Managing Elasticity Across Multiple Cloud Providers

Fawaz Paraiso, Philippe Merle, Lionel Seinturier

Managing Elasticity Across Multiple Cloud Providers

Fawaz Paraiso
University of Lille & Inria Lille - Nord Europe
LIFL UMR CNRS 8022, France
fawaz.paraiso@inria.fr

Philippe Merle
University of Lille & Inria Lille - Nord Europe
LIFL UMR CNRS 8022, France
philippe.merle@inria.fr

Lionel Seinturier*
University of Lille & Inria Lille - Nord Europe
LIFL UMR CNRS 8022, France
lionel.seinturier@inria.fr

ABSTRACT

In the context of cloud computing, elasticity is the capacity to scale computing resources up and down easily. Currently, most Platforms as a Service (PaaS) manage application elasticity within a single cloud provider. However, the not so infrequent issue of cloud outages has become a concern that hinders the availability of cloud-based applications. The most promising solutions to this issue are those based on the federation of multiple clouds. In this paper, we present a *Multi-Cloud-PaaS* architecture. We show how this architecture can be used for managing elasticity across multiple cloud providers.

Categories and Subject Descriptors

H.4 [**Cloud Computing Architecture**]: Multi-Cloud; D.2.11 [**Software Architectures**]: Data abstraction—*Elasticity, Dynamic Load Balancing*

General Terms

Design

Keywords

Multi-Cloud, PaaS, Elasticity, High availability, Dynamic load balancing.

1. INTRODUCTION

Cloud computing allows companies to respond to business opportunities more quickly, without the cost of buying, managing, and maintaining their own computing infrastructure. Cloud computing brings a way to innovate and transform Information Technology (IT) into a set of flexible services that enable a more agile approach than what is possible with traditional computing models. The Infrastructure as a Service (IaaS) layer orchestrates virtualized infrastructures and provides a better utilization of resources (cpu, disk, memory, network, etc.). On the top of IaaS, the Platform as a Service (PaaS) layer supports the development and management of applications. The Software as a Service (SaaS) layer is the application model deployed with PaaS. Cloud computing is a delivery model for IT that provides *elasticity*[1], agile services in an on-demand and pay-as-you-go fashion. Due to its flexibility in provisioning and using services, Cloud computing can optimize IT investment [2], application *availability*[2] and scalability.

These characteristics and benefits make Cloud computing an essential part of the IT infrastructure of many companies. However, as [14] and [15] have already noted, the Cloud computing model is not without service unavailability. A series of news [16, 17, 18] and papers [19, 20, 15, 21] have pointed cloud provider outages. For instance, Table 1 summarizes a list of recent outages classified by cloud provider. Table 1 underlines only the outages that have an high impact and that put services offline for at least one hour. According to a recent report by the International Working Group on Cloud computing Resiliency [22], a total of 568 hours of downtime at thirteen well-know cloud services since 2007 caused financial damage of more than US$71.7 million. The average unavailability of cloud services is 7.5 hours per year, amounting to an availability rate of 99.9%, according to the group preliminary results. These results are far from the expected reliability of mission critical systems (99.999%). As a comparison, the service average unavailability for electricity in a modern capital city is less than 15 minutes per year [23]. The impact of 0.099% unavailability costs US$71.7 million. Besides this economic impact, the downtime also affects millions of end-users. Of course, downtime costs money and dammage, unfortunately protect systems against downtime with 99.999% of availability is not free. Some cloud providers [24, 25, 26] propose several distributed data centres to minimise failure. However, vendor lock-ins (switching cost, etc.) are still a potential risk in the context of a single cloud provider. Additionally their solutions are not effective when all data centres are down. Although most cloud providers today claim to bring availability, but in reality, it is reasonable to assume that even major players [24, 25, 27, 26, 28] have faced availability problems.

In addition, many criteria [29, 30] can be taken into account when choosing the right platform, since most of the time no single cloud provider can meet all the needs. Federating multiple clouds and brokering resources between these different clouds is a solution of choice for these specific needs [31, 32].

To overcome these difficulties, we propose in this paper MCP, a Multi-Cloud-PaaS solution that focuses on the management of

*IUF - Institut Universitaire de France

MultiCloud'13, April 22, 2013, Prague, Czech Republic.

[1] *Elasticity* is the capability to rapidly provision, in some cases automatically, to quickly scale out, and rapidly release resources [1].
[2] *Availability* is a degree to which applications and resources are in an operable and committable state at the point in time when they are needed [3].

Table 1: List of Cloud Outages

	Cloud outages		
Cloud Providers	**What happened & Why**	**When**	**What impact**
Amazon	[4] The event was triggered during a large scale electrical storm which swept through the Northern Virginia area.	29 June 2009	Offline for 5 hours and 30 minutes. Companies affected: Netflix, Instagram, Pinterest, Heroku.
	[5] The loss of power was caused by an electrical ground fault and short circuit in a major power distribution panel that interrupted power to some instances in this particular availability zone.	4 and 8 May 2010	Offline for 7 hours. Data loss for small number of users.
	[6] The problems are focused on Elastic Block Storage (EBS), which provides block level storage volumes for use with Amazon EC2 instances.	21 April 2011	Offline for more than 10 hours. Companies affected: reddit, Quora, HootSuite.
	[7] A power outage at an Amazon Web Services data center in North Virginia knocked some customers offline.	30 June 2012	Offline for more than 5 hours. Companies affected: Netflix, Instagram, Pinterest, HotSuite, Heroku.
	[8] The Amazon Elastic Load Balancing (ELB) Service down in US-East region affected the applications using the ELB.	25 December 2012	Offline for more than 23 hours. Companies affected: Netflix.
Windows Azure	[9] A networking problem during a routine software update interfered with hosted project deployment.	13 March 2009	Offline for 22 hours.
	[10] It is caused by a software bug. The time calculation was incorrect for the leap year.	29 February 2012	Offline for more than one hour.
Rackspace	[11] A power outage at a data center that caused it.	29 June 2009	Offline for about one hour.
Heroku	[12] Amazon outage affected Heroku.	29 June 2009	-
Salesforce.com	[13] No explanation of what went wrong.	4 January 2010	Offline for about one hour and 15 minutes.

elasticity through multiple cloud providers using dynamic provisioning. The MCP architecture is composed of load balancer, node provisioning, monitoring, workload manager and controller components. MCP helps developers to create elastic applications by reducing the complexity of managing multiple different cloud architectures.

The remainder of this paper is organized as follows. In Section 2 we present the motivation of our work. Next, Section 3 describes the architecture promoted by our solution, the integration with existing IaaS/PaaS, and the implementation details of our platform. Then, Section 4 describes some preliminaries evaluation of the load balancer. Section 5 presents and discusses the limitations of this work. Section 6 compares our platform with the state-of-the-art, while Section 7 concludes this paper and presents future work we intend to address.

2. MOTIVATION

As a motivating scenario that will be used later in Section 4, let us consider an IT company in charge of a [3] network in a bank with several agencies. Let us suppose that each camera captures at least one image every δt seconds and sends it in JPG format to a remote server. An application is responsible for checking if the camera sends the image on time and if the JPG format is correct. The application generates an alert message if the camera does not work properly. The IT company deploys the application on a single cloud provider. Fault-tolerance, high availability, resources low cost, and scalability are essentials prerequisites for this application. Thereby, in order to meet the application requirements, the following are questions that the IT company has. How to choose the right cloud platform that meets the application needs? What if the entire data center looses electrical power? What if the data center is destroyed by floods or fire? What if the data center has another kind of outage? At a first glance, these questions point out the limits of what a single cloud provider can deliver. For this reason, managing elasticity across multiple cloud providers is the better way to guarantee high availability when outages occur.

To cope with this variety of requirements and needs, we advocate what the following three main challenges need to be addressed in order to manage elasticity across multiple cloud providers.

High availability.

To ensure the availability despite these outages, multiple instances of the application should be deployed and launched on different cloud providers with a load balancer service to distribute requests among instances of the application.

Automate elasticity through multiple clouds.

With a single cloud provider, when an outage takes in all data centres, resource provisioning is not possible. Cloud elasticity is essential to accommodate the scale (up and down) on demand. A possible solution to avoid outages is to automate the cloud elasticity across different cloud providers.

Transparency.

Each cloud provider has his own API to manage elasticity. The aim is to provide an abstraction support to hide certain aspects of elasticity management to the application developers so that they need only be concerned with the design of their applications.

This paper provides a solution to these challenges, called Multi-Cloud-PaaS (MCP).

3. MULTI CLOUD PAAS DESIGN

In this section we present the MCP platform. We begin by presenting the platform architecture. Next, we describe the implemen-

[3] *CCTV* http://en.wikipedia.org/wiki/Closed-circuit_television

tation of the MCP platform and how the existing IaaS/PaaS solutions are integrated.

3.1 Architecture

Figure 1 gives an overview of the architecture of MCP. This architecture describes all the components that automate elasticity. These include full instruments for monitoring workloads, resources provisioning, load balancing, and all controller services needed to manage elasticity. In order to avoid a single point of failure, the MCP architecture should be deployed at least in two different cloud providers (cf. Figure 1). MCP architecture deployment is shown in Figure 1, in which the *load balancer*, *controller*, *node provisioning*, *workload manager*, *PaaS deployment*, and *SaaS deployment* components are deployed in cloud A. The cloud B contains the replication of these components. The *monitoring* component is deployed with the application in cloud 1 to n (cf. Figure 1). Except the *monitoring* component, there are two components per application deployed.

3.1.1 Load Balancer

The application deployed with MCP is replicated on multiple clouds (APP in Figure 1). The *load balancer* (LB) component routes requests to application instances. The LB dispatches load among different cloud providers. Before distributing load the LB asks for system information (broadcasts an alive-request-message, indicating that it is running), which from the LB point of view is a collection of application instances available for execution. The current implementation of the LB supports the Round Robin algorithm [33]. An API is proposed to extend this default behaviour with other algorithms. A Uniform Resource Locator (URL) is associated with one instance of LB. Each application deployed with the MCP has two instances of LB. The two LB instances belong to the same group. Although, only one instance of LB is active, the second one is passive.

3.1.2 Controller

The *controller* component multiplexes workloads onto an existing infrastructure, allows for on-demand allocation of resources to workloads. The system state is managed by the *controller* component. By state, we mean that information retained in one component that describes something (As example: a small table kept on each instance of LB to associate network addresses with the textual names of available hosts), or is determined by something. The system state offers the potential for improving the coherency, and reliability of the system. For components to work together effectively, they must agree on common goals and coordinate their actions. This requires each part to know something about the other. For example, the *node provisioning* keeps a table of available resources: If the developer wants to deploy an application on the resource, the controller can notify the *node provisioning* to allocate new resources for the application when the available resource is not sufficient. The second potential advantage of the system state is reliability. If information is replicated at several cloud providers and one of the copies is lost due to a failure, then it may be possible to use one of the other copies to recover the lost information. Compared to the *workload manager* and *node provisioning* components, the *controller* takes decision in the system. The Controller component should be self-adaptive in order to respond in a coherent and timely manner to changes in environment, failures of components.

3.1.3 Provisioning

The Node Provisioning (NP) component allocates and deallocates IaaS resources as needed. To accommodate the current or predictive workloads, the NP provides the necessary resources needed. When resources are no longer used or underutilized, they are released. The decision for allocating and deallocating comes from the *controller* component. The resource allocation can be provided from different cloud providers. The ability to provide resources dynamically and quickly from different clouds is essential.

3.1.4 Monitoring

The *monitoring* component provides metrics about the application process and the system on which it is running. Information about currently executing process as well as the system that the monitoring service can be gathered. The *monitoring* component provides the following information:

- os: name, version, kernel, processes, memory, swap, resource limits, uptime and logins.
- cpu: model, name, family, per cpu and average usage.
- filesystem: mounted devices, disk usage, filesystem properties and usage.
- network: usage, bandwidth, interface, routes and connection status.
- process: per process information for cpu, memory, environment, credentials, arguments and other information.

The *monitoring* component notifies on any change in state of the application. The metrics collected in a time interval are sent to the Workload Manager component for analyzing.

3.1.5 Workload Manager

The Workload Manager (WM) component provides some event processing functionality [34]. All events are processed to extract drift indicators (DI). An example of DI can be a CPU consumption is greater than 90% for a period of 2 minutes. The WM is centered on DI tracking perform filtering, transformation, and most importantly aggregation of events. All the metrics (events data) sent by the *monitoring* component are continuously analyzed in terms of drift indicators that are expressed by event rules, and acts upon opportunities and threats in real time, potentially by creating derived events and forwarding it. One of WM major goals is to find a symptom and analyses it to find it root cause. The WM uses a technique called event *correlation*[4] to examine symptoms and identify groups of symptoms that have a common root cause. As an example of event correlation, WM takes multiple occurrences of the same event, examines them for duplicate information, removes redundancies and reports them as a single event. So fifty ”memory consumption greater than 98%” alerts become a single alert that says "memory consumption greater than 98%, fifty times". The WM looks for particular patterns among the events that it monitors. When a drift occurs, the WM reports it to the *controller* component. The ability to derive instant insights into the operations of the resource provisioning is essential. Thus, the dynamic resource allocated and deallocated are important ingredient to build a platform for elastic applications. This feature allows the MCP to handle an increasing volume of transactions, services and persistence data.

[4]See a Computerworld article described event correlation in network and system management at `http://www.computerworld.com/s/article/83396/Event_Correlation?taxonomyId=16&pageNumber=1`.

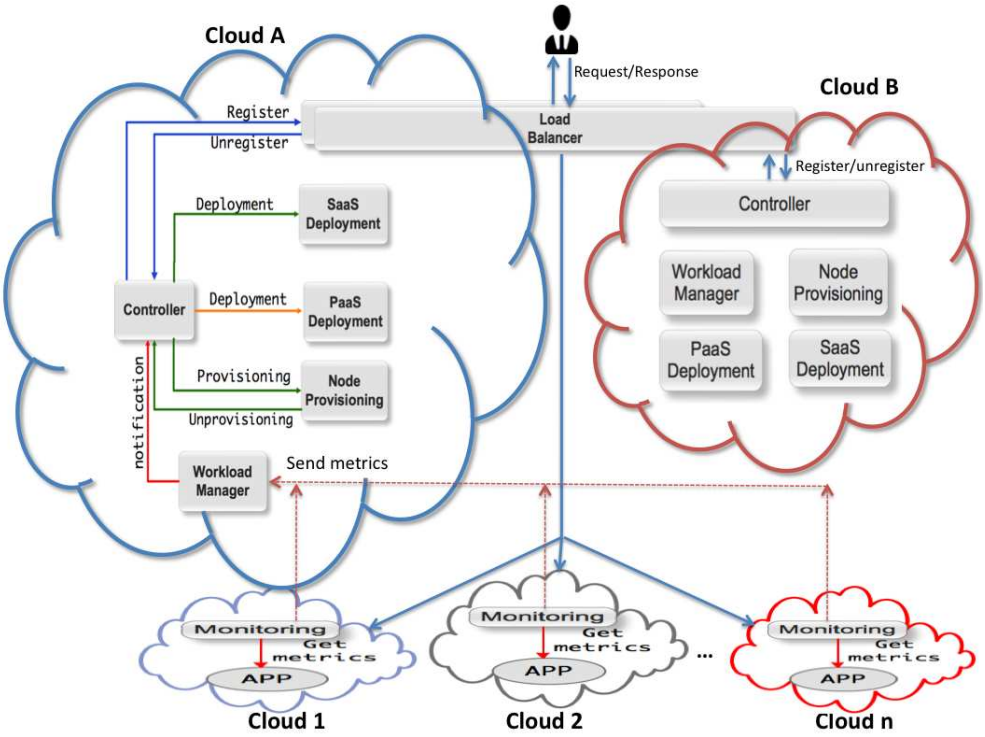

Figure 1: Overview of the Multi-Cloud-PaaS Architecture.

3.1.6 PaaS Deployment Service

This service deploys an instance of the Multi-Cloud PaaS on the target cloud provider. This feature is based on our previous work that deals with a generic solution for deployment in distributed environments [35]. The Multi-Cloud PaaS is deployed as a WAR file.

3.1.7 SaaS Deployment Service

This service allows to dynamically deploy/undeploy SaaS applications running on the Multi-Cloud PaaS nodes. The SaaS deployment service should take into account many complex factors (design of application's architecture, database, etc). It must be able to handle hundreds of simultaneous customers.

3.2 Implementation

This section describes how the implementation of our MCP platform is achieved. MCP relies on FraSCAti [36] that is an open source platform for deploying and executing service-based applications. FraSCAti provides a component-based programming model which simplifies the development, assembly, deployment and management of composite applications. In addition, FraSCAti provides a unified way to build applications. Each component of the MCP architecture is implemented as an SCA component and these components are assembled together to form the platform. The MCP can run in a distributed fashion and managed as virtual unit platform. The Figure 2 shows the SCA architecture of MCP.

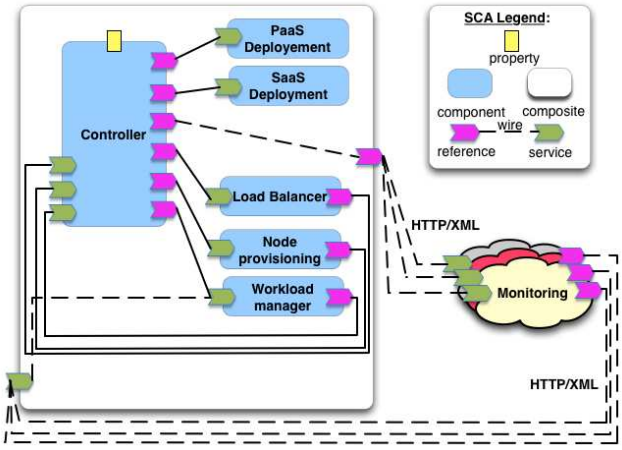

Figure 2: MCP SCA-Based components.

3.2.1 Provisioning

The NP provides an abstraction layer for compute and storage clouds. This component provides a unified way to the MCP to use the resources as needed. The compute and storage clouds are exposed as services. To achieve resource provisioning across differ-

ent cloud providers, the NP component uses JClouds[5]. JClouds is a Java API for compute and storage through multiple clouds. We use this API because it supports many Clouds[6].

3.2.2 Monitoring

Application can run in standalone or distributed fashion depending on their design. The main question is how to monitor a highly distributed and heterogenous environment? The Monitoring component exposes services via REST and JMS to monitor a distributed environment. This component encapsulates a lower-level API called SIGAR developed in C. The Hyperic SIGAR API [37] was chosen because of its cross-platform support and (relatively) small computational footprint. Monitoring application activity that occurs outside of the JVM is not possible using pure Java. The Monitoring component uses a convenient way to access native libraries from pure Java code, using a C binding provided by FraSCAti. The consumer of the services exposed is the *workload manager* component or can be an external API running outside the MCP. Each application deployed with the MCP is automatically monitored. The *monitoring* component uses the aggregation mechanism to monitor standalone or distributed applications.

3.2.3 Workload Manager

Related to the events received from an inbound *monitoring* component, how events can be woven together to pull out the right information? This is accomplished through Complex Event Processing (CEP)[7]. To achieve this, we use DiCEPE, a Distributed Complex Event Processing Engine we have presented in our previous works [39]. The particularity of DiCEPE is the integration of CEP engines in distributed systems, and the fact that they can be exposed via various communication protocols. Event patterns, as the name suggests, allow to define, via expressions, various matching rules that can be applied to incoming events. For instance, a temporal operators can be used to compare inbound events and check whether they arrived in the anticipated order. Patterns enable us to have very fine-grained control over how events are evaluated. Overall, the WM references the *monitoring* component and also exposes services.

3.2.4 Load Balancer

As mentioned below each application has one instance of LB. It is offering high availability, load balancing, and proxying for TCP and HTTP-based applications. To realize the LB component we have implemented a non-blocking server in order to have scalability and performance. Non-blocking servers are not threaded and they use an IO loop and events to handle requests. In order to take full advantage of non-blocking technology, all of our IO and network calls have to be non-blocking as well. The LB implements an event-driven, single thread model which enables support high number of simultaneous connections at very high speed. Multi-process or multi-threaded models can rarely cope with thousands of connections because of memory limits, system scheduler limits, and lock contention everywhere. Event-driven models do not have these problems because implementing all the tasks in user-space allows a finer resource and time management [40]. To achieve the implementation of the LB component, we use the Netty framework[8]. For dynamic load balancing capabilities, a group membership service is used. To avoid a single point of failures, the LB also needs to be replicated. Hence, each application has a DNS name [41] (for example: appname.mcp.net which is associated) with two LB instances. The application deployed with the MCP is associated with the group of LB as shown in Figure 3, specifically with the active instance of the LB. When the active instance of LB fails, the passive instance of LB detects the failure and takes over as the active LB. The passive instance will check if the previous active instance has actually failed, and if so, it report to the *controller* component for instantiating a new passive LB.

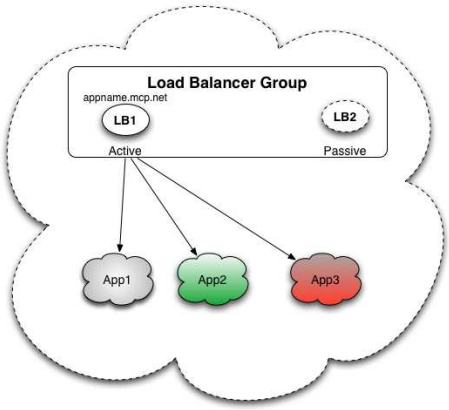

Figure 3: Load Balancer group: active-passive.

3.2.5 Controller

All components in the MCP architecture are registered with the *controller* component. The *controller* component is responsible for taking decisions and managing the behaviour at runtime of each component according to the requirements. All requests handled by a Controller component are processed as transactions. The engine transaction is implemented for the specific needs of the MCP architecture. The goal of transactions is to ensure that all components managed by the *controller* remain in a consistent state when they are accessed by multiple transactions and in presence of crash. The *controller* must guarantee that either the entire transaction is carried out and the results recorded in permanent storage or, in the case that one or more of them crashes, its effects are completely rollbacked. Each transaction is created and managed by a coordinator. Two well-known problems of concurrent transactions can be mentioned: i) lost update and ii) inconsistent retrievals. To avoid these problems we use a serially equivalent[9] executions of transactions. The use of serial equivalence as a criterion for correct concurrent execution prevents the occurrence of lost updates and inconsistent retrievals. The *controller* component is the core of elasticity management, it is made to tolerate failures by the use of redundant components.

3.3 Integration with existing IaaS/PaaS

We report on the existing cloud environments on which the MCP platform has been deployed. The MCP platform is actually deployed on ten target cloud environments that is publicly acces-

[5] http://www.jClouds.org/

[6] http://www.jClouds.org/documentation/reference/supported-providers/

[7] **CEP**: Computing that performs operations on complex events, including reading, creating, transforming, or abstracting them. [38]

[8] https://netty.io

[9] **serial equivalence**: if each of several transactions is known to have the correct effect when it is done on its own, then we can infer that if these transactions are done one at a time in some order the combined effect will also be correct [42].

sible on the Internet[10]. The deployment is done with IaaS/PaaS providers. With IaaS, resources are provisioned from Windows Azure[11], DELL KACE[12], and Amazon EC2[13], we installed a PaaS stack composed of Linux distribution, a Java Virtual Machine, a web container and FraSCAti. We also provide PaaS resources from CloudBees[14], OpenShift[15], dotCloud[16], Jelastic[17], Heroku[18], Appfog[19], Eucalyptus private Cloud. The MCP platform extends an experiment that was presented in our previous work [43].

4. VALIDATION

In this section, the preliminary experiments of the MCP architecture focuses on the Load Balancer performance. In fact, the LB distributes incoming requests that can introduce overhead.

4.1 Load Balancer Overhead

Considering our scenario mentioned in Section 2, we are specially interested in the manager that monitors the video cameras. Since the cameras send the images capture to the server, our manager can monitor for **SEND** events. To evaluate our LB, we try to focus on the introduced overhead. To focus on the overhead introduced by the LB, with Internet lag, all the benchmark experiments were performed with two different cloud providers. The LB and the application were deployed respectively on Windows Azure and DELL KACE.

To evaluate the overhead of the LB instance, 10,000 images were sent to the application. We evaluated two cases of this scenario where the **SEND** event and images were checked: i) directly with the Application, and ii) with the LB. The benchmark was executed ten times on each of the two cases. In Table 2, we present the results of the average execution time for each case, as well as the mean overhead introduced by the LB run-time.

Table 2: Execution time and overhead

Implementation	Avg. exec. time	LB overhead
APP	13.93 sec	-
APP + LB	14.10 sec	1.45%

Overall, these benchmarks show that there is a small overhead introduced by adding the LB layer, the execution time is still acceptable and the benefits provided by the MCP platform (cf. Section 3) outweigh the difference in the execution time.

4.2 Performance

In order to evaluate the performance of our load balancer component, we use a benchmark tool called inject32[20]. Using this tool, we generate $134,021$ requests and continuously connects to one instance of LB to fetch the selected object from the server in loops for $156,000$ ms. Statistics are collected every second, so we have 156 measures. The network bandwidth is measured at the HTTP level and does not account for TCP acks nor TCP headers. This tool measures the load balancer performance with three important factors:

- **The session rate**: This factor directly determines when the load balancer will not be able to distribute all the requests it receives. It is mostly dependent on the CPU.
- **The session concurrency**: Generally, the session rate will drop when the number of concurrent sessions increases. The slower the servers are, the higher the number of concurrent sessions for a same session rate.
- **The data rate**: This factor generally is at the opposite of the session rate. It is measured in Megabytes/s (MB/s), or sometimes in Megabits/s (Mbps).

Table 3: Performance benchmark results

Session rate	Concurrency	Data rate	Failures	Avg. time
850	283	4560 kB/s	0	3 ms

To focus on the performance of our load balancer component in real condition, the benchmarks were performed on a Windows Azure Cloud, using a Virtual Machine with a 2.0 GHz AMD Opteron (tm) processor, 3.5 GB RAM, Ubuntu Server 12.0.4 LTS 64 Bit and Oracle Java 6. As noticed in Table 3, $134,021$ requests were send to the load balancer, the session rate is 850 HTTP requests per second, with 283 concurrent connections which have the data rate (HTTP headers+data only) of 4560 kB/s. All HTTP requests were processed without error. Of course this takes into account the added work induced by network traffic. The load balancer has at average 283 concurrent sessions. This number is limited by the amount of memory and the amount of file-descriptors the system can handle.

Our LB is software-based[21] (Layer-7 of the OSI model) load balancer. Obviously, the best performance can be reached by adding the lowest overhead, which means processing the packets at the network level. One of the most common questions when comparing hardware-based to software-based load balancers is why such a gap between their session count exists. In fact, it depends whether the load balancer has to manage TCP/IP stack or not. TCP/IP stack requires that once a session terminates, it stays in the table in TIME_WAIT state long enough to catch late retransmits, which can be seen several minutes after the session has been closed. After that delay, the session is automatically removed [44]. The sessions in this state do not carry any data and are very cheap. Since they are transparently handled by the OS, a proxy never sees them and only announces how many active sessions it supports. But when the load balancer has to manage TCP, it must support very large session tables to store those sessions.

Overall, given the low resources used by the LB, the results obtained in Table 3 are satisfactory.

5. DISCUSSION AND LIMITATION

This section presents and discusses the evaluation in Section 4 and the limitation of this work. Our preliminaries evaluation only

[10] available at http://multicloudpaas.soceda.cloudbees.net/
[11] https://www.windowsazure.com
[12] https://www.kace.com/
[13] http://aws.amazon.com/ec2/
[14] http://www.cloudbees.com/
[15] https://openshift.redhat.com
[16] https://www.dotcloud.com/
[17] http://jelastic.com/
[18] http://www.heroku.com/
[19] http://www.appfog.com/
[20] Available at http://1wt.eu/tools/inject/
[21] Layer-7 load balancing involves cookie-based persistence, URL switching and such useful features (application availability and scalability).

focus on the LB component. In fact, the LB distributes incoming requests that can introduce overhead. What we expect from this evaluation is that the overhead introduced by the LB should be negligible in order to ensure acceptable response time. As reported in Section 4, the overhead introduced is small and the performance of the LB is also satisfactory. We assume that other aspects of evaluation should be taken into account to validate the MCP architecture.

Currently, the algorithm strategy that our LB uses is round-robin. However, there are other sophisticated load balancing strategies [45, 46] that are not yet tested. Therefore, there is a need to find the load balancing technique that can improve the performance by balancing the workload across all the nodes in a multi-cloud environment. The application replicated through the multi cloud do not interact with data base. In fact, the mechanism of data replication on multi cloud is not taken into account in this work. Several cloud providers have different and increasingly more advanced services for elasticity and also replication, these features are also not necessarily compatible. The solution based on multi cloud tends to be constrained by the least common denominator of features provided. Also, exploitation of new specific features will not necessarily be accessible through a multi cloud solution or their exploitation may be delayed since the multi cloud solution needs to be updated accordingly.

6. RELATED WORK

This section presents some of the related work from different fields of research that are relevant to our contribution. Managing elasticity across multiple cloud providers is a challenging issue. However, although managed elasticity through multiple clouds would benefit when outages occur, few solutions are supporting it. For instance, in [32] the authors present a federated cloud infrastructure approach to provide elasticity for applications, however they do not take into account elasticity management when outages occur. Another approach was proposed by [47], which managed the elasticity with controller and load balancer. However, their solution does not address the management of elasticity through multiple cloud providers. The authors in [48] propose a resource manager to manage application elasticity. However their approach is specific for a single cloud provider. Amazon EC2, Windows Azure, Jelastic already provide a load balancer service with a single cloud to distributed load among virtual machines. However, they do not provide mechanism to allow LB across multiple cloud providers. Different approaches of dynamic load balancing have been proposed in the literature [49], [50], [51], however they do not provide a mechanism to scale the load balancers themselves. The authors in [52], [53] have explored the agility way to quickly reassign resources. However, their approach do not take into account a multi cloud environment.

7. CONCLUSION

This paper presents a Multi-Cloud-PaaS architecture to manage elasticity across multiple cloud providers. Related to the Inter-Cloud Architectural taxonomy presented in [54], our work can be classified into the Multi-Cloud service category. We surveyed each of the concepts related to manage elasticity across multiple clouds and pointed out problematics. To address these problems, this paper proposes an architecture, describes the interactions between each component of this architecture. The purpose of the Multi-Cloud-PaaS architecture presented in this work is to demonstrate the feasibility of our approach. The integration with existing solutions and the evaluation results show significant benefits to cloud users and cloud providers. In future work we plan to address the following two main points. First, evaluate other MCP architecture components and improve the proxy-based load balancer performance. Second, we will investigate how the concept of federated multiple clouds can be used to reduce the resource provisioning cost, while maintaining the Quality of Service (QoS) to customers who use the resources.

Acknowledgment

This work is partially funded by the ANR (French National Research Agency) ARPEGE SocEDA project and the EU FP7 PaaSage Project.

8. REFERENCES

[1] P. Mell and T. Grance, "The NIST definition of cloud computing (draft)," *NIST special publication*, vol. 800, p. 145, 2011.

[2] KPMG, "Clarity in the Cloud : A global study of the business adoption of Cloud," 2011. http://tinyurl.com/chfdzun.

[3] V. Katukoori, "Standardizing availability definition," *University of New Orleans, New orleans, La., USA*, 1995.

[4] Seth Eliot, "A Summary of the Amazon Web Services June 29 Outage," June 2009. http://tinyurl.com/cokv63t.

[5] Rich Miller, "Amazon Addresses EC2 Power Outages," May 2010. http://tinyurl.com/3akfv7q.

[6] Rich Miller, "Major Amazon Outage Ripples Across Web," April 2011. `http://tinyurl.com/3qpp2ts`.

[7] Barb Darrow, "Latest outage raises more question about Amazon cloud," June 2012. `http://tinyurl.com/82oqfh2`.

[8] Amazon, "Summary of Amazon ELB Service outage in the US-East Region," December 2012. `http://tinyurl.com/bjnxn7w`.

[9] "Microsoft MSDN AZURE Outage," March 2009. http://tinyurl.com/clp2x2v.

[10] "Microsoft MSDN AZURE Outage," Feb 2012. http://tinyurl.com/cdy4scn.

[11] "What Went Down At Rackspace Yesterday? A Power Outage And Some Backup Failures," June 2009. http://tinyurl.com/34j9tyf.

[12] "Heroku Outage," June 2009. http://tinyurl.com/cxoomya.

[13] "Salesforce.com Hit by One Hour Outage," June 2009. http://tinyurl.com/yfxfwm5.

[14] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[15] N. Leavitt, "Is cloud computing really ready for prime time," *Growth*, vol. 27, no. 5, 2009.

[16] InfoWorld, "The 10 worst cloud outages (and what we can learn from them)." http://tinyurl.com/br9ck4a.

[17] Zdnet, "Amazon cloud down; Reddit, Github, other major sites affected," October 2012. http://tinyurl.com/95kmk8y.

[18] SearchCLoudComputing, "Cloud computing outages: What can we learn?." http://tinyurl.com/cnnlrg3.

[19] G. Briscoe and A. Marinos, "Digital ecosystems in the clouds: Towards community cloud computing," *CoRR*, vol. abs/0903.0694, 2009.

[20] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: outsourcing computation without outsourcing control," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, pp. 85–90, ACM, 2009.

[21] W. Jansen, "Cloud hooks: Security and privacy issues in cloud computing," in *System Sciences (HICSS), 2011 44th Hawaii International Conference on*, pp. 1–10, IEEE, 2011.

[22] "International Working Group on Cloud Computing Resiliency." http://www.iwgcr.org/.

[23] Maurice Gagnaire, Felipe Diaz,Camille Coti, Christophe Cerin, Kazuhiko Shiozaki, Yingjie Xu, Pierre Delort, Jean-Paul Smets, Jonathan Le Lous, Stephen Lubiarz, Pierrick Leclerc, "Downtime statistics of current cloud solutions," June 2012.

[24] Amazon, "AMAZON WEB SERVICE," December 2012. http://aws.amazon.com/ec2/.

[25] Microsoft, "MICROSOFT WINDOWS AZURE," December 2012. http://www.windowsazure.com/.

[26] Rackspace, "RACKSPACE CLOUD," December 2012. http://www.rackspace.com/cloud/.

[27] Google, "Google App Engine Datastore Java API," December 2012. https://appengine.google.com/.

[28] Salesforce, "Salesforce cloud," December 2012. http://www.salesforce.com/.

[29] J. Staten, S. Yates, J. Rymer, and L. Nelson, "Which cloud computing platform is right for you," *Understanding The Difference Between Public, Hosted, And Internal Clouds, Forrester*, vol. 13, 2009.

[30] P. Louridas, "Up in the air: Moving your applications to the cloud," *Software, IEEE*, vol. 27, no. 4, pp. 6–11, 2010.

[31] J. Tordsson, R. Montero, R. Moreno-Vozmediano, and I. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 358–367, 2012.

[32] R. Buyya, R. Ranjan, and R. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," *Algorithms and architectures for parallel processing*, pp. 13–31, 2010.

[33] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," *SIGCOMM Comput. Commun. Rev.*, vol. 25, pp. 231–242, Oct. 1995.

[34] O. Etzion and P. Niblett, *Event Processing in Action*. Manning Publications Co., 2010.

[35] A. Flissi, J. Dubus, N. Dolet, and P. Merle, "Deploying on the Grid with DeployWare," in *Eighth IEEE International Symposium on Cluster Computing and the Grid*, (France), pp. 177–184, 2008.

[36] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, and J.-B. Stefani, "A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures," *Software: Practice and Experience (SPE)*, vol. 42, pp. 559–583, May 2012.

[37] VMWARE, "Hyperic SIGAR API," December 2012. http://www.hyperic.com/products/sigar.

[38] D. Luckham and R. Schulte, "Event Processing Glossary - Version 1.1," *Processing*, vol. 1.1, no. July, pp. 1–19, 2008.

[39] F. Paraiso, G. Hermosillo, R. Rouvoy, P. Merle, and L. Seinturier, "A Middleware Platform to Federate Complex Event Processing," in *Sixteenth IEEE International EDOC Conference*, (Beijing, China), pp. 113–122, Springer, Sept. 2012.

[40] F. Dabek, N. Zeldovich, F. Kaashoek, D. Mazieres, and R. Morris, "Event-driven programming for robust software," in *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pp. 186–189, ACM, 2002.

[41] "Domain Names - Concepts and Facilities, and related other RFCs." http://www.ietf.org/rfc/rfc1034.txt.

[42] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems: concepts and design*. Addison-Wesley Longman, 2005.

[43] F. Paraiso, N. Haderer, P. Merle, R. Rouvoy, and L. Seinturier, "A Federated Multi-Cloud PaaS Infrastructure," in *5th IEEE International Conference on Cloud Computing*, (Hawaii, United State), pp. 392 – 399, June 2012.

[44] W. Tarreau, "Making applications scalable with load balancing," September 2006. http://1wt.eu/articles/2006_lb/.

[45] H. Qian, M. Rabinovich, and Z. Al-Qudah, "Bringing local dns servers close to their clients," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pp. 1–6, IEEE, 2011.

[46] A. Shaikh, R. Tewari, and M. Agrawal, "On the effectiveness of dns-based server selection," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1801–1810, IEEE, 2001.

[47] L. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 45–52, 2011.

[48] P. Marshall, K. Keahey, and T. Freeman, "Elastic site: Using clouds to elastically extend site resources," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 43–52, IEEE Computer Society, 2010.

[49] V. Cardellini, M. Colajanni, and P. Yu, "Dynamic load balancing on web-server systems," *Internet Computing, IEEE*, vol. 3, no. 3, pp. 28–39, 1999.

[50] M. Harchol-Balter and A. Downey, "Exploiting process lifetime distributions for dynamic load balancing," *ACM Transactions on Computer Systems (TOCS)*, vol. 15, no. 3, pp. 253–285, 1997.

[51] H. Lin and C. Raghavendra, "A dynamic load-balancing policy with a central job dispatcher (lbc)," *IEEE Transactions on, Software Engineering*, vol. 18, no. 2, pp. 148–158, 1992.

[52] W. Zhang, H. Qian, C. E. Wills, and M. Rabinovich, "Agile resource management in a virtualized data center," in *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pp. 129–140, ACM, 2010.

[53] H. Qian, E. Miller, W. Zhang, M. Rabinovich, and C. E. Wills, "Agility in virtualized utility computing," in *Virtualization Technology in Distributed Computing (VTDC), 2007 Second International Workshop on*, pp. 1–8, IEEE, 2007.

[54] N. Grozev and R. Buyya, "Inter-Cloud Architectures and Application Brokering: Taxonomy and Survey," *Software: Practice and Experience*, 2012.