A common API for delivering services over multi-vendor cloud resources

Luís A. Bastião Silva*, Carlos Costa, José Luís Oliveira

University of Aveiro, DETI/IEETA, Portugal

**Abstract**

The increasing pace of evolution in business computing services leads enterprises to outsource secondary operations that are not part of their core business. The cloud computing market has been growing over the past few years and, consequently, many cloud companies are now offering a rich set of features to their consumers. Unfortunately, those cloud players have created new services with different APIs, which imply that cloud-oriented applications might be instantiated in one single cloud provider. This scenario is not desirable to the IT industry because their applications will become provider-dependent. In this paper we present a platform that allows applications to interoperate with distinct cloud providers’ services using a normalized interface. The proposed approach provides a common API that minimizes the present deficit of cloud API standardization and provides secure and redundant services allocation. Moreover, services from different cloud providers can be combined and decorated with additional functionalities like, for instance, redundancy and ciphering on-the-fly.

1. Introduction

The increasing pace of evolution in business computing services leads enterprises to renew their way of operating. Business requirements have also changed and outsourcing has been adopted by many industries, allowing the enterprise to focus more on its core business (Bieberstein, 2006). The cloud computing market has grown significantly over the past few years and, following the natural progress of business models, there is a great interest in the IT industry in migrating services to this kind of infrastructures (Hajat et al., 2010; Leavitt, 2009). In order to respond to this demand, many cloud companies increasingly offer new features to their consumers--using a rich set of services on the server side, and/or providing a platform where users can more easily deploy their applications. For instance, Amazon Web Services (Amazon, 2011) has released services such as Simple Storage Service (S3), Amazon SQS, SimpleDB and many others.

However, despite this evolution, most cloud companies have been creating distinct APIs for their services, which implies that the applications developed for the cloud can only be instantiated in one single provider, i.e. they are locked to each vendor. To achieve portability and interoperability, it is clear that stockholders, i.e. the user community and the marketplace, must adopt a common design (Lee, 2010). Some efforts have been made to create standards, in order to grant interoperability among the various cloud providers, e.g. SNIA (Association, 2011b) and CDMI (Association, 2011a). The desired scenario consists of creating specifications and interfaces that could be used by all cloud service providers. However, these standards are still drafts, with reduced practical impact. Furthermore, several efforts have been made to standardize Infrastructure-as-a-Service (IaaS), while the Platform-as-a-Service (PaaS) still does not have any consistent unified API.

To tackle this issue, a notable effort has been made by several communities, such as jclouds (jclouds, 2010), libcloud (libcloud, 2011) and simplecloud (simplecloud, 2011), to provide a unique programmatic API to deal with multiple cloud solutions. Unfortunately, they do support a limited number of providers and the extension to new ones is not simple. Most of them still focus on IaaS, and there is still a gap in the standardization of other services like storage, database and notification services. The transparent combination of multiple cloud resources will allow applications to communicate with any cloud, even with different providers at the same time, leading to the Sky computing concept (Keahney et al., 2009). Although this paradigm was initially used just for IaaS, recently several efforts have been made to extend it to PaaS (Petcu et al., 2011; Di Martino et al., 2011). However, Sky computing for PaaS is still not fully developed and only possible architecture has been briefly discussed until now.

Standardization may solve many problems in Cloud computing, but there are also some challenges that must be considered during the process. Several usage scenarios imply dealing with critical information, and applications need to manage access to resources to avoid data tampering. Those privacy concerns are a real problem for some corporations as there is information that they intend to keep safeguarded (Kumbhare et al., 2012). Moreover, Amazon released a new service named AWS Storage Gateway, where the
main idea is to cipher and decipher safeguarded data over the cloud, i.e. the data is secure encrypted/decrypted in house. However, it is an Amazon service and it does not work with other cloud providers.

This paper presents a platform that allows client applications to easily interoperate with distinct cloud providers, combining and decorating services like, for instance, dynamic data storage across multiple and incompatible infrastructures. Throughout the paper, we will describe a set of APIs related to cloud services, focusing on storage, columnar databases and publishing/subscription, i.e. the principal resources consumed by Internet applications. Finally, a case study will be presented and a discussion of the advantages and drawbacks of the solution will be provided. The rest of the manuscript is organized as follows. The next section will give a background of Cloud computing, giving special emphasis to storage, columnar data and publishing/subscription services. Also, standardization and interoperability will be discussed. Section 3 presents the system architecture and discusses the proposed abstractions. Section 4 presents a case study, i.e. a clinical solution instantiated with this platform. Section 5 presents a discussion of advantages and drawbacks of the proposed solution, as well as a comparative analysis of other solutions. Finally, the main conclusions of the paper will be presented in a summary in Section 6.

2. Background and related work

2.1. Cloud computing services

IT solutions have been mostly supported by on-premise software, i.e. hosted on private enterprise datacenters. With the emergence of cloud computing, off-premise software has increased its applicability and nowadays applications are offered as services through cloud providers. Thus, cloud computing has been adapted to customers’ requirements, creating dissimilar development models among cloud providers. Clearly, there are self-services offered by these companies, where resources are available at anytime and anywhere for customers. Moreover, the resources can be distributed over multiple locations, in order to improve reliability. In the following sections, we will focus on the description of three of these services: Storage-as-a-Service, Database-as-a-Service and Notification Service.

2.1.1. Storage-as-a-service

Storage-as-a-Service (SaaS) is the ability to offer remote storage in a local virtualized way, for any operating system and application. Nowadays, Cloud providers are offering storage using the Blobstore concept, which, per se, is not new. In the past, these concepts were used in Database Management Systems (DBMS) in the storage and movement of large data blocks. Blobstores are associative memories, i.e. key-value storage providers, where the blob is unstructured data (value) stored in a container and the lookup is performed through a text key. A container is a namespace or domain for the blobs. A blob is always uploaded to a container. The blobstores have a list of containers where the developer can create, remove or rename them. The container holds content, which can be blobs, folders or virtual path. Also, the blobstore in cloud services has an access control list to authorize people to access the data.

In practice, blobstore service allows customers to store data in a container under the Cloud. For instance, Amazon S3, Microsoft Azure and OpenStack have their own blobstore APIs. These services are considered PaaS because they allow developers to take advantage of remote storage service to support their application data in a transparent way. There are many examples of SaaS usage, for instance, the Dropbox application which stores customers’ files in Amazon S3 or commercial web portals that store great quantities of pictures in cloud blobstores.

2.1.2. Database-as-a-service

Database-as-a-Service (DBaaS) is a new paradigm that outsources the burden effort to the cloud provider. Therefore, the database is hosted in a remote datacenter and can be shared between users, in a transparent way. For instance, Amazon AWS, Windows Azure and Rackspace (2011) offer a database as a service where customers pay for what they use. There is a new type of databases, called columnar data, which are organized in a key-value structure. These databases store the information by column, instead of the traditional ones that store the information by row. It has several advantages in computing large amounts of information and cloud providers are now offering these databases. All database operations are supported by these services, for instance, creating tables, loading and accessing data in the tables. Cloud players often supply an API to access the database, and execute operations through a web service API. Furthermore, database maintainers do not need to worry about the server’s redundancy, upgrades, back-up plan and recovery from disaster. Nonetheless, some enterprises are concerned about ensuring data privacy. In fact, this is one of the weaknesses of DBaaS. Despite the provision of Server Level Agreements (SLA) by Cloud Providers, there are legal issues that need a very high level of privacy and confidentiality and these organizations do not have clear data in any case. Also, store procedures and triggers might not be supported in the overwhelming majority of cloud providers supplying DBaaS. Finally, performance might deteriorate because applications will access the data in remote datacenters when located in the public cloud provider.

2.1.3. Notification service

Most information systems and computer applications rely heavily on updated and real time information. In addition, many different applications need to exchange information, i.e. the capability to publish a message from an application and immediately deliver it to other applications that subscribe the same channel. Nowadays, many applications provide this functionality supported on a polling strategy, i.e. checking periodically for new messages. They supply these features in a transparent way to the end-user, although there are extra computational resources requirements and in some cases without the expected efficiency. Cloud providers were influenced by this tendency and have also created services to communicate through a message-based model. Notification services refer to the ability to communicate with another remote entity through Cloud Services. There is some variety of communication services, but essentially they use the Publish/Subscribe model. Several companies are working on notification services, i.e. Amazon SQS (2011), PubNub (PubNub, 2011) and Azure Queue (Corporation). All of them have the same concept: a web service that sends notifications to hanging users through an event-driven workflow.

2.2. Interoperability and standardization

Nowadays there is great competition in the industry to provide better and more services over the Cloud. However, the services provided by different players are not typically compatible (Petcu et al., 2011; Shan et al., 2012; Dillon et al., 2010b). Undoubtedly, interoperability and portability is required to allow applications to be ported more easily between different cloud providers.

Recently, several groups have been formed to create standards and common interfaces that could allow interoperability between distinct cloud solutions (Parameswaran and Chaddha, 2009). For example, Storage Network Industry Association (SNA) (Association, 2011b) has been working on a storage data standard
in the cloud (Association, 2011a). This standard explores the features that vendors are offering and extracts the common domain, aiming for a high quality cloud storage interface in the future. Also, they focus on the financial paradigm of “pay-as-you-go”, considering that such attributes will interest many businesses. Cloud Computing Interoperability Forum is another group that aims to standardize cloud computing. They are an open and vendor-neutral organization, which intends their solutions to be rapid, potentiating successful industry adoption. There are also other committees with the same goals, such as Open Cloud Computing Interface, Open Grid Forum and Open Cloud Consortium.

Most of this standardization work has been related to IaaS, while solutions for PaaS are still emerging (Dillon et al., 2010a). Also, little work has been done regarding standardization in multiple vendor clouds for services such as storage, database and communication. Aneka (Vecchiola et al., 2009), for instance, is a computational platform whose main goal is to support multiple programming models by using specific abstractions for virtual machine deployment. Although cloud services have been used, multi-vendor integration still needs unified specifications (Zeng et al., 2009). Nimbus is an open-source project that aims to offer a Sky computing framework. The main goal is to connect several nodes from different providers, allowing communication between them, transparent migration of machines and ubiquitous management. Nimbus raises the concept of IaaS gateway, working like a deployment orchestrator that allows interoperability to be maintained between several clouds, and creating federated cloud computing, known as Sky computing. Nevertheless, this architecture does not have PaaS services such as databases or communication APIs. mOSAIC (mOSAIC, 2011) is another group that intends to propose a standard API for using Sky-computing. They intend to allow interoperability and portability to support IaaS and also PaaS. However, the project has a wide scope and efforts have been made in the IaaS and processing. They are developing an outstanding architecture that aims to support a Sky in the Cloud computing universe. Moreover, they are designing a language and an API for using multi-cloud resources. However, the project is still ongoing and there are as yet no concerns about the privacy of data. Orleans Bykov et al., 2010 is a programming framework that intends to create a model to outsource the processing, storage and a few other services to the cloud. This framework is quite significant, and is a major contribution to PaaS standardization. However, it does not address interoperability and privacy issues. Cloud4SOA is another European project focusing on achieving semantic interoperability between clouds. They are tackling challenges related with the development of Cloud applications, as well as deployment and migration. This project promises to be very relevant for cloud computing interoperability in the coming years. However, it is still a work in progress. There are other proposals focusing on achieving interoperability through semantic technology. Somasundaram et al. (Somasundaram et al., 2012) created a framework that aims to grant interoperability between Eucalyptus and OpenNebula, proposing a resource description, discovery and submission based on a broker that performs this translation.

Although several organizations have come together to constitute interoperability groups, services over the cloud are not yet compatible. Clients from one cloud storage service cannot easily migrate their data to others. For instance, an application developed to use Amazon SimpleDB would not work with Azure Table. It is not flexible and may be expensive if the application needs data stored on different cloud computing platforms. In the following section we propose an architecture that solves this restriction and is able to deal with three cloud services. Moreover, the platform model copes well with upcoming standards and provides support to service combination, decoration and orchestration. For instance, secure and redundant services allocation.

3. A Service Delivery Cloud Platform

We have developed a common API for delivering services over multi-vendor cloud resources, entitled Service Delivery Cloud Platform (SDCP). This platform has three main goals: (1) grant interoperability between different cloud providers, creating an abstract layer for three cloud services; (2) deliver services using multiple cloud resources, including storage, database management and notification systems; (3) provide service combination, decoration and orchestration. The first goal (1) consists of granting interoperability between cloud players in a transparent way. Basically, an application can work with as many vendors as is desired, taking advantage of existing cloud providers. The SDCP allows creation of cloud provider poll. For instance, it can store data in multiple cloud vendors or cloud free services, creating a federate view of all containers. In addition, it enables the developer to have interoperability with other protocols (2) inside private networks. Cloud services of distinct providers can bundled and decorated with extra functionalities like, for instance, data ciphering on-the-fly (3). Moreover, cache and pre-fetching mechanisms are other examples of value-added SDCP services, extremely important to reduce latency (3).

The presented architecture consists of a hybrid infrastructure that allows “Enterprise to the Cloud” and “Enterprise to the Cloud to Enterprise” applications, i.e. communication between two or more different enterprises, using multiple resources from different cloud vendors. The architecture has basically two main components: the Cloud Controller and the Cloud Gateway (Fig. 1). The Controller contains sensitive information and must therefore be deployed in a trustable provider. This separation in the architecture was necessary to support critical use cases. For instance, some information
The SDCP was designed to make it easier to develop and load new application modules using a plugin approach or web service API. As it is possible to see in Fig. 1, the applications are on top of SDCP. The platform is able to deliver new services using the cloud facilities: data store, databases and communication using cloud providers. In order to extend the platform to support different providers and services (e.g. Google, Amazon, Azure, Rakespace, etc.), we have built a specific model whose structural design sustains the use of different modules under the same interface. We have considered a distributed architecture to support multiple accesses to this data, from distinct points.

We will describe each component of this architecture in the following sub-sections.

3.1. Entities

The platform has its own entities that model the system architecture and describe how it is structured. The fundamental entities and associations of the infrastructure are described in Fig. 2:

- Agent – each gateway has to login using an agent account. Basically, agents are the entities situated inside the enterprise that relay the information to the cloud.
- Domain – is a group of agents belonging to the same enterprise or the same trustable group/enterprise group. Thus, only agents of the same domain can communicate or access the data belonging to its domain.
- Provider – defines a cloud provider and credentials to access them. It can be a storage, database or communication provider. These providers also belong to a domain.
- Private Service – external services that can take advantage of Cloud Controller agents and cloud providers. This service will extend the functionality of the Cloud Controller.

These entities are actors and concepts of the SDCP. The domain is a very important concept because it characterizes the trustable model, i.e. models the relationships and the manages the control of the resources.

3.2. Cloud services

This section describes the implemented Cloud services. We will describe the three implemented cloud services and how the abstraction for these services was applied.

3.2.1. Cloud streams

As expressed, the goal of our platform was to use any resources of the cloud without being locked to a specific provider. To implement this feature for storage services, we used an abstraction to write a set of bytes (i.e. blob) into the Cloud storage using typical Input/Output (I/O) streams. The designed abstraction assures provider independency but also makes it easier to extend to other cloud solutions. Two new I/O entities were implemented: CloudInputStream and CloudOutputStream, Fig. 3. These entities are used to read/write in the storage services as a common Java stream mechanism. An important aspect regarding the writing of a blob is the access policy. By default, we assume that the blob is private, although the user can specify an ACL (Access Control List) to give permission to the blob.

A blobstore API has different features implemented in different cloud providers. Although several features are present in the blobstore API, others are not often presented. Our abstraction will not consider these features, and in that case an extension to the platform will be necessary. Nonetheless, we take into account that several features are just used occasionally, and a trade-off was necessary.

At present, most cloud storage solutions do not offer an option to encrypt data when it is uploaded to the cloud. Our platform has an encryption/decryption layer on the client side, i.e. the cipher and decipher operations are executed on-the-fly on the enterprise side, through our abstraction. In that case, it is ciphered with AES (Advance Encryption Standard) algorithm and the key is stored in
the Cloud Controller. On the other hand, multiple cloud providers can be supplied with a list of CloudSocket being blobs written in both and read from the first one that is available.

The developed Cloud Streams extend the IO Java streams. The Cloud socket contains the identifier of the implementation that will be used to call the most appropriate one for a specific service. JClouds (jclouds, 2010) is an open source framework for cloud development that already provides several cloud players, and as such we decided to build the Cloud Streams as an instance of JClouds blobstores. In addition, we implemented our local storage, following the proposed abstraction. Furthermore, new APIs of different blobstore cloud vendors can be easily implemented using the proposed abstraction.

3.2.2. Columnar data abstraction

As in the previous storage service, we have also developed the same generic API upon cloud databases. This aims to create an abstraction to columnar data, for instance, SimpleDB, Azure Table and other cloud databases publicly available. Nowadays there is a new trend to store information in columnar data instead of the traditional relational system. These tables are very dynamic and the developer does not need to pre-define a model, because the structure auto-fits the data.

There were several problems regarding scalability, which have to be solved in this abstraction. For instance, Amazon SimpleDB uses a mainly horizontal scalability, in opposition to Azure Table, which allows control of the vertical partition. Each partition key represents a different node to have the information. This issue was solved through the Table ID, which identifies the Table name, together with the node label or the location label. The idea was to contain generic features that can be applied in many database services. We implemented two of the available APIs, but it will work for other databases. The Java SDK already uses a high-level abstraction for databases, named JPA (Java Persistence API). Although it is widely used with Object-Oriented databases, we decided to follow this standard for two main reasons: JPA is often used by Java developers to abstract the access to databases, and it is easy to keep compatibility with these applications and the chosen API that fits the JPA abstraction. Thus, it was decided to use the same JPA methods and also add other methods that are specific to the Cloud databases, such as create/remove tables (Fig. 4). Also, for representation of the results, we use a library named Guava (Google Collections), which provides very generic Java collections, e.g., the Table collection. Table is a triple values class <R, C, V> data structure, i.e., Row, Column and Value. This representation is perfect to retrieve the results of queries and also to insert new data into the columnar tables. Fig. 4 shows the architecture of the columnar data abstraction. A Select action is executed quite similarly to the JPA method. It uses a small set of the SQL and just conditions are considered. Complex queries with joins will not be considered in this abstraction, since the columnar tables do not support such a feature.

The data columnar abstraction was deployed on Amazon SimpleDB. The API has a different representation from the JPA abstraction. For instance, each row is called Item, and each item has several attributes that are not structured and can change dynamically for each item. That is why it is called columnar data, because it can be different for each row, i.e., each record can contain different fields. Thus, when creating the table, we do not define a structure as in a common database.

The SimpleDB uses a REST to supply the programmatic interaction with the developer and the results are retrieved in XML files with the responses. So the first step was to create the XML parsers and client communication with the REST AWS interface, as described in their specification (2011). The abstraction for this service is quite similar to the Cloud Streams, and any specific implementation has to be compliant with interfaces described in Fig. 4. In the SimpleDB case, we implemented all functions documented in the abstraction. The model copes with the common API with minor conversions.

3.2.3. Notification abstraction

The notification abstraction aims to dynamically create a message-based communication, based on the Publish/Subscribe mechanism. It is asynchronous, allowing application delivery using this platform to tackle the polling issue often implemented in many applications to simulate an asynchronous system. However, not many Publish/Subscribe public services use only HTTP. We will take the example of PubNub (2011), although a new instance can be implemented, for example using other public services such as Channel API of Google AppEngine, or other protocols like XMPP which support the Publish/Subscribe mechanism. Moreover, the polling approach can also fit the abstraction, and in that case the subscriber has to poll the server until it has a signal message, and then it will call the Receiver callback. Also, for instance, Azure Push Engine (APE) can be installed in a public cloud provider like Amazon EC2, and the service can be used with quite similar behavior to PubNub. Nevertheless, there are also very similar services based on the Publish/Subscribe model, for instance, Amazon SQS and Azure Queue.

In this service abstraction, we used an Observer Pattern, and in the current implementation we created two entities: Publish and Subscribe (see Fig. 5). The channel represents the domain of each agent, and it assures that the communication can only be established between agents of the same domain.

It is important to mention that PubNub specific implementation is quite analogous to the one proposed. So the abstraction classes will call the implementation of PubNub directly using the adapter pattern, similarly to the other previously presented abstractions.

3.3. Cloud Controller

The Cloud Controller is a major component of our architecture responsible for functionalities such as: aggregating providers’ credentials; controlling access to cloud resources; managing
authentication processes with Cloud Gateways; and addition of new services.

This controller provides an API that can be used by third party applications to access their services. The Controller communicates through HTTP, using RESTful specification; thereby it will be much easier for other entities to access services. The Cloud Controller allows us to store credentials of cloud providers for different services, such as blobstore, database and communication (Fig. 6). Also, the ciphered keys used to cipher and decipher the blobs are stored in the Cloud controller, unless the developer explicitly denies the action. Moreover, it also supports addition of external services used by third party applications, extending in this way the Cloud Controller functionality. This platform was instanced with several end-user services associated with Medical Imaging (Repository Data Privacy), particularly the safe storage of medical data in multiple cloud players as described in Section 4.

There is critical information in diverse scenarios. In such cases, the developer can create a new service in a private cloud to keep the more restricted access data. Our platform will be compatible with public or private clouds. Moreover, the Cloud Gateway can cipher the data before sending it to the cloud, and store the keys in these private services.

3.3.1. RESTful API

The interface to external applications is issued as a RESTful web service that provides several interfaces, starting with an authentication mechanism. User validation is based on username and password and if the login is valid, the web service returns a token that will be used to validate subsequent operations. We created functions to get cloud provider and services information.

3.3.2. Dashboard panel

In addition to the web services API, the Cloud Controller also provides a web portal interface (Fig. 7), whereby administrators can add or remove new cloud providers (storage, database, services, etc.) and also check the operation’s logs. This portal was implemented through GWT (Google Web Toolkit) technologies. Also, they can create new domains, add/remove/ban agents and add new services. This dashboard also allows the user to setup a threshold of cloud provider requests because the actions of gateways cloud interactions are sent to the Cloud Controller.

3.4. Cloud Gateway

The Cloud Gateway is a very important component of the architecture. Basically it is an application that loads new services dynamically. It grants authentication from the Cloud Controller and automatically loads the services that are uploaded by the user. Cloud Gateway can run as a daemon. Also, it has an optional external GUI that allows the user to load new plugins/applications or see operation logs. For instance, new adapters for new cloud providers are loaded in the Cloud Gateway.

The architecture of Cloud Gateway (Fig. 8) also uses the SDCP-SDK. Namely, it has access to the plugin core mechanism to load new plugins. Moreover, the interfaces used in API plugins will be instantiated automatically using the Inversion of Control pattern. The plugins to the Cloud Gateway can be services programmed in Java, directly using the SDCP-SDK, but we offer the possibility of external applications, sending information to the cloud through a web service interface. This raises a question: what is the advantage of using the web service API? Third-party application will be allowed to store, access, and use resources from multiple public clouds, using a normalized interface. Thus, third-party applications do not need to be coupled as a Cloud Gateway Java plugin.

Nearly every web application requires an authentication system. The Cloud Gateway is the middleware layer that allows access to cloud resources, and thereby it requires a user validation system. The Cloud Gateway authentication is used through the RESTful web services that access the Cloud Controller web services, previously described. When the gateway application starts, it requires a username and password for the end-user. Next, Cloud Gateway executes authentication and saves the token.

3.5. SDCP-SDK

The end users of the SDCP are allowed to develop applications that use the cloud resources, as well as new plugins to new cloud providers. Thus, the applications can also take advantage of other cloud providers that the developer wants to support. To create these new applications, the developer will use the SDCP-SDK.

The SDCP-SDK defines contracts and specification of the platform, including the communication between the Cloud Controller and the Cloud Gateway. The platform was developed in Java through a set of interfaces. The main idea is that the developer
can take advantage of SDCP-SDK to delegate the authorization process to the platform. Also, the access to the cloud resources is provided by the SDK. The new application will be deployed in Cloud Gateway, the entity responsible for loading the applications. On the other hand, the abstractions of blobstore, columnar data and notification systems are also possible to extend using the SDCP-SDK. For instance, it is possible to the developer write a new plugin for a specific provider based on the SDCP-SDK, only implementing the methods described earlier in Section 3.2. We developed a plugin for notification system based on PubNub in 8 h and now all developed applications with SDCP will benefit of this provider.

3.6. Privacy and confidentiality model

Undoubtedly, cloud computing has several advantages for enterprises, but two major issues need to be addressed: the cost/benefits of the solution and the privacy and confidentiality of the data stored over the cloud.

The first issue depends on the business, and several studies (Armbrust et al., 2009) have been addressing the financial impact of cloud computing. Often associated with data tampering, privacy aspects are still a challenge in these scenarios. Our platform takes those two aspects into consideration, because we can store the information in multiple cloud players and, at the same time, we also tackle the privacy and confidentiality issue. The solution architecture was built taking into account that particular requirement. Our cloud has two main components: Cloud Controller and the cloud players. Thus, for instance, in storage service, we have the opportunity to store the information in a ciphered way.

At present, most cloud solutions do not offer an option to encrypt data when it is uploaded to the cloud. Some companies are already offering this service, for instance, AWS Storage Gateway, but we believe this should be a client service to give more confidence in the cloud solution. Our proposed platform has an encryption/decryption layer on the client side, i.e. the cipher and decipher operations are executed on-the-fly on the enterprise side.

Moreover, this privacy issue is independent of the cloud vendor and the data can be easily sent and accessed in multiple cloud players at the same time. The end-user can do that more easily when writing, specifying a list of cloud providers they intend to use. Use of a common interface should be adopted by services and this will be a contribution to Cloud resources standardization. Another important issue regarding the architecture is that it can be deployed in a hybrid infrastructure. Nonetheless, the Cloud Controller can be deployed in a public or private cloud. Several applications may want to extend the functionality of the Cloud Controller because it may be relevant to host some information in the public cloud.

4. Case studies

4.1. Medical imaging repository

Several use case scenarios that will benefit from the proposed solution can be pointed out. In this article, we will describe one such example: medical imaging repositories. Over the past two decades, the healthcare sector has increasingly adopted ICT to support diagnoses, treatment and patient care. Medical imaging, for instance, produces a huge amount of information and takes advantage of these technologies in daily diagnostic procedures. PACS (Picture Archive Communication System) encompasses several hardware and software technologies for acquisition, distribution, storage and analysis of digital images in distributed environments. The main components are: image acquisition devices, storage archive units, display workstations and databases. The amount of medical images has increased significantly over the last decade as result of the increase in the number and quality of studies. According to some researchers, this trend will continue over the next years. A common PACS archive has two major components: the DICOM (Digital Imaging and Communication in Medicine) object repository and the database system. The object repository typically demands an infrastructure with huge storage capacity to support all DICOM studies. The database module is normally a relational database management system (RDBMS) that supports the DICOM Information Model (DIM) (DICOM-P3, 2001), containing mandatory metadata related to patients, studies, series and images.

Medical institutions have to store a large number of medical studies/images, i.e. DICOM object files. Thus, they need to have large datacenters inside the hospital, a major source of problems for systems’ administrators. The Cloud Computing model fits this scenario. Moreover, healthcare is a critical service and the information must always be available. Therefore, the approach described in this paper is crucial to create solutions that can be instantiated in more than one cloud provider.

Based on the SDCP-SDK presented, a PACS Cloud archive was developed supporting medical image storage and database services. A PACS Cloud Gateway to the Cloud (Bastião et al., 2011) was developed, which provides outsourcing of these two components to the cloud, namely using the new concepts of blobstore and database accessible through web services. Internally, the Gateway is seeded as a common Intranet PACS repository supporting DICOM standard services.
A Cloud Gateway plugin was instantiated and also another component named PACS Cloud Controller to keep sensitive medical information. Those PACS modules were both deployed to interact with two distinct cloud providers, namely Amazon S3 (S3) and Google Storage (Bastiao et al., 2011).

This solution is used to support a distributed archive over a medical institution with multiple centers. The medical repository is deployed over a private cloud, with two distinct gateways supporting the access to the archive. There are an average of 250 exams daily for this archive. Our solution will also replicate a given percentage of the archive to a public cloud provider, in a ciphered way, in order to achieve higher availability taking benefit of the SDCP features, i.e. writing in two distinct cloud providers using our common API. Due to some technical reasons or natural disasters the private cloud might be offline for a certain amount of time. If the solution is also deployed in a public cloud, medical data will be not lost and querying the repository will still be possible. Moreover, since we are working with huge volume of data the latency is one of the main barriers to the adoption of cloud services in medical imaging environments. The SDCP gateway plugin includes cache and pre-fetching mechanisms increasing considerably the quality of service provided.

4.2. Sharing medical images “anytime and anywhere”

Cloud computing is largely used to share files over the Internet, and many examples can be pointed out, such as Dropbox (Dropbox, 2011) and Gmail (Google, 2011). Moreover, Cloud providers offer high availability and scalability of their services. In this case study, we do not outsource the medical repositories to the cloud. Instead, we keep them in the healthcare enterprises, but the goal is to grant external and/or inter-institutional access over organization boundaries. Our DICOM relay architecture takes advantage of the SDCP platform to provide transparent exchange of imagiologic information between several locations.

Communication between the components of the digital medical laboratories is mainly carried out through DICOM. An application deployed in the Cloud Gateway was developed to effect a DICOM protocol forwarder via cloud SDP. This DICOM relay implements the DICOM protocol and waits for new requests. For instance, when it receives storage requests for medical exams, it forwards the messages to another application that is running in another institution. In order to achieve this, the application relies on storage and notification services offered by the SDCP-SDK.

Currently, this solution is supporting teleradiology in a medical institution, with two distinct locations. Physicians are reporting from several places, using the same central medical repository and the DICOM Router to access medical imaging. The solution is online 24/7 and is supporting transferring around 50 exams daily. Our Bridge was deployed in a local private cloud. The direct connection is frequently blocked or needs to be set up in the network, unlike our transparent Web 2.0 approaches.

5. Discussion

5.1. Abstractions

The SNIA project has developed notable work in cloud storage standardization. A large set of important features are supported in the blobstore, including the methodologies to transfer objects to the cloud provider, the mechanism to request and get information and what kind of metadata can be supported. Furthermore, OpenStack has been adopting the SNIA design principles and companies like HP Cloud Services and Rackspace are using OpenStack to support their API infrastructure. However, there are still other players not offering SNIA compatible, which makes the presented platform and the proposed abstractions worthwhile.

“Database-as-a-Service” is also a very valuable service offered by cloud players. However, standardization at this level has not yet been developed. Nonetheless, much effort has been made over the years to make interoperability between databases possible. In turn, implementing several drivers for each provider could solve the problem, using solutions like JPA, which we proposed in SDCP. Although this may be true, we made several contributions to the standardization of these interfaces. However, if organizations like SNIA extend their work to databases, interoperability will become much easier.

Regarding the asynchronous systems for notification, there are no standards yet. Similar to database abstraction, we present a contribution toward standardization, as well as creating an abstraction for it. XMPP is a good standard for Publish/Subscribe and many applications use XMPP as a communication service. Nevertheless, the protocol was written to be a chat and not for notification purposes. Moreover, although the XMPP runs over HTTP, the service is not commonly offered as a web service. Thus, a possible approach to standardization in this service will be achieved by using the Publish/Subscribe pattern with multiple channels to communicate.

5.2. Advantages and drawbacks

Despite the decoupling of the cloud players, the platform still has some weaknesses. Introduction of an abstraction, as a middleware, does not allow support of all features of all cloud providers. Cloud solution vendors are continuously competing for new and distinct features, so they can build a solid position in this growing market. Despite this competition is quite positive for customers, it also raises several problems mostly related with the lack of interoperability. This lack of normalization between interfaces can be easily overpassed with SDP, which takes a conservative view of available services—keeping the maximum common set. However, if these services are not enough, and developers want to take advantage of a specific service for on provider, they can use the service directly through the vendor API, or they can create a plugin through SDP. For instance, common features presented in blob structures, follow a key-value strategy inside a bucket. If a particular provider decided to support blob compression, since the SDCP API do not include that feature, it will be necessary to use the original service API or specific. Our approach explores several APIs and proposes an implemented platform that is able to support many vendors and easier to extend to other new providers.

Using cloud computing, the risk of incidents is reduced because cloud providers have the data in multiple locations. However, there is another risk that developers have to consider: what happens if cloud-computing providers stop supplying their services? Such a situation will certainly harm cloud clients. The proposed approach will greatly minimize those risks, because the data can be redundantly stored in multiple cloud providers, without impact on SDCP API client applications. Moreover, it can automatically forward the resource to another provider, if a cloud provider starts failing.

SDP normalized interface over services of distinct providers, is an important advantage but not the unique one. Providers can be selected on-demand according to client predefined rules or quality parameters. The multi-provider services can be combined, orchestrated and decorated with new functionalities. The result is a value-added service with a high abstraction level.

6. Conclusion

Despite many cloud service applications having been developed, the interoperability and portability of cloud resources is still a major
challenges, and common specifications and standards are needed to enable multi-vendor integration.

In this paper we have presented a solution targeting this goal. The proposed Service Delivery Cloud Platform (SDCP) is a cloud middleware infrastructure that provides a rich set of services using resources from multiple cloud providers. The presented architecture allows decoupling the specificities of each cloud provider API into a unique abstraction. Throughout the article we have extensively detailed the system architecture, and discussed the advantages and drawbacks of the presented solution. The platform was validated with private and public clouds, using three common services: blobstore, columnar data and notification service. In addition, the platform was successfully used in several medical image scenarios that exploit storage redundancy using two databases, in private and public clouds, and the notification service to communicate between multiple data access points.

Acknowledgements

The research leading to these results has received funding from FEDER under the COMPETE programme and by FCT – Fundação para a Ciência e a Tecnologia under grant agreement PTDC/EIA-EIA/104428/2008 and SFRH/BD/79389/2011.

References


Luís A. Bastião Silva is a PhD student of MAP Doctoral Program in Computer Science of University of Aveiro. He received his Master of Science on Computer and Telematics Engineering in 2011. He worked on medical informatics research with partnerships with several medical institutions. His current research interests include healthcare records, medical imaging repositories, health records data federation, integration and cloud computing.

Carlos Manuel Azevedo Costa (http://www.ieeta.pt/~costa/) is Assistant Professor at the Department of Electronics, Telecommunications and Informatics of the Aveiro University. He is also a researcher at the Institute of Electronics and Telematics Engineering of Aveiro (IEETA) and member of the Bioinformatics group at the University of Aveiro. He holds a PhD in Medical Informatics and he is author or co-author of more than 80 publications in this area. His main research activity is in the area of PACS-DICOM (medical imaging systems and networks) and Healthcare Information Systems. He has also interests in other areas of research such as telemedicine, security and access control.

José Luís Oliveira is an associate professor at the Electronics, Telecommunications and Informatics Department, University of Aveiro, Portugal. His research interests include text mining, information retrieval, distributed systems and computational methods in biomedical informatics. He was involved in more than 20 international projects, such as InfoGenMed, Dadalos, EuroNGI, InfoBioMed, EU-ADR (FP7), GEN2PHEN (FP7), RD-CONNECT (FP7) and EMIF (IMI). He has more than 250 publications in book chapters, journals and international conferences.