+Cloud: An Agent-Based Cloud Computing Platform

Roberto González, Daniel Hernández, Fernando De la Prieta, and Ana Belén Gil

University of Salamanca, Computer Science and Automatic Control Department,
Plaza de la Merced s/n, 37007, Salamanca, Spain
{rgonzalezramos,dan_her_alf,fer,abg}@usal.es

Abstract. Cloud computing is revolutionizing the services provided through the Internet, and is continually adapting itself in order to maintain the quality of its services. This study presents the platform +Cloud, which proposes a cloud environment for storing information and files by following the cloud paradigm. This study also presents Warehouse 3.0, a cloud-based application that has been developed to validate the services provided by +Cloud.

Keywords: Cloud Computing, cloud storage, agent-based cloud computing.

1 Introduction

The technology industry is presently making great strides in the development of the Cloud Computing paradigm. As a result, the number of both closed and open source platforms has been rapidly increasing [2]. Although at first glance this may appear to be simply a technological paradigm, reality shows that the rapid progression of Cloud Computing is primarily motivated by economic interests that surround its purely computational or technological characteristics [1].
The majority of well-known Cloud Computing platforms tend to underscore their ability to provide elastic infrastructure services (through virtualized hardware), without taking high level services, such as platform and software, into account.

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 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

S. Omatu et al. (Eds.): *Distrib. Computing & Artificial Intelligence,* AISC 217, pp. 377–384.
DOI: 10.1007/978-3-319-00551-5_46

This study proposes a Cloud architecture developed in the +Cloud system to manage information. +Cloud is a Cloud platform that makes it possible to easily develop applications in a cloud. Information access is achieved through the use of REST services, 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 simplifying the exchange of information among the different services offered by the Cloud. Finally, the study also presents Warehouse 3.0, which is a cloud storage application. This application has been developed with the aim to test and validate the functionality of the services proposed by the +Cloud platform.

This paper is structured as follows: the next section provides an overview of the +Cloud platform, paying special attention to PaaS layer; the Warehouse 3.0 is then presented; and finally the study finishes with the conclusion and future studies.2.1

2 +Cloud Platform

A complete cloud-computing environment was developed for this study. The system has a layered structure that coincides with the widely accepted layered view of cloud-computing [3]. This platform allows services to be offered at the PaaS (Platform as a Service) and SaaS (Software as a Service) levels.

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 next section presents the characteristics and modules of PaaS Layer in +Cloud and +Cloud in greater detail. Both the PaaS and SaaS layers are deployed using the internal layer of the platform, which provides a virtual hosting service with automatic scaling and functions for balancing workload. Therefore, this platform does not offer an IaaS (Infrastructure as a Service) layer. The virtual and physical resources are managed dynamically. To this end, a virtual organisation of intelligent agents that monitor and manage the platform resources is used [4].

2.1 PaaS Layer

The Platform layer provides its services as APIs, offered in the form of REST web services. The most notable services among the APIs are the identification of users and applications, a simple non-relational database, and a file storage service that provides version control capabilities and emulates a folder-based structure.

The services of the Platform layer are presented in the form of stateless web services. The data format used for communication is JSON, which is more easily readable than XML and includes enough expression capability for the present case. JSON is a widely accepted format, and a number of parsing libraries are available for different programming languages. These libraries make it possible to serialize and de-serialize objects to and from JSON, thus facilitating/simplifying the usage of the JSON-based APIs.

2.1.1 File Storage Service (FSS)

The FSS provides an interface to a file container by emulating a directory-based structure in which the files are stored with a set of metadata, thus facilitating retrieval, indexing, searching, etc. The simulation of a directory structure allows application developers to interact with the service as they would with a physical file system. A simple mechanism for file versioning is provided. If version control is enabled and an existing file path is overwritten with another file, the first file is not erased but a new version is generated. Similarly, an erased file can be retrieved using the "restore" function of the API. In addition to being organized hierarchically, files can be organized with taxonomies using text tags, which facilitates the semantic search for information and makes the service more efficient.

The following information is stored for each file present in the system:

- Its virtual path as a complete name and a reference to the parent directory.
- Its length or size in bytes.
- An array of tags to organize the information semantically.
- A set of metadata.
- Its md5 sum to confirm correct transfers and detect equality between versions.
- Its previous versions.

Web services are implemented using the web application framework Tornado[1] for Python. While Python provides excellent maintenance and fast-development capabilities, it falls short for intensive I/O operations. In order to keep file uploads and downloads optimized, the APIs rely on the usage of the Nginx[2] reverse proxy for the actual reads and writes to disk. The actual file content is saved in a distributed file system so that the service can scaled, and the workload is distributed among the frontend servers by a load balancer. The structure of the service allows migrating from one distributed file system to another without affecting the client applications.

File metadata and folder structure are both stored in a MongoDB[3] database cluster, which provides adequate scalability and speed capabilities for this application. Web service nodes deploy Tornado and Nginx as well as the distributed file

[1] http://www.tornadoweb.org/
[2] http://nginx.org/
[3] http://www.mongodb.org/

Table 1 Restfull web services exposed by FSS

REST Web Call	Description
PutFile	creates a new file (or a new version of an existing file) in response to a request containing the file and basic metadata (path, name and tags) in JSON, structured in a standard multipart request.
Move	changes the path of a file or a folder
Delete	deletes a file. Can include an option to avoid the future recovery of the file, erasing it permanently
GetFolderContents	returns a JSON array with a list of the immediate children nodes of a specific directory.
GetMetadata	returns the metadata set of a file or directory providing its identifier or full path.
GetVersions	returns the list of all the recoverable versions of a file.
DownloadFile	returns the content of a file (a specific older version can be specified).
Copy	creates a copy of a file or a recursive copy of a folder.
CreateFolder	creates a new folder given its path.
DeleteVersion	permanently deletes a specific version of a file.
Find	returns a list of the children nodes of a folder (recursively).
GetConfiguration	retrieves the value of a configuration parameter for the application.
SetConfiguration	sets the value of a configuration parameter (e.g. enabling or disabling version control)
GetSize	retrieves the size of a file. If a folder path is passed, then the total size of the folder is returned.
RestoreVersion	sets an older version of a file as the newest.
Undelete	restores a file.

system clients (GlusterFS[4]/NFS), and the access to the MongoDB cluster that can be located either within or exterior to the nodes.

2.1.2 Object Storage Service (OSS)

The OSS is a document-oriented and schemaless database service, which provides both ease of use and flexibility. In this context, a document is a set of keyword-value pairs where the values can also be documents (this is a nested model), or references to other documents (with very weak integrity enforcement). These documents are grouped by collections, in a manner similar to how tuples are grouped by tables in a relational database. Nevertheless, documents are not forced to share the same structure. A common usage pattern is to share a subset of attributes among the collection, as they represent entities of an application model. By not needing to define the set of attributes for the object in each collection, the migration between different versions of the same application and the definition of the relationships among the data become much easier. Adding an extra field to a collection is as easy as sending a document with an extra key. A search on that

[4] `http://www.gluster.org/`

key would only retrieve objects that contain it. The allowed types of data are limited to the basic types present in JSON documents: strings, numbers, other documents and arrays of any of the previous types.

As with the FSS, the web service is implemented using Python and the Tornado framework. By not managing file downloads or uploads, there is no need to use the reverse proxy that manages them in every node; therefore Nginx is used only to balance the workload at the entry point for the service.

Table 2 Restfull web services exposed by OSS

REST Web Call	Description
Create	creates a new object inside a collection according to the data provided. It returns the created object, adding the newly generated identifier. If the collection does not exist, it is created instantly.
Retrieve	retrieves all objects that match the given query.
Update	updates an object according to the data provided (the alphanumeric identifier of the object must be provided).
Delete	deletes all objects that match the given query.

2.1.3 Identity Manager

The Identity Manager is in charge of offering authentication services to both customers and applications. Among the functionalities that it includes are access control to the data stored in the Cloud through user and application authentication and validation. Its main features are:

- Single sign-on web authentication mechanism for users. This service allows the applications to check the identity of the users without implementing the authentication themselves.
- REST calls to authenticate application/users and assign/obtain their roles in the applications within the Cloud.

3 Warehouse

Warehouse is the first non-native application that has been developed for the +Cloud platform. This tool makes intensive usage of both the file and object storage services and it serves the purpose of being the first real-application test for the developed APIs

3.1 Functionality

Using last-generation standards such as HTML5 and WebSockets[5], the tool allows storing and sharing information using the cloud environment. The user interface is shown at Fig. 1. The available mechanisms for file uploading include HTML5's drag&drop technique.

[5] `http://www.websocket.org/`

Fig. 1 Snapshot of the user interface

Every user has a root folder that contains all their data. The information stored by a user can be shared with other users through invitations for specific folders or using the user's "Public" folder. It is also possible to create groups of users, which work in a similar way to e-mail groups, in order to allow massive invitations. The user interface is updated asynchronously by using WebSockets. The changes made by a user over a shared resource are automatically displayed in the browsers of the other users. The application has syntactic and semantic search capabilities that are applied to different types of files (text, images or multimedia) due to the extraction and indexing of both the textual content and the metadata present in those files. Furthermore, the search results are presented next to a tag cloud that can be used to refine the searches even more. Finally, the application allows users to retrieve and manipulate different versions of their files. This function is powered by the mechanisms present in the underlying file storage API that has been previously described.

The contents of the files and the file system structure are stored in the FSS. Additional information is necessary to establish relationships between the data and to maintain the folder-sharing logic. This extra information is stored in the OSS. Due to the scalability and high-performance of the APIs, the application can execute tasks that will mainten the referential integrity of its model and the high number of recursive operations that are necessary to move and copy folders.

When a user creates a folder, three properties are assigned to it automatically: (i) Host user: keeps permissions over the folder. A number of operations are reserved for this user: move, delete, rename and cancel sharing; (ii) A list of invited users, initially empty; and finally, (iii) A list of users with access privileges to the file, initially containing only the host user.

The basic behaviour of the sharing algorithms is shown in the state diagram in Fig. 2. These algorithms are capable of multi-level folder sharing: a children folder of one shared folder can be shared with another list of users. This second group of users will only be allowed to navigate the most-nested folder.

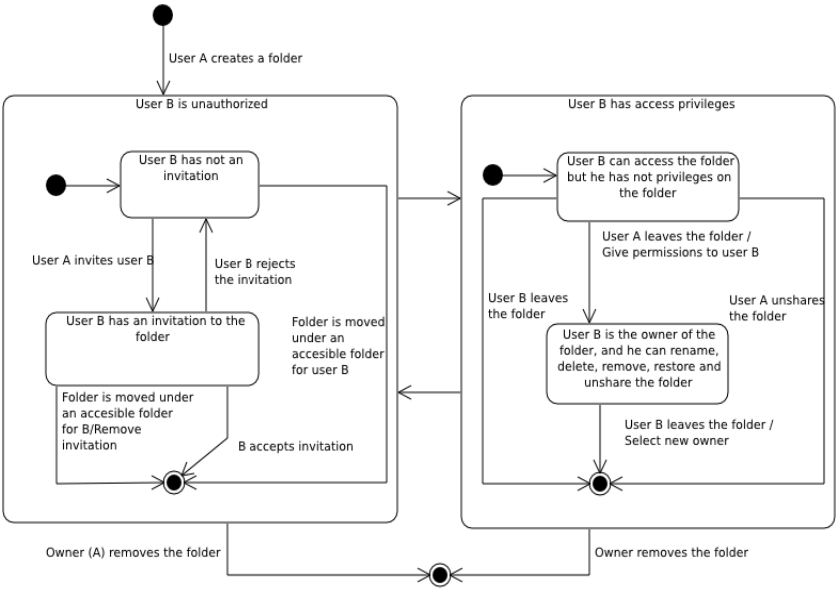

Fig. 2 State diagram for sharing

The actions related to folder sharing include:

- Invite: adds a user to the list of invited users.
- Accept or decline invitation: If the invitation is accepted, the user is added to the list of access-allowed users. Otherwise, the target user is removed from the list of invited users.
- Leave folder: the user that leaves the folder is removed from the list of access-allowed users. If the host user leaves the folder, the folder will be moved to another user's space and that user will be the new host. If there is more than one user remaining, the current host must choose which user will be the new host.
- Turn private: this operation can only be executed by the host user, and deleting all invitations and resetting the access list.
- Move: if the host moves the file, the other users will see a change in the reference to the shared folder. If the operation is done by another user, then only the reference of that user is modified (no move operation is performed).
- Delete: only the host can execute this operation. The shared folder can be moved to the space of another user, or be completely removed.

4 Conclusions

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. 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. 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.

Acknowledgments. This research has been supported by the project *OVAMAH* (TIN2009-13839-C03-03) funded by the Spanish Ministry of Science and Innovation.

References

1. Buyya, R., Yeo, C.S., Venugopal, S.: Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. Department of Computer Science and Software Engineering (CSSE), The University of Melbourne, Australia, pp. 10–1016 (2008)
2. Peng, J., Zhang, X., Lei, Z., Zhang, B., Zhang, W., Li, Q.: Comparison of several cloud computing platforms. In: 2nd International Symposium on Information Science and Engineering, ISISE 2009, pp. 23–27. IEEE Computer Society (2009)
3. Mell, P., Grance, T.: The Nist Definition of Cloud Computing, pp. 1–3. NIST Special Publication 800-145, NIST (2011)
4. Heras, S., De la Prieta, F., Julian, V., Rodríguez, S., Botti, V., Bajo, J., Corchado, J.M.: Agreement technologies and their use in cloud computing environments. Progress in Artificial Intelligence 1(4), 277–290 (2012)