2010 IEEE 3rd International Conference on Cloud Computing

QoS–aware Clouds

Stefano Ferretti, Vittorio Ghini, Fabio Panzieri, Michele Pellegrini, Elisa Turrini
Department of Computer Science
University of Bologna
Bologna, Italy
{sferrett | ghini | panieri | pellegri | turrini}@cs.unibo.it

***Abstract*— In this paper we discuss the design and experimental evaluation of a middleware architecture that enables Service Level Agreement (SLA)-driven dynamic configuration, management and optimization of cloud resources and services. This architecture has been designed in order to respond effectively to the Quality of Service (QoS) requirements of the cloud customer applications. Typically, an application can be hosted in an execution platform constructed out of (real and virtual) cloud resources. In this context, the application QoS requirements can be specified in a SLA that binds the application to its hosting platform. Our architecture incorporates a load balancer that distributes the computational load across the platform resources, and monitors the QoS the platform delivers. If this deviates from that specified in the SLA, so as to violate it, the platform is reconfigured dynamically in order to incorporate additional resources from the cloud. In contrast, if the SLA is honored and platform resources result unused, platform reconfiguration occurs to release those unused resources.**

Keywords: cloud computing, Service Level Agreement, Quality of Service, load balancing, resource optimization, resource dynamic reconfiguration.

I. INTRODUCTION

We believe that the success of next-generation cloud computing infrastructures will depend on how effectively these infrastructures will be able to instantiate and dynamically maintain computing platforms, constructed out of cloud resources and services, that meet arbitrarily varying resource and service requirements of cloud costumer applications. Typically, these applications will be characterized by Quality of Service (QoS) requirements, such as timeliness, scalability, high availability, trust, security, specified in the so-called Service Level Agreements (SLAs). In essence, an SLA is a legally binding contract which states the QoS guarantees that an execution environment, such as a cloud based computing platform, has to provide its hosted applications with.

To the best of our knowledge, current cloud technology is not fully tailored to honor possible SLAs, although both the industrial and the academic research communities are showing growing interest on issues of QoS assurance within the context of cloud computing. In general, honoring an SLA requires an accurate assessment of the amount (and characteristics) of the needed resources. Application services hosted in clouds (e.g., Web applications, Web services) are often characterized by high load variance; hence, the amount of resources needed to honor their SLAs may vary notably over time.

As of today, in order to ensure that an application SLA is not violated, a resource *overprovision policy* is often adopted, that is based on evaluating (either through application modeling or through application benchmarking) all possible resources a hosted application can require in the worst case, and then statically allocating these resources to that application. This policy can lead to a largely suboptimal utilization of the hosting environment resources. In fact, being based on a worst-case scenario, a number of allocated resources may well remain unused at run time. This limitation can be overcome by developing a middleware architecture, as we propose, that can be integrated in a cloud computing platform so as to manage dynamically the cloud configuration, and honor the SLAs of the applications that platform hosts. This is accomplished by adding to, and removing from, the cloud resources at run time, as needed.

Thus, in this paper we discuss the design and evaluation of a middleware architecture that enables SLA-driven dynamic configuration, management and optimization of cloud resources and services, in order to respond effectively to the QoS requirements of the cloud customer applications. For the purposes of our discussion, we term "QoS-aware cloud" a cloud computing environment augmented with our middleware architecture.

A QoS-aware cloud will be able to change dynamically the amount of resources made available to the applications it hosts, in a proactive way. Optimal resource utilization will be achieved by providing (and maintaining at run time) each hosted application with the number of resources which is sufficient to guarantee that the application SLA will not be violated.

It is worth mentioning that the architecture we propose can be particularly attractive for private clouds, where the resource management may become quite critical if a limited number of resources is shared among several applications. Within this scenario, in case the total amount of the resources every application hosted in the cloud *requires* in the worst case be larger than the total amount of resources *available* in the cloud, a resource over-provision policy is possible only through the acquisition of additional resources (with the relative additional costs).

The architecture we discuss in this paper consists of the following principal services; namely a Configuration Service and a Load Balancer, which in turn includes a Monitoring Service and an SLA Policy Engine. This architecture extends

978-0-7695-4130-3/10 $26.00 © 2010 IEEE
DOI 10.1109/CLOUD.2010.17

an earlier middleware architecture we developed as part of the project TAPAS [4] in order to support clustering of QoS-aware application servers [2]. A preliminary experimental evaluation we have carried out shows that the design principles we adopted in the TAPAS project are equally appropriate in the context of cloud computing in order to construct what we have termed QoS-aware clouds.

This paper is structured as follows. The next section introduces the notion of SLA and illustrates how an SLA can be used to express QoS application requirements. Section III describes the architecture we propose, and discusses the principal issues we have addressed in the design of the Configuration Service and the Load Balancer mentioned earlier. Section IV examines the results of an evaluation of our architecture we have carried out through simulation. Section V compares and contrasts our approach with relevant related works. Finally, Section VI proposes some concluding remarks and introduces some future developments.

II. SERVICE LEVEL AGREEMENTS

QoS requirements are usually specified in so-called Service Level Agreements (SLAs). In general, an SLA consists of a collection of contractual clauses between a service provider and a service customer that detail the nature, quality and scope of the service to be provided, and its authorized use [11,12]. Thus, both service customer and service provider responsibilities can be stated, and their relative rights and obligations defined.

Typically, within the context of cloud computing, the service provider is a cloud computing provider, and the service customer is an application requesting a hosting service from that provider. In this context, the hosting service can be instantiated by an execution platform, constructed out of cloud resources and services, for use from the application. The service customer and service provider responsibilities mentioned above specify the level of QoS the application requires to be delivered from that platform, and the obligations the application is to honor.

Note that an application hosted by a cloud based platform may itself deliver a service to clients of its own, and be bound by a SLA to those clients. Thus, for the sake of completeness, in the following we discuss both the SLA between a customer application and its hosting environment, and the SLA between that application and its own clients.

The XML code below illustrates a SLA fragment which specifies the following specific customer obligations and rights: namely, the maximum request rate at which an application may issue requests to the virtual execution environment in the cloud (in this example the request rate must not surpass the limit of 100 requests per second), and the operations the application is allowed to invoke from this environment (these operations can be listed as children elements of the <*Operations*> element, not shown in the example below).

```
<ContainerServiceUsage name="HighPrority"
                requestRate="100/s">
    <Operations>
    ...
    </Operations>
    ...
</ContainerServiceUsage>
```

The following code shows some typical responsibilities of a service running in the cloud.

```
<ServerResponsabilities
                serviceAvailability="0.99"
                efficiency="0.95"
                efficiencyValidity="2">
    <OperationPerformance
                name="HighPriority"
                maxResponseTime="1.0s">
                <Operations>
                ..
.
                </Operations>
    </OperationPerformance>
    ...
</ServerResponsabilities>
```

In this case, a *serviceAvailability* attribute is provided that specifies the probability with which the hosted application must be available over a predefined time period (in the example above, the daily availability is to be no less than 99%). In addition, a response time attribute is defined, i.e. *maxResponseTime*, that specifies the required service responsiveness [2].

Finally, an SLA may also specify the percentage of SLA violations that can be tolerated, within a predefined time interval, before the service provider incurs a (e.g., economic) penalty. The *efficiency* and *efficiencyValidity* attributes in the example above capture this requirement. These attributes specify that no less than 95% of client requests are to be served, within any two-hours interval. Hence, we assume that the SLA QoS attributes (such as the response time) can be violated during the SLA efficiency validity period (two hours in the above example), provided that the violation rate is maintained below the SLA efficiency limit (in our example the percentage of tolerated violations is 5%).

The issues discussed above derive from middleware solutions for distributed applications running, for instance, on application servers [2]. As mentioned earlier, the cloud computing paradigm introduces what can be thought of as an additional level of SLA, between the service provider and the cloud infrastructure hosting that service. In other words, the virtual execution environment offered by the cloud infrastructure to the applications must provide some performance guarantees. Examples exist of performance metrics and monitoring of virtual machines, e.g. [1]. For instance, the cloud infrastructure may provide the applications it hosts with guarantees on specific hardware and network parameters, e.g. CPU power, available RAM and storage, network bandwidth, as well as scheduling and allocation policies, and so on. These guarantees can require a specific SLA between the application and the cloud infrastructure, as shown in the following code fragment.

```
<InfrastructureSLA>
    <node typename="alpha">
```

```
<cpu num="4" type="x86-64"
        performance="X"/>
<ram dim="64GB" performance="Y"/>
<storage dim="640GB"
         performance="Z"/>
<network>
        <if name="eth0">
                <incoming rate="5Mbit"
                peak="10Mbit"
                maxburst="10MB"/>
                <outgoing rate ... />
                ...
</network>
</node>
<outgoingTraffic rate="10Mbit"
        peak="15Mbit"
         maxburst="100MB"/>
...
</InfrastructureSLA>
```

In the example above, all the hardware characteristics of a given node are listed, such as the number and typologies of the CPUs, RAM and storage dimensions. Moreover, network characteristics are included, such as the average download rate of the node, its possible peak, and so on.

III. CLOUD ARCHITECTURE

In this section, we first discuss the principal design issues that are to be addressed when designing a QoS-aware distributed (middleware) architecture. Then, we overview the main principles and features typical of a cloud computing system. Finally, these two views are merged into an architectural proposal for QoS-aware clouds. Due to space limitations, we will not go into technical details of the software components, focusing on the functionalities they must provide.

A. On the Design of a QoS-Aware Distributed Architecture

When an organization decides to exploit the computing resources and services available from cloud service providers for executing its information services and applications, its main goal is to maximize the efficient use of the resources while minimizing the provider bill. This is particularly true in a cloud computing context where resources can be acquired dynamically by considering both the run time conditions of the computing environment and the application QoS requirements that are to be honored. In other words, a cloud service provider has to meet the following two principal requirements:

1. guaranteeing that the application QoS requirements are met;
2. optimizing the resource utilization in meeting the above requirements.

To this end, a QoS-aware cloud architecture should incorporate the following three principal components.

Dynamic Configuration – This component is responsible for instantiating an application hosting platform that can meet the application QoS requirements and is constructed out of resources obtained from a pool of available resources (e.g., a cloud). This component must maintain the hosting platform and manage the resource pool in a proper manner; that is, it may need to allocate new resources to the platform, or release those currently allocated to the application, at run time. The resource allocation/release policy must guarantee that the QoS requirements of the hosted application are met using as little resources as possible. Hence, in general, resources must be allocated at the time the application is deployed in the hosting platform, and possible resource reconfiguration may occur at run time.

The initial configuration process consists of requesting the minimal set of resources that ensure the service availability requirements are met. The run time reconfiguration process consists of dynamically resizing the resources in the hosting platform, by adding/removing resources to/from it, as needed. Adding resources can be necessary in order to handle a dynamically increasing load and in case a resource is no more available and needs to be replaced by an operational one (or possibly more than one). Releasing resources may be necessary to optimize the use of the resources themselves and, according to the pay-as-you-go principle, reducing the costs for running the application. If the load on a hosted application significantly decreases, some of the resources allocated to that application can be dynamically released and included in the pool of spare resources of the cloud possibly available to other applications.

Monitoring – monitoring the platform at application run time enables the detection of possible 1) variations in the platform configuration, 2) variations in platform performance, and 3) violations of the QoS requirements. Thus, a monitoring service should be able to check periodically the platform configuration to detect whether resources join or leave their home platform, following failures or voluntary connections to (or disconnections from) it. In addition, it monitors data such as the response time of each resource in the platform, the current application load, and whether the QoS delivered by the monitored platform deviates from that required by the running application.

Dispatching and load balancing – some specific service is needed to dispatch requests to the resources in the hosting platform. Examples of requests can be a client request for an HTTP page (in case of a Web application), a mail that has to be filtered (in case of an anti-spam filter), etc. When the platform is offered to clients like a set of separate (possibly virtual) resources, so that each client has the visibility of each single resource allocated to it, such service should balance the load of the application among the resources of the platform. This may contribute to meeting the QoS requirement by preventing the occurrence of resource overload and avoiding the use of resources that have become unavailable (e.g., failed) at run time.

The load balancing service can easily detect specific resource conditions, such as resource response times and pool membership configuration. If for instance, the hosted application is a Web application running in a cluster of application servers, the load balancing service can be thought of as a reverse proxy server that essentially intercepts client

HTTP requests for an application and dispatches these requests to the application server hosting that application [2]. Such service should include support for both request-based and session-based load balancing. With request-based load balancing, each individual client request is dispatched to any clustered node for processing; in contrast, with session-based load balancing, client requests belonging to a specific client session are dispatched to the same clustered node (this client-server correspondence is termed session affinity).

In general, the load balancing service is responsible for:

1. intercepting each client request;
2. selecting a target resource that can serve that request by using specific load balancing policies;
3. deftly manipulating the client request to forward it to the selected target resource;
4. receiving the reply from the selected target resource, and, finally
5. providing a reply to the client that sent the request.

The load balancing policy should be adaptive in order to balance the load dynamically among resources in the hosting platform. Lightly loaded resources should be selected in order to serve novel client requests. Hence, when an incoming client request is intercepted at the load balancing service level, that request must be forwarded to the resource with the shortest pending requests queue. This policy allows one to evaluate dynamically the expected performance of each resource in the platform, even in the presence of load imposed by other services running on those resources.

B. *On the Design of a Cloud Architecture*

Typical cloud architectures are designed to provide a clean separation of concerns among the layers operating at different abstraction layers. Usually, some manager is present in the architecture, which interacts with the application providers that wish to run their applications in the cloud. Then, a different component should be in charge of organizing how different applications (or execution environments) must be executed within the cloud. Finally, resources must be adequately managed and monitored.

Different cloud computing systems follow this general view. For instance, the RESERVOIR project foresees the following software components [3]. A module termed Service Manager is considered, which represents the software component that interacts with service providers. Based on the received requests, it dynamically derives a list of required resources to handle the job, their configuration, as well as constraints on the environment of execution. To execute the requests coming from the applications, the Service Manager interacts with a Virtual Execution Environment Manager (VEEM) running at a lower level of abstraction than the Service Manager, responsible for optimizing the scheduling of the execution environments into cloud resources. Hence, the VEEM decides where the execution environments must be run. Finally, the lowest level of abstraction is the Virtual Execution Environment Host (VEEH). This layer is responsible for the basic control and monitoring of execution environments and their resources. A VEEH usually encapsulates some kind of virtualization technology, in order to guarantee full interoperability between the applications to be run on the cloud and the employed physical resources.

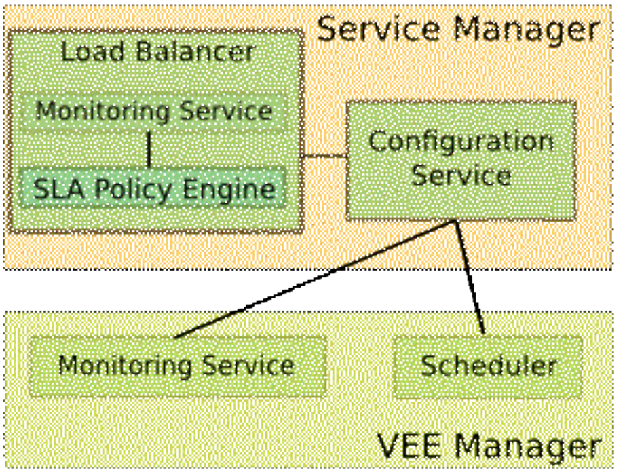

Figure 1. QoS-aware Cloud Architecture

Our software architecture has been developed in order to be integrated within RESERVOIR. This architecture is depicted in Fig. 1, and introduced in the next Subsection.

C. *A QoS-Aware Cloud Architecture*

The software modules depicted in Fig. 1 cooperate with each other to ensure QoS enforcement and monitoring. These modules work at the Service Manager and VEEM levels of abstraction, leaving the configuration and management of the hosts and resources to the classic virtualization technologies typically utilized in cloud computing architectures.

Basically, the Load Balancer is responsible for implementing the load dispatching and balancing functionalities introduced earlier in Section III.A. As it is implemented at the Service Manager level of abstraction, it is independent of the specific virtual node that will execute the specific request. The Load Balancer simply receives requests from the application clients and dispatches these requests appropriately to platform resources, balancing the load. A Monitoring Service is included within the Load Balancer. It is in charge of monitoring incoming requests and related responses, so as to check whether the SLAs associated to these requests are satisfied or violated. In order to control SLA policies, a SLA Policy Engine is exploited, which analyzes logs of the Monitoring Service to identify possible SLA violations. Based on its outcomes, the system decides whether a system reconfiguration is necessary, i.e. if the cloud needs additional virtual nodes (scaling up) or if some virtual nodes may be released (scaling down). In such a case, it invokes the Configuration Service so as to reconfigure the virtual resource pool.

The Configuration Service acts upon the VEEM in order to add new virtual resources. Its purpose is to identify an upper boundary above which adding new resources does not introduce further significant performance enhancements. This can be caused by factors such as increased coordination costs for pool management and bottlenecks due to shared

resources such as a centralized load balancing service or a centralized DBMS. In this particular case, it is in fact important to avoid the unwanted situation when, as performance degrades (due to the problems mentioned above), the system needlessly continues to add nodes in the face of SLA violations that are not caused by a lack of virtual resources; thus, the client is billed for an additional, unnecessary use of resources. Needless to say, the Configuration Service manages the virtual resources by interacting with software modules that specifically control them, i.e. a specific Monitoring Service and the scheduler which directly control each specific virtual resource.

Note that the Configuration Service can augment the set of employed resources by introducing one new virtual resource at a time or more than one in a single action. When adding one node at a time, a waiting time elapses between the Configuration Service reconfigurations following each resource inclusion. This time may be useful for handling the transient phase of a new added resource. The transient phase represents the time elapsed from the introduction of the new resource in the application hosting platform until it reaches a steady state enabling it to serve the client requests. On the other hand, adding more than one resource at a time can be useful to deal with possible flash crowd events. In fact, these events may not be fully resolved by adding just one resource at the time to the platform, owing to the above-mentioned transient phase.

By interacting with each specific Monitoring Service of each specific virtual resource, and with the Monitoring Service of the Service Manager, the Configuration Service may detect if the allocated virtual resources are effectively responding to the injected client load (that is, the violation rate trend of the virtual pool is significantly below the QoS limit). In this case, the Configuration Service might decide to release some allocated virtual resources, as they are no longer necessary (see below).

In configuring/reconfiguring the application hosting platform, the Configuration Service produces a resource plan object. This object includes the reference of each resource belonging to the built platform. In essence, the resource plan specifies the resources to be used in order to construct the platform capable of meeting the application QoS requirements. Resource failures and voluntary connections to the platform are detected by the Monitoring Service, which then raises an exception to the Configuration Service. In both cases, the Configuration Service reconfigures the platform; that is, it updates the resource plan by removing (or adding) the resource(s) that have become unavailable (or available); in addition, in case of resource failures it adds new resources as the modified platform configuration may be unable to meet the QoS requirements.

To conclude this subsection, it is worth mentioning that in our scenario, we assume that the virtual resources used for the reconfiguration of the application hosting platform are virtual machines on which the applications can be deployed. This assumption is motivated by the observation that the virtualization technology is establishing itself as the predominant technology for a large number of cloud computing providers [10]. Moreover, given the growing popularity of virtualization, many commercial and/or open source products such as OpenNebula [13], are being developed in order to provide the abstraction of virtual machines and to enable their dynamic management.

D. Managing SLA Violations

Our approach to the detection of possible SLA violations at the Service Manager level, is based on continuously assessing the trend of SLA violations in the system. In this sense, our approach is proactive, since its goal is to dynamically reconfigure the resource allocation before the violation rate reaches a predefined threshold. It is worth noticing that our current design focuses on performance SLA rules, such as response times and availability. Other QoS aspects, like security and privacy, are not yet handled in our architecture. We are planning to address these issues in our future work.

In particular, for each completed request, the response time is measured. This value is utilized to obtain and update a measure of an efficiency index the system maintains. When the response time surpasses the maximum response time in the related SLA, then the efficiency index is decremented. Otherwise, if the response time respects the SLA and the efficiency index is still below a maximum value, then the efficiency index is increased [9].

When this index becomes smaller than a warning threshold, the system is reconfigured and novel resources are added. Conversely, if the efficiency is equal to the maximum possible value and the virtual resource pool results underutilized, then a proper number of virtual resources are released. Note that releasing virtual resources is possible provided that the remaining allocated resources are sufficient to meet the service availability requirement specified in the SLA.

IV. Experimental Assessment

In this section we discuss an assessment of our architecture we have carried out through simulation. The objective of this assessment was to verify whether our approach enables to instantiate application hosting platforms that meet their SLAs, and optimize the use resources in the cloud. In the following, we outline the main characteristics of our simulation exercise. Next, we discuss the results we have obtained from this exercise.

A. Simulation Scenario

We created a simulation environment able to validate a general cloud architecture including the one proposed in this paper. It is a discrete-event based simulator, comprising different software components, as outlined below.

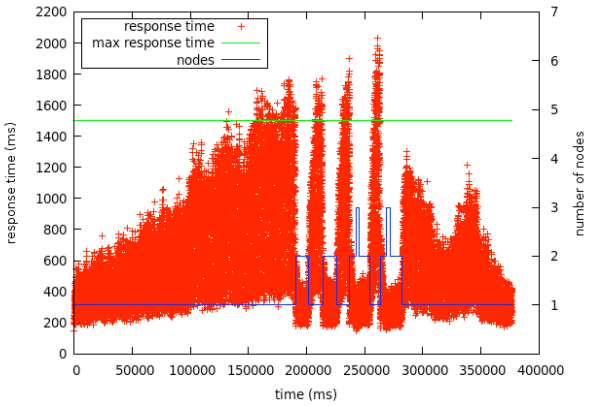

Figure 2. Resource utilization and response time.

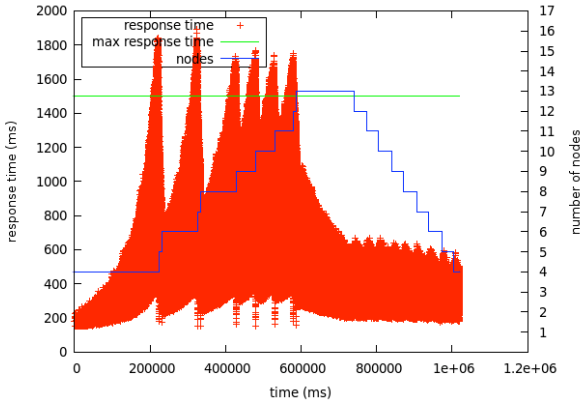

Figure 4. Resource utilization and response time.

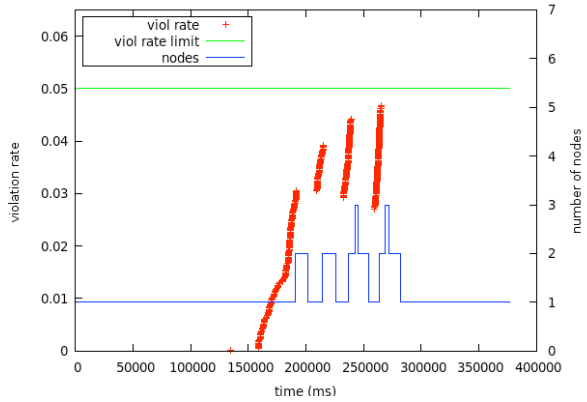

Figure 3. Resource utilization and violation rate.

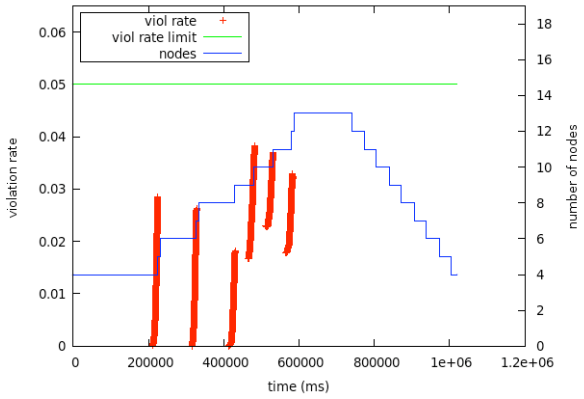

Figure 5. Resource utilization and violation rate

Firstly, a *request generator* module has been developed to simulate requests from the applications, depending on some workload distribution. This request generator interacts with the Load Balancer we devised, placed at the Service Manager layer of abstraction. This software component includes the Monitoring Service and the SLA Policy Engine. These modules manage the workload in the cloud, taking into consideration QoS constraints and trying to avoid that SLAs are violated. It is worth mentioning that they are fully implemented and could be in general included in a real system, so as to work within a cloud. The SLA Policy Engine manages SLAs declared as XML documents, as discussed in the previous section.

Secondly, the simulated architecture includes the Configuration Service in charge of regulating the number of active virtual resources. These resources are simulated by means of a *response generator* that generates the request response times according to the number of available virtual resources and the load produced by the *request generator* introduced above.

It is worth mentioning that in cloud computing, allocating resources has a non-negligible cost, due to the fact that applications are run on top of virtual execution environments [8]. In fact, scheduling algorithms and physical resource monitoring are employed to identify where these virtual execution environments can be executed. Thus, modeling allocation times before the execution of a job is an important aspect in the simulation. Owing to this observation, we varied the allocation time for setting up a virtual execution environment. As we will show, this may have some important impact on the provision of QoS in a cloud.

In our simulation environment, we assume that our middleware can manage a pool of spare Virtual Machines (VMs) available for allocation. If our middleware detects that the SLA is being violated, it allocates to the application hosting platform a new VM (removing it from the pool). In this case the hosting platform (exploiting the new VM) will become more responsive and the average response time will return under the limit specified in the SLA. Similarly, resource optimization principle requires to deallocate VMs (and re-insert them in the pool of spare VMs) when they are no longer necessary in order to meet the SLA.

Note that VMs are allocated/deallocated dynamically, monitoring at run time the performance of the execution environment and comparing it to the value specified in the SLA. In our experiment the maximum number of VMs allocated is three. Therefore, in an over-provision policy, all three VMs should be used for the entire duration of the test;

in contrast, to honor the hosting SLA, our middleware allowed us to dynamically allocate a minimum of one up to three VMs depending on the imposed load at different time intervals. Note that cloud provider will bill, for every VM, only the time it has been allocated to run the application (and not the time the VMs are in the pool of spare resources).

B. Results

The purpose of the experiments we carried out was to assess the ability of our middleware to optimize resource utilization without causing hosting SLA violations.

In view of these assumptions, for our test we set the SLA efficiency attribute to 95 percent; thus, the SLA response time requirement could be violated by 5% over a predefined timeframe, at most. The allocation time, i.e. the time necessary to set up a new VM and to include it in the execution environment, was set to 2 seconds.

We injected artificial load into the cloud following a simple request distribution: the load has been progressively increased until it reached about 90 requests per second (the limit specified by SLA was 100 reqs/sec) and then progressively decreased. This test ran for approximately 400 seconds of simulated time; the results obtained are illustrated in Fig. 2. As the chart shows, our middleware components dynamically adjust the hosting platform size as necessary, augmenting the number of clustered VMs as load increases and releasing VMs as load decreases. However, note that the SLA Policy Engine tends to deallocate a VM and to re-allocate it again quickly. Although this behavior could cause additional overhead, it optimizes the resource utilization as it permits to use as little VMs as possible to honor the SLA.

In this test, we also measured the percentage of SLA violations (see Fig. 3). Here, the peaks correspond to the instant in which a new node had to be added to the hosting platform for not incurring SLA efficiency requirement violations; however, as shown in Fig. 3, the SLA violation rate is maintained below the limit imposed by the hosting SLA.

Finally, two observations are in order. The first one is concerned with the maximum number of allocated VMs. In other tests we have carried out (results are shown in Fig. 4 and Fig. 5) we have augmented the load imposed to the cloud so as to cause the allocation of up to 13 VMs. However, it is worth noticing that if the number of VMs included in the virtual execution environment exceeds a given limit, some scalability problems may arise for certain distributed applications. As an example, a shared DB could become a bottleneck for the system, hence causing performance degradation.

The second observation concerns the allocation time, i.e. the time a VM requires to be included in the virtual execution environment running the application. In the test we have just described, we set a mean allocation time equal to 2 seconds. We have carried out other tests that have shown that augmenting the allocation time (and leaving unchanged the other simulation parameters) the SLA requirements may not be honored. Specifically, the longer the allocation time, the greater the SLA violation rate. In our opinion this is an important result as it suggests that the adoption of VMs to deploy applications (practice that characterizes the cloud computing technology) might entail application performance degradation if not properly managed.

In view of the above observations, with the aim of avoiding SLA violations caused by allocation time values greater than 2 seconds, we have modified the value of the parameters that are used to take reconfiguration decisions. Thus, we set such parameters so that the longer the time to allocate a new VM, the sooner the SLA Policy Engine will trigger the reconfiguration. A preliminary test we have carried out produced encouraging results but further investigations are necessary.

V. Related Work

In the last few years, a number of papers has been published on the subject of cloud computing (e.g., [14,15,16,17,18] to mention a few). However, to the best of our knowledge, the topic of QoS provision in cloud computing environments has not received much attention as yet. Nevertheless, some scientific papers recently published reveals a growing interest in this topic in both the industrial and academic research communities. In this section we briefly review a few papers on this topic that show some analogy with our approach to development of QoS-aware clouds. (However, please note that we are not proposing an exhaustive evaluation of the state of the art on this topic as it would be out of scope of this paper).

In [5] the authors describe a method for achieving resource optimization at run time by using performance models in the development and deployment of the applications running in the cloud. Although the overall goal of that work appears to be quite similar to ours, the methodology used and the case study they present are very different from those we propose. In fact, their approach is based on a LQM performance model (Layered Queuing Models, a kind of extended queueing network model) that predicts the effect of simultaneous changes (such as resource allocation/ deallocation) to many decision variables (throughputs, mean service delays, etc.). Moreover this model seems to take no account of variables that could heavily affect the performance of the applications hosted in the cloud. Examples of such variables include: (i) possible bottlenecks produced by load balancers, databases, and applications replication; (ii) the allocation times required for the introduction of a new VM in the virtual pool; (iii) the time elapsed from the introduction of the new VM in the virtual pool until it reaches a steady state.

The RESERVOIR project previously mentioned notably influenced our solution. From this project we derive the architectural model of the cloud and the levels of abstraction. However, the RESERVOIR project mainly "*addresses the limited scalability of a single-provider cloud, the lack of interoperability among cloud providers, and the lack of built-in Business Service Management support in current cloud offerings"*. Although it addresses also issues of SLA management, the available documentation does not provide one with sufficient details as to the implementation of the RESERVOIR SLA management features.

Other works are complementary to ours as they focus only on issues related to the definition and monitoring of the SLAs in a cloud computing environment and do not address issues of QoS enforcement and resource optimization. In [6] the authors present a methodology for adding or adjusting the values of non-functional properties in service descriptions depending on the service run time behavior, and then dynamically deriving adjusted SLA template constraints. In contrast, in our proposal the SLA is defined offline and cannot be modified at run time. Issues related to the SLA monitoring are presented in [7]. In that paper the authors introduce the notion of Service Level Management Authority (SLMA), a third independent party that monitors the SLA and facilitates the interaction between the Cloud vendor and the end customer. This approach differs from ours as in the solution we propose the monitoring facilities are implemented by a component of our middleware platform rather than by an external entity. (However it is worth noticing that due to the modularity of our architecture, one could investigate the possibility of integrating a SLMA in our solution).

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have described a middleware architecture that we have designed in order to enable platforms, constructed out of cloud computing resources, to meet the QoS requirements of the applications they host. An initial evaluation of this architecture, carried out through simulation, has provided us with very encouraging results that confirm the adequacy of our design approach.

We are planning to carry out further experiments using a real cloud computing environment as a test bed. In addition, we are planning to extend our investigation on issues of QoS in cloud computing environments by including in our assessment dependability issues such as fault tolerance, trust and security. In particular, we would like to examine those issues in the emerging context of cloud federations. In addition, in this context we shall address issues of scalability in cloud computing. Specifically, we shall investigate the design of a Service Manager architecture that, in contrast with that described in this paper, can cope effectively with the scaling out of a cloud as this joins a cloud federation.

Last but by no means least, our future research plans include investigating the integration of cloud computing and mobile technology, as discussed in [19].

REFERENCES

[1] K. Keahey, M.O. Tsugawa, A. M. Matsunaga, and J.A. B. Fortes, "Sky computing", IEEE Internet Computing 13(5): 43-51, 2009.

[2] G. Lodi, F. Panzieri, D. Rossi, and E. Turrini, "SLA-Driven Clustering of QoS-Aware Application Servers", IEEE Trans. Soft. Eng. 33(3): 186-197, 2007.

[3] Rochwerger, B., Galis, A., Levy, E., Cáceres, J. A., Breitgand, D., Wolfsthal, Y., Llorente, I. M., Wusthoff, M., Montero, R. S., and Elmroth, E. 2009. "RESERVOIR: management technologies and requirements for next generation service oriented infrastructures", Proc. of the 11th IFIP/IEEE international Conference on Symposium on integrated Network Management (New York, NY, USA, June 01 - 05, 2009). IEEE Press, Piscataway, NJ, 307-310.

[4] TAPAS, IST Project No. IST-2001-34069, http://tapas.sourceforge.net, 2006.

[5] J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai. "Performance model driven QoS guarantees and optimization in clouds", Proc. of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009.

[6] J. Spillner and A. Schill. "Dynamic SLA Template Adjustments based on Service Property Monitoring", Proc. of the 2009 IEEE Conference on Cloud Computing, 2009.

[7] A. Korn, C. Peltz, and M. Mowbray. "A Service Level Management Authority in the Cloud". Technical Report, 2009.

[8] B. Sotomayor, K. Keahey, and I. Foster, "Overhead Matters: A Model for Virtual Resource Management", 2006 in Second International Workshop on Virtualization Technology in Distributed Computing (VTDC 2006).

[9] F. Raimondi, J. Skene, and W. Emmerich, "Efficient Online Monitoring of Web-Service SLAs", Proc. of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, 2008

[10] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud computing". Technical Report No. UCB/EECS-2009-28, University of California at Berkley, USA, Feb. 10, 2009.

[11] J. Skene, D. Lamanna, and W. Emmerich. "Precise Service Level Agreements", Proc. of 26th International Conference on Software Engineering (ICSE '04), Edinburgh, Scotland (UK), 25 May 2004

[12] A. Keller and H. Ludwig. "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services". Technical report, IBM Research RC22456, March 2003.

[13] OpenNebula, http://www.opennebula.org/

[14] R. Moreno-Vozmediano, R. S. Montero, and I. M.. Llorente. "Elastic management of cluster-based services in the cloud", Proc. of the 1st Workshop on Automated Control For Datacenters and Clouds (ACDC '09), ACM, June 2009, pp.19-24.

[15] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. "The Eucalyptus Open-source Cloud-computing System", Proc. of Cloud Computing and Its Applications, 2008.

[16] S. Han, M. M. Hassan, C. Yoon, and E. Huh. "Efficient service recommendation system for cloud computing market", Proc. of the 2nd international Conference on interaction Sciences: information Technology, Culture and Human, (ICIS '09), 2009.

[17] A. Weiss, "Computing in the clouds," netWorker , Dec. 2007, pp.16-25.

[18] R. Buyya, C.S. Yeo, and S. Venugopal. "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities", Proc. of the 2008 10th IEEE international Conference on High Performance Computing and Communications (HPCC.2008), IEEE Computer Society, 2008, pp. 5-13.

[19] S. Ferretti, V. Ghini, F. Panzieri, E. Turrini, "Seamless Support of Multimedia Distributed Applications Through a Cloud", Proc. of the IEEE 3rd Int. Conf. on Cloud Computing (Cloud 2010), Work-in-Progress Track, Miami, Florida USA, 5 - 10 July, 2010.