# Engineering Multi-Cloud Service-Oriented Applications

Fawaz Paraiso, Philippe Merle, Lionel Seinturier

# Engineering Multi-Cloud Service-Oriented Applications

Fawaz Paraiso Philippe Merle Lionel Seinturier

**Abstract**

Cloud platforms are increasingly being used for hosting a broad diversity of services from traditional e-commerce applications to interactive web-based IDEs and crowdsourcing systems. However, the proliferation of offers by cloud providers raises several challenges. Developers will not only have to deploy applications for a specific cloud, but will also have to consider migrating services from one cloud to another, and to manage distributed applications spanning multiple clouds. In order to address these challenges, we present a federated multi-cloud PaaS infrastructure that is based on three foundations: i) an open service model used to design and implement both our multi-cloud PaaS and the SaaS applications running on top of it, ii) a configurable architecture of the federated PaaS, and iii) some infrastructure services for managing both our multi-cloud PaaS and the SaaS applications. We report on the deployment of this cloud-based infrastructure on top of thirteen existing IaaS/PaaS.

**Keywords**: Multi-cloud computing, Platform as a Service, Portability, Provisioning, Elasticity, High availability, Service Component Architecture.

## 1 Introduction

Cloud computing is a major trend in current research for building scalable distributed computing environments. In particular, Cloud computing emerged as a way for *"enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."* [35]. Several layers of cloud computing exist, including the infrastructure, platform, and application layers, which provide to end-users functionalities referred to as IaaS, PaaS, and SaaS, respectively [48]. Amazon Elastic Compute Cloud (Amazon EC2), Windows Azure, and Google App Engine are three of most well-know cloud platform providers, yet the offer has increased rapidly over the last months and tens of solutions are now available[1]. Besides, many key players in the IT business are also offering private cloud solutions for their data centers. In this context, multi-cloud computing [8] as the federation of multiple heterogeneous cloud computing environments is a promising paradigm to support very large scale world wide distributed applications.

[1] `http://upon2020.com/2011/04/the-ever-growing-list-of-paas-companies-and-paas-projects`

However, multi-cloud computing has to face several key challenges: *portability*, *provisioning*, *elasticity*, and *high availability*. Multi-cloud portability means write applications once and run them on any clouds. Most existing cloud providers are typically offered through proprietary APIs services and limited to a single infrastructure provider. In such situations, vendor lock-in is a primary concern for moving towards a cloud provider. Multi-cloud provisioning refers to the capability to deploy a distributed application spanning multiple cloud providers. Deploy a distributed application in a multi-cloud context is not easy task. Furthermore, existing deployment tools are limited to use of single clouds as differences in contextualization mechanisms [24, 25] hinder multi-cloud deployment. Multi-cloud elasticity means the capability to scale applications across multiple clouds. With the rapidly increasing size of computing systems provided from different clouds and the growing complexity of interconnected systems in multi-cloud environments, elasticity becomes more complex to implement. Applications are often grouped and deployed across multiple cloud environments, elasticity can be defined according to the requirement of each part of these applications. Currently, there is no convenient way to express specific application elasticity rules for each part of a distributed application as need. Multi-cloud high availability refers to the degree to which application is operable across multiple clouds. As [4] and [27] have already noted, the cloud computing model is not without concern. In fact, cloud provider services can become unavailable due to outages or denials of services. High availability needs to be analysed and set across multiple clouds in order to reduce the probability of outages that could affect services deployed in a single cloud system.

In this article we discuss the design and implementation of soCloud. soCloud is a multi-cloud PaaS to overcome the four key challenges presented in the previous section. soCloud is a service-oriented PaaS to create service-oriented applications. soCloud is a distributed PaaS, that provides a model for building distributed applications. This model is on an extension of the OASIS's SCA standard [38]. soCloud is a multi-cloud PaaS that provides services to ensure portability, provisioning, elasticity, and high availability across multiple clouds. Our ongoing approach to address portability and provisioning in a multi-cloud context is the use of a SCA standard. Our elasticity management approach is based on autonomic computing [26] with the overall aims of creating self-managed elastic multi-cloud applications. High availability is achieved in two ways. Firstly, soCloud provides a multi-cloud load balancer service that fronts traffic for applications deployed across multiple clouds and makes a decision about where to route the traffic when cloud nodes fail. Secondly, the soCloud architecture uses redundancy at all levels to ensure that no single component failure in a cloud provider impacts the overall system availability. We describe a way to annotate SCA artifacts with deployment information needed to optimal use of services in multiple cloud environments. These annotations also allow to express elasticity rules that ensure the appropriate adjustment decisions made in timely manner to meet service needs in the presence of cloud service failures. The soCloud architecture is composed of SCA components: service deployer, constraints validator, PaaS deployment, SaaS deployment, load balancer, node provisioning, monitoring, workload manager and controller components. soCloud is deployed and evaluated on ten existing cloud providers such as Windows Azure, DELL KACE, Amazon EC2, CloudBees, OpenShift, dotCloud, Jelastic, Heroku, Appfog, and an Eucalyptus private cloud.

The remainder of this article is organized as follows. Section 2 describes the main features of the soCloud platform. Section 3 compares soCloud with the state-of-the-art. Section 4 concludes this article and presents future work we intend to address.

# 2 Overview of soCloud principles

In this section we present an overview of soCloud. We first discuss background details of SCA and FraSCAti. Next, we describe how soCloud addresses the challenges of *portability*, *provisioning*, *elasticity*, and *high availability*. Finally, we describe how to design a soCloud SaaS distributed application.

## 2.1 SCA

soCloud is based on the SCA standard [38]. SCA is a set of OASIS's specifications for building distributed applications and systems using Service Oriented Architecture (SOA) principles [18]. SCA promotes a vision of Service-Oriented Computing (SOC) where services are independent of implementation languages (Java, Spring, BPEL, C++, COBOL, C, etc.), networked service access technologies (Web Services, JMS, etc.), interface definition languages (WSDL, Java, etc.) and non-functional properties. SCA thus provides a framework that can accommodate many different forms of SOC systems.

soCloud is built on top of FraSCAti [45]. FraSCAti is an open source framework for deploying and executing SCA-based applications. FraSCAti provides a component-based approach to support the heterogeneous composition of various interface definition languages (WSDL, Java), implementation technologies (Java, Spring, EJB, BPEL, OSGI, Jython, Jruby, Xquery, Groovy, Velocity, Fscript, Beanshell, UPNP.), and binding technologies (Web Services, JMS, RPC, REST, RMI.). Moreover, FraSCAti introduces reflective capabilities to the SCA programming model, and allows dynamic introspection and reconfiguration via a specialization of the Fractal component model [7]. These features open new perspectives for bringing agility to SOA and for the runtime management of SCA applications. FraSCAti is the execution environment of both the soCloud multi-cloud PaaS and soCloud SaaS applications deployed on the top of this multi-cloud PaaS.

## 2.2 Main features of soCloud

soCloud is a service-oriented component-based PaaS for managing portability, elasticity, provisioning, and high availability across multiple clouds. soCloud is a distributed PaaS, that provides a model for building distributed SaaS applications based on an extension of the OASIS's SCA standard. To address the challenge of *multi-cloud portability*, soCloud promotes SCA as the model to design and develop multi-cloud SaaS applications. To address the challenge of *multi-cloud provisioning*, soCloud offers a service to provision applications across multiple cloud providers. To address the challenge of *multi-cloud elasticity*, soCloud offers an autonomic service which provides a

global mechanism to manage elasticity across multiple clouds and also offers the possibility to define application specific elasticity rules. To address the challenge of *multi-cloud high availability* despite outages, soCloud provides high availability in two ways. Firstly, with the applications deployed with a soCloud platform, the high availability is ensured by using a load balancer service which distributes requests among instances of the application deployed on multiple cloud providers. Secondly, the soCloud architecture uses redundancy at all levels to ensure that no single component failure in a cloud provider impacts the overall system availability.

**Multi-cloud portability** The different layers of a cloud environment (IaaS, PaaS, SaaS) provide dedicated services. Although their granularity and complexity vary, we believe that a principled definition of these services is needed to promote the interoperability and federation between heterogeneous cloud environments. Hence, our multi-cloud infrastructure uses SCA both for the definition of the services provided by the PaaS layer and for the services of SaaS applications that run on top of this PaaS. soCloud uses an open service model called SCA to provide portability.

**Multi-cloud provisioning** soCloud provides a consistent methodology based on the SCA standard that describes how SaaS applications are modelled. soCloud provides services based on FraSCAti to deploy and allows runtime management of SaaS applications and hardware resources. This consistent methodology offers flexibility and choice for developers to provision and deliver SaaS applications and hardware resources across multiple clouds. soCloud provides a multi-cloud service enabling policy-based provisioning across multiple cloud providers.

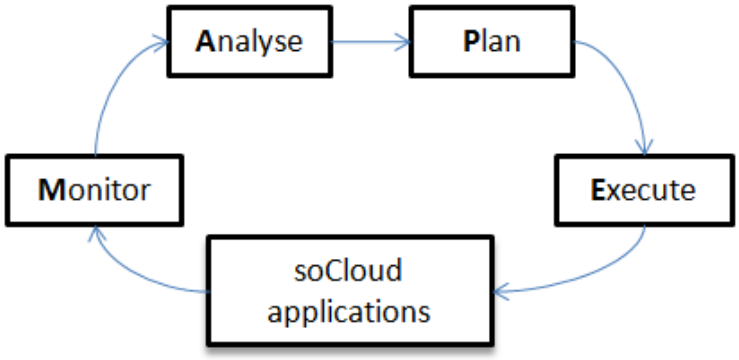


Figure 1: soCloud elasticity management with FeedBack Control Loop.

**Multi-cloud elasticity** The management of elasticity across multiple clouds is complex and appears to be approaching the limits of what is done with managing elasticity in a single cloud. In fact, systems become more interconnected, diverse, latency and outages can occur at any time. The soCloud architecture focuses on the interactions among components, leaving such issues to be dealt with at runtime. Particularly, any

automated set of actions aimed to modify the structure, behaviour, or performance of SaaS applications deployed with the soCloud platform while it continues operating. An appropriate option remaining is autonomic computing [12]. As described in Fig. 1, soCloud elasticity is managed with the MAPE-K (**M**onitor, **A**nalyse, **P**lan, **E**xecute, -Knowledge) reference model [12] for autonomic control loop.

**Multi-cloud high availability** soCloud provides high availability in two ways. Firstly, by using a multi-cloud load balancer service to switch from one application to another when failure occurs. Secondly, the soCloud architecture uses replication strategy. Specially, soCloud uses redundancy at all levels to ensure that no single component failure in a cloud provider impacts the overall system availability.

## 2.3 soCloud SaaS applications

**Application specification** soCloud applications are built by using the SCA model. As illustrated in Fig. 2, the basic SCA building blocks are software components [46], which provide services, require references and expose properties. The references and services are connected by wires. SCA specifies a hierarchical component model, which means that components can be implemented either by primitive language entities or by subcomponents. In the latter case the components are called composites. Any provided services or required references contained within a composite can be exposed by the composite itself by means of promotion links. To support service-oriented interactions via different communication protocols, SCA provides the notion of binding. For SCA references, a binding describes the access mechanism used to invoke a remote service. In the case of services, a binding describes the access mechanism that clients use to invoke the service. We describe how SCA can be used to package SaaS applications. The first requirement is that the package must describe and contain all artifacts needed for the application. The second requirement is that provisioning constraints and elasticity rules must be described in the package. The SCA assembly model specification describes how SCA and non-SCA artifacts (such as code files) are packaged. The central unit of deployment in SCA is a contribution. A contribution is a package that contains implementations, interfaces and other artifacts necessary to run components. The SCA packaging format is based on ZIP files, however other packaging formats are explicitly allowed. As shown in Fig. 2, a three-tier application is packaged in as a ZIP file (SCA contribution) and its architecture is described. This application is composed of three components. One component represents the frontend-tier of the application. The second one represents the computing-tier of the application. The last one represents the storage-tier of the application. Each component of this three-tier application is implemented as an SCA contribution (ZIP file package).

**Annotations** Some cloud-based applications require more detailed description of their deployment (c.f. Fig. 2). The deployment and monitoring of soCloud applications are bound by a description of the overall software system architecture and the requirements of the underlying components, that we refer to as the *application manifest*. In particular, the *application manifest* should be platform-independent. Basically, the

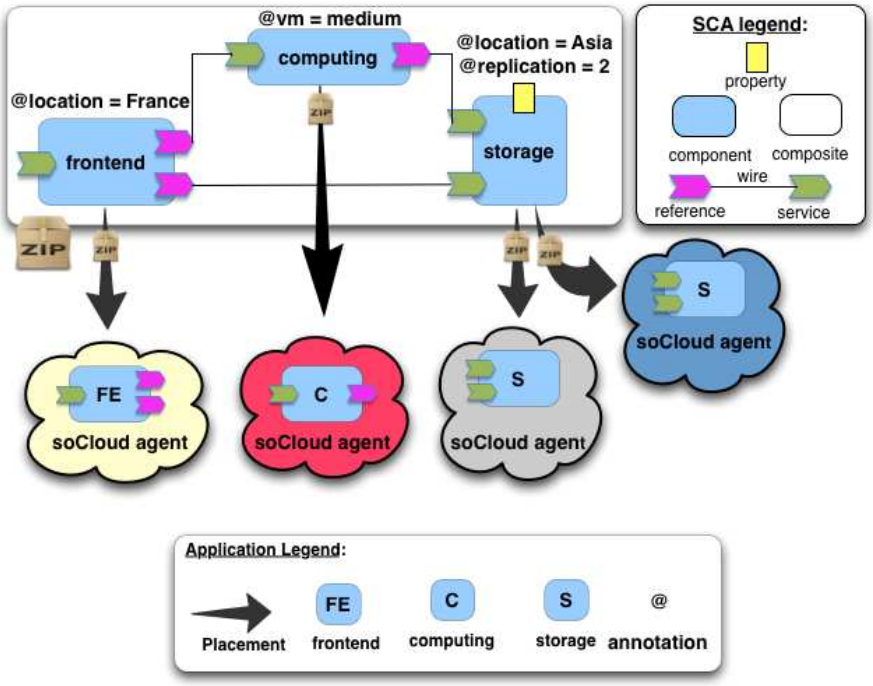


Figure 2: **An annotated soCloud application.**

*application manifest* consists of describing what components the service is composed with functional and non-functional requirements for deployment. In fact, the application can be composed of multiple components (c.f. Fig. 2). The application manifest defines elasticity rule for the service component (*e.g.* increase/decrease instance of component). Commonly, scale up or down, is translated to a **condition-action** statement that reasons on performance indicators of the service deployed. In order to fulfill the requirements for the soCloud application descriptor, we propose to annotate the SCA components with the four following annotations:

1. **placement constraint** (*@location*) allows to map components of a soCloud application to available physical hosts within a geographical datacenter in multi-cloud environments.

2. **computing constraint** (*@vm*) provides necessary computing resources defined for components of a soCloud application in the multi-cloud environments.

3. **replication** (*@replication*) specifies the number of instances of the component that must be deployed in multi-cloud environments.

4. **elasticity rule** (*@elasticity*) defines a specific elasticity rule that should be applied to the component deployed on multi-cloud environments.

For example, let us consider the three-tier web application described in Fig. 2, which consists of a frontend, computing, and storage components. The annotation (*@location=France*) of the frontend component indicates to deploy this component on a cloud provider located in France. Next, the annotation (*@vm=medium*) on the computing component specifies the computing resources required by this component and can be deployed on any cloud provider. Finally, the annotations (*@location=Asia and @replication = 2*) on the storage component indicate to deploy this component on two different cloud providers located in Asia. The soCloud automates the deployment of this three-tier application in a multiple cloud environment by respecting a given annotations.

## 2.4 Integration with existing IaaS/PaaS

We report on the existing cloud environments on which the soCloud platform has been deployed. The soCloud platform extends an experiment that was presented in a previous work [40]. The soCloud platform is actually deployed on ten target cloud environments that are publicly accessible on the Internet[2]. The deployment is done with IaaS/PaaS providers as illustrated in Fig. 3. With IaaS, resources are provisioned from Windows Azure[3], DELL KACE[4], Amazon EC2[5], and our Eucalyptus private cloud, we installed a PaaS stack composed of a Linux distribution, a Java Virtual Machine, a

[2]available at http://multicloudpaas.soceda.cloudbees.net/
[3]https://www.windowsazure.com
[4]https://www.kace.com/
[5]http://aws.amazon.com/ec2/

web container and FraSCAti. soCloud is also deployed on PaaS such as: CloudBees[6], OpenShift[7], dotCloud[8], Jelastic[9], Heroku[10], and Appfog[11] as a WAR file.

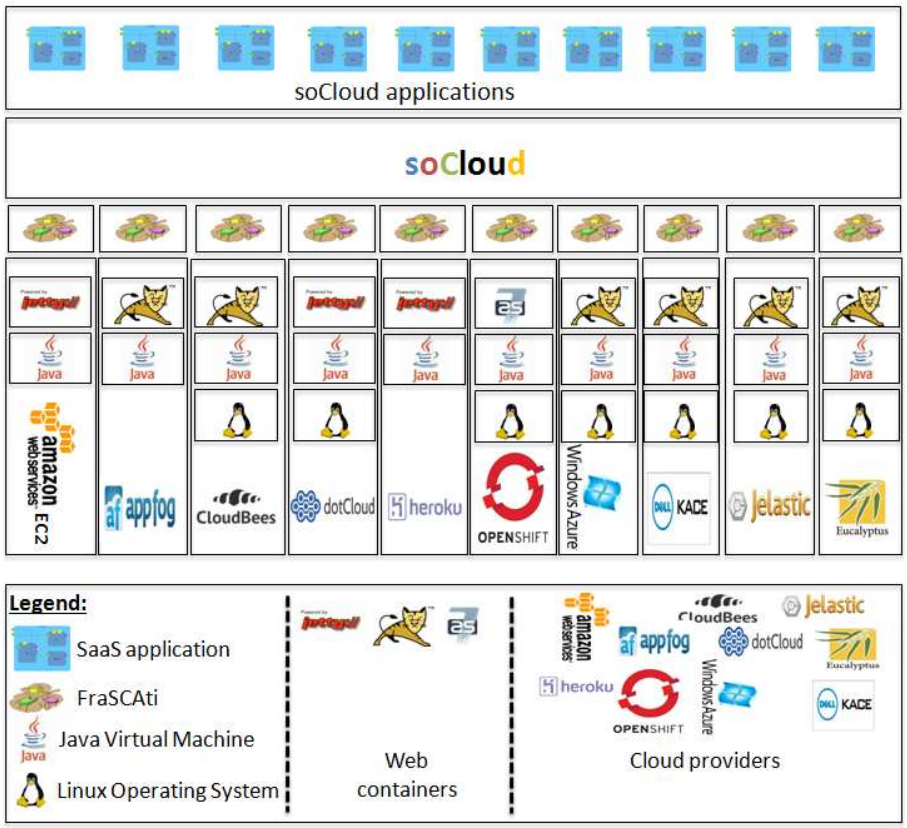


Figure 3: soCloud deployment with ten cloud providers.

# 3 Related work

Related to the Inter-Cloud Architectural taxonomy presented in [20], soCloud can be classified into the Multi-Cloud service category. This section presents some of the related work to multi-cloud computing challenges discussed in Section 1: *portability*, *provisioning*, *elasticity*, and *high availability* across multiple clouds.

[6] http://www.cloudbees.com/
[7] https://openshift.redhat.com
[8] https://www.dotcloud.com/
[9] http://jelastic.com/
[10] http://www.heroku.com/
[11] http://www.appfog.com/

**Multi-cloud portability** Portability approaches can be classified into three categories [39]: *functional portability*, *data portability* and *service enhancement*. The authors [41] with their mOSAIC solution deal with *service enhancement* portability at IaaS and PaaS levels. moSAIC provides a component-based programming model with asynchronous communication. However, mOSAIC APIs are not standardized and are complex to put at work in practice. Our soCloud solution deals with *service enhancement* portability among of an interface which runs on PaaS. soCloud supports both synchronous and asynchronous communications offered by the SCA standard. Moreover, SCA defines an easy way to use portable API. The Cloud4SOA [14] project is worked for the portability between PaaS using a semantic approach. soCloud intends to provide portability using API based on the SCA standard.

**Multi-cloud provisioning** A great deal of research on dynamic resource allocation for physical and virtual machines and clusters of virtual machines [3] exists. The work of dynamic provisioning of resource, in the cloud computing may be classified into two categories. Authors in [36] have addressed the problem of provisioning resources at the granularity of VMs. Other authors in [11] have considered the provisioning of resources at a finer granularity of resources. In our work, we consider provisioning at both VM and finer granularity of resources.

Some libraries such as DeltaCloud [15], jClouds [23], daseing [1], LibCloud [29], CompatibleOne [28], exist for provisioning resources across multiple cloud providers. Most of them act as wrappers of other technologies (e.g. libCloud, DeltaCloud, or jClouds). They propose an uniform API to different cloud services. CompatibleOne implements the Open Cloud Computing Interface (OCCI) standard [16]. Each library is implemented in its own programming language (e.g. LibCloud uses python; jClouds uses Java; DeltaCloud uses Ruby). As mentioned already the Node provisioning of soCloud provides a high level abstraction layer that hinds the complexity of provisioning across multiple clouds. Specifically soCloud offers the possibility to interact with jClouds, LibCloud, CompatibleOne which are implemented in different programming languages.

The authors in [19] have addressed the problem of deploying a cluster of virtual machines with given resource configurations across a set of physical machines. While [13] defines a Java API permitting developers to monitor and manage a cluster of Java VMs and to define resource allocation policies for such clusters. Unlike [19, 13], soCloud uses both an application-centric and virtual machine approaches. Using knowledge on application workload and performance goals combined with server usage, soCloud utilizes a more versatile set of automation mechanisms.

**Multi-cloud elasticity** 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 [8], 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 both a controller and a load balancer. However, their solution does not address the

management of elasticity through multiple cloud providers. The authors in [34] propose a resource manager to manage application elasticity. However their approach is specific for a single cloud provider.

In some cloud provider environments such as Amazon Elastic Load Balancing [21], the quality of service metrics (e.g., request count and request latency) is watched by Amazon Cloudwatch. The Amazon scalability mechanisms depend on initiating a VM instance as a load balancer routing the traffic into many similar VMs instances. This approach have two limitations. First, it is limited to specific applications like web servers and not applicable to the other applications like databases. Second, it is not possible to support a complex application that needs specific elastic rules.

Nevertheless, authors in [10] propose an elastic service definition language. Their language is based on Open Virtual Format (OVF) [33], a DMTF standard for packaging and deployment of virtual services. soCloud uses Event Processing Language (EPL) which is based on the SQL standard. As comparison to OVF, the EPL converges event stream processing (filtering, joins, aggregation) and complex event processing (causality) into a single language. Additionally, based on SCA properties, soCloud has the possibility to define constraints on applications deployed.

The authors in [42] propose a solution to deploy a complex application composed of several services using a manifest. They design a proprietary manifest language to specify the entire structure of the application deployed. In soCloud, the manifest used to define applications structure is based on the SCA standard language. Thus, we eliminate the use of a new proprietary language.

The authors in [17] focus on the description of the management problems in multi-cloud architectures. soCloud has addressed these challenges. [44, 30] proposed a framework that allows users to deal with applications by defining elasticity rules and based on automation. However any scalability management system is bound to the underline cloud API (the problem of ”discrete actuators” as named by [30]). One essential task for any application-level elasticity controller is, thus, mapping user scaling policies from the appropriate level of abstraction for the user to the actual mechanisms provided by IaaS clouds. The authors in [31] propose a solution that focuses on the problem of building external controllers for dynamic applications hosted on the cloud by using feedback control. With this solution in multiple cloud environments, they must define as many controller as cloud providers. soCloud provides an uniform way to manage elasticity across multiple cloud providers based on feedback control. [5] developed an agent-based solution to automate system tuning, their agents do both controller design and feedback control, however, slow converges of the system (i.e., 10 minutes for MaxClients), makes it unsuitable for sudden workload changes.

**Multi-cloud high availability** Cloud providers as 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 load balancing across multiple cloud providers. Different approaches of dynamic load balancing have been proposed in the literature [9, 22, 32], however they do not provide a mechanism to scale the load balancers themselves. The authors in [49], [43] have explored the agility way to quickly reassign resources. However, their approach does not take into account a multi

cloud environment. Most existing membership protocols [6, 37, 2] employ a consensus algorithm to achieve agreement on the membership. Achieving consensus in an asynchronous distributed system is impossible without the use of timeouts to bound the time within which an action must take place. Even with the use of timeouts, achieving consensus can be relatively costly in the number of messages transmitted, and in the delays incurred. To avoid such costs, soCloud uses a novel Leader Determined Membership Protocol that does not involve the use of a consensus algorithm.

# 4 Conclusion

In this article, we have proposed soCloud a service-oriented component-based PaaS for managing portability, provisioning, elasticity, and high availability across multiple clouds. soCloud is a distributed PaaS that provides a model for building any multi-cloud SaaS applications. This model is based on an extension of the OASIS's SCA standard. We surveyed each of the concepts related to express specific elasticity rules, ensure high availability across multiple clouds and pointed out problematics. To address these problems, this article proposes an architecture, and describes the interactions between each component of this architecture. We explain how the components in a SaaS application descriptor can be annotated with elasticity rules, placement constraints, computation constraints. Based on these annotations, deployable contributions can be loaded and deployed in a suitable manner. The article described the approach used by the soCloud platform to ensure high availability. In particular soCloud takes a wait-free approach to the problem of coordinating components in different clouds and uses load balancer to switch from one application instance to another in case of failures. In comparison of soCloud availability with public cloud availability, we demonstrate that soCloud ensures high availability in minutes. We analyse the flash crowd phenomenon on a use case, and demonstrate how the soCloud platform increases the elasticity of the application. This approach is proactive in the case that the content replications is performed when detecting a traffic surge and anticipating a flash crowd. In future work 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.

# References

[1] The Dasein Cloud API. http://dasein-cloud.sourceforge.net.

[2] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Membership Algorithms for Multicast Communication Groups. In *Distributed Algorithms*, pages 292–312. Springer, 1992.

[3] P. Anedda, S. Leo, S. Manca, M. Gaggero, and G. Zanetti. Suspending, Migrating and Resuming HPC virtual clusters. *Future Generation Computer Systems*, 26(8):1063–1072, 2010.

[4] Armbrust, M. and Fox, A. and Griffith, R. and Joseph, A.D. and Katz, R. and Konwinski, A. and Lee, G. and Patterson, D. and Rabkin, A. and Stoica, I. and others. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[6] K. P. Birman, R. Van Renesse, et al. *Reliable distributed computing with the Isis toolkit*, volume 85. IEEE Computer Society Press Los Alamitos, 1994.

[7] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The Fractal component model and its support in Java: Experiences with Auto-adaptive and Reconfigurable Systems. *Softw. Pract. Exper.*, 36(11-12):1257–1284, Sept. 2006.

[8] 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*, pages 13–31, 2010.

[9] V. Cardellini, M. Colajanni, and P. Yu. Dynamic load balancing on web-server systems. *Internet Computing, IEEE*, 3(3):28–39, 1999.

[10] C. Chapman, W. Emmerich, F. G. Marquez, S. Clayman, and A. Galis. Elastic service management in computational clouds. *CloudMan 2010*.

[11] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 103–116. ACM, 2001.

[12] A. Computing et al. An architectural blueprint for autonomic computing. *IBM White Paper*, 2006.

[13] G. Czajkowski, M. Wegiel, L. Daynes, K. Palacz, M. Jordan, G. Skinner, and C. Bryce. Resource management for clusters of virtual machines. In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 1, pages 382–389. IEEE, 2005.

[14] F. Dandria, S. Bocconi, J. G. Cruz, J. Ahtes, and D. Zeginis. Cloud4SOA: Multi-Cloud Application Management Across PaaS Offerings. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on*, pages 407–414. IEEE, 2012.

[15] A. DeltaCloud. DeltaCloud API that abstract difference between clouds. http://deltacloud.apache.org.

[16] A. Edmonds, T. Metsch, A. Papaspyrou, and A. Richardson. Toward an open cloud standard. *Internet Computing, IEEE*, 16(4):15–25, 2012.

[17] E. Elmroth, J. Tordsson, F. Hernández, A. Ali-Eldin, P. Svärd, M. Sedaghat, and W. Li. Self-management challenges for multi-cloud architectures. In *Towards a Service-Based Internet*, pages 38–49. Springer, 2011.

[18] T. Erl. *Soa: principles of service design*, volume 1. Prentice Hall Upper Saddle River, 2008.

[19] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang. Virtual clusters for grid communities. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 513–520. IEEE, 2006.

[20] N. Grozev and R. Buyya. Inter-Cloud Architectures and Application Brokering: Taxonomy and Survey. *Software: Practice and Experience*, 2012. http://dx.doi.org/10.1002/spe.2168.

[21] D. Guide. Amazon elastic load balancing. 2010.

[22] M. Harchol-Balter and A. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems (TOCS)*, 15(3):253–285, 1997.

[23] A. jclouds. The jclouds API that uses use cloud-specific features. http://jclouds.incubator.apache.org.

[24] K. Keahey and T. Freeman. Contextualization: Providing one-click virtual clusters. In *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, pages 301–308. IEEE, 2008.

[25] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes. Sky computing. *Internet Computing, IEEE*, 13(5):43–51, 2009.

[26] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[27] N. Leavitt. Is cloud computing really ready for prime time. *Growth*, 27(5), 2009.

[28] L. Lefevre, O. Mornard, J.-P. Gelas, and M. Morel. Monitoring energy consumption in clouds: the compatibleone experience. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 794–795. IEEE, 2011.

[29] A. Libcloud. Apache Libcloud a unified interface to the cloud. http://libcloud.apache.org.

[30] H. C. Lim, S. Babu, and J. S. Chase. Automated control for elastic storage. In *Proceedings of the 7th international conference on Autonomic computing*, pages 1–10. ACM, 2010.

[31] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh. Automated control in cloud computing: challenges and opportunities. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, pages 13–18. ACM, 2009.

[32] H. Lin and C. Raghavendra. A dynamic load-balancing policy with a central job dispatcher (LBC). *IEEE Transactions on, Software Engineering*, 18(2):148–158, 1992.

[33] D. management task force. Open Virtualization Format Specification DSP0243 v1.0.0, February 2009.

[34] 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*, pages 43–52. IEEE Computer Society, 2010.

[35] P. Mell and T. Grance. The NIST Definition of Cloud Computing. Technical report, National Institute of Standards and Technology, 2009. `http://www.nist.gov/itl/cloud/upload/cloud-def-v15.pdf`.

[36] R. Mietzner and F. Leymann. Towards provisioning the cloud: On the usage of multi-granularity flows and services to realize a unified provisioning infrastructure for saas applications. In *Services-Part I, 2008. IEEE Congress on*, pages 3–10. IEEE, 2008.

[37] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4):54–63, 1996.

[38] OASIS. Reference Model for Service Oriented Architecture 1.0, August 2006. http://oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf.

[39] K. Oberle and M. Fisher. ETSI CLOUD–initial standardization requirements for cloud services. In *Economics of Grids, Clouds, Systems, and Services*, pages 105–115. Springer, 2010.

[40] 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*, pages 392 – 399, Hawaii, United State, June 2012. http://hal.inria.fr/hal-00694700.

[41] D. Petcu, G. Macariu, S. Panica, and C. Crăciun. Portable Cloud applications From theory to practice. *Future Generation Computer Systems*, 2012.

[42] G. Pierre and C. Stratan. ConPaaS: a platform for hosting elastic cloud applications. *Internet Computing, IEEE*, 16(5):88–92, 2012.

[43] 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*, pages 1–8. IEEE, 2007.

[44] L. Rodero-Merino, L. M. Vaquero, V. Gil, F. Galán, J. Fontán, R. S. Montero, and I. M. Llorente. From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, 26(8):1226–1240, 2010.

[45] 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)*, 42(5):559–583, May 2012.

[46] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. ACM Press and Addison-Wesley, New York, NY, 1998.

[47] L. Vaquero, L. Rodero-Merino, and R. Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011.

[48] Q. Zhang, L. Cheng, and R. Boutaba. Cloud Computing: State-of-the-art and Research Challenges. *Journal of Internet Services and Applications*, 1(1):7–18, May 2010.

[49] 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*, pages 129–140. ACM, 2010.