DOI 10.12694/scpe.v16i2.1086
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 2, pp. 139–151. http://www.scpe.org © 2015 SCPE

A SCALABLE AND DISTRIBUTED CLOUD BROKERING SERVICE

ALBA AMATO[*] AND SALVATORE VENTICINQUE[†]

Abstract. Cloud computing can be considered the key of the development of ICT systems business. It allows to work with the basic tools, but in a dynamic, mobile and technologically advanced way. Taking a good choice of Cloud provider, is the key to safely embracing the Cloud and the benefits that it provides. To do that is necessary to identify the Cloud service needs based on the application requirements. Some applications will be more critical than others. Moreover it is necessary to select the provider which best fits those requirements. In fact, as happens to a more traditional delivery model in which organizations use different product packages from different providers because no single supplier can meet all of their needs, there are different Cloud providers that offer different features and perks, and many of them are not directly comparable. In this paper a scalable distributed multi-users version of a Broker As A Service solution is proposed, mainly suitable for exploiting the capability of a distributed environment and for addressing the related issues. The idea is to decompose global broker into a set of distributed brokers that cooperate and dynamically scale, together with the computing infrastructure, to support unforeseeable workloads produced by the interactions with large groups of users. The brokering problem is divided into simpler tasks, which are distributed among independent agents, whose population dynamically scales together with the computing infrastructure, to support unforeseeable workloads produced by the interactions with large groups of users. This solution is also suitable in case of Multi-Cloud brokering. Several experimental results have been performed to verify the effectiveness of the proposed system.

Key words: Intelligent Agents; Distributed Brokering; Cloud Computing

1. Introduction.

In the IT world as we know it today computers become exponentially more powerful and cost per unit of resources is rapidly decreasing, so that IT can be considered a commodity. On the other hand IT becomes more and more pervasive in organizations and the increasing complexity in managing the entire infrastructure, made up of different software architectures within which the information is distributed, has led companies to spend a growing amount in IT. The diffusion of the phenomenon of Cloud Computing has generated a strong interest in the companies, in fact Cloud computing can be considered the key to the development of ICT systems business. It is a fundamental revolution in the way Information Technology (IT) services are created, developed, deployed, updated, maintained and paid for.

In particular, among the benefits of adopting Cloud models, elasticity plays a relevant role. In fact it allows for providing the required resources to the deployed applications according to the current workload and paying just for their usage.

Especially in a service oriented context the exploitation of such a facility is desired. A service must support access to multiple users simultaneously ensuring always the required level of availability and performance. The application workload can vary during the day, with regular or unforeseeable bursts on special periods. Moreover the service can be invoked also by applications and robots and not only by human users.

However scalability of a service with its changing workload is not enough to guarantee the exploitation of Cloud elasticity. Providing a service with self-adaptivity to scale dynamically with a changing computing infrastructure is not trivial for any new developed applications or legacy ones when they are ported in the Cloud.

This paper discusses this issues presenting a scalable distributed Broker As A Service solution that has been designed for exploiting the elasticity of a Cloud computing infrastructure. Such a broker service supports the users to take a good choice among the many alternatives of Cloud services offered by the increasing number of different providers [19].

The idea is to decompose a global broker into a set distributed brokers, which cooperate and dynamically scale, together with the computing infrastructure, to support unforeseeable workloads produced by the interactions with large groups of users. This solution is suitable also in case of Multi-Cloud brokering, where the optimal choice is a collection of services provided by heterogeneous providers. The remainder of this paper is structured as follows. Some related approaches are introduced in 2. Section 3 motivates, introduces, and

[*]Department of Industrial and Information Engineering, Second University of Naples, Aversa, Italy (alba.amato@unina2.it).
[†]Department of Industrial and Information Engineering, Second University of Naples, Aversa, Italy (salvatore.venticinque@unina2.it).

Table 2.1
Brokering Solutions

Reference	**Properties**	**Limitations**
RightScale	select, migrate and monitor different Clouds	missing negotiation and optimization features
CloudSwitch	make simpler the process of migrating an application or workload to the Cloud	missing proper brokerage service
JamCracker	aggregates and distributes on-demand services through a global ecosystem of Service Providers	missing a proper brokerage service
[9]	based on a Cloud service register catalog where it performs a search of the desired proposal	missing a proper brokerage service
[12]	focused on the scheduler	only cost optimization
OPTIMIS	under development	only cost optimization
[14]	optimize placement of VMs across multiple cloud providers	dynamic cloud scheduling not supported

analyzes the problem, whose solution's requirements, approach and implementation are then investigated in Sections 4, 5 and 6. Sections 7 and 8 concludes with an overview and interpretation of the results.

2. Related Work.

2.1. Background. In recent years, Gartner has analyzed in great detail the role of the Cloud Service Brokers [6]. In the first place, Gartner defines Cloud Services Brokers (CSB) it as an IT role and a business model where a company or other entity adds value to one or more Cloud services (public or private) on behalf of one or more consumers of this service through three roles primary: aggregation, integration and customization. The activity of the CSB has become necessary in recent years for the diffusion of solutions of hybrid Cloud involving multiple Cloud services. Furthermore, the role of the CSB has a positive impact both for business and IT operations. In fact organizations are increasingly adopting Cloud solutions, so the supply of services is increasing more and more. Companies must therefore address issues of greater complexity in the adoption and must be able to handle a large number of providers.

2.2. Brokering Solutions. For those reasons several companies offers brokering solutions which are summarized in Table 2.1. RightScale [17] Self-Service is a solution that enables IT users to deliver self-service access to a curated catalog of commonly used components, stacks, and applications and an easy-to-use interface for developers and other Cloud users to request, provision, and manage these services across public Cloud, private Cloud, and virtualised environments. RightScale Self-Service includes a Cloud service catalog that enables users to design a set of Cloud applications that can be configured with the application versions, security patches, and software configurations that match company standards. It also includes built-in cost management features that enable users to set usage quotas in order to stay within set project budgets. Nevertheless it is focused on select, migrate and monitor different Clouds from a single management environment and does not provide negotiation and optimization features.

CloudSwitch [18] is a Cloud broker software vendor that helps enterprises to move Cloud data more easily. It advertises itself as a the enterprise gateway to the Cloud and provides the ability to integrate public application program interfaces (APIs) with Cloud providers to make the process of migrating an application or workload to the Cloud as simple as a drag-and-drop action in a web browser. Nevertheless it does not really provide a proper brokerage service but it gives customers an interface to port applications to several different providers.

The JamCracker system, available in [7], provides a platform where it aggregates and distributes on-demand services through a global ecosystem of Service Providers, Resellers, System Integrators, and ISVs but it does not provide a proper brokerage service whereby an entity looks at the actual QoS requirements of the service under question, the various IPs that could potentially meet them, rank them against parameters like cost, trust, eco-efficiency, risk etc. and provide functionality to on board these applications in to the various IPs finally selected as stated in [15]. The brokering problem is also investigated in several scientific communities and European Project.

The Cloud-service broker, presented in [9] as an emerging technology, intermediates heterogeneous multiple Cloud services for both providers and consumers. The Cloud-service-broker portal enables the Cloud service providers to specify that their services are available. In addition, the Cloud-service-consumers may find the most suitable services by negotiating the agreements on the services. It is based on a Cloud service register catalog where it performs a search of the desired proposal. Nevertheless it does not assist users with decision making to evaluate and select a Cloud vendor or solution based on specific requirements.

A Cloud broker architecture for deploying virtualised servers across available Clouds is presented in [12]. The paper is focused specially on the scheduler, that is one of the three components. Other components are the VM manager and the Cloud manager. In particular the paper addresses the placement challenge by using different algorithms developed for optimizing the cost of the required infrastructure. Specifically, it utilizes a prediction model which takes into account the historical prices of available Cloud providers in order to calculate, for the next hour deployment, the best Cloud provider to move the VMs to. Nevertheless it is focused on cost optimization and does not consider the other different criteria.

In the scope of the EU FP7 project OPTIMIS, an architectural design of a framework capable of powering the brokerage of Cloud services is proposed and is under development. It is described in [13] that introduces the problem and the architectural design, but it does not provide an implementation or algorithms to achieve the brokering. The main features are to ensure data confidentiality and integrity to service customers, to match the requirements of Cloud consumer with the service provided by the provider, to negotiate with service consumers over Service Level Agreements (SLAs), to maintain performance check on these SLA's and take actions against SLA violation. Moreover it aims to effectively deploy services provided by the Cloud provider to the customer, to manage the API so that provider does not learn anything about the identity of the service consumer, and to manage security problems.

The paper [14] proposes an architecture for cloud brokering and multi-cloud VM management, describes algorithms for optimized placement of applications in multi-cloud environments. The proposed model incorporates price and performance, as well as constraints in terms of hardware configuration, load balancing, etc. An evaluation against commercial clouds demonstrates that compared to single-cloud deployment, a multi-cloud placement algorithms improve performance, lower costs, or provide a combination thereof. It becomes evident that capabilities of optimization and decision making to evaluate and select a Cloud vendor or solution based on specific requirements, which is the focus of the proposed approach have little or no support in Cloud service brokerage available today.

In preliminary work [3, 4], the authors present the architecture of the Broker Agent and its implementation within the EU FP7 mOSAIC project for provisioning of brokering service at Cloud platform level. The proposed Cloud broker provides several facilities in order to optimize different variables like overall infrastructure cost, service performance, or others, depending on the strategies selected by the user. Moreover it gives the possibilities to set different optimization criteria and several restrictions to support user in the choice of the provider that is more adapt to their applications. According to [16], the agility and flexibility are the main elements that encourage people to adopt the Cloud, so the Cloud broker will assume a key role in the future deployment of solutions as a service. But this kind of multi-criteria optimization problems are hard to compute, so it is necessary a scalable, multi-user, distributed solution. In this paper is described a new scalable and distributed approach that overcomes the performance limitations of paradigm as a service. Some experimental results are also presented.

3. Problem Statement.

Deployment of Cloud services makes it necessary to select the provider that best fits the application requirements. Usually it is not a trivial task because there are different Cloud providers that offer different features and perks, and many of them are not directly comparable. Besides it could be necessary to consider composition of services from different providers, as it happens when organizations use different product packages because no single supplier can meet all of their needs.

The broker realized in the mOSAIC FP7 project [1] is designed and developed to support decision making by a Cloud deployer. It is centralized on a single brokering process, which takes care of evaluating all available

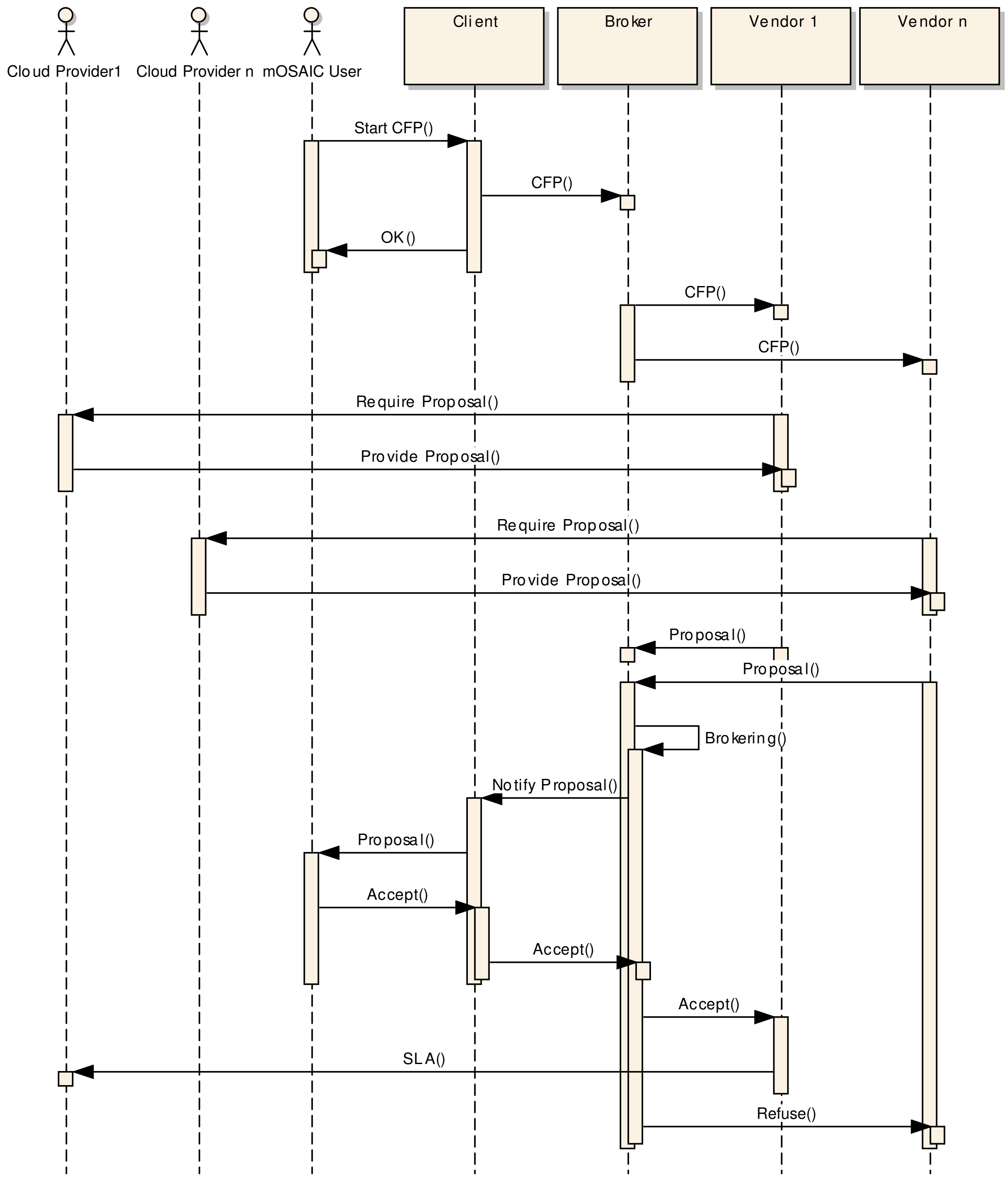


Fig. 3.1. *mOSAIC Broker*

alternatives limited to a single user's preferences. Moreover in this project the brokering of service composition has not been addressed.

The mOSAIC Broker is shown in Figure 3.1. For each received call for proposal (CFP), cloud agency creates a broker that searches for vendors that can offer resources with the required QoS (Quality of Services). As shown in Figure 3.1, the broker collects a number of proposals described in a vendor agnostic way and chooses the best one(s) according to the brokering rules.

To provide such an application *as a service*, with the aim to overcome the single-user delivery and the atomic service brokering, new requirements and challenges will be introduced. The former limitation deals with the need of handling increasing and dynamically changing workloads. The latter constraint affects the response time and the complexity of a single brokering problem.

With this premise a centralized approach is not feasible anymore. But Cloud allows to scale the computing resources according to the measured workload, so improving also their utilization. The idea is to exploit such elastic computing model for building a distributed Cloud broker over a Cloud infrastructure. But the application must be able to reconfigure itself autonomically to keep the QoS level above the desired threshold. Due to lack of control both of the network and of the compute utility, not only performance issues should be addressed, but also reliability and availability when the service execute over such kind of distributed platform.

At a glance it is possible to identify three different scenarios for our brokering solution with improving capabilities.

1. A *mOSAIC Broker* is a *single-user application* that gets *one CFP* and returns *many atomic proposals.*
2. A *brokering As A Service* is a *multi-user application* that gets *many CFP* and returns *many atomic proposals.*
3. A *multi-Cloud brokering As A Service* is a *multi-user application* that gets *many CFP* and returns *many compositions of proposals.*

Moving from the first scenario to the second ones implies the handling of parallel requests from multiple users. The service must be capable to distribute the workload among many worker nodes. A trivial solution would be the execution of brokering processes shown in Figure 3.1 across the nodes according to a task parallel approach. This solution would limit a fine grain balancing of the workload among working nodes because different brokering problems could have different completion time. In addition it prevents the utilization of the distributed resources to solve a single brokering problem, that can become computationally hard when we move from the second to the third scenario.

4. Requirements for an elastic brokering and design guidelines.

Addressing the design of a Cloud service two different requirements have to be taken into account: the interaction with the service requestor at front-end and the processing of service requests at back-end. Service levels at front-end are measured by the availability and responsiveness. The first one regards the capability to accept service requests from requestors. It could be independent from the processing capability at back-end, above all if the results must not be provided in real time. The second one is about as much interactive should be the service. It affects in advance the quality perceived by the requestor beyond the time within the response is expected to be available. In the presented case the idea is to accept the brokering requests and to provide information to the requestor about the status of the job, or about intermediate results with a certain level of responsiveness. Design solutions for exploitation of elasticity at front-end are quite common. Availability is the first service level that every Cloud provider grants into the SLA. Moreover most of Cloud services are conceived to provide web interface and interactive interfaces.

The design guidelines suggest to implement a synchronous front-end that is deployed on multiple virtual machines. The Cloud infrastructure will monitor the resource utilization of each virtual machine and will start/stop new instances. A load balancer will distribute the requests between the running replicas. The development of a stateless pool of web services is a mandatory choice to allows for a simple automatic scaling of the front-end. Besides the status of the web session is shared by Cloud data services. Distributed cache will provide reliability, allow transparency between service instances and load balancer, improving responsiveness.

Service levels at back-end are throughput and the completion time of each job. Here it is important that the utilization of the computing resources is maximized. The effective scheduling for load balancing of jobs and their fair allocation is the most relevant issue for scalability. A stateless solution is again desired for allowing any new idle worker to get any waiting job and to execute it using available resources. The usage of NOSQL solutions is quite common to partition the data providing availability, and facilitating the scheduling. Key value stores, queues or big tables [10] are example of services that store the application data. Queue services are used for synchronization and to route the message among workers. Asynchronous solutions improve utilization and facilitate the scheduling. The routing algorithm of the queue can be used to add intelligence to the scheduling by distributing the workload.

In the presented case it is important to scale the system to collect as many brokering requests (call for proposals) as they come. The only limitation will be the number of virtual machine instances to use and pay for, both to run the service front-end and the service back-end. Although many load balancing schemes have been presented in Cloud computing, there is no scheme providing the elasticity and adaptive adjustment in Cloud computing. This allows us to address fine-grained load balancing and scalability at design time and at component level. It is possible to model the brokering as a data parallel problem. The pool of stateless worker will be composed of vendors, brokers and SLA managers. Vendors will run on behalf of providers. They receive CFPs and are in charge to generate correspondent proposals. Brokers will evaluate the score of a proposal comparing it with the correspondent CFP, without caring about the specific transaction and about who is the requestor or the vendor, producing a new SLA candidate. SLA manager compare the score of each new produced SLA candidate with the ones belonging to the same transaction, filtering only the best ones.

In this scenario the data flow generates events which trigger asynchronous workers. A new CFP trigger idle vendors. New proposals triggers idle brokers and new SLA candidates activate SLA managers.

To conclude this section it is necessary to discuss about real time requirements, which can become relevant for such a kind of service. Let us suppose that we must integrate services form different providers, when constraints and preferences are not satisfied by only one. In this case it is necessary to consider all the possible compositions of services, being aware about how the Service Levels and cost of the composite service change. The brokering problem can become computationally hard as it is shown in [2] because of the exploration of a wide solution space. In fact the number of solutions to be evaluated grows exponentially and an exhaustive evaluation of all combination can become unfeasible. Heuristic approaches such as multi-objective genetic algorithms can reduce the computational requirements and the execution time, but introduce a loss of optimality. In this case it is possible to provide a sub-optimal solution within time constraints that can be acceptable for the user.

To integrate such approach within our Cloud service it is necessary to add an expiration time to the proposals and the capability for the SLA manager to trigger a new CFP that allows the vendors to produce the next generation of proposals.

This solution have been described in [5] and [11] where also experimental results have been shown. Experimental results show that Multi-Cloud brokering service can reduce the computational requirements and the execution time, but introduce a loss of optimality. The optimality is affected by the specific genetic algorithm, by its parameters such as the crossover, by the number of individuals evaluated (that means the execution time), the number of objectives to be minimized/maximized, the kind of problem. Performance evaluations showed the feasibility of the approach and that solutions approximated well the true Pareto front in all the cases at least in some condition. In any case the generality of the discussion about the scalability of the Cloud brokering service is not affected by the typology of solution.

5. System Overview.

The front-end architecture is composed of a web service that allows for

- Registration: Users can create a profile and have a unique personal account having the possibility to choose his credentials (username and password).
- CFP submission: the logged users can insert one or more Call For Proposal. Users that have inserted a CFP can run a new request, in a completely asynchronous way.
- Proposals upload: Cloud Providers can insert proposal and update the portfolio of their offer.
- Brokering results: Users can consult their requests, the status of the requests and the available results.

Figure 5.1 shows the front end section with several service instances that receive requests from end users and access in-memory shared information. The in-memory session manager will allow for the exploitation of elastic capability of the Cloud infrastructure. The warm copy of the session manager allows for improving reliability.

The back-end architecture is composed of stateless and asynchronous agents whose tasks are scheduled by service queues. Those agents have to be able to run on every available computing resources and perform the brokering. Nevertheless the front-end must be implemented by a service interface that accepts synchronous requests and forward them to an asynchronous back end. Those new pending tasks are stored in order to be handled by pools of asynchronous working agents that will return the current status of service elaboration.

As it is shown in Figure 5.1 agents will implement the back-end of the new agency, that is completely free from management of the interactions with clients. Cloud storages (database ad queues) keep persistent information of the distributed applications and implement communication channel between synchronous front-

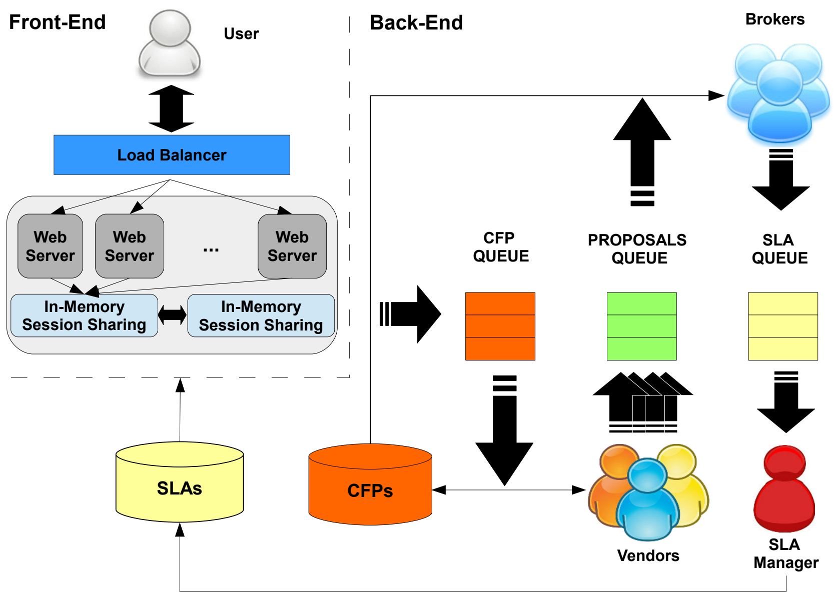


Fig. 5.1. *Distributed architecture*

end handlers and back-end workers. This design choice has been done in order to obtain an easy distribution of the workload using a task parallel programming model.

Stateless agents get new problems by a common Bag-of-Tasks in a concurrent and parallel way, executing wherever there are available computing resources, and updating the computing results if they complete successfully. If there are unsolved problems which remain in the Bag-of-Tasks because of any failures or delay, they are re-scheduled, so ensuring reliability.

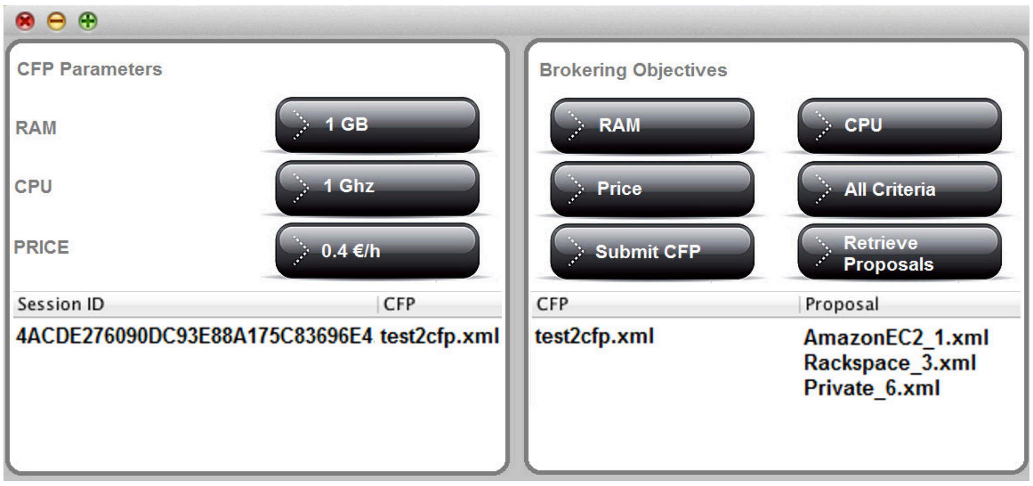
The vendors agent sign up to the CFP_QUEUE to receive the CFPs received from users. Each of them submits its proposal to the PROPOSALS_QUEUE. Idle brokers are waiting for proposals. A single proposal is dispatched to one broker that executes its matching with the correspondent CFP for evaluation purpose. The matching results is stored into the SLA_QUEUE if it belongs to the Pareto front of optimal solutions. ActiveMQ has been used as queue services for communication and synchronization. Brokering results are stored also into an in-memory session, together with information of each related user's request.

6. Prototype Implementation. The implementation of a prototype shows the utilization of open-source tools which results in a rapid implementation with minimal or no software input cost.

Users are provided with the web interface shown in Figure 6.1. It allows for composing the CFP and listing, for each session, the best SLAs according to different brokering objectives. Users log into a web page, the web page makes a request to a REST service and submits the call for proposals.

The Call for Proposals are included, along with information of the session and the user, in CFP queue. The characteristic of the CFP queue is that all consumers that join the queue, receive all the submitted CFPs. Apache Tomcat has been used as web and application server to run the web service at front-end. It has been specifically configured for working with the Terracotta Framework for the transparent distribution and sharing of web sessions. Jersey API have been used to implement the RESTFull service that provide methods for authentication, CFP submission and SLA retrieval.

Meanwhile their requests are pending, users can wait for the result of brokering or may poll periodically to get the status of their request. When one of the brokers has found a feasible proposal for a certain request, the current results are updated so that user can get it. The brokering terminates when there are no more proposal



Session ID	CFP
4ACDE276090DC93E88A175C83696E4	test2cfp.xml

CFP	Proposal
test2cfp.xml	AmazonEC2_1.xml Rackspace_3.xml Private_6.xml

Fig. 6.1. *Web GUI*

suitable for optimizing the user's query to be evaluated. Then the termination is notified to the user.

CFPs and SLAs are stored in the RDBMS Mysql databases, that is used as persistent storage.

The Web and Application Server chosen to run the web service at front-end is Apache Tomcat that is the most popular open-source Java Web application server [20]. Tomcat, typifying object-based servers, has some unique runtime properties including intensive threading, high read/write ratios, extensive object sharing via collections framework and fine-grained irregular object access patterns [8].

Terracotta [21] is a JVM-level clustering product. It works within an aspect-oriented programming (AOP) framework and has to instrument product-specific classes. Users need to manually specify shared classes as distributed shared objects (DSOs) and their cluster-aware concurrency semantics.

Tomcat has been specifically configured for working with the Terracotta Framework for the transparent distribution of web sessions.

Jersey API, that allow the creation of Asynchronous RESTFul web service, have been used by the service requestor and by vendor agents to handle HTTP service requests and responses. The relational database management system (RDMS) is implemented by Apache Derby. Finally Apache ActiveMQ provided us a distributed open source queue service. The queue service is very important because the scalability of the proposed architecture is inherently tied to the assumption that the message queue can scale perfectly. Figure 6.2 shows the sequence diagram of the system. The first action that is performed is the login phase. As one can see, the client sends an http request (post) to the server, which creates an instance of access for managing users and their profiles. After the authentication phase, users can enter their CFPs. In particular, the client sends a request for insert a CFP to the server, which creates an instance of Services class that uses the Producer class to access a Queue. Similarly, the client sends a request for insert a CFP to the server, which creates an instance of VendorServices class that uses the Producer class to access a Queue. A generic client sends a request for insert a proposal to the server, which creates an instance of VendorServices class that uses the Producer class to access a Queue. Indexing agents and broker agents are created to get elements respectively from the PROPOSAL QUEUE and from the CFP QUEUE as they have been populated by vendors and broker clients. In particular, a consumer process (which runs in the background) withdraws proposals from the proposals queue and inserts them from time to time in a database. The Consumer send a getProposal() message to Queue and obtains a proposal to be sent to the DB for store it. Brokers operate in background and send requests to the class queue to get a CFP and class DB to get the proposals inserted. After that, they start the brokering and send the result to the Session class that, using a database, returns the result to the user with the corresponding ID.

7. Experimental Results. In order to evaluate performances of the proposed approach the following testbed were set up.

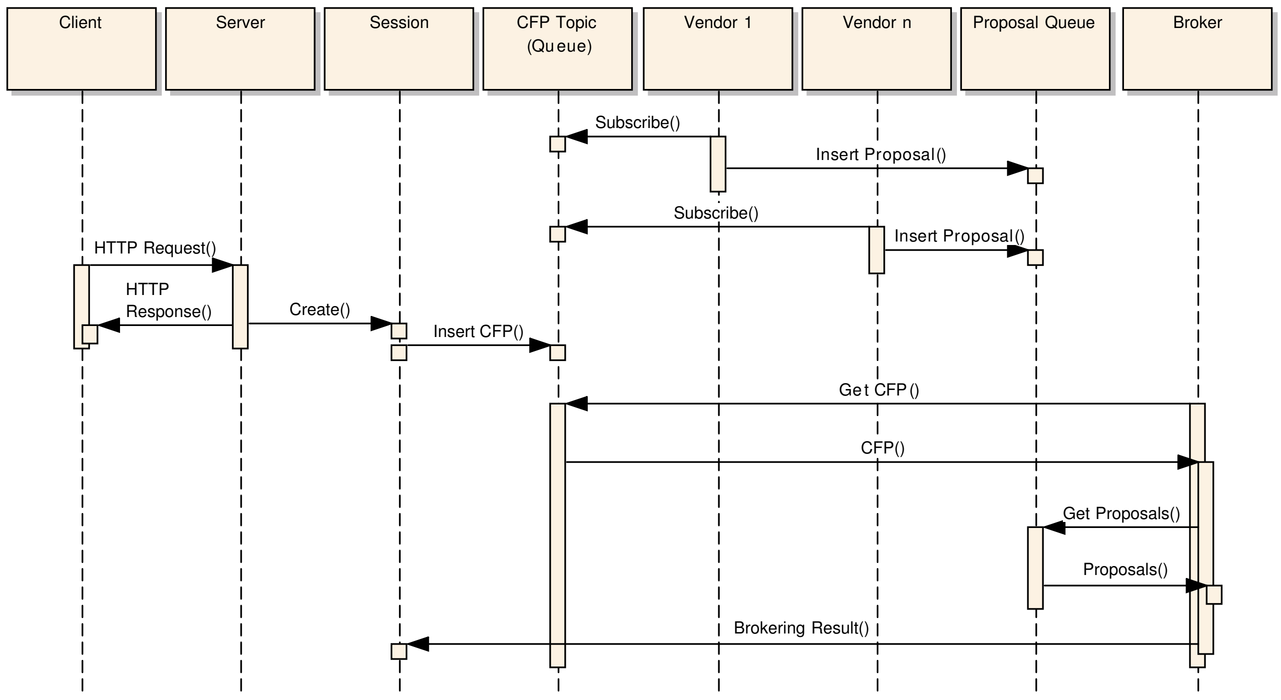


Fig. 6.2. *Diagram*

7.1. Description of Testing Environment and Test Cases. A Linux physical machine hosts the ActiveMQ 5.6 service, with a Topic, named CFP_QUEUE, that receives CFPs from concurrent clients, which run on a different physical machine in the same 1GB Ethernet local network. The server is 64bit Intel Core^{TM}2 Quad Processor Q9300 (6M Cache, 2.50 GHz, 1333 MHz FSB) with 4GB RAM. Oracle Java7 is the runtime environment. Concurrent clients send 10 CFPs, each one, according to a Poisson process with different mean time of arrivals. Vendors run on the server and wait for incoming CFPs. All vendors get the same CFP and generate their proposal, which is sent to the PROPOSALS_QUEUE. A Poisson distribution has been used to generate a synthetic workload. The average queue time has been measured by configuring the native trace log facility of Active MQ and analyzing the output. The measurements run on at least three machines. The database is only used at the start-up to load the current offers by the vendors and to save request and response (CFPs and SLAs). All the brokering computation does not use the RDBMS Mysql databases but information is stored in memory and in the queues.

7.2. Scenarios and Test Data. In a first scenario all brokers run on the server itself. They receive a different proposal from the QUEUE and evaluate the compliance with the correspondent CFP. The result is sent to an SLA_QUEUE from which only the best ones are notified to the clients. The performance of such configuration have been evaluated changing the number of clients, the number of vendors and the number of brokers. All the measures are taken at server side. In the following the most significant figures are explained. Figure 7.1 presents the number of evaluated proposals per millisecond for each experiment.

The first series of Figure 7.1 shows the case of 2 clients that send CFPs with a mean time of arrivals of 2 seconds to the server that hosts 4 vendors and only one broker. It is possible to see that to evaluate 80 proposals the server takes about 24 seconds in this case.

7.3. Results. It is possible to conclude that the application scales well in all the cases where the workload does not exceed the server capability.

The second series represents the same scenario, but with an mean time of arrivals that is 1 second. In this case it is straightforward to observe that only one broker is able to process all the requests in about 13 seconds.

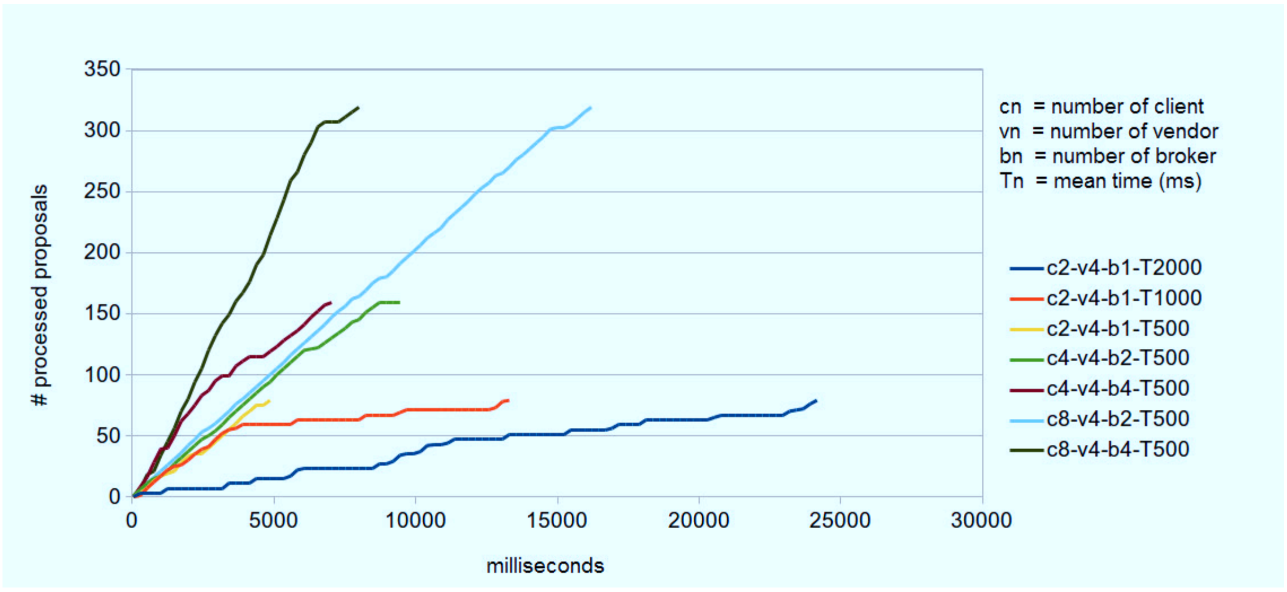


FIG. 7.1. *Proposals processing*

The slope of the line shows that the proposals are processed faster as they arrive with greater frequency and the workload is below the capability of the server. The same happens in about 5 seconds when the mean time of arrivals is 0.5 seconds (third series).

In the fourth series the number of clients and the number of brokers have been doubled. So there are 160 CFPs, which arrive with a doubled rate, but the throughput of the server does not change as the number of broker threads have been doubled. Just in the end it is possible to observe a slightly shift of the line because the clients stopped and the queue are going to be empty.

In the fifth series the number of brokers have been doubled resulting in a faster throughput at the beginning, if compared with the previous case, but after that the slope of the line follows again the trend of the previous case.

Finally, in the case of 8 clients with 2 and 4 brokers (sixth and seventh series) it is possible to observe that the slope of the line depends on the number of brokers again and it is constant till when the queue has proposals ready to be dispatched or the overhead makes the machine slow.

A loss of performance is observed either when the number brokers is greater than 4, as they exceed the number of cores of our server, and when the number of clients double, as the workload overcome the capability of our machine. Figure 7.2 shows the maximum throughput in the case of 8 clients and 4 brokers.

The effect of the overhead, due both to the need of handling the high rate of incoming CFPs and to the schedule of a number of broker greater than the number of cores can be observed. In particular 2 clients never overload the system. The workload increases both because the high number of concurrent clients and because of the scheduling overhead. For this reasons there is a random delay for both the traced events starting from when the server reaches the overload condition.

A noticeable effect of this behavior is the average waiting time of a proposal into the queue, that affects directly the response time of the system and service level perceived by the client.

Moreover it is possible to notice that, with the same mean time of arrivals, configuration with 8 clients and 4 brokers performs better than configuration with 8 clients and 8 brokers, and similar combinations. It happens because CPU exceeds the number of cores because of the overhead due to the process scheduling and to the context switch. In fact the Intel Q9300 quad processor has a single quad core CPU that does not support hyper threading. When the number of threads becomes 8 two of them share the same cache and the same core, affecting the performance.

In Figure 7.3 the occurrence of the overload condition can be observed, with 8 clients and 4 vendors varying

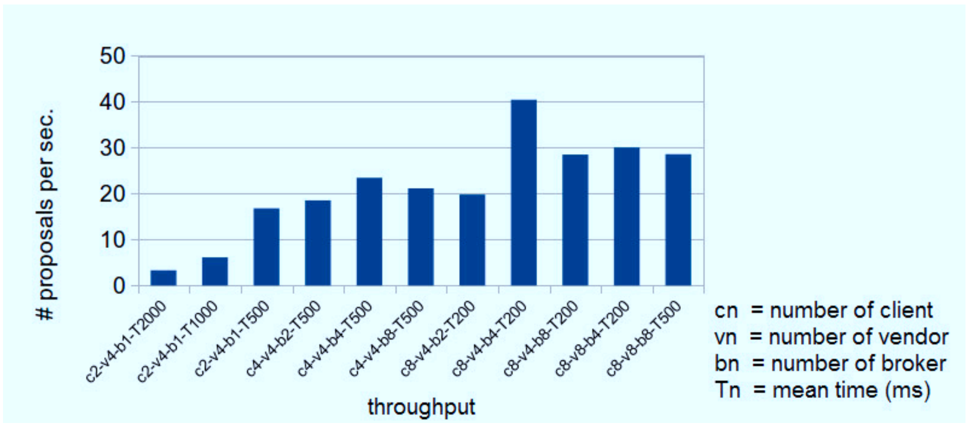
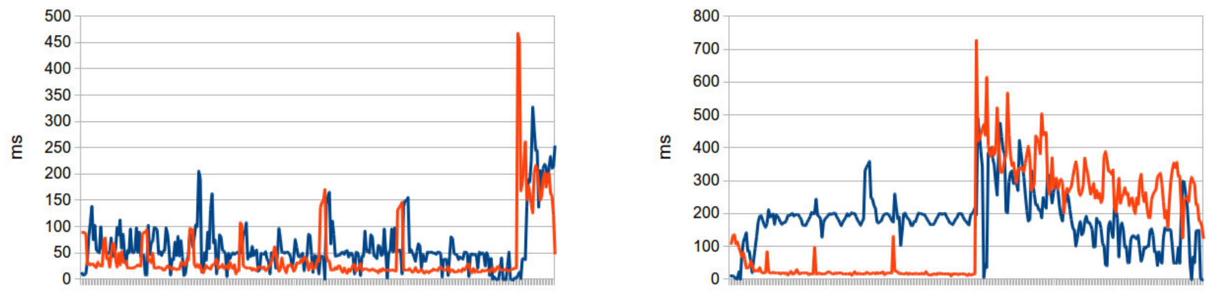


Fig. 7.2. *Broker throughput*



(a) c8-v4-b4-T500 (b) c8-v4-b8-T500

Fig. 7.3. *Mean time of arrivals vs service time of proposals*

the number of brokers. The x-axis shows the different proposals in order of arrival. The blue series shows the elapsed time between the arrival of a proposals into the queue and its withdrawal by an idle broker. The red series shows the time between receiving of a proposal by the broker and the next arrival into the SLA_QUEUE. In Figure 7.3 (a) 4 brokers pull proposals from the queue, keeping the processing time comparable with the enqueued time. At the end of the time series the rate of incoming proposals, which is slightly higher that the brokers' throughput, is going to introduce an overhead that affects the performance. However the overhead rapidly decreases because brokers have processed almost all proposals when clients finished to send their requests. In Figure 7.3 (b) the number of brokers exceeds the number of supported threads. Their scheduling, over a limited amount of resources, causes a waiting time of proposals greater than their processing time. The processing time is not affected till when the high rate of incoming proposals overloads the system. Experimental results show that the time elapsed from the beginning of the experiments and the overload condition is almost the same, as it depends on arrival rate of incoming requests. However in the first case almost all the requests have been processed, but in the second experiment many requests still need to be consumed and their processing time is much affected by the overload condition. These results demonstrate that when the rate of incoming requests grows it is necessary to look for a distributed solution that scales well when new computing resources are added.

Table 7.1 shows the average enqueued time of a proposals in the case of the server. It is much more than the mean time needed by a broker to process the proposal that is about 55 ms. It is possible to observe that before the performance degradation the speedup is 2.38 with two brokers and 2.52 with 3 brokers.

In order to improve this parameter, and to address the problems related to the overload caused by a too high arrival rate of requests, the possibility to offload part of the workload using a Cloud Infrastructure have

Table 7.1
Average enqueued time on server

configuration	8c-8v-1b	8c-8v-2b	8c-8v-4b	8c-8v-8b
average time	428ms	180ms	170ms	171ms

been investigated. An OpenStack installation in the same local network have been used. This private Cloud provided a Linux virtual machines. The Linux OS sees just one processor with a 64bit virtual Intel Core 2 Duo P9xxx (Penryn Class Core 2) 2.5GHz with 2K L1 cache and 2GB RAM. In order to estimate the processing capability of such computing resource tests have been performed on the scenario defined above with no brokers running on the server and some brokers executing only on the virtual machine. The estimated processing time for a proposal evaluation is equals to 100ms when only one broker is running. Table 7.2 shows that the average enqueued time for a proposals improves when two brokers are used and goes worse when 4 threads run concurrently. Also in this case the reason of the performance degradation is a number of thread greater than the number of cores, as the virtual processor does not support hyper threading. The speedup is 1.55 in the case of two brokers.

Table 7.2
Average enqueued time on VM

configuration	8c-8v-1b	8c-8v-2b	8c-8v-4b
average time	1430ms	920ms	11296ms

Cloud support have been tested to evaluate the improvement of the best working configuration presented before, that has 8 clients, 4 vendors and 4 brokers on the server. Table 7.3 shows that it is possible to improve the performances when 2 broker threads run on the VM. About the distribution of workload between machines, it has been delegated to the ActiveMQ service that distributes the messages to the consumers according to their capability to consume. It has not been studied further. Using heterogeneous machines it is not relevant the speedup as scalability measure. Moreover the benefits of using only one less powerful machine provide a not relevant improvement. The availability of many nodes, even if with reduced computational resource could be useful.

Table 7.3
Average enqueued time on hybrid computing infrastructure

configuration	8c-8v-4b	8c-8v-4b-1b	8c-8v-4b-2b	8c-8v-4b-4b
average time	170ms	165.578ms	160ms	167ms

8. Conclusion.

Cloud Computing is an emerging trend that will become even more widespread especially in companies and enterprises but some changes are needed to allow users to take advantage of these innovations. When moving to the Cloud it is important that the requirements for the move are understood and that the Cloud services are selected to meet these needs.

8.1. Background.

The paper presents a scalable and distributed solution for the SLA-based brokering of Cloud resources from different Cloud vendors and for Multi-Cloud brokering. It also describes the system architecture and implementation details of a prototype of this distributed broker that overcomes the problems of the centralized broker so allowing a Multi-User and a Multi-Cloud utilization.

8.2. Main Findings and Future Works.

The great advantages of the proposed approach is the scalability, obtained dividing the brokering problem into simpler tasks, which are distributed among independent agents, whose population dynamically scale together with the computing infrastructure, to support unforeseeable workloads produced by the interactions with large groups of users. Besides it allows the concurrency, is open and available to enable easy extensions of existing components and to add new components. A study of the scalability of the system and of the behavior of its components is described. In future work, we plan to perform a more extensive performance study in a wider range of scenarios.

REFERENCES

[1] A. Amato, B. Di Martino, and S. Venticinque, *Agents based multi-criteria decision-aid*, Journal of Ambient Intelligence and Humanized Computing, 5 (2014), pp. 747–758.

[2] ———, *Multi-objective genetic algorithm for multi-cloud brokering*, in Euro-Par 2013: Parallel Processing Workshops, D. an Mey, M. Alexander, P. Bientinesi, M. Cannataro, C. Clauss, A. Costan, G. Kecskemeti, C. Morin, L. Ricci, J. Sahuquillo, M. Schulz, V. Scarano, S. Scott, and J. Weidendorfer, eds., vol. 8374 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2014, pp. 55–64.

[3] ———, *Evaluation and brokering of service level agreements for negotiation of cloud infrastructures*, in ICITST, 2012, pp. 144–149.

[4] A. Amato and S. Venticinque, *Multi-objective decision support for brokering of cloud sla*, in The 27th IEEE International Conference on Advanced Information Networking and Applications (AINA-2013), Barcelona, Spain, March 25-28 2013, IEEE Computer Society.

[5] ———, *Modeling, Design and Evaluation of Multi-Objective Cloud Brokering*, In: International Journal of Web and Grid Services (IJWGS), Inderscience, Vol. 11, No. 1, 2015, ISSN: 1741-1114

[6] M. Cantara, *Hype cycle for cloud services brokerage*, 2013.

[7] Jamcracker, *Jamcracker cloud management.*

[8] K. T. Lam, Y. Luo, and C.-L. Wang, *A performance study of clustering web application servers with distributed jvm*, in Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on, Dec 2008, pp. 328–335.

[9] J. Lee, J. Kim, D.-J. Kang, N. Kim, and S. Jung, *Cloud service broker portal: Main entry point for multi-cloud service providers and consumers*, in Advanced Communication Technology (ICACT), 2014 16th International Conference on, Feb 2014, pp. 1108–1112.

[10] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber , *Bigtable: A Distributed Storage System for Structured Data*, in OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006.

[11] A. Amato, B. Di Martino, F. Xhafa, and S. Venticinque, *Brokering of Cloud Infrastructures driven by Simulation of Scientific Workloads*, in Proceedings of the XI Conference of the Italian Chapter of AIS, (2014).

[12] J. L. Lucas-Simarro, R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, *Cost optimization of virtual infrastructures in dynamic multi-cloudscenarios*, Concurrency and Computation: Practice and Experience, (2012).

[13] S. K. Nair, S. Porwal, T. Dimitrakos, A. J. Ferrer, J. Tordsson, T. Sharif, C. Sheridan, M. Rajarajan, and A. U. Khan, *Towards secure cloud bursting, brokerage and aggregation*, in Proceedings of the 2010 Eighth IEEE European Conference on Web Services, ECOWS '10, Washington, DC, USA, 2010, IEEE Computer Society, pp. 189–196.

[14] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, I. M. Llorente, *Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers*, in Future Generation Computer Systems, Volume 28, Issue 2, February 2012, pp. 358367.

[15] Optimis, *Optimis-project*, 2013, URL: www.optimis-project.eu/.

[16] S. Ried, *Cloud broker a new business model paradigm*, 2010, URL: https://www.forrester.com/go?docid=57809.

[17] RightScale, *Cloud portfolio management by rightscale*, 2013, URL: http://www.rightscale.com/cloud-portfolio-management/benefits.

[18] Verizon, *Cloudswitch*, 2011, URL: https://home.cloudswitch.com/.

[19] Talkin' Cloud, *2014 Talkin' Cloud 100: Top Cloud Service Providers, Aggregators and Brokers*, 2014, URL: http://talkincloud.com/tc100.

[20] A. Zeichick, *Tomcat, eclipse named the most popular in SDTimes study.*, 2013, URL: http://www.sdtimes.com/content/.

[21] A. Zilka, *Terracotta - jvm clustering, scalability and reliability for java.*, 2013, URL: http://www.terracotta.org.

Edited by: Marian Gusev
Received: Dec 18, 2014
Accepted: Mar 31, 2015