2013 International Conference on Cloud Computing and Big Data

Load Balancing approach for QoS management of multi-instance applications in Clouds

Mohamed Mahmoud OULD DEYE
Faculty of Science and Technology
University Cheikh Anta Diop
Dakar, Senegal
Email: mohamed.oulddeye@ucad.edu.sn

Yahya SLIMANI
Institut Supérieur des Arts Multimédia
Université de la Manouba
Manouba, Tunisie
Email: yahya.slimani@fst.rnu.tn

Mbaye SENE
Faculty of Science and Technology
University Cheikh Anta Diop
Dakar, Senegal
Email: mbaye.sene@ucad.sn

***Abstract*—The success of the technology of cloud computing lies mainly in its business model of pay-as-you-go where users pay only for the resources really consumed. However it is known that there is no warranty for the QoS that these resources will provide at runtime. In this paper, we suggest an approach to make load balancing more dynamic to better manage the QoS of multi-instance applications in the Clouds. This approach is based on limiting the number of requests that, at a given time, can be effectively sent and stored in queues of virtual machines through a load balancer equipped with a queue for incoming user requests. This limitation is intended on the one hand to allow requests to go on to the faster instances, and on the other hand to better mitigate the effects of interference of sharing resources by the fact that a large part of the requests which were intended to instances that have become affected by degradation are still stored at the load balancer and can be allocated to non-affected instances or to new instances which will be created. A performance study using the simulator CloudSim showed the gain that this approach can generate, compared to classical approaches of load balancing.**

Keywords*—*Cloud Computing; QoS; Load Balancing; Instance;

I. Introduction

Cloud architectures represent one of the most recent developments in computer systems [1]. This is a new model of computer systems, where data and computation are processed and kept somewhere in a computer network, which constitutes the clouds. The latter refers to a set of data centers owned and maintained by third parties. They are very dynamic systems accessible on the Internet and are characterized by a very large computing and storage capacity. From an architectural point of view, they look like a new type of mainframe available on the Internet, but they are distinguished primarily by the fact that their capacities are not limited as in the case of a mainframe, but they give the illusion of the existence of an infinity of resources through mechanism scalability and infrastructures distributed across the globe. They are also distinguished by the diversity of services and resources provided, ranging from simple computing power to the provision of an entire IT infrastructure.

The Clouds represent a technology that leverages the advancement of information technology and communication. They constitute the evolution and exploitation of parallel and distributed systems capabilities such as Clusters and Grids [2]. They are the result of the evolution of a wide range of technologies [3] : the technologies related to hardware architectures, virtualization technologies, the technologies for distributed architectures and also the technologies of the Internet. They are considered by some researchers as a step towards the realization of the idea of having the computing power as a public service in the same way as water, electricity and telephone are. The idea of utility computing is to provide computing and/or storage according to the needs of users who are not concerned with the location where their data will be stored or their applications will be executed. From an economic standpoint, the Clouds allow to pay only the resources that are really consumed unlike the Grids.

There is no single definition for Clouds, but most definitions refer to the provision of material resources, software systems, and applications as a service. Buyaa et al. in [4] propose the following definition :

> *"A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers."*

There are three types of Clouds: (1) **Public Cloud**, which is a computing architecture available to the general public over the Internet; (2) **Private Cloud** is also a computing architecture that provides hosted services to a limited number of people behind a firewall; (3) **Hybrid Cloud** is a composition of at least one private Cloud and at least one public Cloud. We have also three main models of services that can be provided by a cloud: Software as a Service **(SaaS)** is one that provides applications running on infrastructure of the Cloud. An example of this model is Google Apps. Platform as a Service **(PaaS)** is a service that provides a computing platform hosted on infrastructure of the Cloud. The most famous example of PaaS is Google App Engine. Finally, Infrastructure as a Service **(IaaS)** provides both computing resources and virtual machines. Amazon is one of the main actors of IaaS Cloud.

Despite the many promises of Clouds systems, their adoption in the business world is still hampered by some difficulties. One of these difficulties that interest researchers, users and providers of Clouds is the QoS management problem in the Clouds. The QoS management in Clouds provides gains for different actors of Clouds. For users, the gain is to ensure the quality of the execution of their tasks in Clouds infrastructure,

978-1-4799-2830-9/14 $31.00 © 2014 IEEE
DOI 10.1109/CLOUDCOM-ASIA.2013.69

which are often shared on a large scale. From the point of view of infrastructure providers, the gain is to remain competitive by offering the guarantee of a certain QoS, but also by optimizing the use of resources by allocating only those which are necessary for a given user.

Indeed, in current Clouds, users are charged on the basis of the resources used or reserved and no guarantee is given about the QoS that these resources will provide at runtime. Also, in the case of contracts of SLA (for Service Level Agreements), non-violation of constraints is often provided through a resource over-provision policy which is determined for the worst case [5], where a double deficit is caused : an inefficient use of infrastructure resources, and increased costs billed to customers.

In reality, the QoS management is at the same time confronted to the challenges of QoS managing of centralized and distributed systems but they are distinguished by the use of virtualization that involves the implicit sharing of physical resources and that have more accentuated these challenges. The main challenges for the management or QoS guarantee in Clouds are [6] :

1) The estimation errors at the time of the design for the physical or virtual quantities of resources which are required to achieve a given level of application quality. Indeed, with many parameters (CPU time, memory resources, I/O) that are used during the execution of an application, it is difficult, or even impossible, to accurately predict the optimal combination of resources to accomplish the execution with respect to a quality constraint such as the response time, for example.
2) The highly dynamic nature of these infrastructures where the advent of a certain unexpected behavior cannot be excluded, such as for example the failure of a server or the loss of a network connection;
3) The highly variable loads that characterize the applications of these systems (social networks, web hosting, etc.) make it impossible to maintain a static allocation of resources and require the implementation of an adaptation policy of these allocations in order to maintain an acceptable level of performance.
4) The phenomenon of performance interference, which results from sharing some physical resources of the Cloud infrastructure and causes interference between the virtual machines that share the same physical servers. In fact, virtualization, which is a main technical base for the Clouds, offers a number of advantages, such as the coexistence of multiple systems on a single physical machine, and isolation of failures within each virtual machine ensuring the protection of other machines running on the same server. However, virtualization technology does not guarantee the isolation of performance [7], which means that the performance of a virtual machine application may change due to the existence of other virtual machines on the same physical server.

In this dynamic and distributed environment, one solution of QoS management must allow for efficient use of infrastructure resources, relying on the one hand, on effective mechanisms of adaptation of resource allocation, and on the other hand, on a predictive model to calculate at any time the amounts of resources that are to be reserved for responding to changes in the load intensity of existing services and applications or to ensure the QoS of new services deployed.

The other sections of the paper are organized as follows : In Section 2, we summarize some related works. In Section 3 we present the proposed load balancing approach. In Section 4, we discuss the simulation of our proposed approach and summarize the results. Section 5 is the conclusion of this paper.

II. Related Works

The QoS management of applications and services in the Clouds are recently been of particular interest to researchers, as evidenced by the increasing number of papers on the subject [6], [8]–[14]. In this section, we briefly present the most similar approaches to our proposal.

Li et al. [10] suggested a method for optimizing resources at runtime by using performance models in the development and deployment of applications of Clouds. Their approach is based on a queue network model, called LQN (for Layered Queueing Network performance model) where, for each new deployed application, a LQN model must be generated and maintained for predicting the effects of changes in resource allocations. However, this method may not be transparent to existing applications due to the fact that it is based on measurements that must be made in advance by the developer for the generation of LQN models.

Nathuji et al. [11] have demonstrated that the performance of an application running in a Cloud environment, can be significantly affected by the presence of other virtual machines sharing the same physical server. They present a framework "Q-Cloud" which must ensure that the performances experienced by applications are the same as those carried out in an environment without interference performance. Their approach is to maintain a number of free resources (called "head-room") on each physical server to compensate its affected virtual machines on the basis of a MIMO model (Multi-Input Multi-Output) capturing the relationship between resource allocation and QoS experienced by the virtual machines. This approach, unlike ours, is not independent of the placement policy of virtual machines at the level of physical servers in the cloud infrastructure.

Calheiros et al. [6] proposed a provisioning technique based on an admission controller of user requests. That controller can accept or deny requests, depending on the number of requests already submitted at the instances of each application. Their technique is also based on a Workload analyzer to calculate the number of suitable instances for the incoming flow of requests. Their proposal is more similar to ours except the fact that with our approach the capacity of instances are best used especially in case of performance degradation where instances unaffected by these degradations can be used to execute requests that were previously intended for affected or failing instances. They also ignored the fact that the Clouds are generally remote systems and that the latency must be taken into account in performance calculation, especially with the fact that the virtual machines in the Clouds can migrate from one datacenter to another without users notification.

III. Proposed Approach

In our proposed approach, we place ourselves in the case of an Application Service Provider (ASP) who wants to enjoy the capabilities of IaaS Clouds type for the execution of its applications, while providing a level of QoS to its users and at the same time optimizing resource costs that are billed by infrastructure providers of Clouds. However, the current Clouds do not offer performance guarantees, but they have large capacities and can provide at any time any additional resources that are deemed necessary for the hosted applications in order to achieve some degree of performance. In this case, it belongs to the ASP to manage and adapt at runtime the resource reservation as are necessary to ensure their desired performance.

This choice has the advantage of being practical in the sense that it corresponds to the reality of Clouds and particularly to public Clouds which are black boxes and their users have no access to what happens either inside these systems or on their configurations. This choice also allows the abstraction of internal technology of Clouds in favour of one independent solution that is portable for all types of Clouds. However, it should be noted that the QoS cannot be effectively guaranteed without the use of a placement policy of virtual machines capable of limiting the level of degradation that can be caused by the sharing of physical resources.

The main idea of our proposal comes from the fact that the sharing of physical resources in the Clouds can affect the performance of virtual machines and thus affect the QoS of the applications and services running on them. On the basis of this observation, we believe it is necessary to mitigate the effects of these degradations in order to guarantee QoS in the Clouds. For this reason we propose a load balancer equipped with a queue allowing to temporarily keep some of the accepted requests before they are dispatched subsequently on the fastest instances (see Figure 1). This way of doing aims to limit the number of requests that can be effectively stored at a given instance to a minimum. However, this minimum of requests that can be effectively stored must be chosen with the aim of ensuring an acceptable rate of use of virtual machines. Limiting the number of requests that can be effectively stored in the queue of one instance has the advantage of :

- Increasing the chance that the incoming requests will be executed by the fastest instances and therefore to best exploit their capabilities.
- Mitigating the effects of any degradation by the fact that a large part of the requests which were intended to instances that have become affected by degradation are still stored at the load balancer and can be allocated to non-affected instances or to new instances which will be created.

We propose a distributed architecture consisting of two major components : a Load Balancer and a Response Dispatcher. The behavior of the system can be described as follows : First of all user requests are received by the Load Balancer which decides for each new request: its acceptance or its rejection. In case it accepts it, it decides whether it is going to store it in its queue or send it directly to an instance. Inturn, the responses are received by the Response Dispatcher which forwards them

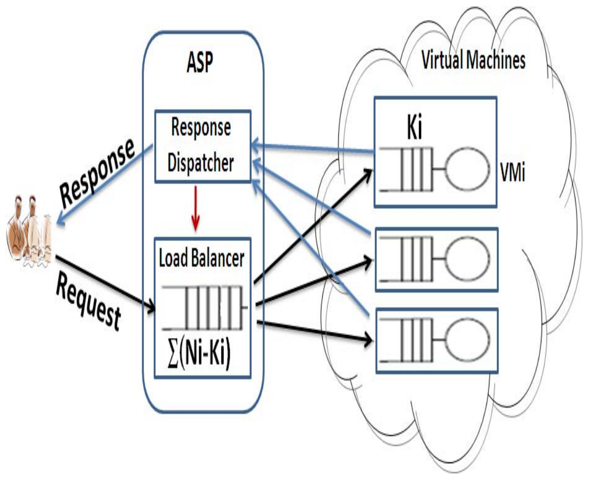

Fig. 1. Components of the proposed architecture

to users. Response Dispatcher monitors the service time and also the propagation time to the instances and informs the Load Balancer in case of change of these values. Meanwhile, the Load Balancer monitors the intensity of incoming requests and can create new instances if the arrival rate increases as it can remove some other otherwise. However, determining the number of instances to create or to remove depending on the traffic intensity of user requests will be studied in a future work and its out of the scope of this paper.

A. The Load Balancer

As mentioned above, the Load Balancer intercepts user requests and rejects those which their QoS may not be guaranteed. To accept or reject a request, the dispatcher maintains for each instance i the maximum number of requests N_i that this instance can take using the following equation :

$$N_i = \frac{T_{Qos}}{2 * TP_i + TS_i}$$

Where T_{Qos} is the negotiated response time and TP_i is the propagation time to the instance i and TS_i is the average service time of the instance i.

Upon receiving a request, the Load Balancer checks if there is still an instance that can take it, otherwise the request is rejected. In other words, if the number of requests that are in the system (the Load Balancer and instances) is smaller than that of N ($N = \sum N_i$), then the request is accepted, otherwise it is rejected. For an accepted request, the Load Balancer decides whether to store it in its queue or to send it directly to an instance. An accepted request is automatically stored on the Load Balancer if its queue is not empty. If the queue of the Load Balancer is empty, the query will be stored if no instances can take it immediately and in this case it is placed first in the queue of the Load Balancer managed with the First-Come-First-Serve policy (FCFS).

To check if there is an instance that can immediately take a new request that happens, the Load Balancer is based on a K_i indicator which is recalculated for each instance i according to its delay and its average service time. This indicator K_i must be fewer in number or equal to N_i and means, for each instance i, the maximum number of requests that can be submitted and effectively stored in its queue. In other words, K_i represents the length of the queue of instance i. If the number of requests that are already present in an instance i is fewer than K_i then that instance can still store another request, otherwise the queue of Load Balancer may be used especially if no other instance is available for that task. In other words, for each instance i, there are K_i requests that can be immediately stored on the instance and $(N_i - K_i)$ requests that can be stored on the Load balancer. The K_i indicator that we propose is obtained by the following equation:

$$K_i = \begin{cases} \frac{2*TP_i}{TS_i} + 1 & if \quad \frac{2*TP_i}{TS_i} + 1 < N_i \\ N_i & if \quad \frac{2*TP_i}{TS_i} + 1 >= N_i \end{cases}$$

This indicator K_i gives the minimum number of requests needed to maximize the use rate of virtual machines. To determine the appropriate value of K_i, we considered the special strategy, in which a single request is sent to a virtual machine. Upon receiving response from this virtual machine a new request is sent to the same virtual machine (see Figure 2). We found that in this special strategy the virtual machine is free for a time T where :

$$T \approx 2 * TP_i$$

From this observation, we found that to maximize the utilization rate of virtual machines, it is sufficient to just send on each instance i, K_i requests and when receiving a response from an instance, zero, one or more of requests stored on the Load Balancer can be sent to this instance. Indeed, upon receipt of each response, the indicators K_i and N_i are updated to reflect any change in the service time or propagation time.

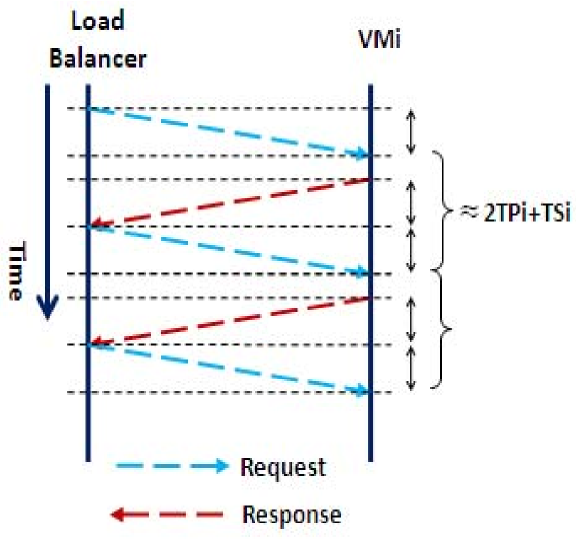

Fig. 2. Special strategy : single request on each instance

B. The Response Dispatcher

The Response Dispatcher is the second component of our architecture and its function is to monitor the performance of different instances used. Also as its name suggests, it is the component that receives the responses from intances before they are transmitted to users. Upon receipt of a response, it recalculates the N_i and K_i indicators before they are sent to the Load Balancer. It computes also the number of requests KR_i from the local queue of the Load Balancer that are to be sent against each response received from one instance i with the following formula:

$$KR_i = \begin{cases} K_i - (OLDK_i + 1) & if \quad OLDK_i <= K_i \\ 0 & if \quad OLDK_i > K_i \end{cases}$$

Where $OLDK_i$ is the old value of K_i

IV. Simulation

To investigate the effectiveness of our proposal, we have used the simulator CloudSim [15] known as the reference tool for the simulation of the cloud environment. It is an extensible simulation framework that enables modeling and simulation of Cloud computing systems and their management policies.The general architecture of a CloudSim model consists of one or more brokers and one or more datacenters. A Broker represents a client and can simulate the behavior of one or more users. Datacenters represent the internal infrastructure of simulated Cloud. The connection between a Broker and a Datacenter or between Dataceners can be defined with a connection speed and delay.

A. Simulation Set Up

The CloudSim model that we have used is composed of a data center with 50 physical servers each of type Quad-Core of 1000 MIPS (Millions of Instructions Per Second) with 16GB of RAM. The virtual machines used are each of type One-Core with 1000 MIPs and 2GB of RAM. These virtual machines are distributed onto physical servers following a simple allocation policy that chooses, as the host for a virtual machine, the host with less Processing Elements in use. This is the default policy of CloudSim for the allocation of VMs to physical servers and its choice was dictated by the fact that the placement of virtual machines is not the purpose of this work. Also, the allocation of processing cores to VMs is done on the basis of time-shared policies, which means that at a given moment a core can be shared between multiple virtual machines if the load requires that. This choice is also motivated by the fact that we want to simulate and see the effects of interference from sharing physical resources of studied approaches.

In our experiments, we considered several incoming flows of user requests, and for each flow we first studied the case without resource sharing where our virtual machines are running only in the data center. Then, we studied the case with resource sharing where other users share with us the physical resources of the data center. We simulated in particular cores shared between virtual machines through time-shared policies mode. As first flow of incoming requests, we studied the flow shown in Figure 3 and which has been studied in the paper of

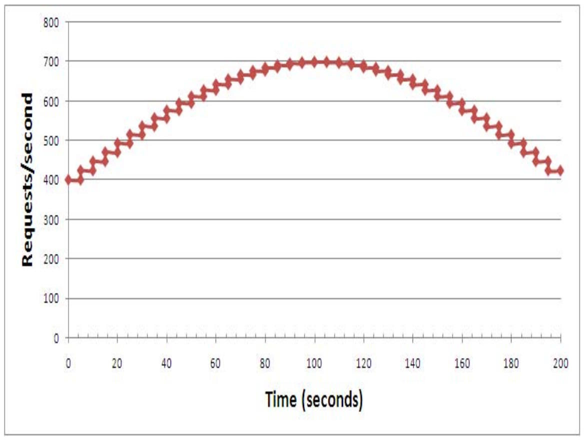

Fig. 3. Arrival rate of requests per second (regular flow)

Calheiros et al. [6]. There is a regular flow whose arrival rate r is computed by the following equation:

$$r = R_{min} + (R_{max} - R_{min}) * \sin(\frac{\pi t}{T_{sim}})$$

Where R_{min} and R_{max} are respectively, the minimum and maximum number of requests of the simulation interval T_{sim}. This equation is evaluated every 5 seconds to vary the arrival rate r. In our experiments, we took $R_{min} = 400rq$ and $R_{max} = 700rq$ and the simulation time is $T_{sim} = 200s$. Second, we considered the flow shown in Figure 4 whose intensity varies randomly every 5 seconds. For this flow, the arrival rate r is chosen randomly between 400 and 700 requests/s. We also considered independent requests which have an execution time that varies randomly between 100ms and 110ms. The connection between the Brokers and Datacenters is defined with a debit of 10MB and a propagation delay of 60ms. In all experiments the negotiated response time is $T_{Qos} = 1s$.

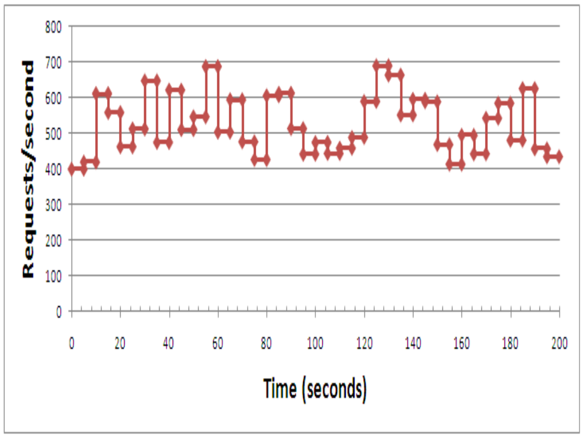

Fig. 4. Arrival rate of requests per second (random flow)

By way of comparison, we implemented the load balancing approach of Caheiros [6], with the difference that we considered the propagation delay in calculating the number of requests to accept. For this approach, all accepted requests are passed directly to virtual machines and no request is stored at the Load Balancer. The QoS criteria of comparison that we have considered are the rate of rejected requests and the average response time.

In our simulations we studied, with each flow of requests two configurations: The first configuration with 50 VMs and the second with 100 VMs. The simulation scenario for each flow and for each configuration starts first by an execution without sharing, where our Broker runs only in the infrastructure. In figures 5 to 12, this case is indicated by the label "*Without other clients*". Then, another execution with shared resources, but without sharing of cores, i.e., the number of all virtual machines is equal to the number of cores; In figures 5 to 12, this case is indicated by the label "*With other clients*". Finally, we considered the sharing of cores, through the creation of a number of virtual machines greater than the number of cores of all physical machines. In this case some of our virtual machines undergo performance degradation due to sharing cores with those of other clients. The simulated degradation here is the fact that a quad-core physical machine hosts five virtual machines instead of four. In figures 5 to 12, these cases are indicated by a label that indicates the number of virtual machines affected by the sharing cores "*N VM affected*" where N takes 5,10,20,30,40 and 50 for the first configuration and 10,20,40,60,80 and 100 VM in the second configuration.

B. *Results*

All results, which are presented in Tables I and II, show a high scalability of our approach. Our proposal was better in all cases without resource sharing in terms of rate of rejected requests and also in terms of average response time and that with the two used flows. With the regular flow, the results are shown in Figures 5 - 8. The case without sharing is reported with the wording "*Without other clients*". On these Figures, we can see that the approach of Calheiros has rejected 36% of requests received, while our approach has rejected only 23% (See Figure 5), and that with the same flow and the same number of virtual machines (50 MVs). With 100 MVs, that observation is also confirmed with a rejected rate of 4% against 0% for our approach (See Figure 7). With the random flow, the results are illustrated in Figures 9 - 12. With this latter flow, the configuration with 50 MVs gave as rejected rate 30% to Calheiros approach against 14% for our proposal (See Figure 9) and with the 100VMs configuration there was a rate of 1% for the Calheiros approach against 0% for our approach (See Figure 11).

In the case with resource sharing, our proposal has proved better, even with a large number of virtual machines affected by degradations relative to the total number of the virtual machines used. For example, in the case with 50 VMs, our approach remains the best, even with 30 MVs affected among the 50 MVs used. Similarly, with 100 VMs, our approach remains the best with 80 MVs affected among the 100 MVs used with the regular flow and also remains the best with 60 MVs affected in the case with the random flow. A gap of 4% to 16% of rejected requests was recorded in favour of our approach compared to the approach of Calheiros in the case of 50 MVs. Also, in the case of 100 MVs, a gap ranging from 1% to 11% of rejected requests has been realized in favor of our approach.

We can also see that our approach has achieved a rate of rejection in some cases with sharing of resources, smaller than the rate of rejection of Calheiros' approach, in the case without sharing. For example, with the first input flow (regular flow) and with 50 MVs, Calheiros' approach has rejected more than 36% of requests received, in the case without sharing, while our rejected rate is less than 36% in the case of sharing with 5 MVs affected. Also, with the second flow and 50 MVs, Calheiros has rejected, in the case without sharing, more than 30.5% of received requests, while our rate of rejection is less than of 30.5% in the case of sharing and with 5 MVs affected.

The same gain is more visible with the results of 100 MVs where we can see that the rejection rate of our approach in the case of sharing and with 40 MVs affected (0.1% with the regular flow and 0.2% with the random flow) is smaller than the rate of rejection of Calheiros in the case without sharing (4% with the regular flow and 1% with the random flow).

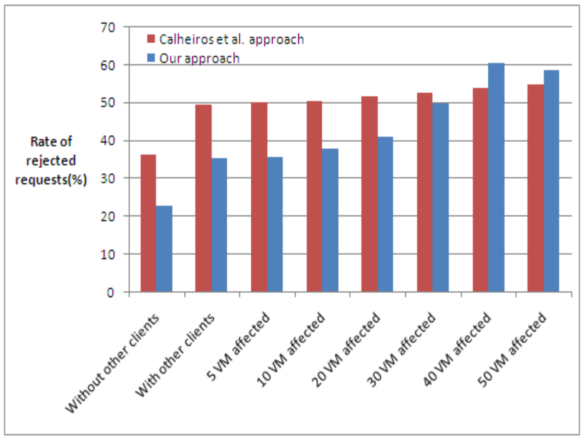

Fig. 5. Rate of rejected requests (regular flow with 50 VMs)

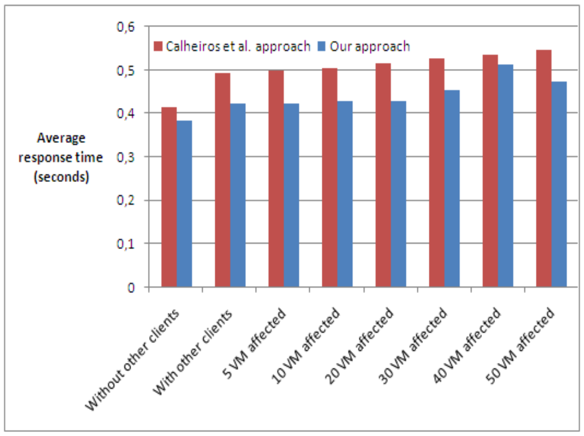

Fig. 6. Average response time (regular flow with 50 VMs)

By cons, where the vast majority of virtual machines are affected by degradation, the Calheiros' strategy proved more effective, however such cases must be very rare with the use of placement policies of virtual machines in the Clouds.

V. Conclusion

In this paper, we proposed a new load balancing policy that takes into account the interference of performance, which may affect the QoS of virtual machines in the Clouds. Our proposal was more scalable and could significantly reduce the rate of rejected requests in both cases: with and without resource sharing.

We have shown that it is possible to reduce the rate of rejected requests through optimal use of capacities of virtual machines and we were able to mitigate the degradations of some virtual machines using other non-affected virtual machines without creating an additional burden.

However, we intend to complete this work by the introduction of a policy to predict the evolution of the intensity of flow of requests to dynamically obtain the appropriate number of instances to create or maintain to meet the desired QoS.

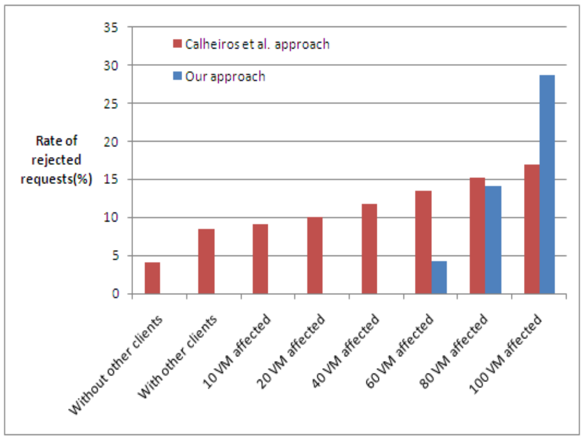

Fig. 7. Rate of rejected requests (regular flow with 100 VMs)

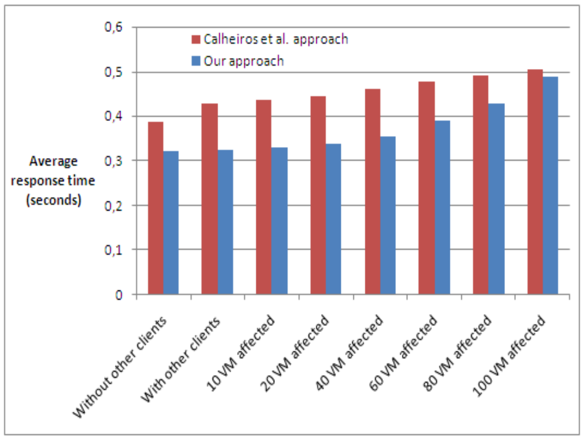

Fig. 8. Average response time (regular flow with 100 VMs)

TABLE I. Results of the configuration with 50 MVs

	First configuration with 50 MVs							
	With regular flow				With random flow			
	Rate of rejected requests(%)		Average response time(s)		Rate of rejected requests(%)		Average response time(s)	
	Calheiros et al.	Our appr.	Calheiros et al.	Our appr.	Calheiros et al.	Our appr.	Calheiros et al.	Our appr.
Without other clients	36.38023087	23.04969705	0.414686277	0.383954839	30.52908630	14.25401522	0.413871682	0.365914836
With other clients	49.55860674	35.36702438	0.494716391	0.422682357	43.23516982	27.89241114	0.491096770	0.412797386
5 VM affected	50.14398848	35.80330511	0.500041940	0.422931257	43.85757646	30.17867435	0.495733101	0.424767758
10 VM affected	50.65051939	38.05842219	0.504730973	0.430043614	44.52896880	30.87415946	0.500986983	0.426256189
20 VM affected	51.79385649	41.14495834	0.515878981	0.428933452	45.76033502	34.63688761	0.509631907	0.431173803
30 VM affected	52.80606123	50.01885693	0.526049184	0.453064016	46.97537268	42.54755043	0.516578800	0.462713728
40 VM affected	53.83883575	60.69616347	0.536385729	0.513979239	48.17792377	54.84630163	0.522670341	0.533942976
50 VM affected	54.91017861	58.82332773	0.547408134	0.473514024	49.41217151	54.57348703	0.526906162	0.507099359

TABLE II. Results of the configuration with 100 MVs

	Second configuration with 100 MVs							
	With regular flow				With random flow			
	Rate of rejected requests(%)		Average response time(s)		Rate of rejected requests(%)		Average response time(s)	
	Calheiros et al.	Our appr.	Calheiros et al.	Our appr.	Calheiros et al.	Our appr.	Calheiros et al.	Our appr.
Without other clients	4.067291129	0.055704577	0.387953494	0.321167767	1.137324214	0.107584723	0.357782231	0.321221182
With other clients	8.521086325	0.095983271	0.426892019	0.325050584	2.528046719	0.049946212	0.378495964	0.330125478
10 VM affected	9.148405564	0.033422746	0.436156605	0.331177693	3.010219763	0.089326879	0.386489549	0.340561438
20 VM affected	10.04224978	0.035136733	0.444655287	0.338026920	3.532733979	0.172890733	0.392412819	0.346527213
40 VM affected	11.79565847	0.101125232	0.460891726	0.353294316	4.470185954	0.203626863	0.405603688	0.358656277
60 VM affected	13.51564442	4.190698193	0.476101863	0.389089878	5.534424466	2.200514830	0.418869920	0.400364617
80 VM affected	15.21677650	14.20123921	0.490738204	0.428358087	6.613070539	9.272706316	0.433567487	0.457040143
100 VM affected	16.96675722	28.69128523	0.505174113	0.487234262	7.815621638	22.97621792	0.449540143	0.502292300

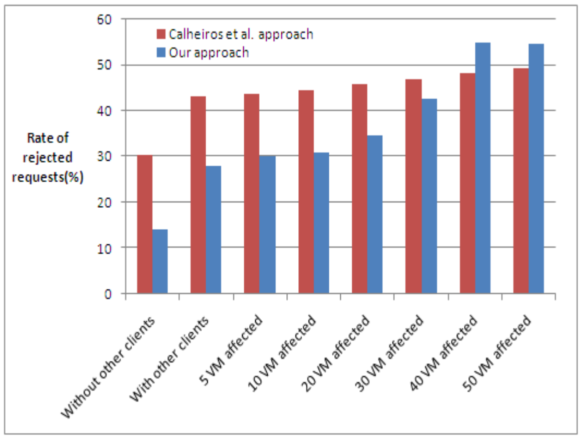

Fig. 9. Rate of rejected requests (random flow with 50 VMs)

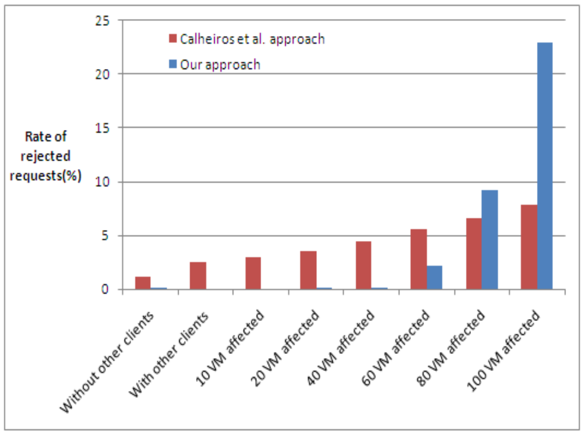

Fig. 11. Rate of rejected requests (random flow with 100 VMs)

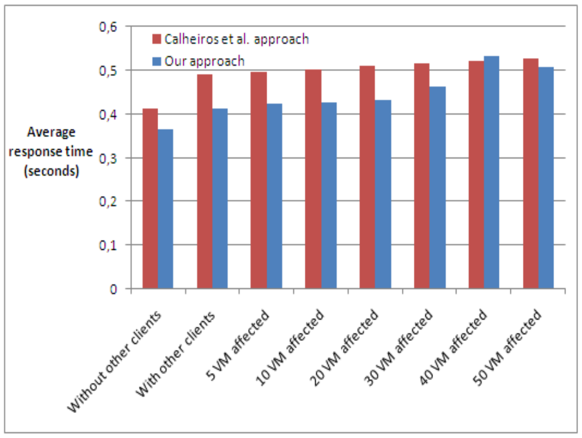

Fig. 10. Average response time (random flow with 50 VMs)

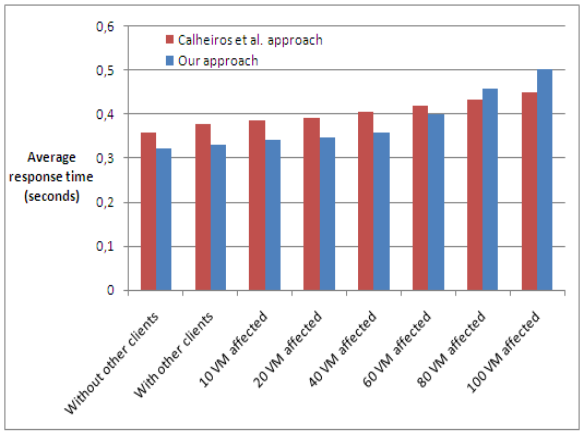

Fig. 12. Average response time (random flow with 100 VMs)

References

[1] A. E. Borko Furht, *Handbook of Cloud Computing*. Springer, 2010.

[2] I. T. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," *CoRR*, vol. abs/0901.0131, 2009. [Online]. Available: http://arxiv.org/abs/0901.0131

[3] R. Buyya, J. Broberg, and A. M. Goscinski, *Cloud Computing Principles and Paradigms*. Wiley Publishing, 2011.

[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Comp. Syst*, vol. 25, no. 6, pp. 599–616, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.future.2008.12.001

[5] S. Ferretti, V. Ghini, F. Panzieri, M. Pellegrini, and E. Turrini, "Qos-aware clouds," in *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, ser. CLOUD '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 321–328. [Online]. Available: http://dx.doi.org/10.1109/CLOUD.2010.17

[6] R. N. Calheiros, R. Ranjan, and R. Buyya, "Virtual machine provisioning based on analytical performance and QoS in cloud computing environments," in *ICPP*, G. R. Gao and Y.-C. Tseng, Eds. IEEE, 2011, pp. 295–304. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6046213

[7] Y. Koh, R. C. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *ISPASS*. IEEE Computer Society, 2007, pp. 200–209. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ISPASS.2007.363750

[8] T. Cucinotta, D. Giani, D. Faggioli, and F. Checconi, "Providing performance guarantees to virtual machines using real-time scheduling," in *Euro-Par Workshops*, ser. Lecture Notes in Computer Science, M. R. Guarracino, F. Vivien, J. L. Träff, M. Cannataro, M. Danelutto, A. Hast, F. Perla, A. Knüpfer, B. D. Martino, and M. Alexander, Eds., vol. 6586. Springer, 2010, pp. 657–664. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-21878-1

[9] V. C. Emeakaroha, I. Brandic, M. Maurer, and I. Breskovic, "SLA-aware application deployment and resource allocation in clouds," in *COMPSAC Workshops*. IEEE Computer Society, 2011, pp. 298–303. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6032032

[10] J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai, "Performance model driven qos guarantees and optimization in clouds," in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, ser. CLOUD '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 15–22. [Online]. Available: http://dx.doi.org/10.1109/CLOUD.2009.5071528

[11] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: managing performance interference effects for QoS-aware clouds," in *EuroSys*, C. Morin and G. Muller, Eds. ACM, 2010, pp. 237–250. [Online]. Available: http://doi.acm.org/10.1145/1755913.1755938

[12] H. N. Van, F. D. Tran, and J.-M. Menaud, "SLA-aware virtual resource management for cloud infrastructures," in *CIT (1)*. IEEE Computer Society, 2009, pp. 357–362. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/CIT.2009.109

[13] Z. Wang, X. Tang, and X. Luo, "Policy-based SLA-aware cloud service provision framework," in *SKG*. IEEE, 2011, pp. 114–121. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6088022

[14] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," in *HPDC*, S. Hariri and K. Keahey, Eds. ACM, 2010, pp. 304–307. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851516

[15] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw, Pract. Exper*, vol. 41, no. 1, pp. 23–50, 2011. [Online]. Available: http://dx.doi.org/10.1002/spe.995