International Journal of Cloud Computing and Services Science (IJ-CLOSER)
Vol.2, No.4, August 2013, pp. 249~275
ISSN: 2089-3337

OPENICRA: Towards A Generic Model for Automatic Deployment and Hosting of Applications in the Cloud

Gadhgadhi Ridha*, Cheriet Mohamed*, Kanso Ali, Saida Khazri***
*Synchromedia Lab. for Multemedia Communication in Telepresence,
École de Technologie Supérieure, Montréal (QC), Canada
** Ericsson Research, Montréal (QC), Canada

Article Info

Article history:

Received Jun 12th, 2013
Revised Jul 20th, 2013
Accepted Jul 30th, 2013

Keyword:

Cloud computing, Generic Model, Automated Deployment, Cloud Migration, Scalability, Vendor lock-in.

ABSTRACT

Cloud Computing offers a distributed computing environment where applications can be deployed and managed. . It is characterized by its scalability, elasticity and widely-spread use. Although the choice of such an environment may seem advantageous enough, several challenges still remain, mainly in terms of the automated deployment process of applications. This paper focuses on the design and the implementation of a new generic model for application automatic deployment, called OpenICRA, to mitigate the effects of barriers to entry, to reduce application development complexity and to simplify cloud services deployment process. We conducted two case studies to validate our proposed model. Our empirical results demonstrate the effectiveness of OpenICRA to automate and orchestrate the deployment process of different applications without any modification in their source code and optimize their implementation in terms of performance in heterogeneous Cloud environments.

Corresponding Author:

Gadhgadhi Ridha
Synchromedia Laboratory for Multimedia Communication in Telepresence
École de Technologie Supérieure, Montréal (QC), H3C 1K3 Canada
E-mail : rgadhgadhi@synchromedia.ca

1. INTRODUCTION

Cloud computing is a computing paradigm based on the “on-demand” provision of computational resources via the network, with a pay-as-you-use billing model. This concept, which we refer as a springboard for technological innovation, is essentially based on grid computing, SOA technology, and obviously on virtualization [1]. This paradigm is revolutionizing the computing world and is considered as the architecture of the next generation of entrepreneurial activities. For the time being, many companies are seeing substantial interest in the cloud. It is primarily a financial aspect, but it also allows a company to focus on the heart of the business. One of its most attractive features is the on-demand scalability of capacities and resources with no upfront investment. With the advent of commercial solutions such as Amazon EC2 and S3 [2], Google App Engine [3] and Microsoft Azure [4], the cloud has become more than just a concept, it is a reality. In addition, the combination of the latest technologies of cloud computing can considerably reduce costs, ensure scalability and flexibility of services as well as improving application performance. Furthermore, in a cloud infrastructure, users can access applications from any device everywhere, and pay only for the services that they have really used. Due to its advantages, companies are more likely to extend their technical infrastructure by adopting cloud computing. Although the choice of such an environment may seem advantageous enough, developers are faced with many challenges; especially with regard to migration and deployment in the cloud of applications and services, regardless it is a private, public or hybrid cloud. Indeed, the deployment of applications in the cloud is a very complex process given the large number of operations required to enable a successful deployment [5]. The restructuring of each application layer for the cloud, the automating of the deployment process, the auto-scaling of services depending on the demand and the optimization of various application services to take advantage of the cloud benefits are among these operations.

According to the survey conducted by North Bridge [6], vendor lock-in and interoperability are respectively the third and fourth major concern inhibiting adoption of cloud computing. Which brings us to the main questions, how can we avoid vendor lock-in? And how can we ensure the interoperability and portability of applications in the cloud? Similarly, according to HP [7], deployment activities occupy up to 40% of the operator working time, while the repetition of routine tasks occupies up to 70% of network and database administrator's time. This is explained by the fact that most applications use several services and they have a complicated architecture, often composed of a database, a middleware and other configuration components. Also, each service imposes unique requirements on the IT environment that hosts it. In the same way, to deploy an application in the cloud, a good preparation of the target environment is essential to be compatible with its architecture; however, to migrate an existing application to the cloud, a difficult reconfiguration of the connectivity of the application's different entities and the scalability is often required. Further, unlike traditional solutions, where IT services are in a good physical, logical and personnel control, cloud computing outsources the applications and the databases to large data centers of specialized providers. Consequently, the adaptation of existing applications and the development of new services as well as data management are becoming increasingly complex tasks. These processes of migration and deployment constitute a challenge for cloud users, not only with regard to the compatibility and interoperability of applications, but also to the development and deployment complexity in the cloud where the main issues such as the applications scalability, the repetition of recurring tasks and the management of data storage have not indeed well been evaluated.

As reported in [8], there is not enough work in the literature on the support of applications deployment process in the cloud. In order to make the effective use of such an environment, it is interesting to study how we can overcome the challenges and automate the key tedious tasks of applications deployment in a reproducible manner with an optimization of used resources. To contribute to this field of research, our main objective is to design and develop a new generic model of application automatic deployment in the cloud, in order to reduce the complexity of application development, costs and time of implementation; and to simplify the deployment process of services in the cloud. Additionally, supporting and automatically deploying applications in the cloud by assuring elasticity, auto-scaling of applications and interoperability with all platforms are the primary objectives of this paper. More specifically, we aim to answer all the issues raised previously, namely i) vendor lock-in and interoperability of applications, ii) scalability of the deployment model and auto-scaling of applications, and iii) reduction of applications deployment complexity in the cloud.

For this purpose, we begin by developing a Platform as a Service (PaaS) based on open sources technologies. This platform, namely OpenICRA, is a cloud model that will have to provide developers a programming environment, available immediately to automatically create and deploy applications on the cloud, in a record time, with less complexity and constraints at both development and deployment levels. This offers the developers the opportunity to focus on the implementation of the application logic and adding features rather than configuring operating systems, servers and development tools. The focus on the use of open source stacks of software and technology and excluding proprietary solutions allow building an extensible, elastic and operational platform on any cloud environment. This ensures the freedom of choice, including the freedom to deploy any application developed on other PaaS (if the application is not using proprietary PaaS APIs), the independence and the autonomy of the deployment model. By completely isolating the application from the environment of the underlying cloud and hiding the APIs and the complexity of configuring the runtime environment, applications can be easily moved from one cloud to another without restriction or modification of the source code. This ensures the portability of applications both on and off the platform OpenICRA, preventing vendor lock-in on our proposed model.

The auto-scaling technology is necessary to ensure the scalability of applications deployed in the cloud. Indeed, this technology allows to significantly improving resources utilization at the server level and quality of service to end-user level. Therefore, the developer will be able to automatically configure the increase or decrease of the resources capacities based on the traffic or according to the configuration of well-defined rules of scaling. This is possible via a simple administration web interface of our platform. The integration of load balancing technology with auto-scaling also contributes to ensure scalability and elasticity of the OpenICRA platform and thus guaranteeing the QoS.

The automatic deployment challenges are solved by the development of modules and extensible and customizable classes describing the functioning and the dependence of the different components and application services to be deployed on the cloud and by that resolving the problem of the repetition of tedious tasks and allow automating the applications deployment process. The classes developed should include a description of the monitoring and the automatic scaling rules of services and computing resources and storage. This simplifies the deployment, accelerate the implementation of applications and reduce the deployment complexity and errors as well as costs. Furthermore, our model incorporates real time monitoring tools to monitor the evolution of the deployment and the performance of applications deployed by our platform on heterogeneous cloud environments such as Amazon Web Service [9], GSN [10], etc.. By this, we verify the functioning of services and trigger the appropriate corrective methods in case of need.

The remainder of this paper is organized as follows: in Section 2, we survey the litreture for related work. In Section 3 we present the different methods we used to implement our approach, and the details of applications automatic deployment‘s process in the cloud. Section 4 illustrates the assumptions and the design of the proposed platform, OpenICRA. In section 5, we present the storage solution of big data management in the Cloud. Section 6 illustrates two concrete case studies of automation deployment and migration to real cloud environments, and discusses the experimental results conducted to validate our appraoch. Finally, our main conclusions and future work are drawn in section 7.

2. RELATED WORK

As there are no standards that define the use and the development of cloud services, Application Services Providers (ASP) are facing several difficulties in migration, deployment and portability of their applications. Indeed, there is several initiatives of standardization aimed to normalizing the situation, but any consensus has been reached [11]. Numerous solutions have been proposed based on the use of intermediate layers for isolating applications from the variability of some services offered by cloud providers. However, when these tools are proprietary, they can build applications that work only on a specific infrastructure. These approaches are therefore a partial solution to this problem, because they contribute to the risk of moving the effect of vendor lock-in from the cloud provider to the deployment tools. In this Section we survey the various solutions for the deployment problem and discuss their advantages and limitations.

In [12], the authors proposed an approach to migrate “.Net” applications of n-tier architecture to Windows Azure platform. The proposed solution based on a classification of different migration tasks in categories indicating the main factors that affect the migration cost. This proposal is a good initiative to simplify for the developers the migration of their application in the cloud. However, it only provides some techniques to migrate a particular application to be deployed in a specific environment (Windows Azure) and it does not provide any reduction of deployment complexity and requires a proficiency of the application to be migrated and a lot of effort to reconfigure it in the target environment.

The mOSAIC Project [13], is an initiative reference financed by the European Commission under the FP7-ICT program, that propose a solution to offer cloud users the freedom to choose the programming languages as well as the IT resources. It mainly deals the mechanisms normalization of applications development and deployment in the cloud. This work is characterized by its robustness and is based on an open source API and platform for developing applications in the cloud. However, this approach supports only the new applications compatible with cloud environment and doesn't provide any mechanism to migrate existing applications. In addition, it doesn’t include either APIs or tools for automatic deployment and management of applications on the cloud to avoid the recurring administrative tasks.

The Elastic-R solution proposed in [14] is a platform as a service that allows teachers and students the allocation of virtual machines on the cloud through a standard Web browser to use statistical and mathematical environments such as R [15] and Scilab [16]. This solution provides a distributed services for collaboration, resources sharing and interactive teaching techniques that allow educators to create their own cloud e-learning tools. However, it's only compatible with Amazon's EC2 [2] and it's limited to provide only mathematical and statistical computation services based on the R and Scilab environment. Moreover, it is not extensible to support other services and does not offer portability of created services, since they can work only with Amazon Web Services.

In [17], the authors developed a PaaS platform to manage computing resources of university laboratories in a private cloud. The main goal of this work is to resolve the problem of limited access of softwares. Using virtualization and resource sharing mechanisms, the proposed solution allowed to meet the needs of students to remotely access to the applications. However, this solution is implemented in a specific way and adopted to a particular environment. In addition, the architectural model is not generic and cannot be implemented for other IT infrastructure.

In [18], a survey is presented, whose the main goal is to understand how to use efficiently PaaS models in cloud environment. A detailed comparison of different cloud platforms has been exposed based on several aspects such as architecture, features, supported applications, etc. The authors showed through this work the importance of the use of open source platforms. Indeed, they explained that this type of PaaS has the potential to democratize web development by allowing everyone who can use a browser to program, test and easily ameliorate their Web applications. In addition, they demonstrated that an open source PaaS solution adapts with the industry standards and enables applications to be deployed across multiple cloud providers. This investigation promotes and motivates developers to use open source PaaS to benefit from their portability and flexibility characteristics. However, none of the solutions presented in this survey is based on an approach to automatic deployment applications on the cloud.

Other works focus on the analysis of existing challenges such as; the problems of interoperability and portability of applications in the cloud. Various solutions are proposed in this field: Open Cloud Computing Interface [19] is a set of specifications that enables the tools development to perform common management tasks of cloud systems including; deployment, auto-scaling, monitoring and network management. The proposed API by OCCI is supported by various cloud computing projects like Eucalyptus, OpenNebula and OpenStack [20]. This approach of standardization aims to ensure both interoperability that allows different cloud providers to work together without any restriction or reserve, and application portability allowing customers to switch easily between providers based on business objectives (example, cost) to promote competition. It should be mention that OCCI began as an API for managing cloud infrastructures [19] and can be used with other cloud models such as PaaS and SaaS, but the author in [21] affirmed that the OCCI standard is not yet ready for PaaS.

In [11], the authors have shown that there is not enough efforts that have been made in the standardization of development methodologies to ensure portability and compatibility of applications in cloud environments. Furthermore, they explored an alternative solution based on the use of software adaptation (SA) techniques whose different guidelines were presented as the basis of an approach that promotes interoperability and application migration to the cloud thereby avoiding vendor lock-in. This approach can be applied to identify and resolve the sources of mismatch of application deployment code in cloud environments. Despite the importance of the adaptation of software components in different execution environments, this work lack of being integrated into a PaaS platform, so that developers can use this algorithm. An alternative proposal for aplications development based on software adaptation presented in [22]. It allows applications to be deployed indifferently on any cloud environment. The segmentation of the application modules serves to facilitate the deployment and redistribution of the application in heterogeneous environments. Interoperability between interdependent components deployed in different clouds is achieved by automatically generating the necessary communication services. This solution provides developers the portability of their applications and the interoperability of related services. However, it does not offer any orchestration services of tasks, although it uses a deployment mode where application components are distributed in different cloud environments. This takes considerable time specifically at the configuration and establishment of communication services, because they come from different sources and geographical areas. Thus, this approach may increase application developpement complexity.

In [23], the authors described the challenges of cloud computing, considering the industrial efforts and propose an interoperability model between cloud environments. Among these challenges, we mention the need to achieve cloud services integration through the principles of SOA technology. The authors proposed an orchestration service as an intermediate middleware layer to solve this issues. Indeed, portability was seen as a particular challenge of interoperability. From this perspective, the incoherent use of the term "interoperability" to refer to the two concepts can be justified. However, two main definitions were proposed in [24] in terms of interoperability. The first is the horizontal interoperability defined as the ability of communication between two services of the same cloud model (IaaS/PaaS/SaaS). And, the second is the vertical interoperability defined as the ability to deploy a service on another model of lower cloud like a SaaS application ported in different platforms PaaS.

The common factor in all this works is the desire to create an intermediate abstraction layer that decouples applications from proprietary tools imposed by cloud providers. In this case, developers create their applications using this intermediate layer which is an independent platform that hides proprietary APIs of vendors. Thus, the intermediate layer prevents developers from linking to programming languages, file formats or storage of specific data. In summary, some studies proposed an abstraction of specific infrastructure to support applications migration from one cloud to another, but they have not addressed the issues of portability and interoperability. Others have addressed the problem of interoperability by proposing solutions to standardize cloud services. However, no consensus has been established. The difficulties in adopting these standards are the consequences of the lack of providers' acceptance like Google, Microsoft, Amazon and others because they want to offer differentiated services to attract more customers. Furthermore, the definition of standards at IaaS level, which is not an easy task, does not fix the problem of application portability, since each platform has its own interoperability with a specific tools and frameworks. Similarly, most of the proposals aim to create and develop new applications in the cloud, but none of these proposals provides services that allow the migration of existing applications. Eventually, none of these works supports the automation of application deployment in the cloud and there are no orchestration services to avoid repetitive and tedious tasks.

In this context and being built on open source technologies, our proposed model OpenICRA is designed to allow the developer to choose to move from a traditional environment to the cloud and vice versa. For this purpose, only open source languages and standard tools are used without any integration of proprietary API, technology or resources in our model. This ensures interoperability between cloud environments and applications portability both within and outside of OpenICRA, preventing vendor lock-in on our platform. OpenICRA includes a set of command line tools that provide full access to the developer interface. These tools are easy to use and scriptable for automated interactions. This proposed solution also includes a rich web console's interface for management and orchestration in parallel and in real time for all nodes in a cluster, reducing the complexity of applications development and management tasks. Our generic model is able to migrate and deploy existing applications in heterogeneous cloud environments and provides automation services of recurring tasks by triggering actions in real-time in a cluster of nodes. To fully benefit from the elasticity of the cloud, OpenICRA provides horizontal auto-scaling of resources based on the applications load variation which eliminate the manual operations by adding or deleting instances of virtual machines or application containers. The table below give a summary about the main services provided by our proposed platform, OpenICRA, compared with the major existing solutions in the littréture and which related to the field of application deployment and migration in heterogeneous cloud computing environments.

Table 1. Comparison between the main services offered by OpenICRA with those of the major existing solutions of application deployment in the Cloud

Platform / **Services**	**mOSAIC [13]**	**Elastic-R [14]**	**OCCI [19]**	**Cloud Development Framework [22]**	**Amazon Elastic Beanstalk [9]**	**OpenICRA**
Interoperability / Portability	+	-	+	+	-	+
Auto-scaling	-	+		-	+	+
Automatic deployment	-	-	-	-	-	+
Application developpement	+	+/-	+	+	+	+
Existing application migration	-	+/-	+	-	-	+
Storage Management	-	+	-	-	+	+

3. AUTOMATIC DEPLOYMENT OF APPLICATIONS IN THE CLOUD

In the light of the issues and objectives described in the first section, we have proposed a new generic model for the automatic application deployment in the cloud. This model allows the reduction of programming efforts, time and costs required to implement applications using a management and automation functions. This accelerates the deployment process and ensures the flexibility and scalability of deployed applications with high availability. In the following, we will present the various methods used to achieve our goals as well the automatic applications deployment process details.

Our proposed model is composed of three key architectural components: Cloud Manager, Cloud Controller and Datanodes. The Cloud Manager is responsible of the provision of computing resources and storage as well as the execution of tasks requested by the Cloud Controller. The latter is the brain of the model and handles the automation of application deployment, the scaling, the management and the control of the system. The datanodes are the nodes that host the framework templates and applications data. All the comminucation is carried based on the Advanced Message Queuing Protocol (AMQP), hence the components are interconnected through the ActiveMQ queues server, and the client and servers of MCollective orchestration software. The redundancy and scaling methods are used to provide high availability, scalability and extensibility of the model and applications deployed by our proposed PaaS model. The utilization of an open ecosystem of deployment and the integration of frameworks and extensible modules allow the model to ensure application portability and avoid vendor lock-in.

3.1. Automatic application deployment process in the Cloud

In order to reduce the deployment complexity and errors, to accelerate the applications implementation and to reduce costs, we used extensible and customizable modules and classes to automate applications deployment in the cloud, allowing developers to describe their applications, services and their interdependencies as well as monitoring and scaling rules of their services, computing resources and storage. Consequently, the deployment and the configuration management of an application in the cloud becomes a simple process, which occurs in three main steps as shown in Figure 1.

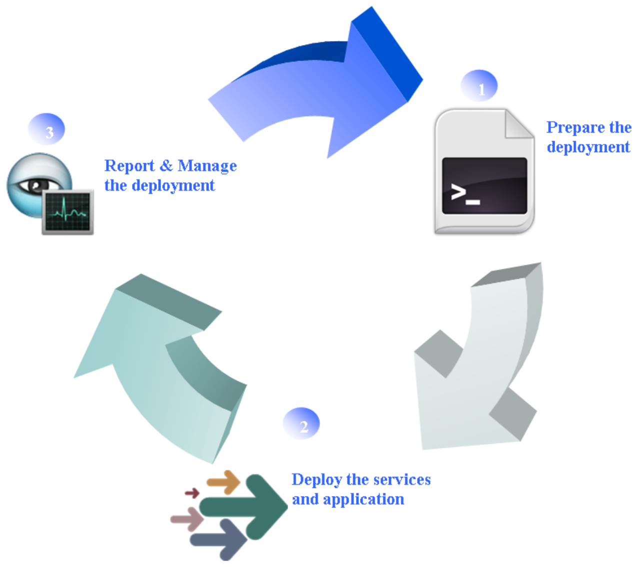

1
Prepare the deployment
2
Deploy the services and application
3
Report & Manage the deployment

Figure 1.Automatic application deployment process

Step 1: Application deployment's preparation

The first step of the deployment process consists to:

- describe the provision of required computing resources to the application deployment in cloud environment (public or private);
- prepare binary files or source code required for application services;
- describe the life cycle of the application and its services in modules and classes;
- describe the monitoring and scaling rules of the application and its services.

Regarding the description of the resource provisioning, life cycle of the application and its services as well as monitoring and scaling rules, the developer can reduce the development time of modules and classes using existing and reusable modules and personalize them as needed. Alternatively, if he has some prerequisite knowledge, he can build a typical module using the declarative configuration language of the Puppet. These modules are used to define the desired deployment state of application and its services in an automatic manner. In addition, the developer can use these modules to deploy distributed applications in physical or virtual cluster nodes as well as in traditional or cloud environments.

Step 2: Automatic deployment of services and applications

The second step is that of the automatic application deployment in the cloud. It consists to automatically trigger the deployment by a single action of a clic on a button using the web console or by typing a simple command using the command line tool "synchro". Among commands used, we cite « *synchro run X -I rgadhgadhi@synchromedia.ca -p qwerty* », « *synchro app create -s icra ctad mysql-5.1 phpmyadmin-3.4 -I rgadhgadhi@synchromedia.ca -p qwerty* », etc. The first command is used to trigger the automatic deployment of an appplication by launching the X script created by the developer, while the second command is used to create a scalable and customizable template of CTAD type with MySQL and PHPMyAmin applications for database creation. This CTAD template type allows the developer to install any application compatible with Linux systems. The "-I" and "-p" options are used for developer's authentication by the cloud controller.

In short, this step consists to:

- automatically provision computing and storage resources from selected Clouds using Cloud manager ;
- auto-install Puppet agents in provisioned containers or virtual machines;
- trigger the application deployment by the Cloud Controller using MCollective application.

After the triggering of deployment by the Cloud Controller, each Puppet agent contacts the server (master) and downloads the latest configuration and applies it in order to achieve the desired and defined state by the developer. The Puppet clients are the installed agents by the Cloud Manager in the provisioned resources (containers or virtual machines), while the Cloud Controller is the one who orchestrates all the components of the platform through MCollective. When clients start to apply the new configuration, they create the template corresponding to the application type that will be deployed and they install its frameworks and services. Subsequently, they set up the application, the monitoring and scaling rules and they ensure that this configuration is synchronized with that desired by the developer. Once completed, each client starts to send the deployment status indicating whether there are errors to be corrected or successful deployment. It is important to point out even for the development of new applications or new components of an existing application, the update can be done in an automatic manner through the differential synchronization techniques between local versions "on-premise" and those in the cloud.

Step 3: Deployment monitoring and management

This final step consists to monitor and manage the deployement using the web console interface or the terminal (CLI) of the platform. Thus, the developer can evaluate the automation and supervise the availability and performance of the application using the default monitoring tool. It is possible to integrate other tools for effective monitoring with open source API of Puppet to generate detailed reports of the deployment result to ensure governance, conformity and control.

3.2. Avoiding vendor lock-in and enabling portability

In order to ensure freedom of choice, including the freedom to deploy developed applications on any other PaaS, our proposed model shall be designed and built using open source software and technology stacks. For this purpose, only open source frameworks and technologies will be used and no proprietary API, technology or resource will be integrated into our automatic deployment model. Based on developer request, the proposed PaaS should provide a higher-level control, since it will have access to the underlying infrastructure. Otherwise, this platform will hide the resource configuration complexity using an abstraction layer and best practices to meet the needs of users who have less advanced requirements. In addition, the integration of computing resource provision tools such as Boxgrinder and VMbuilder with Cloud Manager will allow the designed platform to support various public Clouds (GSN, Amazon EC2, Windows Azure, Rackspace...) and private clouds (OpenStack, VMware vCenter, Citrix XenServer...). By completely isolating applications from underlying cloud environment and hiding APIs and the complexity of configuring the runtime environment, applications are easily moved from one cloud to another without any restrictions at any time. This allows us to achieve our goal of minimizing vendor lock-in.

The proposed platform includes free and unmodified programming languages and frameworks. This means that the developed application on this platform can be easily moved to other environments supporting the same programming languages. For example, Ruby or JBoss applications running on our platform can be moved to independent implementations of Ruby or JBoss in data centers. The developer can deploy his application in the cloud without making changes in the source code regardless of the used software stack (Java/spring, Java EE, Ruby on Rails, PHP...), database (relational like MySQL or Non relational such as MongoDB or Apache Cassandra), or any other component that the application uses. Also, the developer can use its own programming languages and different types of databases by leveraging the powerful of Cloud extensibility and the ability to customize templates if the offered support of preconfigured languages by the platform not suits him. Using the CTAD and Puppet integration with the distributed version control system Git, the developer can deploy in the cloud almost any program or any binary that runs on a Linux or even on Windows platforms. This allows us to achieve our goal of deploying applications in the cloud without any code change with portability guarantee of applications both in our platform or on any other platform, which preventing the vendor lock-in in our generic model of automatic applications deployment in the cloud.

3.3. Redundancy approach

Both stateful and stateless modes can be used to describe whether a machine or service is designed to note and retain one or more preceding events in a given interactions sequence with user, program, or any other external elements. Furthermore, *stateful* means that the computer or program keeps track of the interaction state, by setting values in a storage field designed for this purpose. While *stateless* means that there is no record of past interactions and each request must be handled or treated based only on the information that comes with this request.

In this context, our proposed model will be designed with redundancy approach in mind by using stateless components. Redundancy serves two goals, which are high availability and scalability. The main components of the system certainly can all be separated in different hosts and multiplied for redundancy. Thus, each architectural element may be implemented redundantly. Data nodes are by default scaled and should operate in a redundant architecture. Cloud Controller and Cloud Manager Applications are stateless and are implemented behind a simple load balancer. The messaging service is also stateless and Puppet/MCollective is configured to use multiple ActiveMQ endpoints. Multiple MongoDB instances are combined in a cluster of servers (Replica Set), which implements the master-slave replication and automatic failover for high availability and fault tolerance. Figure 2 shows the redundancy approach of the proposed platform.

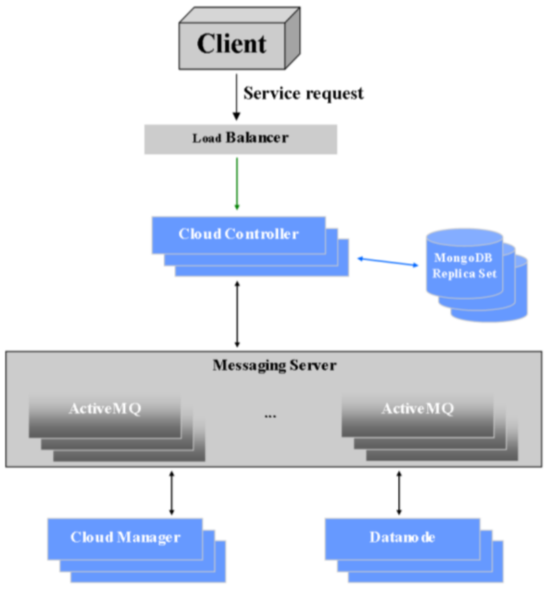


Figure 2. Proposed architecture with redundancy of key components

3.4. Auto-scaling applications

The horizontal scaling allows an application to react to the traffic evolution and automatically allocate resources according to the demand. This is accomplished by using the load balancer. At the time of application creation or deployment, the developer must specify whether the application is scaled or not. If deployed or created application is not scaled, it occupies only a single container in a data node and all traffic will be sent to this container. However, when the developer creates an application with scaling, it provisions two containers, one for the load balancer, and the other for the real application. If the developer adds other templates like MongoDB or MySQL to its application, they will be installed within their own containers. As shown in figure 3, the load balancer hosted in a dedicated container is located between the application container and the Internet and forwards web traffic to the developer application.

When traffic increases, the load balancer notifies the PaaS's cloud controller that it needs additional computing resources. The cloud controller checks the available resources in the PaaS system and then sends a request to the cloud manager to create another copy of the developer application in a new container. In the case that there are no available resources, the controller still sends a request to the cloud manager to provision one or more data nodes. Afterward, cloud manager must create a new container for the new copy of application. To synchronize the updates made it by the developer of all application copies, the source code in the repository of the distributed version control tool Git will be copied in each new container. When the new copy of the application is started by invoking automatically the hooks, the load balancer starts routing web requests to this new instance. If the developer pushes a code change to the application via the Git tool, all its running instances will be updated automatically.

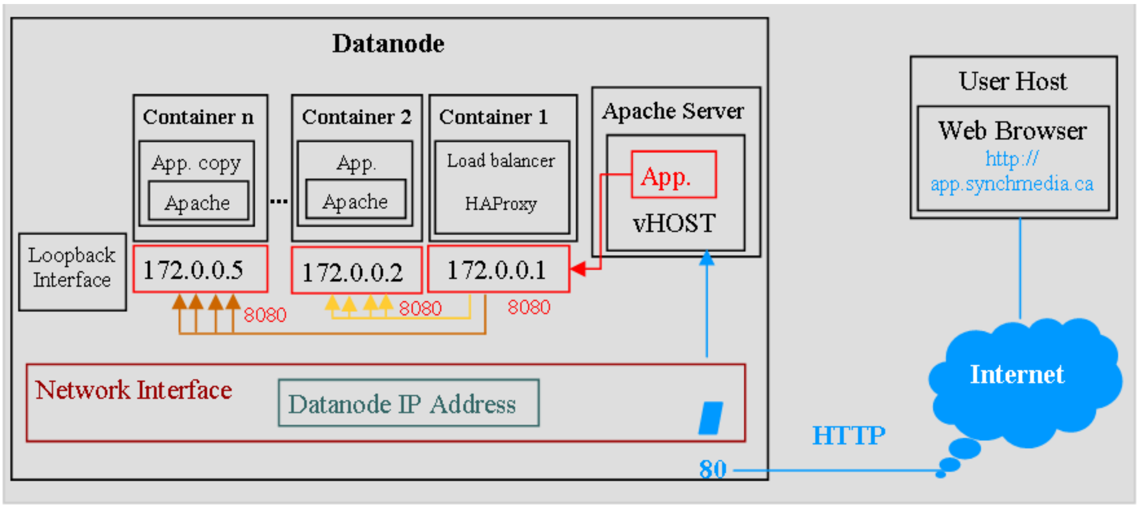


Figure 3.Auto-scaling : Inter-container TCP communication

The scaling algorithm can be based on the average response time, on the CPU utilization or on that of the memory. It allows automatically increasing or decreasing the capacity of resources based on the conditions defined by the developer. With the method of auto-scaling, the developer can make sure that the number of application instances increases continuously during peak demand to maintain performance, and automatically decreases when the applications load decreases to minimize costs. For example, the developer can set a condition to add new application instances to the application cluster when the average CPU utilization reaches or exceeds 80%, and similarly, he can set a condition to remove application instances when the average CPU usage drops below 20%. The developer can also define a condition of autoscaling based on the average response time. Thus, the load balancer adds another container in the application cluster if the average response time exceeds a threshold, for example, of 0.5 ms, and it removes the container if the average response time drops under this threshold for several minutes. This process is repeated and the load balancer continues to add or remove containers based on the application traffic. It is important to note that there are other approaches to implement the auto-scaling technique, which are based on the memory usage and the number of simultaneous requests that can be used by the load balancer. Also, the addition and deletion thresholds of containers are configurable and can be adjusted according to the needs of the developer**.**

4. ARCHITECTURAL DESIGN OF THE PROPOSED PAAS MODEL: OPENICRA

Our research work consists in designing and developing a new generic model of automatic applications deployment in the cloud. To reach this goal, we propose to use three main architectural elements, which are:

- a PaaS allowing migration and deployment of applications in the cloud;
- a configuration management tool to automate the deployment and avoid the repetition of tedious tasks;
- and a distributed file system to facilitate data sharing and management in the cloud.

Given the huge variety of deployment scenarios, platforms, configurations and applications in production environments, it is difficult to found a full customizable tool, which meets our needs. An alternative solution for personal and commercial purposes involves using Open Source softwares. These open source softwares offer two key advantages for organizations, first of all, they are open and extensible, and secondly, they are free. It is important to note that the choices of technological solutions for the design of our generic model of application deployment are based on high-level qualitative comparisons. These solutions are the key architectural components of our proposed model, and provide a high level of flexibility, scalability and performance. Unlike commercial solutions, these components are characterized by the free access to their source code. This allows us to guarantee the portability of deployed applications in any runtime environment, ensure evolutivity of the model, avoid vendor lock-in and facilitate the process of applications deployement in the cloud. In the following, we present assumptions and design of the proposed model as well as architectures of its various components.

4.1. Assumptions

This section describes in detail our design assumptions of the proposed model.

- The OpenICRA system is built only from open source components such as HAProxy, Puppet, Boxgrinder, etc. Thus, no proprietary technology, API or resource is integrated into the system.
- The system uses the resources of the GreenStar Network (GSN) project as an infrastructure as a service controlled by the cloud management software, OpenGSN. The system is also compatible with any cloud environment such as Amazon EC2, Rackspace and GoGrid.
- The system uses a distributed file system in IaaS level to meet the increasing demand of data processing of the underlying infrastructure.
- The telecommunications module developed within OpenICRA interacts with external VoIP providers such as Google Voice for outgoing calls and IPKall for incoming calls. It also uses a local PABX based on Asterisk server to provide SIP calls. These servers are designed to interconnect the deployed application with VoIP and PSTN networks.

4.2. Global architecture of the proposed model

The proposed model OpenICRA is a platform as a service (PaaS) that automates and orchestrates as much as possible the deployment of applications in the cloud. It incorporates a large variety of services and advanced technology to ensure the scaling of applications and a good quality of service (QoS) for the end user. This model sits between the application and cloud environment chosen in caring for infrastructure, management and configuration of required resources, leaving the developer focused only on the development and the improvement of his applications.

OpenICRA implements a layerwise architecture, which hides the implementation details and simplifies deployment process as shown in Figure 4. This architecture is composed of a Cloud Controller, a Cloud Manager and several Datanodes. Each component is a Linux machine configured with SELinux security module and which performs communication processes to form a cluster of nodes. Famework templates of different programming languages are hosted in datanodes. They are characterized by their extensibility allowing developers to add new frameworks to support the deployment of any type of application. The hosted modules in the Cloud Controller allow developers to migrate their applications to the cloud without making any changes in the source code, regardless of programming language (PHP, Java/Spring, Python, Ruby on Rails...), databases (relational such as MySQL, or NoSQL such as MongoDB and Cassandra Apche) or any other software stack used.

Application is composed at least of one framework which is contained in a template and running in one or more application containers. Additional templates can be added to the application in one or more different containers. Using SELinux, users have only permission to change their own application files. Thus, user accounts are accessible via SSH for secure communication between the developer's machine and OpenICRA environment. OpenICRA is based on open source components and does not use any proprietary technology. As a result, it supports portability and can run applications on any cloud infrastructure such as GSN, Amazon EC2 or with cloud management software, like OpenStack. In addition, user application can be exported from OpenICRA platform to be deployed on any other platforms.

4.3. Cloud Controller

Cloud Controller is the brain of the OpenICRA system. It handles the creation, management and general orchestration of the automatic deployment of applications, including user authentication and communication with Cloud Manager and data nodes. It manages DNS service and applications metadata. Therefore, users do not have direct contact with the controller, but they use the web console or the CLI tools to interact with the API via the REST architectural style or SSH. The controller handles itself services offered to applications and users. These services are divided into three sections, which are authentication, metadata and domain name service (DNS). In addition, it uses three separate interfaces in order to interact with different systems as shown in Figure 5.

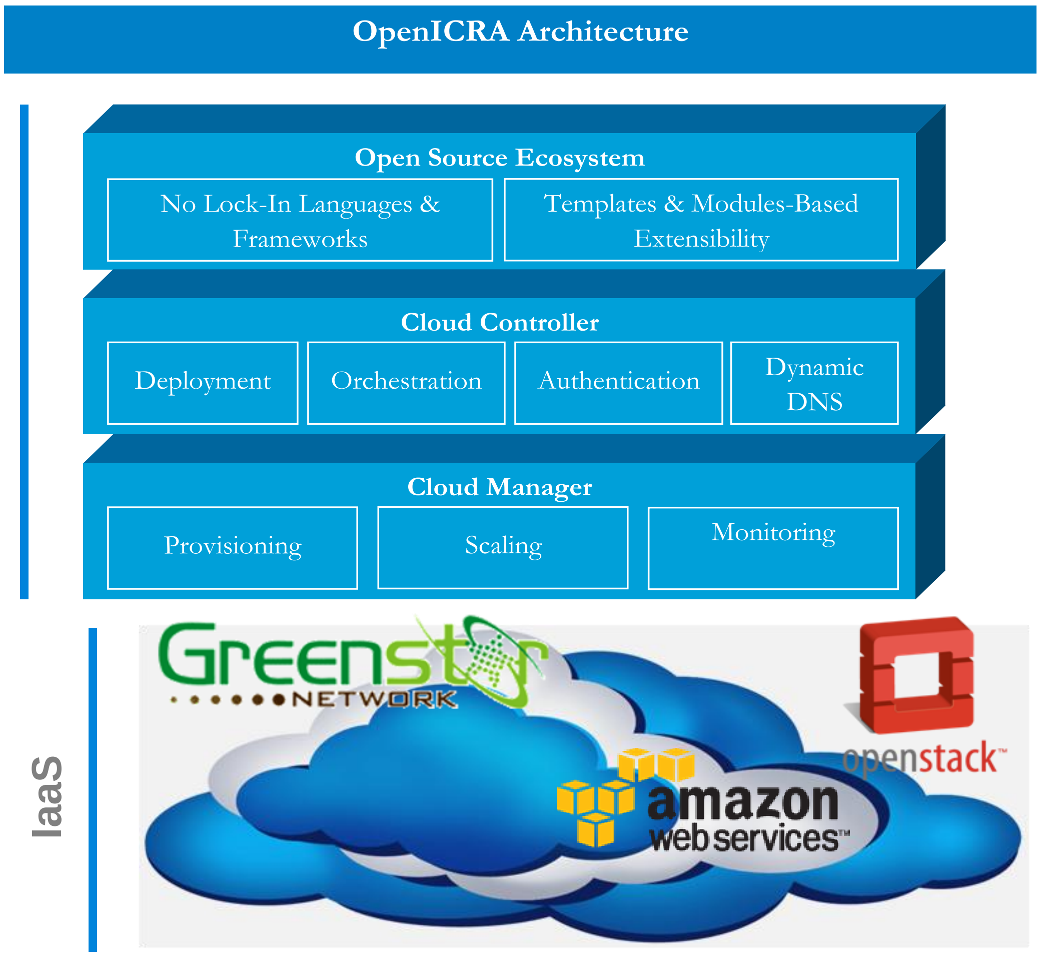


Figure 4. OpenICRA's Global architecture

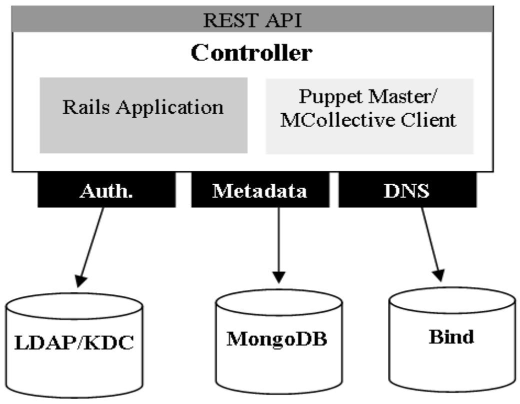


Figure 5.Cloud Controller

Authentication: This interface is used to manage authentication and authorization of users using LDAP or Kerberos KDC.

Metadata: Metadata of applications, users and datanodes are stored in a NoSQL database of MongoDB.

DNS: BIND server is responsible for the dynamic management of application URLs deployed by the developer.

All these services can be customized according the needs. Indeed, metadata storage, authentication, and DNS server services can be implemented as plug-ins. They can be in the same machine as the controller or in separate machines (physical or virtual). The Rails application hosted in the controller machine used to transfer user requests from the REST API to automated configuration manager, Puppet, which is responsible for the automation of application deployment. Thus, user can transfer deployment module of its application to Puppet through SSH or via the web console interface. Then, he can use its CLI tool to interact with MCollective in order to trigger the automation deployment process. Subsequently, Puppet analyzes the user module, supplies the resources via the Cloud Manager and finally launches the automatic deployment of the application in the cluster of datanodes. Depending on the provisioning type of resources requested, user can have full control of the virtual machines or only the rights to change files of his applications.

4.4. Cloud Manager

Cloud Manager is the manager of cloud resources. It is an abstraction layer for the underlying cloud environment. It communicates with the cloud controller through MCollective to meet the needs of users in terms of provision of computing and storage resources. Figure 6 shows a logical view of its various components.

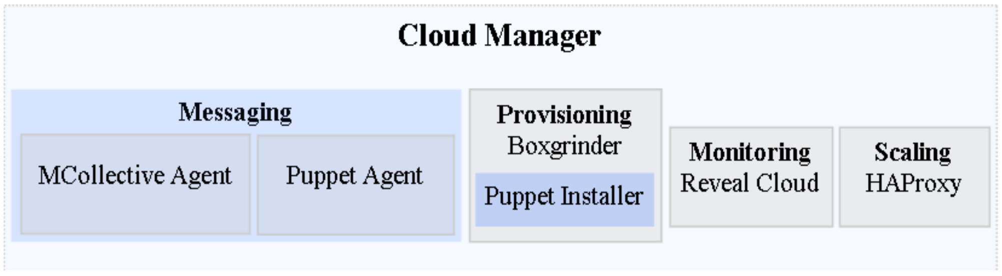


Figure 6. Cloud Manager

Cloud Manager is responsible for interfacing with cloud infrastructure to provide on demand computing resources, which are required for application functioning. It uses Boxgrinder and VMBuilder tools to create new virtual machines or containers, where application files will be hosted. Thus, it connects via SSH to the provisioned ressource to automatically install the Puppet configuration manager tool. The latter connects to the Puppet Master to check for new manifests to compile. The Cloud Manager also provides a permanent supervision for applications using a monitoring tool API. The tracking results of applications can be displayed on a dashboard using a simple browser. During resources provision, Cloud Manager provides a load balancer template with the requested container or virtual machine. Application processes run in a container other than that of the load balancer. This load balancer sets between the application and Internet, and routes the requests to the application container. When demands increase, it triggers auto-scaling rules by creating copies of the application in another container and it begins to load balance between the different containers of the application. In the case where the demand decreases, load balancer removes all unused application instances.

4.5. System resources

4.5.1. Data nodes

Datanodes are physical or virtual machines. These nodes are partitioned to several containers using SELinux security module. This partionning method is based on a technique used by the OpenShift platform team to create their containers. The datanodes host the framework templates used to deploy applications in the cloud. They store also application files and run their processes.

4.5.2. Application Container

Application Containers are virtual partitions or virtual environments (VE) managed by the SELinux security module and also known by Virtual Private Server (VPS). They provide limited IT resources to perform one or more modules/templates and limit the amount of RAM and storage space for applications. Therefore, OpenICRA system creates several containers in each datanode and dynamically affects them to applications. Figure 7 illustrates the relationship between different system resources.

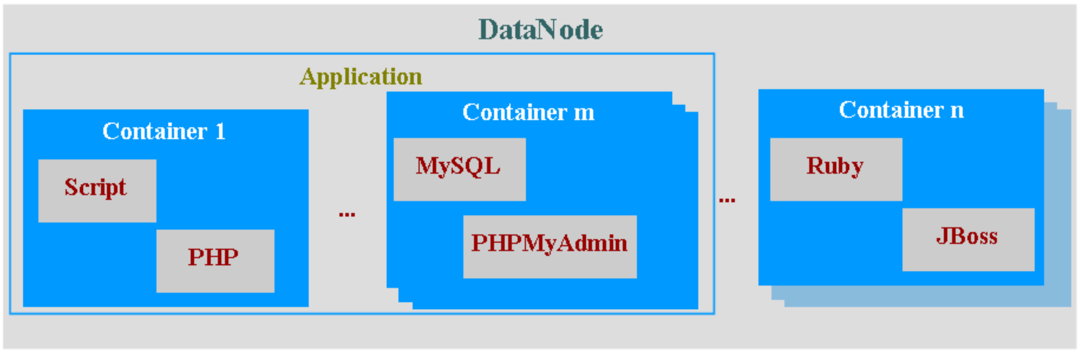


Figure 7. Relationship between different system resources

In general, an application is composed at least of a framework contained within a template which runs in one or more containers in the case of a cluster or a scaling mode. Additional templates can be added on demand to the application in the same or different containers.

4.6. Modules / Templates

Modules are used by the Puppet tool, which is integrated with OpenICRA to automate application deployment, while templates are hosted in the data nodes and usually are used to create or develop a new application in the cloud or to deploy an existing application. We define modules as directories used by the Puppet Master to trigger automatic deployments in the data nodes through Puppet Agent and MCollective Server or to migrate an existing application to the cloud. Templates can also be defined as containers or directories that host framewoks or components that can be used to create or deploy applications. They are stored in the data nodes and they support also the deployment of any binary application without source code modification. The templates provide the real functionality needed to run the user application. Supported programming languages are PHP, Ruby, Java, Python, etc., and several types of databases such as Postgres, MySQL Mongo, etc.

4.7. Mechanisms of system communication

Communications from any external client (eg, SSH, Web Console) towards the OpenICRA PaaS are performed through the REST API, which is hosted by the cloud controller. The latter communicates with datanodes and Cloud Manager through the ActiveMQ message broker. MCollective is used to facilitate the interrogation and orchestration of datanodes and Cloud Manager as well as to secure the communication with the individual nodes.

5. BIG DATA STORAGE MANAGEMENT IN THE CLOUD

According to [25], a shared storage system usually refers to a shared storage disk over the network allowing anyone on the same network the access to the server files. This means that the architecture of shared system does not allow the horizontal scaling by adding new storage servers; it only allows the vertical scaling in a single server. In addition, it stores all the files from the entire network infrastructure, which limits the number of files to store. Moreover, a shared storage system cannot store all or part of a large file that exceeds the capacity of the shared disk. These limitations make the traditional storage system non-scalable and unable to meet the needs of a system based on the cloud computing concept. Because of the lack of dynamic management system for storage solutions, a single storage system may encounter bottlenecks in the case of need of a storage space to host a large image of a virtual machine or fror mass data processing.

Thus, in cloud computing environment like the GreenStar Network (GSN) [10], it is necessary to have a computing high-speed and a scalable and efficient storage system. Moreover, in a production environment where there are multiple servers trying to access to the same files, it is always difficult to establish an adequate storage management architecture. However, there are various manners to share files with multiple servers. The most beneficial ways is to implement a Distributed File System (DFS) offering to the distributed computer users the ability to share data and storage resources. Also, distributed file systems are made of different software components running on multiple computers, but operate as a single system. They can be advantageous since they facilitate the distribution of documents to multiple customers and they provide a centralized storage system.

Based on the distribution management tool of real-time data and the techniques of provision and virtualization resource optimization, OpenICRA offers a high efficient environment for high performance and intensive I/O applications. Therefore, an integration of a DFS with OpenICRA will provide an excellent environment for virtual machines and applications where scalability is an issue both in terms of storage and computing speed.

5.1. The Distributed File System (DFS)

In order to facilitate the integration of chosen distributed file system with our proposed model, it is important to establish a set of the desired selection criteria. For this purpose, a comparative study between the most popular and common DFSs was performed taking into account their implementation in large scale networks of computer systems such as cloud computing, grid computing, etc. Thus, the comparison process considers the following distributed file systems: The Google GFS [26], the Apache HDFS [27], the IBM GPFS [28] and the Sun NFS [29]. In the literature, there are several other distributed file systems such as AFS [30], Lustre [31], Amazon S3 [9], etc. However, they do not meet the established evaluation criteria.

The comparative study revealed that Hadoop Distributed File System (HDFS) is the best adaptive solution to implement a management system of distributed storage in terms of performance, reliability and adaptability. Sustained by major industrial palyers such as Facebook, Yahoo, Twitter and Cloudera, HDFS has emerged as a solid solution that uses a highly scalable technology to meet the challenges of massive data. Thus, it is an open source system compatible with cloud computing environment and it can be easily integrated without any restriction with Cloud middleware of resource management, and it may also be very useful to manage data of deployed applications by platforms as a service in a cloud environment. In addition, HDFS is a scalable distributed file system and designed to reliably store very large files on different machines in a large cluster of nodes. It is inspired by GoogleFS publications [26]. Its three main components are: **NameNode** that stores metadata information of file blocks and the list of data nodes in the cluster, **Datanodes** which host file blocks, and, **Secondary NameNode** which is the primary NameNode replication and it is used in case of NameNode failure.

The HDFS integration with our OpenICRA platform provides an ideal environment for applications that have requirements of large storage space and a high calculation speed. Examples of such applications are hypervisors, applications of log files analysis, and applications which need high availability. Thereby, HDFS is a better choice for use with other applications on an ordinary file system in order to have a distributed and fault-tolerant storage.

5.2. Proposed storage system: Integration of HDFS with OpenICRA

HDFS is designed to be deployed on a less expensive hardware and it provides a high-speed access to application data and is suitable for applications that use large data files. It offers efficient use of storage space through compression and distribution of stored data in its cluster of nodes. Its fault tolerant architecture provides a rapid access to large amounts of data spread across multiple servers and it provides high scalability. Thus, our proposed storage system consists to integrate the HDFS distributed file system with our OpenICRA platform to solve the problem of the ratio between the growth in the volume of VM images with the performance of the compute nodes and the cloud manager.

In our proposed architecture, we configure the HDFS cluster and we implement the fuse-dfs extension of the file system in user space FUSE on the compute node. This “fuse-dfs” module provides an interface for transferring data between compute nodes and data nodes of HDFS. In addition, virtual machines created by the cloud manager can directly access to their virtual hard disks stored in the HDFS cluster using the fuse-dfs module.

Not being a veritable file system compatible with the POSIX standard, HDFS cannot be mounted directly by the operating system and the access to HDFS files is done via Shell commands. However, the FUSE module for HDFS addresses this limitation by exposing the latter as a mount point on the compute node. This means that we can mount the files from HDFS system to any POSIX service. Also, we can use a remote visual client like WinSCP to explore HDFS files. Indeed, using fuse-dfs, which is one of applications that based on the Fuse file system, we can mount HDFS as if it was a traditional Linux file system.

In order to integrate HDFS with OpenICRA, a common interface for data exchange is required that allows the Cloud Manager to retrieve and store data in the HDFS cluster. The solution consists to use the module “fuse-dfs” as a communication channel, which connects the HDFS NameNode machine to mount the entire distributed file system. Hence, HDFS can be mounted on the compute nodes as a traditional Linux file system such as ext3 or ext4 and can be used to store and retrieve files using fuse-dfs. Indeed, the FUSE module provides a "bridge" to the compute node kernel interface and, instead of using the HDFS client tool, the standard terminal of compute node will act as a HDFS client to access its data. Also, the OpenICRA Cloud Manager will be able to interact directly with HDFS to extract and update application data.

In our proposed storage system, we set the node name, the data nodes of the HDFS cluster and the cloud manager on different servers. This is done to improve performance and optimize resource utilization. Given that the fuse-dfs module should interact directly with the cloud manager, these two components must run on the same system to avoid performance problems and to enable it access to files of HDFS system. As a result, the fuse-dfs module is configured on the same node as the cloud manager assuming that Java JDK is already installed on the host. Such a configuration allows undoubtedly better resource utilization for the global storage system. The architecture of the proposed solution of storage management is shown in Figure 8. Indeed, the Namespace manages directories, files and blocks of files and it supports file system operations such as creating, editing, deleting and listing of files and directories. The Block Storage includes two parts, which are the Block Management and Physical Storage (datanodes). The Block Management maintains Datanodes adhesion in the cluster and it supports operations linked to the blocks, such as creating, deleting, modifying and localizing of the blocks. It also supports replication of blocks. While the Physical Storage handles the data hosting and the reading and writing access to the blocks.

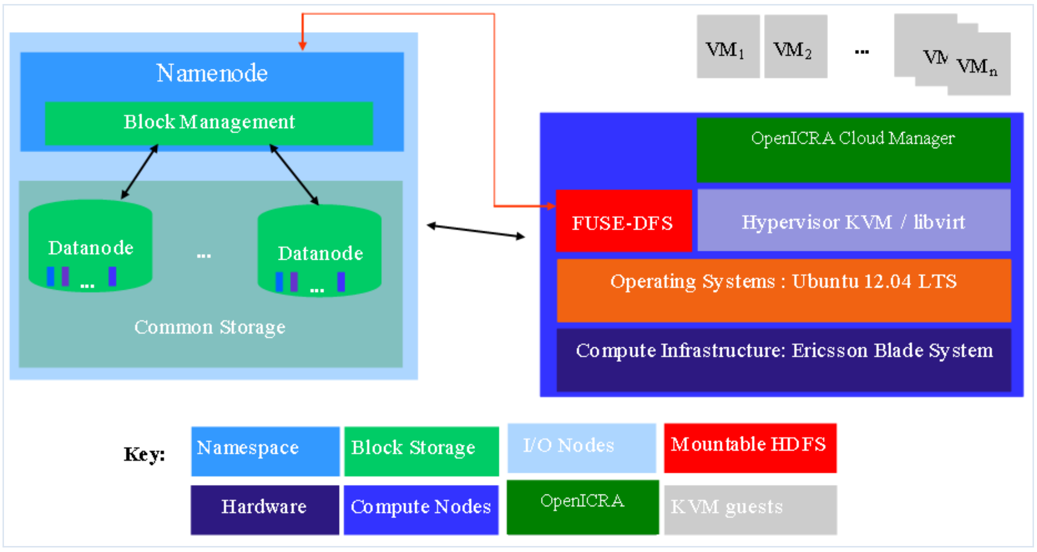


Figure 8. Integration of HDFS distributed file system with OpenICRA.

The fuse-dfs module is the responsible component of HDFS mount to the compute nodes. It uses the APIs of *libhdfs* library and FUSE to appear HDFS as an ordinary file system on the host machine. It also allows arbitrary programs to access data stored in HDFS system. The Compute node is the server on which the Cloud Manager and KVM hypervisor are running.

The proposed storage system can easily be scaled by adding or removing nodes on demand. It does not need to make any changes to existing components of cloud infrastructure. The adding or removing of node or of storage space is transparent to the infrastructure of cloud provider. The proposed system is therefore elastic in nature. Thus, it can be used as a warehouse for the very large virtual machine images.

6. EXPERIMENTS AND VALIDATION

In order to test and validate our proposed model, OpenICRA, we illustrate two concrete case studies, where the first consists to automate the deployment of OpenSAF distributed middleware across a cluster of nodes within Ericsson Blade System [32], and the second case aims to migrate to the Amazon EC2 environment [2] the ICRA collaborative system developed by the research team of Synchromedia [33]. To do this, we follow the automatic deployment process of applications described in section 3.1 to perform the two case studies and present the results obtained from the real cloud environments.

6.1. Case study 1: OpenSAF deployment automation

OpenSAF is an open source middleware actively supported by major telecommunications companies in the world such as Ericsson, HP, Nokia Siemens Networks, Sun Microsystems, etc. This middleware helps to develop a runtime environment with high availability for applications and telecommunications equipment compatible with the SAF (Service Availability Forum) specifications and require uninterrupted service [34, 35]. As part of the Ecolotic project [36], Ericsson wanted to deploy OpenSAF in a large cluster of nodes in a cloud environment to take advantage of cloud-simple functionality wherever possible, and to provide a high availability service for developed distributed applications in the cloud. However, due to the deployement complexity and huge varieties scenarios of OpenSAF, also the repetitive and tedious configuration tasks of applications and services, Ericsson decided to automate its deployment in order to significantly reduce errors and costs, to accelerate the implementation of applications and to ensure compliance, control and governance. In this context, we used our proposed model for OpenSAF middleware deployement automation in a cloud environment.

6.1.1. OpenSAF automation process

Based in our generic model OpenICRA designed to support different deployment scenarios, OpenSAF automation deployment may be carried out in a process of three steps according to the application deployment process described in section 3.1 and as shown in Figure 9. The first step is to prepare the deployment of OpenSAF middleware. According to the official documentation, the OpenSAF recommended configuration which allows to demonstrate its various capabilities of high availability (HA) with better performance requires at least four servers, two machines act as controllers configured in Active/Standby mode and at least two other machines for payloads (entities where OpenSAF manages its running applications). In an OpenSAF cluster, a node or server may be a controller or payload and a maximum of two controllers is allowed. Any additional node must be configured as payload in the cluster [34, 35].

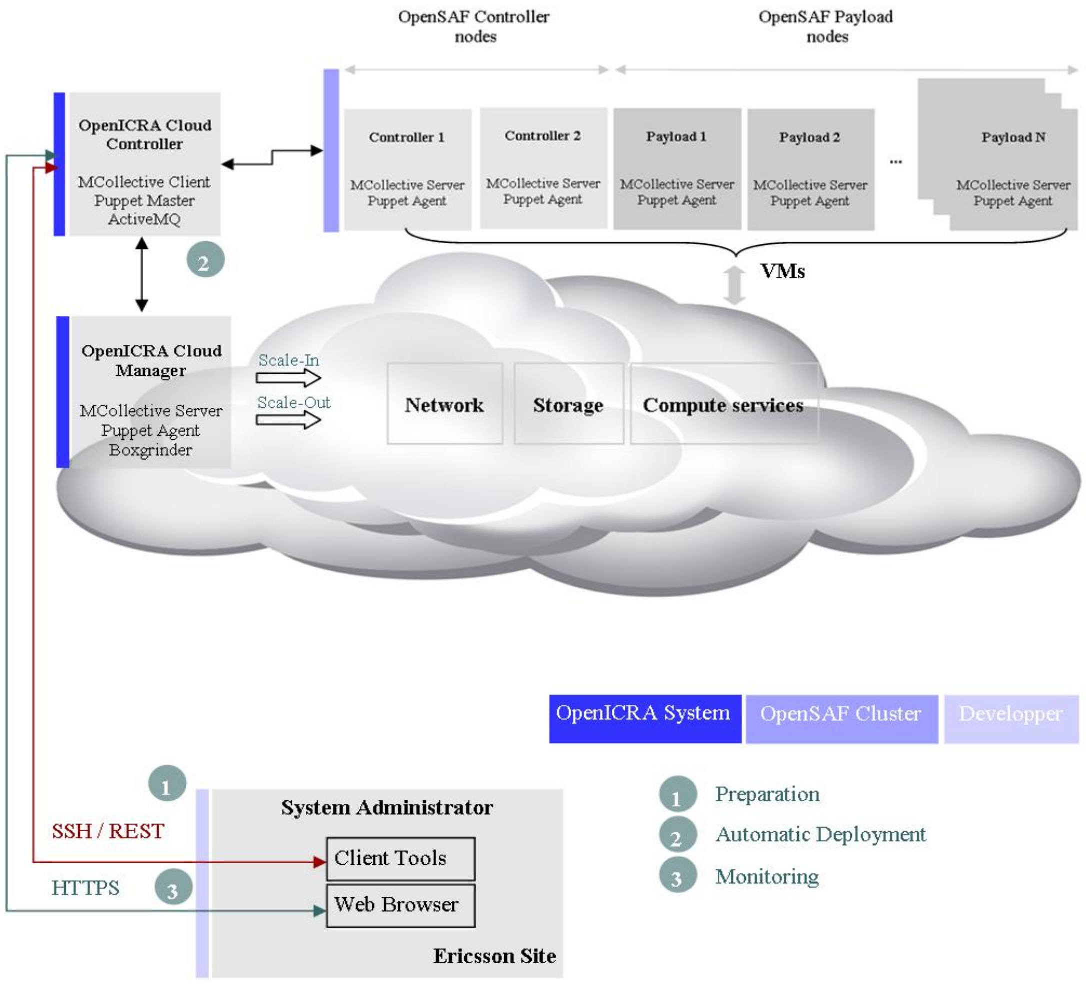


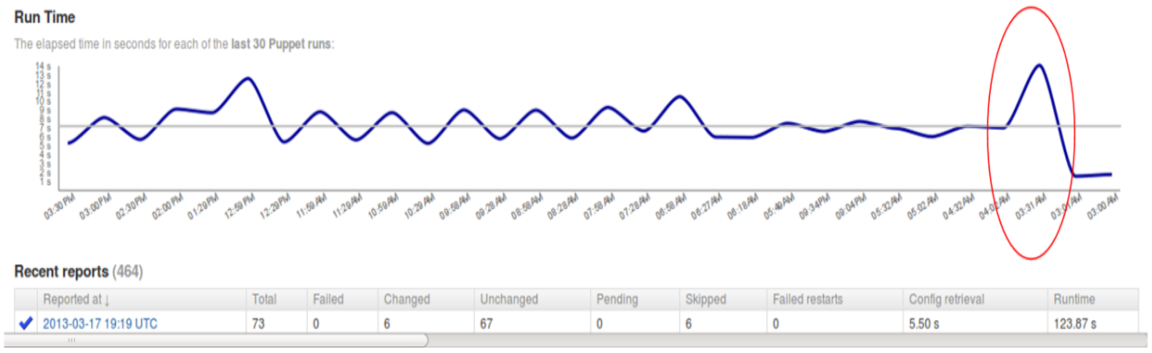
Figure 9. Deployment process of OpenSAF middleware

To automate a successful deployment of OpenSAF, we developed a module called "opensaf" which describes all the instructions required for deployment automation. This module is written in a declarative language and it uses classes, configuration files and binary files. The binary files are used to install dependencies when there is no Internet connection in the deployment environment. Otherwise, they will be ignored and dependencies will be installed from the official packages repositories. The configuration files are preconfigured templates used to ensure proper functionning, compatibility and communication between the different OpenSAF components. However, the Classes are used to:

- request the Cloud Manager to provision the essential computing and storage resources for the deployment and automatically install the Puppet configuration tool;
- update and install the necessary dependencies packages;
- orchestrate the cluster nodes;
- apply the recommended configuration by the system administrator.

After the preparation, the second step consists to trigger the script of resources provisioning and the automatic deployment of OpenSAF. This program must be launched by the developer or the administrator from his local machine via the command line tool "synchro" or through SSH session to triggering the deployment. As shown in figure 9, the Cloud Controller is the central point of the deployment process. It orchestrates in parallel and in real time all the components of OpenICRA system and controls all the main deployment tasks. Indeed, it contacts the Cloud Manager to provision the required computing and storage resources. The Cloud Manager installs and runs sequentially and automatically Puppet agent in each virtual machine created. During the virtual machine startup, each Puppet agent contacts the controller (Puppet Master) to download the configuration described in the "opensaf" class and apply it locally. The "opensaf" class is developed so that it is capable to deploy OpenSAF and automatically solves all the common problems without any human intervention.

The final step consists to monitor and evaluate the deployment. Once the deployment step is complete, each Puppet agent sends a report to the controller. In addition, when data nodes (Puppet agents) run to look for new configurations or updates from the controller, they return the inventory data and a report of their executions. These reports and statistics can be viewed by the developer in the web page of each node through the web console interface. Figure 10 shows the execution time of the Puppet demon in the controller node.



Recent reports (464)

	Reported at ↓	Total	Failed	Changed	Unchanged	Pending	Skipped	Failed restarts	Config retrieval	Runtime
✓	2013-03-17 19:19 UTC	73	0	6	67	0	6	0	5.50 s	123.87 s

Figure 10. Execution Time of Puppet demon in the controller node

The graph of the execution time shows how long each of the last 30 Puppet executions took to complete the configuration or the intended deployment. A long execution usually means that there are changes have been made, but could also indicate that the server load was heavy or there are some other circumstances. In our case, the last run which is the longest corresponds to the OpenSAF deployment. For the other executions, they are not long, because there is no configuration or updates to be made.

6.1.2. Result of the first case study

The main contribution of this case study is to relieve users from reading the complex installations guides and manual troubleshooting of OpenSAF by automating its deployment in order to avoid repetitive tasks on each server. In this section, we present the results of OpenSAF automatic deployment process in a cluster of nodes. According to a recent survey [37] which had interrogated about 95 users from several companies such as Ericsson, Oracle, Connectem etc., OpenSAF users are concerned by the complexity and long duration of its manual deployment. The survey revealed that the average time required to manually deploy OpenSAF in a cluster of nodes is approximately 138.95 minutes. Figure 11 illustrates the results of this survey and indicates that 63.16% of respondents exceed the average duration of OpenSAF's manual deployment.

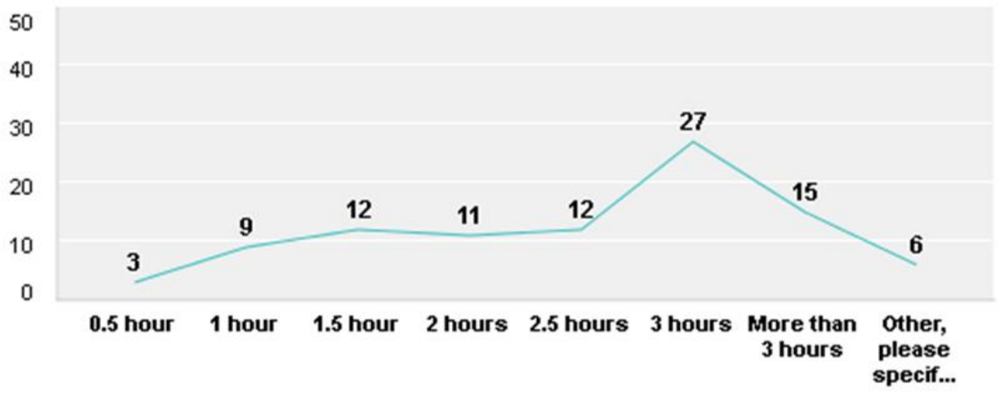


Figure 11. Survey results of OpenSAF's manual deployment

With our proposed model, OpenICRA, we have greatly reduced the required time of OpenSAF deployment by automating key tasks of configuration, installation, updates and resolution of common problems. Indeed, we have successfully deployed OpenSAF in cloud environment in less than 166 seconds. This approach of OpenSAF automatic deployment has produced measurable results where the efficiency improvement reached 91.51%. These results are presented in table 2 which summarizes the comparison of deployment duration before and after the automation as well as the efficiency improvement of deployment.

Table 2. Deployment duration before and after automation

Task	Before automation	After automation	Efficiency improvement
OpenSAF deployment	≈ 2.31 h	≈ 11.76 min	91,51 %

As a conclusion to this case study, we notice that OpenICRA has reached 91.59% of the effort and time reduction of OpenSAF deployment and has provided an automated deployment without any human intervention after the triggering of the process. This reduces costs and effectively accelerates the OpenSAF middlware implementation in the cloud.

6.2. Case study 2: Automatic migration of ICRA system to Amazon EC2

ICRA (Internet based Collaboration and Application Resource) is a collaborative application based on the web, developed by the research team of Synchromedia [33]. It allows users to establish virtual meetings and video conferences remotely from different locations around the world to facilitate their research and team works. Indeed, the application is equipped with advanced telecommunications interactive technologies, enabling users to have productive discussions and to easily share digital documents, data and annotations in real time.

ICRA and Asterisk [38] are considered as competing applications for various services, while in reality, Asterisk can be complementary to ICRA application. The integration and deployment of the two applications in a cloud environment create essentially a more scalable and more efficient collaboration system. The selection of a free IP PBX was not complicated, because currently the major actor of the free IP PBX is the Asterisk softswich. The aim of this case study is not to innovate on the IPBX technology. We naturally decided to use this one since it is considered one of the most popular open sources PBX in the world. It is also a key element of the VoIP service and it will allow the management of outgoing and incoming voice calls.

This integration gives users an opportunity to test and become familiar with the VoIP (Voice over IP) technology in a cloud environment and collaborative work such as real-time annotation, document and presentation sharing, instant messaging and web conferencing. Also, it offers the ability to easily and rapidly use advanced services such as VaaS (Voip as a Service) and CaaS (Cooperation as a Service) with reduced cost and without client-side required software. However, one of the complexities associated with the integration of Asterisk and ICRA is related to the large number of components to configure and migrate to the cloud and the lack of coherent management tools. Indeed, these applications use several different services and require a difficult reconfiguration during the migration process to be compatible with the target runtime environment.

The main objective of this case study is to integrate the collaborative application ICRA with the PABX Asterisk. All must be automatically deployed in the cloud environment of Amazon EC2. Thus, we will use our proposed model OpenICRA to automate the integration and deployment of the two applications in the cloud. This simplifies the migration process and ensures scalability, control and governance of all system components.

6.2.1. Automatic migration process

Accordance the proposed model functioning described in section 3.1, the integration and migration process of both applications ICRA and Asterisk can be performed in three simple steps. In our case, the goal is to integrate the Asterisk VoIP server with the ICRA collaborative application to allow users to make incoming and outgoing calls using centralized client web interface to any telephone systems such as IP networks, PSTN, cellular, etc. All must be deployed on the cloud. Based on our analysis, we determined that Amazon is the most appropriate provider due to its flexibility and elasticity. It provides virtual machines with plenty of highly available services that utilize advanced technology and a secure access to resource via RSA keys, as well a monitoring service such as CloudWatch. Figure 12 illustrates an overview of ICRA's migration automation process.

The first step consist to understand the architecture and functionning of the collaborative system's various services to be migrated to the cloud in order to ensure a successful deployment. For the softswich Asterisk inteconnection with PSTN and cellular networks, we look for providers that allow us to receive and make calls freely. The solution was to find a provider that includes the two features. However, none met our expectations. So, we decided to divide the provider tasks. To interconnect the IP network with PSTN, we need a provider which offers us a phone number.

We easily determined what would be our line provider. IPKall [39] meets our needs, it provides a telephone number in the United States (Washington state) for free and also serviceable in Canada. For outgoing calls, our attention will focus on the Google Voice provider as it offers the opportunity to call the United States and Canada phone numbers for free. Therefore, we used IPKall for incoming calls and Google Voice for external (outgoing) calls. Regarding the architecture of ICRA application, it is mainly based on open sources components namely, MySQL, SWFTools, ImageMagick, OpenOffice.org, Red5, middleware java (jar) and a client web interface, etc. All these components can be installed with the command line interface on any Linux system. Therefore, their installation can be automated by OpenICRA deployment hooks.

The ICRA client web interface is the interconnection point of all system components. It connects to the database server and ICRA middleware through web services and ActiveMQ queue server. Also, it connects to the streaming server via the configuration file "conf.xml" to guarantee the video conferencing services, instant messaging, real-time annotation, etc. To automate the integration of different services and the deployment of the collaborative system, we created a deployment hook that describes all the requirements and instructions of the migration automation process of the collaborative system in the EC2 environment. This hook will be used in the next step "automatic deployment" by the automatic deployment script to trigger the integration and deployment of ICRA and Asterisk applications.

The second step consists to automatically migrate the Collaborative system to the EC2 cloud environment. The developer can trigger the deployment from his local machine using OpenICRA command line or via SSH connection. In our case, the developer must use the "synchro" command line tool to trigger the deployment. A simple execution of the deployment script can trigger the automatic migration of collaborative system. Once the deployment is triggered, the OpenICRA PaaS intervenes through its Cloud Controller and Cloud Manager components to ensure a successful deployment. The main command behind ICRA migration is: ***« synchro app create -s icra ctad mysql-5.1 phpmyadmin-3.4 -I rgadhgadhi@synchromedia.ca -p qwerty »***. This command allows creating a scalable and customizable template of CTAD type with MySQL and PHPMyAmin applications for the database creation. Thus, this template type allows the developer to install any application compatible with Linux systems. In our case, this template is created in a separate container to host ICRA applications, Asterisk and PHPMyAdmin. The load balancer HAProxy which ensures the auto-scaling is hosted in another container. Also, the database is hosted in a different container to ensure the collaborative system scalability in case of a variation in the demand. The "-I" option and "-p" are used for the developer authentication by cloud controller.

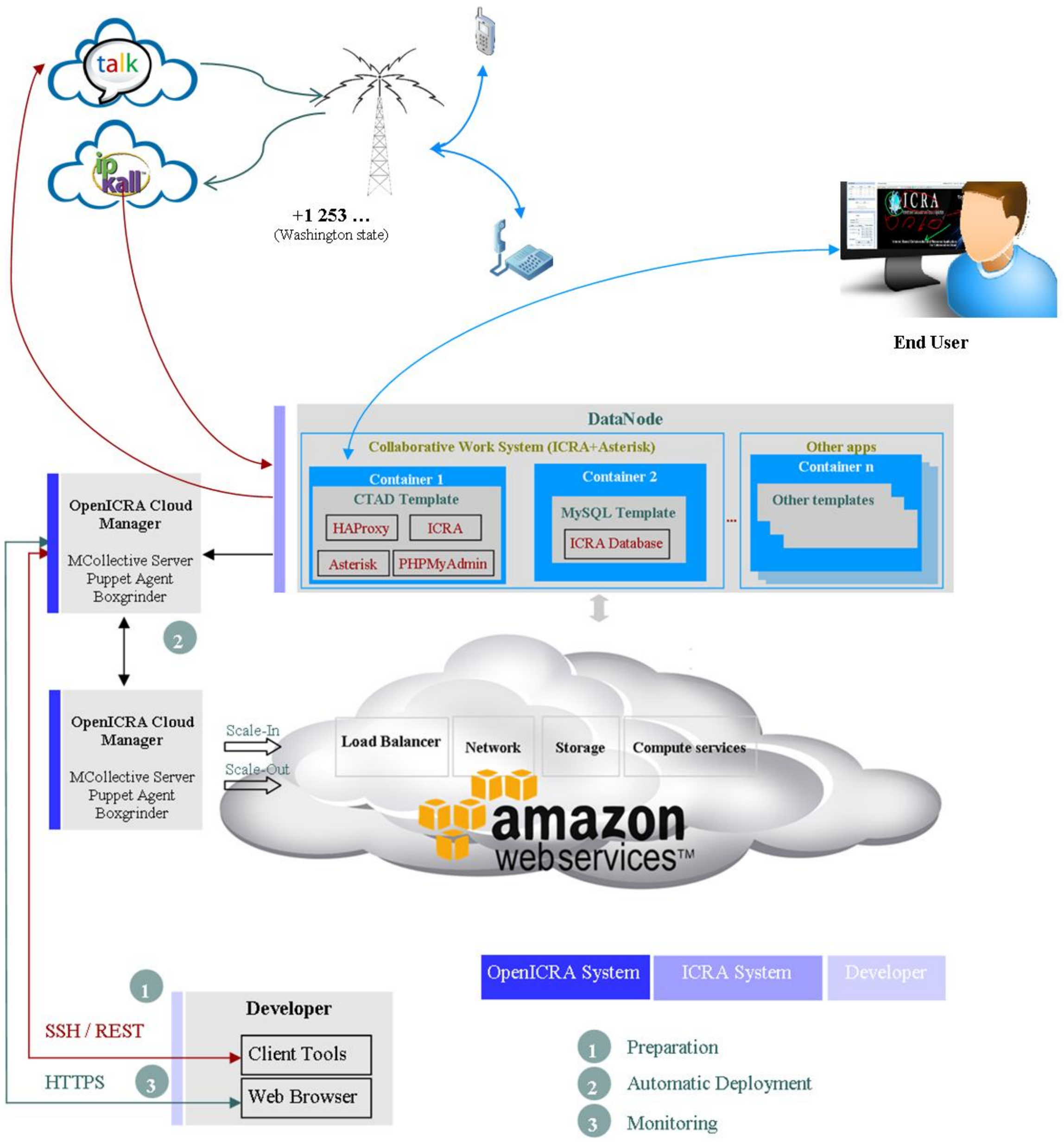
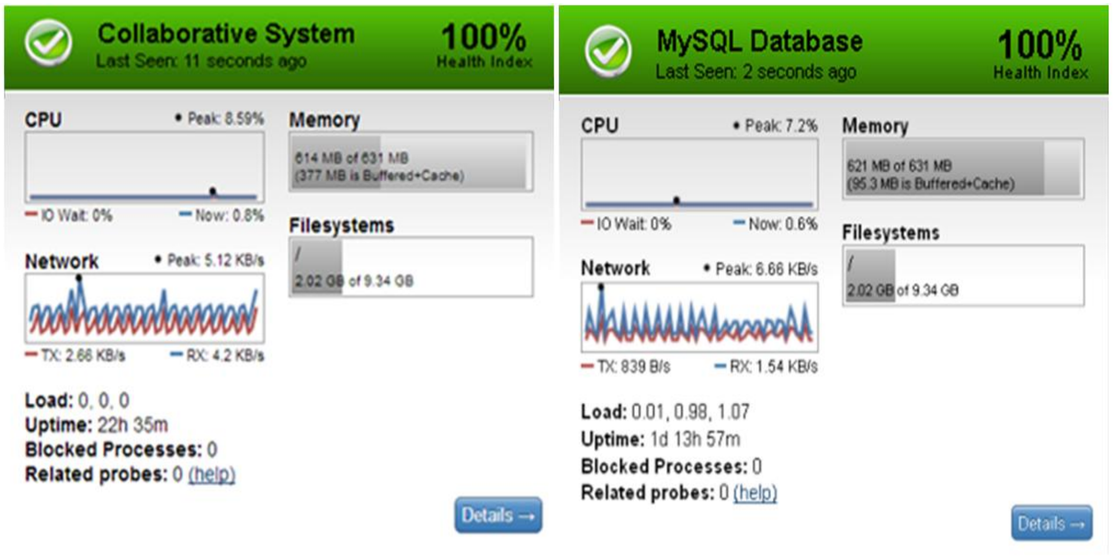


Figure 12. Overview of ICRA's migration automation process

Furthermore, the deployment hook of ICRA and Asterisk installs in automatic way the prerequisites and services system, configure the application as desired by the developer and automatically fixes the common problems. Once the deployment of the collaborative system is complete, the deployment hook returns a report to the Cloud Controller which will be available to the developer through the monitoring tool web interface.The final step consists to utilize monitoring tools provided by the PaaS platform, OpenICRA, to ensure the progress of the proper deployment. In fact, with monitoring tools, the developer may easily detect errors that have occurred or check if the deployment is successful. Figure 13 illustrates the overall status of the collaborative system resources after their deployment on EC2 and shows that the two instances are healthy with a variation of CPU usage between 0.6% and 8.59%.


Collaborative System
Last Seen: 11 seconds ago
100%
Health Index

CPU • Peak: 8.59%
IO Wait: 0% Now: 0.8%

Network • Peak: 5.12 KB/s
TX: 2.66 KB/s RX: 4.2 KB/s

Load: 0, 0, 0
Uptime: 22h 35m
Blocked Processes: 0
Related probes: 0 (help)

Memory
614 MB of 631 MB
(377 MB is Buffered+Cache)

Filesystems
/
2.02 GB of 9.34 GB

Details →

MySQL Database
Last Seen: 2 seconds ago
100%
Health Index

CPU • Peak: 7.2%
IO Wait: 0% Now: 0.6%

Network • Peak: 6.66 KB/s
TX: 839 B/s RX: 1.54 KB/s

Load: 0.01, 0.98, 1.07
Uptime: 1d 13h 57m
Blocked Processes: 0
Related probes: 0 (help)

Memory
621 MB of 631 MB
(95.3 MB is Buffered+Cache)

Filesystems
/
2.02 GB of 9.34 GB

Details →

Figure 13. Overall status of containers system

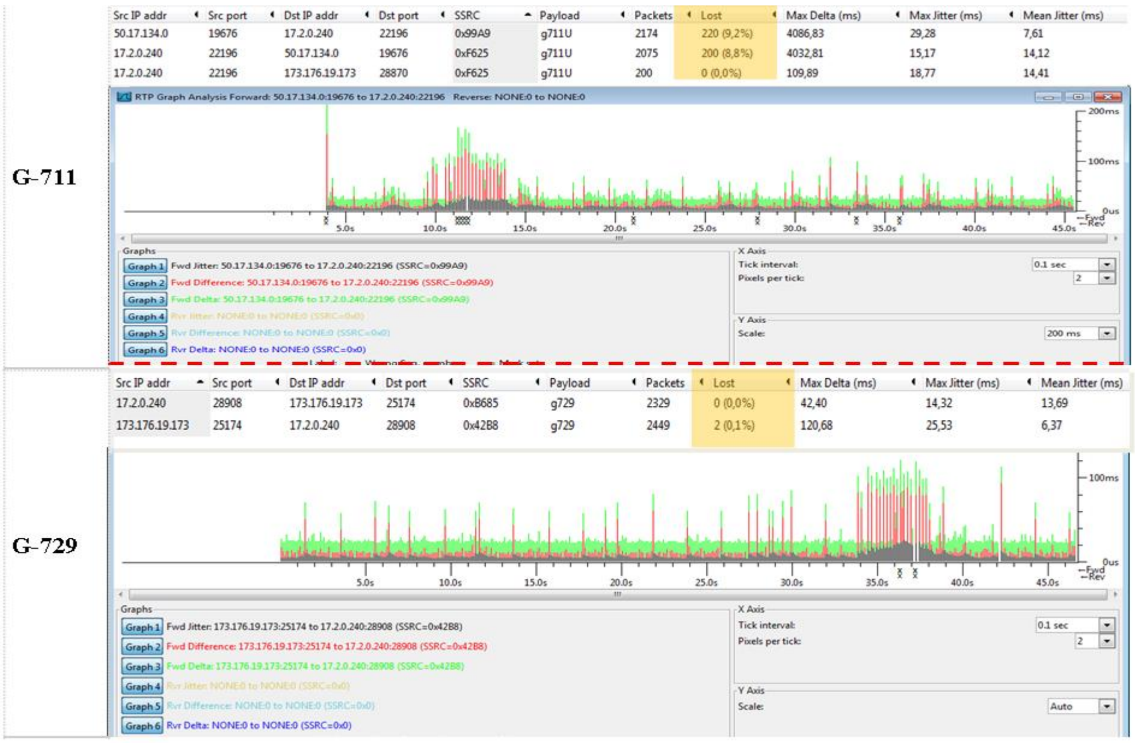
For a deployment on Amazon EC2, OpenICRA offers to developers the ability to use both monitoring services, CloudWatch EC2 of Amazon [40] and RevealCloud of CopperEgg [41]. These tools are used to provide in real time detailed information on the services and the migration process status of collaborative system. Thus, the developer can use these services through the web interface of the controller to check in more detail the progress status of the migration process by consulting logs tables of different services.

6.2.2. Evaluation of collaborative work system's overall performances

In order to evaluate the stability and scalability of the collaborative system deployed by OpenICRA on Amazon EC2, we firstly present a quantitative comparison of G-711 and G-729 codecs of integrated VoIP module, and secondly, a stress test of the whole system.

6.2.2.1. Comparison of codecs and moving from G711 to G729

In this section, we compare the G711 and G729 codecs to identify which is the optimal for the collaborative work system. The tests were performed with WireShark and its analysis module of RTP flow and one of the client machines captures RTP traffic. Both catches were made at 10 minute intervals; we can therefore affirm that the traffic and the saturation of different links involved between both SIP clients were constant. The analysis shows a great gain in terms of the packet loss reduction since that it passes from 8-9% of loss for the G-711 to 0% with the G-729 as shown in figure 14. As a reminder, G-711 codec uses a large bandwidth (56-64 Kbit/s) versus G-729 which is optimized for WAN (8 Kbit/s). Regarding the latency and jitter, there is no significant change. Our architecture depends on several providers and it is difficult to be optimizable at this level. We can still note the correct performance, usually below 100ms with an average around 20ms (measured values are not present in this figure).



G-711

Src IP addr	Src port	Dst IP addr	Dst port	SSRC	Payload	Packets	Lost	Max Delta (ms)	Max Jitter (ms)	Mean Jitter (ms)
50.17.134.0	19676	17.2.0.240	22196	0x99A9	g711U	2174	220 (9,2%)	4086,83	29,28	7,61
17.2.0.240	22196	50.17.134.0	19676	0xF625	g711U	2075	200 (8,8%)	4032,81	15,17	14,12
17.2.0.240	22196	173.176.19.173	28870	0xF625	g711U	200	0 (0,0%)	109,89	18,77	14,41

RTP Graph Analysis Forward: 50.17.134.0:19676 to 17.2.0.240:22196 Reverse: NONE:0 to NONE:0

Graphs
- Graph 1 Fwd Jitter: 50.17.134.0:19676 to 17.2.0.240:22196 (SSRC=0x99A9)
- Graph 2 Fwd Difference: 50.17.134.0:19676 to 17.2.0.240:22196 (SSRC=0x99A9)
- Graph 3 Fwd Delta: 50.17.134.0:19676 to 17.2.0.240:22196 (SSRC=0x99A9)
- Graph 4 Rvr Jitter: NONE:0 to NONE:0 (SSRC=0x0)
- Graph 5 Rvr Difference: NONE:0 to NONE:0 (SSRC=0x0)
- Graph 6 Rvr Delta: NONE:0 to NONE:0 (SSRC=0x0)

X Axis
Tick interval: 0.1 sec
Pixels per tick: 2

Y Axis
Scale: 200 ms

G-729

Src IP addr	Src port	Dst IP addr	Dst port	SSRC	Payload	Packets	Lost	Max Delta (ms)	Max Jitter (ms)	Mean Jitter (ms)
17.2.0.240	28908	173.176.19.173	25174	0xB685	g729	2329	0 (0,0%)	42,40	14,32	13,69
173.176.19.173	25174	17.2.0.240	28908	0x42B8	g729	2449	2 (0,1%)	120,68	25,53	6,37

Graphs
- Graph 1 Fwd Jitter: 173.176.19.173:25174 to 17.2.0.240:28908 (SSRC=0x42B8)
- Graph 2 Fwd Difference: 173.176.19.173:25174 to 17.2.0.240:28908 (SSRC=0x42B8)
- Graph 3 Fwd Delta: 173.176.19.173:25174 to 17.2.0.240:28908 (SSRC=0x42B8)
- Graph 4 Rvr Jitter: NONE:0 to NONE:0 (SSRC=0x0)
- Graph 5 Rvr Difference: NONE:0 to NONE:0 (SSRC=0x0)
- Graph 6 Rvr Delta: NONE:0 to NONE:0 (SSRC=0x0)

X Axis
Tick interval: 0.1 sec
Pixels per tick: 2

Y Axis
Scale: Auto

Figure 14. Performance test of G-711 and G-729 codecs

6.2.2.2. Scalability test of collaborative system and OpenICRA

To evaluate the automatic scaling capabilities of OpenICRA model as well as the scalability and stability of collaborative system when demand increases with the number of users, we conducted a stress test in order to measure under severe operating conditions the overall performance of containers hosted in a datanode of OpenICRA on Amazon. Further, we present in this section the followed steps to perform the stress test and the different results recorded.

In fact, a good stress test for the ICRA collaborative work system consists in generating a large load with multiple users. For this, we have chosen to perform this test in the Synchromedia laboratory [10] using its facilities and a customizable Shell script [42] to generate a large number of customers for overloading the collaborative system. On each workstation, we launched several customers (between 10 and 50 clients) connected to the same conference for overcharging the server and to monitor its behavior. Thus, measures of CPU utilization, throughput and memory of the server were done using the CloudWatch [40] and RevealCloud [41] monitoring tools.

In the following, we present the configuration of the stress test and the results of the CPU utilization, memory and throughput. In addition, the datanodes configuration in our system is as follows:

- 613 MB of memory;
- 2 EC2 Compute Units (1 virtual core with two EC2 Compute Units);
- EBS storage;
- Platform 64-bit Ubuntu 12.04 LTS;
- 71 Mbit/s of maximum throughput (network saturation when the interface usage exceeds 70%);
- Automatic scaling up by adding one or more instances, when the CPU utilization exceeds 90% (rules are configured in the load balancer).

Regarding the results of the stress test, figure 15 shows in detail the evolution of the CPU usage rate during the entire test period. In addition, it has increased from 6.5% when we have only 5 clients going through 17.49% with 60 clients and up to 86% when we have 105 clients. With 115 clients, the CPU usage rate has reached 92%, which is exceeded 90%. As a result, the load balancer has detected this overrun and it responded by creating a new instance of the ICRA application. At the same time, the number of clients continued to grow and the CPU usage rate too. At 10:46 p.m. UTC, the CPU rate begins to decrease. This is due to the load balancing between the two instances of collaborative system after the auto-scaling service outbreak.

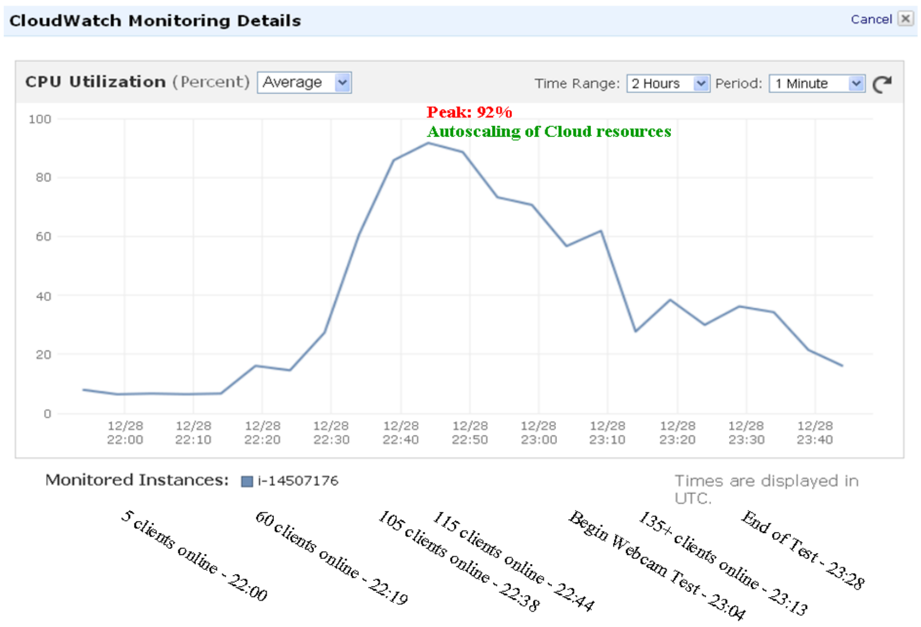


Figure 15. CPU utilization rate of ICRA system

For network performances, figure 16 illustrates the evolution of the used throughput of data transfer where bytes are sent and received via the network interface of the container hosted in the datanode. We notice that this container has made a peak with an upload speed of 5.76 MB/s while the download speed is around 120 KB/s. Thereafter, the upstream and downstream speeds were gradually decreased due to the automatic scaling triggered by the load balancer of OpenICRA platform. In addition to the 90% threshold of CPU, the load balancer is configured with an incrementing rule by an application instance when the network interface's utilization rate exceeds a threshold of 70% of the maximum throughput. As the peak value of the used throughput (5.76MB/s of upstream speed) did not exceed the threshold value (70% of maximum throughput ≈ 6.21MB/s), the network is not considered in a saturation situation. It is for this reason, that we have not had an outbreak of the second auto-scaling when the upstream speed has reached its maximum value.

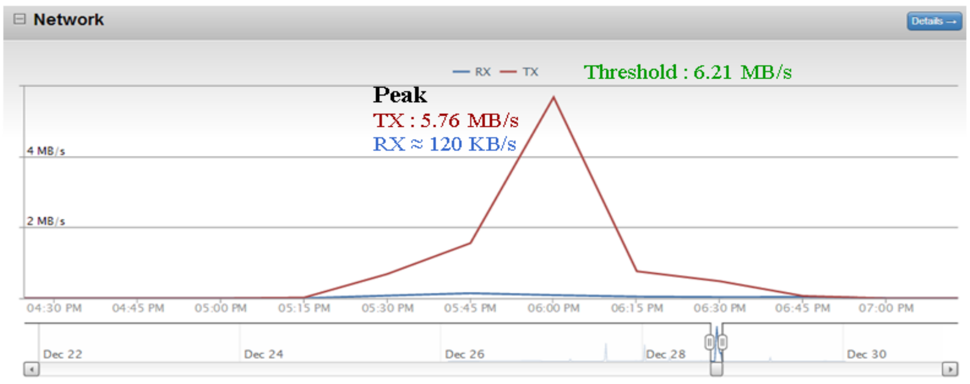


Figure 16. Used average throughput

Furthermore, the memory utilization rate is shown in figure 17. The active memory was initially around 31.6% for the first group of five users. Upon reaching 60 users in the group, active memory utilization rate has been increased to 37.8% and we even noticed an increase in the cache memory from 30.9% to 34.34% to reach a total value of 91.02%, including 18.88% of the buffer. With 115 users, the memory total value used is around 605 MB, which is the equivalent of 98% (sum of active memory, cache and buffer). Although the size of the active memory remains stable, we notice an increase in the cache memory and the one of the buffer which is perfectly consistent with the CPU usage decrease just after the outbreak of the auto-scaling by the OpenICRA load balancer. Note that the buffer is a zone of the RAM used to temporarily store data between two processes during their moving, to compensate for differences in the receiving rate and processing speed between two processes which do not operate the same rate, while the cache contains only duplication of the data to reduce the data access time.

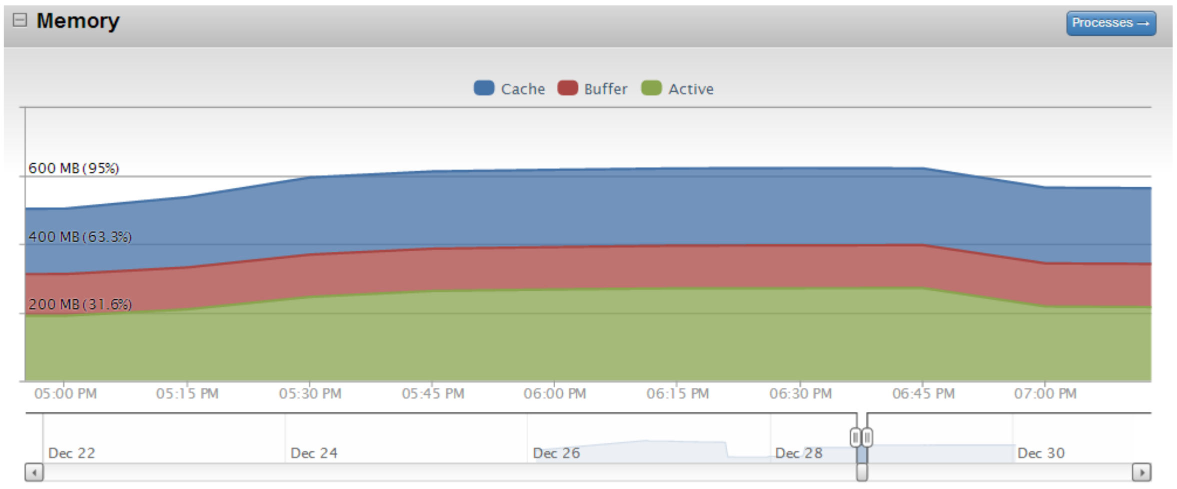


Figure 17. Memory utilization rate of ICRA server

According to this test, with 135 users connected at one time to the server, the CPU usage rate has reached 92%, which leads to an automatic provisioning of additional resources signaled by the load balancer to respond to the massive increase of client requests. A result of this reaction, the collaborative system has reestablished its normal functionning through the load balancing, which contributed to mitigate the overload of ICRA system containers. This clearly shows the scalability and flexibility of our OpenICRA system. It is important to mention that we have stopped adding more than 135 users for reasons of material resources lack.

In short, through this complex case study that combines several decentralized services depend on several providers such as Google Talk, IPKall, and AWS, our generic deployment model has proven its ability to automatically deploy in the cloud of any type of application with a process of three simple stages including preparation, deployment and monitoring. In addition, the result shows the ability of our model to react accurately in the event of increased demand and exceeding preconfigured thresholds by triggering an automatic scaling to reestablish the normal functioning of the deployed application and to balance the load.

7. CONCLUSION

This paper focuses on the major issues related to application deployment in the cloud, which is the most important challenge for developers when creating a new application or migrating an existing application to a cloud computing environment. In order to avoid vendor lock-in, to reduce applications development's complexity and implementation delays, to simplify service deployment's process, and also to ensure the scalability of applications in the cloud, we designed and developed a new generic model for automatic applications deployment in the cloud. Being built on open source technologies, our proposed model, OpenICRA, is designed to enable the developers move from a traditional environment to the cloud and vice versa. For this purpose, only open source languages and standard tools have been used in our model and there no proprietary APIs or technology resources that have been integrated. This ensures interoperability with cloud platforms and portability of applications both on OpenICRA platform and on other platforms, preventing vendor lock-in.

OpenICRA includes a full set of command line tools that provide full access to the developer interface. These tools are easy to use and also scriptable for automated interactions. Our proposed solution includes a rich Web console's interface for management and orchestration of cluster nodes in parallel and in real time, which reduce the complexity of application developement and management. Two concrete cases were considered to test and validate our model of automatic application deployment. The first case study consists in automating the deployment of the OpenSAF middleware across a cluster of nodes with a full control at infrastructure level within Ericsson environment [32], and the second case aimed to migrate the collaborative system ICRA to Amazon EC2 [2]. The results demonstrate the superiority and efficiency of our model to deploy different types of applications without making any changes in the source code of applications. In addition, our generic model has shown its performance in application migration and deployement on the cloud environment and its ability of providing automation of repetitive tasks by invoking services with real-time actions in a cluster of nodes. To fully benefit from the elasticity of the cloud, OpenICRA provides horizontal auto-scaling of resources based on the application load; eliminating the need of manual operations to add or delete instances of applications or application containers. The integration of load balancing technology with auto-scaling ensures scalability and elasticity of OpenICRA platform and hence guarantees the QoS. Furthermore, unlike other available cloud computing solutions, such as Cloud Foundry, OpenShift or CloudBees, this platform is not limited to the manual deployment of new applications in the cloud. But, it can be used to automate deployment of existing applications in the cloud using customizable and extensible modules and easily adaptable to different application architectural requirements.

This work therefore provides cloud users the opportunity to test and deploy their applications and to be familiar with cloud services. It also provides the ability to automatically and rapidly create applications in the cloud. Moreover, it not only contributes to the familiarization with the cloud, which is considered the architecture of the next generation of enterprise computing, but also to the environment protection. Indeed, it is compatible with the resources and ecoresponsible technologies of GSN, the first international project that aims to build the first green network with zero carbon [43]. Thus, this model supports the integration of VoIP service providers like Gtalk, IPKall, etc., and telecommunication applications deployment, particularly Asterisk and Freeswicth IPBX that allow interconnection between IP and PSTN networks, OpenIMS and webphone such as Mizu, ozeki, among others. Therefore, it contributes along the future vision of Ecolotic project toward improvement of migration approaches [36].

In our future works, several other researches will be considered. As this paper aims to develop a new generic model of automatic applications deployment in the cloud and in the vision of the Ecolotic project, significant efforts have been invested to support the automatic migration of telecommunication applications and services such as IMS (IP Multimedia Subsystem), WebRTC APIs and VoIP services to interconnect PSTN and IP networks. One objective of the OpenGSN middleware is to design an autonomous cloud management system, capable to ensure migration of services and applications to the cloud. Hence, integration of our generic deployment model with OpenGSN could achieve this goal. This makes the solution more independent and efficient. Also, this helps to stabilize our deployment model by considering concrete case studies such as the migration of the open source implementation of IMS core to a real cloud environment in Ecolotic project. Such integration will open undoubtedly the path towards other requirements, which require more advanced system in terms of speed of application implementation in the cloud system and deployment efficiency.

Finally, regarding the storage management in cloud environments, the way that we store data continues to evolve in an impressive manner at both the speed and the volume levels. Therefore, companies are looking for more effective approaches to manage big data such as images of VMs' Virtual Hard Disks (VHD), etc. The DFS (Distributed File System) can be a solution to resolve the problem of the ratio between the growths in the volume of VMs images and the compute nodes performance. Indeed, its integration at the IaaS level of cloud environments allows us to offer an excellent solution for applications and virtual machines where scalability is an issue in terms of both storage and computing speed. Concerning the storage pool accessibility, the integration of some DFS not fully POSIX compliant, such as HDFS, with the mountable module fuse-dfs resolves the data access problem by virtual machines and simplifies the management of large VHDs images by making the storage system more efficient and more scalable. Thus, in the perspective to obtain a powerful, a scalable and an automatically deployable storage system, that combines the strengths of the automating process of application deployment in the cloud computing and those of reliable warehousing of very large data files in the cloud and the high-speed access to data, we suggest to develop a generic storage model for cloud environments using HDFS, GlusterFS, Lustre, etc. as a distributed file system and OpenICRA as a PaaS of automatic application deployment in the cloud.

ACKNOWLEDGEMENTS:
The authors thank Rachid Hedjam, Reza Ferrahi Moghaddam and Guoqiang Zhong for their unconditional support of this work and the Synchromedia laboratory team for the help in implementing the experiments. The authors would like also thank the NSERC and the Canarie of Canada for their financial support.

REFERENCES

[1] L. Youseff, M. Butrico, and D. Da Silva, "Toward a unified ontology of cloud computing," in *Grid Computing Environments Workshop, 2008. GCE'08,* 2008, pp. 1-10.
[2] Amazon. Amazon Elastic Compute Cloud (Amazon EC2) [Online]. Available: http://aws.amazon.com/ec2, 2012.
[3] Google. Google App Engine [Online]. Available: https://developers.google.com/appengine/, 2012.
[4] Microsoft. Microsoft's Cloud Platform [Online]. Available: http://www.windowsazure.com, 2012.

[5] G. Juve and E. Deelman, "Automating application deployment in infrastructure clouds," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, 2011, pp. 658-665.
[6] North Bridge, "2012 Future of Cloud Computing Survey Exposes Hottest Trends in Cloud Adoption " in *2012 Future of Cloud Computing Survey*, SAN FRANCISCO, USA, 2013.
[7] HP, "Four steps to better application management and deployment," in *Business white paper*, 2012.
[8] M. A. Chauhan and M. A. Babar, "Towards Process Support for Migrating Applications to Cloud Computing," in *Cloud and Service Computing (CSC), 2012 International Conference on*, 2012, pp. 80-87.
[9] Amazon. Amazon web service (AWS) [Online]. Available: http://aws.amazon.com/, 2012.
[10] Synchromedia. GreenStar Network | Building a Zero Carbon Network [Online]. Available: http://www.greenstarnetwork.com/, 2010.
[11] J. Miranda, J. M. Murillo, J. Guillén, and C. Canal, "Identifying adaptation needs to avoid the vendor lock-in effect in the deployment of cloud SBAs," in *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups*, 2012, pp. 12-19.
[12] V. Tran, J. Keung, A. Liu, and A. Fekete, "Application migration to cloud: a taxonomy of critical factors," in *Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing*, 2011, pp. 22-28.
[13] B. Di Martino, D. Petcu, R. Cossu, P. Goncalves, T. Máhr, and M. Loichate, "Building a mosaic of clouds," in *Euro-Par 2010 Parallel Processing Workshops*, 2011, pp. 571-578.
[14] K. Chine, "Learning math and statistics on the cloud, towards an EC2-based Google Docs-like portal for teaching/learning collaboratively with R and Scilab," in *Advanced Learning Technologies (ICALT), 2010 IEEE 10th International Conference on*, 2010, pp. 752-753.
[15] I. Ross and G. Robert. The R Project for Statistical Computing [Online]. Available: http://www.r-project.org/, 2013.
[16] S. L. Campbell, J.-P. Chancelier, and R. Nikoukhah, *Modeling and Simulation in SCILAB*: Springer, 2006.
[17] W. Tian, S. Su, and G. Lu, "A framework for implementing and managing platform as a service in a virtual cloud computing lab," in *Education Technology and Computer Science (ETCS), 2010 Second International Workshop on*, 2010, pp. 273-276.
[18] N. K. Salih and T. Zang, "Survey and comparison for Open and closed sources in cloud computing," *arXiv preprint arXiv:1207.5480,* 2012.
[19] OCCI. Open Cloud Computing Interface [Online]. Available: http://occi-wg.org/, 2012.
[20] Favoritemedium.com, "KVM host with gateway guest using port-forwarding," 2011.
[21] K. Mike, "Adopt emerging specifications for cloud resource control," in *Red Hat*, 2012.
[22] J. Guillén, J. Miranda, J. M. Murillo, and C. Canal, "A service-oriented framework for developing cross cloud migratable software," *Journal of Systems and Software,* 2013.
[23] S. Dowell, A. Barreto, J. B. Michael, and M. T. Shing, "Cloud to cloud interoperability," in *System of Systems Engineering (SoSE), 2011 6th International Conference on*, 2011, pp. 258-263.
[24] D. Petcu, "Portability and interoperability between clouds: challenges and case study," *Towards a Service-Based Internet,* pp. 62-74, 2011.
[25] Small Tree. Shared Storage [Online]. Available: https://www.small-tree.com/shared_storage_a/212.htm, 2013.
[26] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM SIGOPS Operating Systems Review*, 2003, pp. 29-43.
[27] D. Borthakur, "HDFS architecture guide," *Hadoop Apache Project. http://hadoop. apache. org/common/docs/current/hdfs_design. pdf,* 2008.
[28] F. B. Schmuck and R. L. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in *FAST*, 2002, p. 19.
[29] T. Haynes and D. Noveck, "Network File System (NFS) version 4 Protocol," *Network,* 2013.
[30] V. I. Miloushev and P. A. Nickolov, "Aggregated opportunistic lock and aggregated implicit lock management for locking aggregated files in a switched file system," ed: Google Patents, 2011.
[31] T. Zhao, V. March, S. Dong, and S. See, "Evaluation of a performance model of Lustre file system," in *ChinaGrid Conference (ChinaGrid), 2010 Fifth Annual*, 2010, pp. 191-196.
[32] Ericsson. Ericsson - A world of communication [Online]. Available: http://www.ericsson.com/, 2013.
[33] M. Cheriet. ICRA [Online]. Available: http://tokra.synchromedia.ca/icra/, 2012.
[34] OpenSAF team. Getting Started with OpenSAF [Online]. Available: http://devel.opensaf.org/wiki/GettingStartedWithOpenSAF, 2013.

[35] Business Wire. OpenSAF Reports Large Gains in Market Traction [Online]. Available: http://python.sys-con.com/node/1753543, 2011.
[36] Ericsson, CGI, FUJITSU, IBM, MIRANDA, TELEDYNE, and DALSA. A major GreenICT Initiative [Online]. Available: http://www.equationict.com/, 2011.
[37] Synchromedia. Manual deployment duration of OpenSAF [Online]. Available: http://www.surveymonkey.com/s/Y5Q9DSL, 2013.
[38] L. Sun, I.-H. Mkwawa, E. Jammeh, and E. Ifeachor, "Case Study 1—Building Up a VoIP System Based on Asterisk," in *Guide to Voice and Video over IP*, ed: Springer, 2013, pp. 193-213.
[39] IPKall. Free Washington state phone number to your Internet phone [Online]. Available: http://www.ipkall.com/, 2013.
[40] Amazon. Amazon CloudWatch [Online]. Available: http://aws.amazon.com/cloudwatch/, 2013.
[41] CopperEgg. Real-time insight into server performance and services deployed in public and private clouds. [Online]. Available: http://www.copperegg.com/revealcloud-server-monitoring/, 2013.
[42] D. Fred. Stress Testing the BigBlueButton Server with Many Clients [Online]. Available: https://github.com/bigbluebutton/bigbluebutton/blob/master/labs/stress-testing/bbb-test, 2013.
[43] F. F. Moghaddam, M. Cheriet, and K. K. Nguyen, "Low carbon virtual private clouds," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 2011, pp. 259-266.