenticated and reroutes the user to the IdentityManager by following these steps:
    a. Request a temporary authentication identifier with a REST call to GetAuthenticationToken, using the necessary parameters for this call.
    b. Reroute the user to the web authentication module, accompanying the token as a GET parameter in the request.
3. IdentityManager now performs the authentication and returns the application to the user who requested it, providing a session identifier within the +Cloud system.
4. The application accepts the user and confirms the validity of the sesión identifier in +Cloud with a REST call to TokenIsValid. It also gets the user roles within the application.

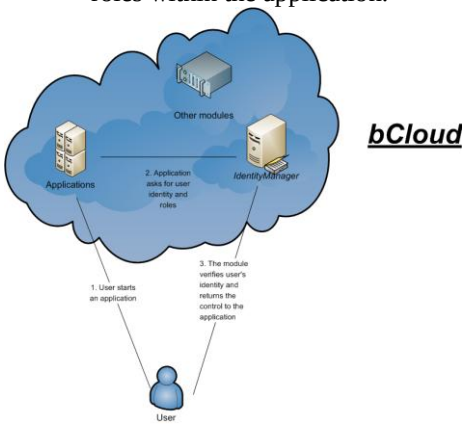

Figure 4. User authentication process

3.5. Scalability and high availability

In order for the platform level services to respond to the increasing demand of requests and volume of data, or to respond appropriately to failures in the individual nodes, the use of replication mechanisms and information division is applied, allowing the problems to be approached on three separate fronts: MongoDB clusters, distributed file systems, and API web services.

1. Replication and division of information in the MongoDB systems: uses the mechanism provided by SGBD. The information is divided into segments that are distributed among replica sets. The nodes from one replica set contain copies of the same information. The queries are made to a special node (or various nodes for security) that act as a load balancer and sent the queries to nodes that are live and contain the necessary segment of information.
2. Distributed file system: with GlusterFS, nodes can be added to the file system in a configuration similar to that of MongoB, although without needing to have special nodes to balance the workload.
3. API web servers: the API servers are replicated in order to manage large workloads and to overcome node failures. The Nginx reverse proxy is used at the system point of entry to function as a workload balancer between them.

4. Sharing computational resources among services

The development a Cloud Computing platform like + Cloud does not only allow to store information without schema, but also to share the computational resources provided by the cloud environment among the storage services (FSS, OSS).
Thus, +Cloud incorporates an information service, wherein each of the computing resources and offered services periodically have to send status information, as shown in Figure 5.

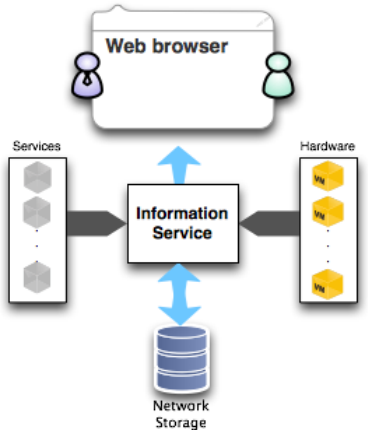

Figure 5. Service information model

As shown in the following graphs, it is possible to have information about different services centralized in a centralized web.

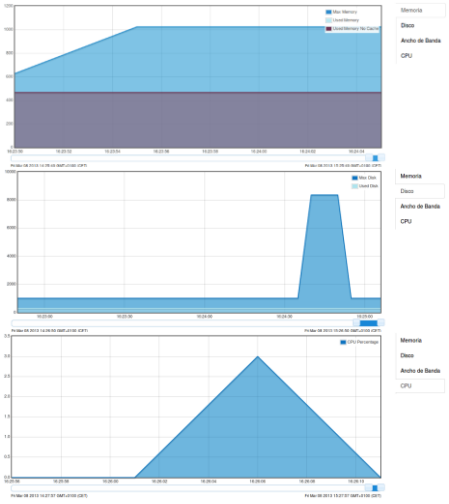

Figure 6. Usage of resources (Memory, disk and CPU)

With this model, it is possible to not only display information (Figure 6), but also to manage the computational resources and share them about the offered services.

The following graphs show how the system adapts itself to cope with the peaks on the demands. Figure 7 shows how the current memory of an individual virtual machine (red line) is increased to cope with the demand (blue line)

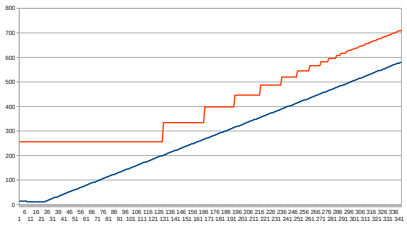

Figure 7. Redistribution resources: Memory

In the same way, Figure 8 shows how the CPU is adapting depend on the demand.

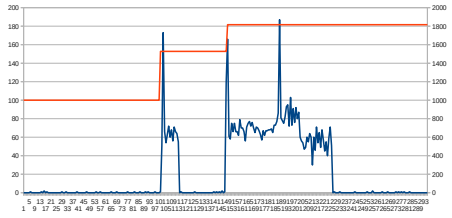

Figure 8. Redistribution resources: CPU

5. Conclusions and future lines of work

The cloud architecture defined in +Cloud has made it possible to transparently store information in applications without having previously established a data model.

The storage and retrieval of information is done transparently for the applications, and the location of the data and the storage methods are completely transparent to the user. This characteristic makes it possible to change the infrastructure layer of the cloud system, facilitating the scalability and inclusion of new storage systems without affecting the applications.

JSON can define information that is stored in the architecture, making it possible to perform queries that are more complete than those allowed by other cloud systems.

The storage system can easily compare information between different applications that can fuse data from different groups and process them as indicated. In this case, the use of JSON is important to be able to retrieve stored information without previous knowledge of the information stored in the different entities.

As future lines of work, we would like to add semantic search mechanisms to the files storage system, but not limited to those based on descriptions established by the metadata sent by the user.

6. References

[1] Amazon elastic compute cloud (Amazon EC2). Amazon. http://aws.amazon.com/ec2/

[2] Amazon Web Services. Retrieved April 20, 2010, from Amazon: http://aws.amazon.com/

[3] Apache Lucene Core. http://lucene.apache.org/core/

[4] Apache Sorl. http://lucene.apache.org/solr/

[5] Armbrust, M., et al. Above the clouds: A Berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, U.C. Berkeley, Feb 2009.

[6] CHAPPELL, D.2009. Introducing the Azure Services Platform. David Chappell & Associates

[7] Dave T (2008) Enabling application agility - Software as a Service, cloud computing and dynamic languages. Journal of object technology:29-32

[8] Dawoud W, Takouna I, Meinel C (2010) Infrastructure as a service security: Challenges and solutions. 2010 7th international conference on informatics and systems, INFOS2010, March 28, 2010 - March 30, 2010, Cairo, Egypt,

[9] Espadas J, Molina A, Jimenez G, Molina M, Ramirez R, Concha D (2011) A tenant-based resource allocation model for scaling Software- as-a-Service applications over cloud computing infrastructures. doi:10.1016/j.future.2011.10.013

[10] Foster I, Zhao Y, Raicu I, Lu S (2008) Cloud computing and grid computing 360-degree compared. Grid computing environments workshop, GCE 2008, November 12, 2008 - November 16, 2008, Austin, TX, United states,

[11] GlusterFS. http://www.gluster.org/

[12] Mahmood Z (2011) Cloud computing for enterprise architectures: concepts, principles and approaches. In: Mahmood Z, Hill R (eds) Cloud computing for enterprise architectures. Springer, pp 3-10

[13] Mell P, Grance T (2009) The NIST definition of cloud computing. National institute of standards and technology,

[14] Membrey, P., Plugge, E., & Hawkins, T. (2010). The definitive guide to MongoDB: the noSQL database for cloud and desktop computing. Apress.

[15] MongoDB. http://www.mongodb.org/

[16] NGiNX. http://wiki.nginx.org/Main

[17] Pautasso, C., Zimmermann, O., & Leymann, F. (2008, April). Restful web services vs. big'web services: making the right architectural decision. In*Proceedings of the 17th international conference on World Wide Web* (pp. 805-814). ACM.

[18] TurnerM,BudgenD,BreretonP(2003)Turningsoftware intoaservice. Computer 36 (10):38-44. doi:10.1109/mc.2003.1236470

[19] Wu X, Wang W, Lin B, Miao K (2009) Composable IO: A novel resource sharing platform in personal clouds. 1st international conference on cloud computing, CloudCom 2009, December 1, 2009 - December 4, 2009, Beijing, China,

[20] Yang H, Wu G, Zhang J (2005) On-demand resource allocation for service level guarantee in grid environment. 4th international conference on grid and cooperative computing - GCC 2005, November 30, 2005 - December 3, 2005, Beijing, China,

[21] Zhang Q, Cheng L, Boutaba R (2010) Cloud computing: State-of-the-art and research challenges. Journal of Internet Services and Applications:7- 18. doi:10.1007/s13174-010-0007-6