A QoS App-SLO Manager for Virtualized Infrastructure

Fernando Rodríguez-Haro, Felix Freitag, and Leandro Navarro

***Abstract*—The management of infrastructure for supporting Cloud Computing presents the challenge of automated service provisioning, which addresses the problem of mapping high-level requirements expressed in end-user terms to low-level resources such as CPU, memory, and network bandwidth. Current infrastructure is supported through virtualization via hypervisors. In this paper, we describe the formal specification of a high-level component for enhancing hypervisors. With this component, applications running in a Virtual Machine can receive a Quality of Service defined by Service Level Objectives. The manager is aware of the application's needs and requests the CPU resources through the lifetime of the Virtual Machine. The implementation of our proposal achieves to manage computing-oriented and net-oriented applications.**

***Index Terms*—Hypervisor, QoS, SLO.**

I. Introduction

NOWADAYS, virtualization infrastructure is a common solution for supporting Cloud Computing, Grid, and High Performance Computing. An important challenge in these infrastructures is the automated service provisioning of Virtual Machine (VM) based resource providers for the execution of applications. When we review the literature [1], some interesting questions arise from end users willing to deploy applications in VM-based resource providers.

– How can we predict (or have some degree of certainty) that deadline execution time requirements for a given job will be met?
– How can we compute the amount of resources that are needed to increase (or decrease) the number of transactions to a certain required level?
– Moreover, how can we provide the needed resources and at the same time minimize the degradation of the externally-perceived response times?

In this paper, we describe the formal specification of a high-level component for enhancing hypervisors. The component is named QoS App-SLO manager and allows end users to express the application's requirements in terms of a Service Level Objective (SLO).

With our component, applications running in a virtual machine can receive a Quality of Service (QoS) defined by two types of SLOs (slo_2, slo_3).

Manuscript received on September 19, 2012; accepted for publication on December 10, 2012

Felix Freitag and Leandro Navarro are with the Department of Computer Architecture (DAC) at the Universitat Politècnica de Catalunya (UPC), Jordi Girona 1-3, 08034 Barcelona, Spain, and Fernando Rodríguez-Haro is with the University of Colima (e-mail: felix, leandro, frodrigu@ac.upc.edu)

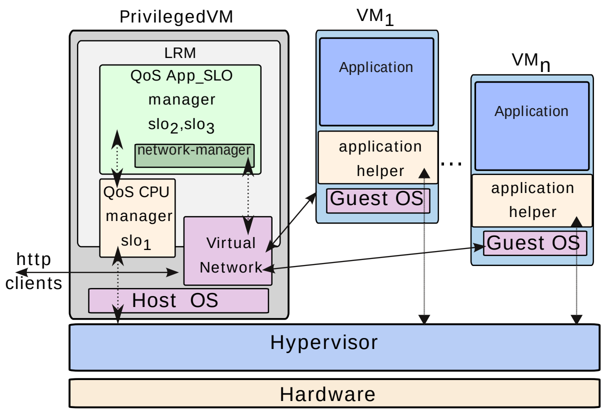


Fig. 1: Interactions of the QoS App-SLO manager in a VM-based resource provider

The manager is aware of the application's needs and requests the CPU resources through the lifetime of the Virtual Machine. The implementation of our proposal achieves to manage computing-oriented and net-oriented applications, i.e. it meets at least the agreed application requirements, and provides self-management for external modifications in the application's SLO (e.g. a user requests more transactions per secod (tps)).

The proposal relies on the services offered by the hypervisor, the host OS, and the low-level component QoS CPU manager [2] which manages the SLO slo_1.

II. QoS App-SLO manager

In this section, we describe a QoS App-SLO manager. Figure 1 shows the interactions of this manager in a VM-based resource provider. It allows handling the high-level requirements of user's applications, i.e. it runs in the scope of the privilegedVM and manages the resources needed by each guest VM.

Additionaly, the definitions of the proposal are build upon two systems to acquire knowledge about information of each application:

– An inter-VM messaging system, which is a communication system that allows writing and reading information in both ways: privilegedVM from/to guest virtual machines.
– An application-helper which runs in the guest virtual machine with an independent period τ_{ah}. It measures

the metric *tps* transactions per second and writes the measured metrics through the inter-VM messaging system.

A. Characteristics

The formal specification that we propose has the following features: Application rate (app-rate) guarantees for granting a requirement expressed in transactions per second tps, user-perceived service guarantees for providing a SLO in (user-perceived) response times, net-rate guarantees for granting a requirement expressed in requests per second rps, and a request admission control for applying a policy in the incoming requests to meet a SLO.

B. Definitions

Definition 2.1: Let ϱ be the set of online guest virtual machines identifiers and let $Apptype_{vm}$ be the sequence indexed by ϱ that defines the type of application that is running in the guest virtual machine so that

$$Apptype_{vm} \in \{\text{computing-oriented}, \text{net-oriented}\} \forall vm \in \varrho.$$

Definition 2.2: Let $slotype_{vm}$ be a set that has the type of resquested SLOs slo_2 or slo_3 for a given vm. Let slo_2 be a service level objective of the form ”ensure that the application running in a guest virtual machine vm will be able to achieve an application rate of at least 95% of the target tps or rps”. Let slo_3 be a service level objective of the form ”ensure that the application running in a guest virtual machine vm will be able to achieve a % of the served requests with response times below a given threshold expressed in seconds”. Let $appslo$ be the set of parameters of the requested service level objectives for all managed virtual machines

Inspecting the throughput that was achieved by each computing-oriented application can be done as follows.

Definition 2.3: Let tps_{vm} be the throughput achieved by the computing-oriented application between the time interval Cah_t, Cah_{t-1} and let T be the number of composite transactions measured by the application-helper so that

$$tps_{vm} = \frac{T}{Cah_t - Cah_{t-1}}.$$

C. Network-manager

With this subsystem we aim to measure the burst behaviour of the net-oriented application. Web technologies are based on the transport protocols Hypertext Transfer Protocol (HTTP) and the secure HTTP protocol (HTTPS), both main function is to move data between Web servers and browsers. Despite the fact that it is a stateless protocol it is nowadays the facto transport protocol for technologies based on web services. Common protocols for deployment of web services are SOAP (Simple Object Access Protocol), REST (Representational state transfer), and XML-RPC.

The approach for measuring the throughput of a web application assumes that requests are atomic and represents a unit of work which ends with the successful transfer of the results. As Web servers are the target application, for managing the net-oriented applications we propose the following approach.

Trace the http requests of each VM by inspecting the connection states of the incoming HTTP packets. Depending of the hypervisor [3], [4], tracing could be done in the privilegedVM or the guest VM. Thus, we can measure the response time of each connection and build a distribution array $http$ at every trace period of length $\tau_{http} = Cht_t - Cht_{t-1}$. This array has n elements, each element is a counter of successful (served) requests according to its response time rt. The granularity of n depends of the rt ranges that need to be grouped. We propose a granularity of $n = 9$ which is mapped to the ranges shown in Table I. The metric λ_{vm} is the arrival rate observed in the virtual machine vm for the net-oriented application.

TABLE I: SERVICE TIME GROUPS OF HTTP REQUESTS USED TO BUILD THE DISTRIBUTION ARRAY.

array position	response time range
0	$rt < 1\mu s$
1	$1\mu s \leq rt < 10^{-5}s$
2	$10^{-5}s \leq rt < 10^{-4}s$
3	$10^{-4}s \leq rt < 1ms$
4	$1ms \leq rt < 10^{-2}s$
5	$10^{-2}s \leq rt < 10^{-1}s$
6	$10^{-1}s \leq rt < 1s$
7	$1s \leq rt < 10s$
8	$10s \leq rt < 100s$

In order to measure the metric mean response time of the served requests it makes use of a circular buffer CB with a history length l for each vm so that

Definition 2.4: Let CB_{vm} be the history of response times of the last l served requests for a given vm. Let $meanRT_{vm}$ be the computed mean response time observed for virtual machine vm so that

$$meanRT_{vm} = \frac{\sum_{k=l}^{1} CB_{vm,k}}{l} \forall vm \in \varrho.$$

Using the distribution array $http$ the inspection of throughput achieved by each net-oriented application can be done as follows.

Definition 2.5: Let rps_{vm} be the throughput achieved during the last period τ_{http} and R the number of successful completed requests measured by the *network-manager* so that

$$R = \sum_{k=1}^{n} http_{vm,k},$$

$$rps_{vm} = \frac{R}{Cht_t - Cht_{t-1}}.$$

Finally, the metrics are advertised through the inter-VM messaging system.

D. App-rate metrics collector

It is in charge of getting the observed metrics of the running application. It interacts with the inter virtual machine messaging system and the network manager. Its function is to keep a snapshot of the measured metrics for the learning component.

Definition 2.6: Let TPSapp$_{vm}$ be the set of measured metrics app-rate expressed in requests/transactions per second so that TPSapp$_{vm} = m(vm)$

$$m(x) = \begin{cases} tps_{vm} & Apptype_{vm} = \text{computing-oriented} \\ rps_{vm} & Apptype_{vm} = \text{net-oriented} \end{cases} \quad (1)$$

E. Learning component

Its function is to compute online parameters that profiles the current application rate. We use a multi queue system. Each virtual machine is modeled using Little's law from queueing theory.

By having the CPU consumption of each virtual machine and the application rate we obtain the service demand SD as follows

Definition 2.7: Let $avgmets$ be a set of mean CPU metrics for each virtual machine computed by the QoS CPU manager during the previous controller period. Let SD_{vm} be the mean CPU time spent per transaction/request during the previous period

$$SD_{vm} = \frac{avgmets_{vm}}{TPSapp_{vm}}.$$

In order to obtain the learned service demand we apply a forecasting method. First, we obtain a trend of the pasts service demands by applying exponential moving average (EMA) [5] which technically can be classified as an Auto-Regressive Integrated Moving Average ARIMA(0,1,1) model with no constant term [6]. Second, the method enhances the trend by measuring the volatility of the sampled metrics using a trading mechanism with a configurable parameter ϖ. We propose to apply Bollinger bands [7] in order to capture the burst behavior of the running applications and improve the reactiveness of the QoS CPU manager.

Definition 2.8: Let $SDtrend$ be an Exponential Moving Average function to compute the trend for the service demand of a given vm so that $SDtrend_{vm,t}$ is defined as follows

$$SDtrend_{vm,t} = SD_{vm,t}, t = 0,$$

$$SDtrend_{vm,t} = \alpha{*}SD_{vm,t}{+}(1{-}\alpha){*}SDtrend_{vm,t-1}, t > 0.$$

Definition 2.9: Let N be the length of history needed to forecast the next service demand. Let $SDforecast$ be a forecasting function to compute the next service demand of a given vm so that $SDforecast_{vm,t}$ is defined as follows

$$\sigma_{vm,t} = \sqrt{\frac{1}{N}\sum_{i=t}^{t-N}(SD_{vm,i} - \overline{SD_{vm}})^2},$$

$$\overline{SD_{vm}} = \frac{1}{N}\sum_{i=t}^{t-N} SD_{vm,i},$$

$$SDforecast_{vm,t} = SDtrend_{vm,t}{+}\varpi{*}\sigma_{vm,t} \qquad 0 \le \varpi \le 2.$$

The service demand forecasted is a snapshot of the needed resources for the next controller period. However, we introduce the notion of Number of Rounds To Learn $NRTL$ parameter in order to find out a tradeoff between reactiveness and disturbance. A $NRTL = 1$ means that the $SDforecast_{vm,t}$ will be used at each controller period in order to compute a new CPU requirement for the application, the reactiveness of the learning phase is high but the accuracy of the forecast is affected by the disturbances of so frequent changes in the slo_1 (CPU resource). On the contrary, for $NRTL > 1$ we introduce the notion of learning phase (or window) which helps in the smoothing of the service demand forecasted and also improves the accuracy of computed values.

Definition 2.10: Let NRTL be the length of the controller window needed to learn a smoothed value of $SDforecast$. The length of the period of each NRTL window is given by controller period times. Let NRTL be a counter which decreases at each controller period.

Now we use an approach to find out the burst behaviour of the requests in the net-oriented application. It is proposed to use an array of percentiles $perc$ of $p = n - 1$ elements (see section II-C) for each vm with a net-oriented application. Each element has a circular buffer of length $NRTL$. The position of the element in the array accounts, in the circular buffer, the percentage of requests that were served below the threshold defined in the position of each element (response times) in the array $http$.

Definition 2.11: Let $perc$ be an array of circular buffers. Each circular buffer of length NRTL for each virtual machine vm so that

$$\text{put}\left(\frac{\sum_{i=0}^{j} http_{vm,i}}{rps_{vm}}\right) \text{in}\, perc_{vm,j}, 1 \le j \le n.$$

For instance, $perc_{vm,6}$ is a circular buffer with the percentage of requests that were successful server below 1 second, see table I.

The QoS CPU manager uses β to set the reactiveness of the manager in order to climb and achieve the requested slo_1 for each vm. The default value of the parameter β is set to 0, though can be dynamically configured by the type of application running in the virtual machine, e.g. computing intensive applications have a value of 0. However, net-oriented applications need mechanisms to detect the behavior of bursty applications and configure a properly value of β. Therefore, we use $perc$ to measure the bursty behavior as follows.

Definition 2.12: Let β be the degree of burst behaviour detected in the (served) requests of the traced application running in the virtual machine vm so that

$$burst_{vm} = \sum_{j=1}^{n} \sigma_{vm,j},$$

$$\sigma_{vm,j} = \sqrt{\frac{1}{N} \sum_{i=1}^{NRTL} (perc_{vm,i} - \overline{perc_{vm}})^2},$$

$$\beta_{vm} = b(burst_{vm}) \forall vm \in \varrho \quad \text{iff NRTL} = 0.$$

$$b(x) = \begin{cases} 0.5 & 0.00 \leq x \leq 0.03 \\ 0.6 & 0.03 \leq x \leq 0.06 \\ 0.7 & 0.06 \leq x \leq 0.09 \\ 0.8 & 0.09 \leq x \leq 0.12 \\ 0.9 & 0.12 \leq x \leq 0.15 \\ 1.0 & 0.15 \leq x \end{cases} \tag{2}$$

F. CPU-rate estimator

When a learning phase ends, that is $NRTL = 0$, this component sets the $newQoS$ estimated for the running application. The approach of the process involves asking an increase or decrease in the amount of assigned resources, which in fact is and admission control procedure. Due that higher decisions (such as migration of VMs) are leave to the global resource manager, the CPU-rate estimator works as follows.

– It stores the forecasted service demand.
– It stores the modification of the QoS.
– It computes and stores the percentage of QoS granted with the current state of the CPU resources.
– It uses the notion of premium services, via differentiated service, to prioritize the assignment of the CPU resources.

The CPU-rate estimator computes the needed raw CPU power for all virtual machines with computing-oriented and net-oriented applications defined in the sets tps and rps. The needed raw CPU power is transformed into a SLO of type slo_1 and requested via the QoS CPU manager.

Definition 2.13: Let S be the current state (available resources) of the CPU capacity of the resource provider at time t. Let ϑ be a parameter required by the end user for the SLO which express the degree of tolerable (soft,...,hard) reduction in the requested slo_1. Let ac an admission control mechanism that is defined in the QoS CPU manager as follows:

$$ac(MHzSLA, \vartheta) = \begin{cases} 1 & MHzSLA \leq S_t \bigwedge \vartheta = 1 \\ \frac{S_t}{MHzSLA} & MHzSLA * \vartheta \leq S_t \bigwedge 0.5 \leq \vartheta < 1 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

This function helps to manage the admission control of new virtual machines and online virtual machines that request internal updates of slo_1, i.e. all virtual machines with $ac(MHzSLA, \vartheta) = 0$ can be rejected and the status is informed to the QoS App-SLO manager.

Definition 2.14: Let $\hat{S}_\varrho$ be an array of virtual machines ordered by differentiated service. Let *mhz* be the needed CPU in order to achieve a given application rate (i.e. MHz to serve the target reference $appslo$ or λ) for a given vm. Let $newQoS$ be the CPU that is granted by the QoS CPU manager according to ϑ, if is not specified then $\vartheta = 1$. Let φ be the configurable node capacity, and let Φ be the raw CPU capacity of the resource provider. Let ψ the minimum reservation of φ for the guest virtual machines. Let Ξ be the absolute CPU capacity of the node, i.e. for 4 processors $\varphi = 4 * 100 = 400$. The sets $appslo$ and $slotype$ were defined in Def. 2.2. Now, we define $negotiateslo1$ as the function that sets the raw CPU (in % of φ) for each VM. In other words, an slo_1 is computed so that the running application receives slo_2 or slo_3.

$$newQoS_{vm} = negotiateslo1(mhz(vm), \vartheta) \forall vm \in \hat{S}_\varrho \quad \text{iff NRTL} = 0 \tag{4}$$

$$negotiateslo1(MHzSLA, \vartheta) = max(\frac{MHzSLA * ac(MHzSLA, \vartheta)}{\varphi}, \psi) \tag{5}$$

$$mhz(vm) = \begin{cases} \Phi * min\left(\frac{appslo_{vm} * SDforecast_{vm,t}}{\Xi}, 1.0\right) \\ \quad \text{if } vm \in \text{tps} \bigwedge slotype_{vm} = slo_2 \\ \Phi * min\left(\frac{appslo_{vm} * SDforecast_{vm,t}}{\Xi}, 1.0\right) \\ \quad \text{if } vm \in \text{rps} \bigwedge slotype_{vm} = slo_2 \\ \Phi * min\left(\frac{\lambda_{vm} * SDforecast_{vm,t}}{\Xi}, 1.0\right) \\ \quad \text{if } vm \in \text{rps} \bigwedge slotype_{vm} = slo_3 \end{cases} \tag{6}$$

In the third case of Equation 6, we have introduced the notion of automatic sizing for net-oriented applications which request an slo_3. In this case, the requested slo_3 takes into account the observed arrival rate λ of each traced net-oriented application and it increases or decreases its demand according to the observed behaviour.

G. Net-rate estimator

For net-oriented applications we follow an approach to implement an admission control mechanism. It is applied at the end of a learning phase. Only if the target net-oriented application has a computing-oriented behaviour it is likely, applying queueing theory, to find a relation between the number of requests served and the CPU consumed. However, net-oriented applications have a burst behaviour with different resource consumption patterns. Therefore, we propose to use the user-perceived service as a measure of the quality served by the net-oriented application. Even if it can be seen as a black-box that ignores the inner bottlenecks which can cause a bad perceived service, with this approach we aim to size the resources according to the current configuration of the net-oriented application and scale up the aggregation of VMs. We assume that a load balancer can manage the external requests and distribute them to the online virtual machines, thus by aggregating VMs we increase the number of served requests. However, additional inner optimizations in the configuration of the web application server can be applied out of the band and the effect of this optimizations will be seen as an increase/reduce of the resources assigned. Finally, the $capacity$ is granted according to the differentiated service.

Definition 2.15: Let γ be the CPU resources, expressed in percentage of the full node capacity, that are available for all VMs. Let $capacity_{vm}$ be the granted capacity, expressed in rps, of the virtual machine so that

$$capacity_{vm} = n(vm) \forall vm \in rps.$$

$$n(vm) = \frac{newQoS_{vm} * \gamma}{SDforecast_{vm,t}} \tag{7}$$

Now we compute the admission control parameter for the net packets. We use the metrics of the network manager, i.e. the dynamics of the external arrival rate of the net-oriented application clients and the queue length of the current pending requests for each traced net-flow that was observed during the last controller period.

Definition 2.16: Let $perc_{vm,6}$ be a circular buffer with the percentage of requests that were successful server below 1 second. Let $slo_{vm,target}$ the service level objective requested by the user. Let Δ_{slo} be the adjustment positive/negative in the number of requests per second admitted to reach the virtual machine. Let $reqadmission_{vm}$ be the required admission control parameter that limits the amount of accepted requests.

$$abovecapacity_{vm} = \lambda_{vm} - rps_{vm},$$

$$slo_{vm,level} = \left(\frac{\sum_{i=0}^{NRTL} perc_{vm,6}}{NRTL} \right),$$

$$\Delta_{vm} = (slo_{vm,level} - slo_{vm,target}) * abovecapacity_{vm},$$

$$reqadmission_{vm} = capacity_{vm} + \Delta_{vm}.$$

Next step is to adjust the admission control according to the number of waiting requests in the system.

Definition 2.17: Let QL be an array of circular buffers. Each circular buffer for each virtual machine vm has a length of NRTL elements and it stores the observed queue length of waiting/pending requests inside the system during the last controller period. Let maxQL a sort-term memory value of the maximum queue length observed in the NRTL samples. Let $admission_{vm}$ be the rps that will be admitted in the next learning phase.

$$\text{put}\left(ObservedQL_{vm,t}\right) \text{ in} QL_{vm,t}, \forall vm \in rps,$$

$$admission_{vm} = steadystate(vm) \forall vm \in rps.$$

$$steadystate(vm) = \begin{cases} max\left(reqs_{vm}, reqadmission_{vm} - maxQL\right) \\ \qquad \text{if } \frac{\text{QL}_{vm}}{reqs_{vm}} > 0.05 \\ max\left(reqs_{vm}, reqadmission_{vm} - \frac{maxQL}{2}\right) \\ \qquad otherwise \end{cases} \tag{8}$$

The next step is the application of the network level admission-control through the network manager. By controlling the admission of incoming net packets before arriving the target application we aim to ensure a given level of user-perceived service in the response times. Therefore, with this approach we do not only size the resources according to a given external demand but also according to an expected level in the quality of service.

III. Evaluation

We implement the abovementioned proposal in a Local Resource Manager (LRM) to test the resource management of computing-oriented and net-oriented applications. In the following, we have the characteristics of the QoS App-SLO manager.

– Application-aware. It uses inspection of high-level application metrics in order to learn the CPU needed (slo_1) to achieve app-rate level requirements (tps, rps).
– Service negotiation. It acts on behalf of administrators in order to request the new slo_1 through the QoS CPU manager interfaces. However, it depends on reservations (leases) to grant or revoke the assigned resources as well as policies to detect and limit the misbehavior of virtual machines. If a new slo_1 can not be fully granted then it is informed via a VMstate information system. The VM-state agent is in charge of informing about this issue to external agents so that decisions about the migration of virtual machines can be managed by, for instance, global resource managers.
– Learning. It acquires online knowledge about the consumed resources. It also constructs a CPU profile for the resource consumed by the running application.

We set up three experiments in two physical machines with Fedora Linux and Xen interconnected through a gigabit switch, both with Intel Quad CPU Q6700 with 8GB of memory and a 750GB SATA Disk.

A. Evaluation of computing-oriented applications

The experimental setup test the following festures:

– Create four virtual machines which request, as a bootstrap mechanism, different CPU-rate SLO guarantees.
– Deploy a math-application in all VMs.
– Handle virtual machines which request different app-rate guarantees with fixed differentiated services.

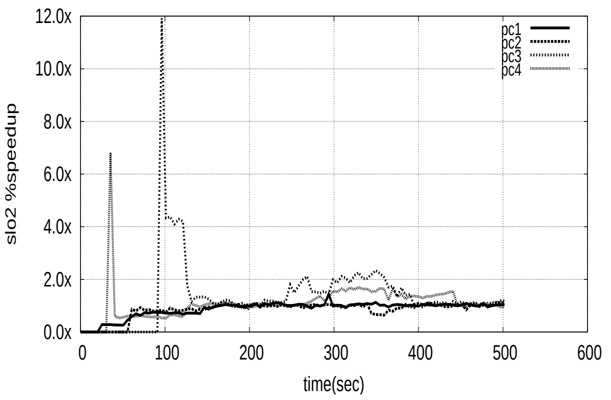


Fig. 2: Application throughput relative to agreed SLO. An user's view of the application performance.

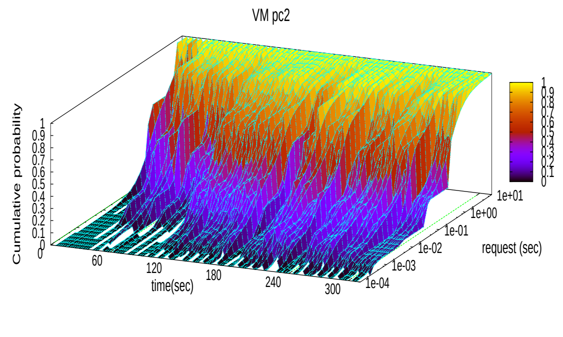
– Provide dynamic management by responding to external agents that change the initial VM's differentiated service (pc3 and pc4).
– Provide dynamic management by responding to external agents that change the initial VM's app-rate.

The agreed app-rate requirement is managed according to each requested parameter. The initial CPU-rate parameter is an initial guess of the needed resources though it can be obtained from previous executions. The learning procedure obtains and requests new CPU-rates which are managed according to their respective differentiated service. Applications can benefit from having hard and soft guarantees about the expected performance (tps). Additionally, each user has a real view of its application throughput. Figure 2 shows results for this experiment. The speedup graph represents the transactions per second relative to its agreed app-rate. From the results for the SLO type 2 (slo_2) we obtained the following mean relative errors: pc1 -0.01, pc2 -0.01, pc3 -0.07, and pc4 -0.03. Therefore, we can observe that each VM achieves its agreed SLO, and additionally we can observe that VMs with premium services receive their corresponding aggregated resources, thus they achieve a better throughput.

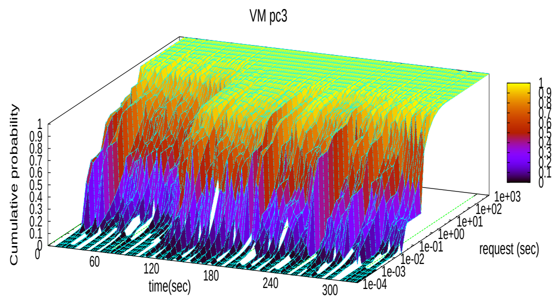
B. Evaluation of net-oriented applications

We setup a web application server that renders 3D images and a web image server in two VMs, and for http benchmarking we use Siege [8]. Figure 3 shows the results of this experiment. In this experiment we see the controller changing the CPU resources of two web-based services which have different resource intensive requirements. Despite that pc3's workload is network intensive with low CPU consumption and pc2's workload is CPU intensive with low network consumption, the controller is able to manage both type of applications. At the same time, the network QoS policy adjusts the acceptance in the number of allowed connections that can reach each VM.

A closer look in Figure 3(b) shows that after time 95s more than 90% of the requests are served by the web image server



(a) Web application server in pc2



(b) Web image server in pc3

Fig. 3: Distribution vector of served requests. Http packets are controlled.

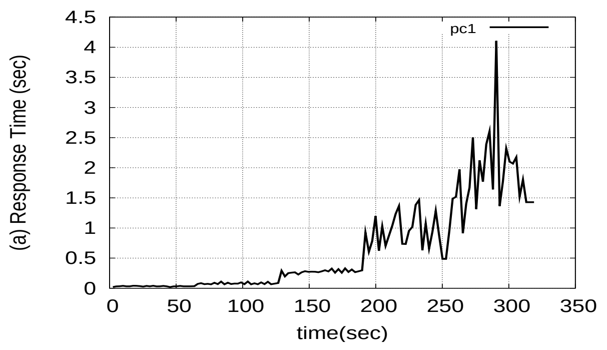
with a response time below one second. The same behaviour is observed for the web application server in Figure 3(a).

C. Automatic sizing

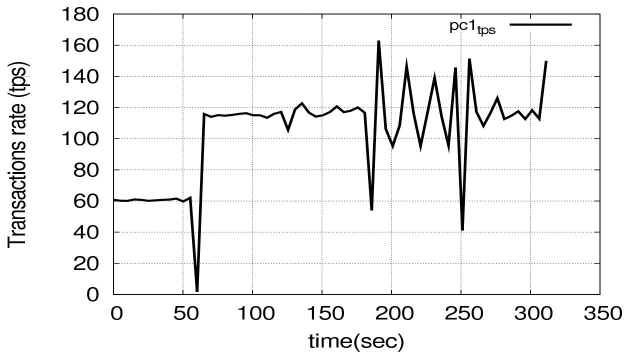
We test in this experiment a net-oriented application, i.e. a virtual machine with the web application server described in the previous experiment. The goal is to evaluate that the QoS App-SLO manager is able to find out the VM's resource configuration parameters so that the application, at any moment, can reach its maximum throughput and at the same time meet the user's perceived service-time requirements slo_3.

First, as a baseline experiment, we evaluate the web application server without our QoS App-SLO manager, for this experiment we launch the benchmarking tool Siege with an incremental load in the number of simulated web clients (2,8,32,128,256). Each incremental load has a think-time equal to zero and a duration of 60secs. The results show the saturation points which can be observed in metrics CPU consumption and successful requests. The surges correspond to the start and end times of the load generated by the benchmarking tool, each one with a duration of 60secs.

The maximum transaction rate is achieved with 8 clients starting at second 61 and the saturation point of response times at second 190 with 32 clients, which can be seen more clearly in Figure 4(b) and Figure 4(a) respectively. Therefore, the

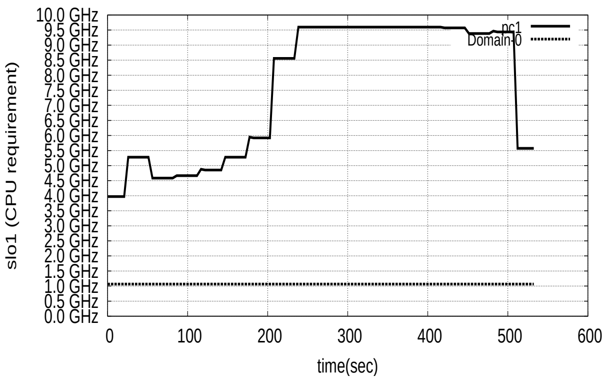


(a) Online mean response time as the mean user perceived experience

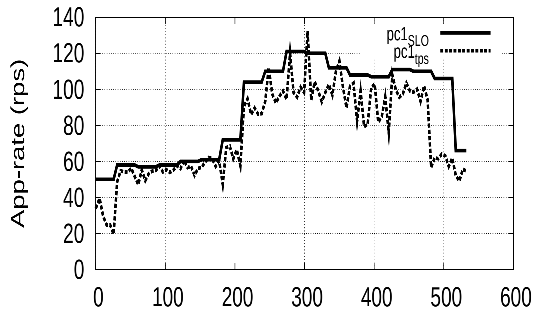


(b) Online application metrics

Fig. 4: Application web server.



(a) Learned CPU-rate



(b) App-SLO manager

Fig. 5: Automatic sizing of an application web server.

mean response times perceived by remote users start climbing at second 190.

In the second part of this experiment we enabled the QoS App-SLO manager in order to evaluate its automatic sizing capability. The workload used in this experiment has an incremental/decremental traffic pattern with the following number of remote clients: 2,8,32,128,256,128,32,8,2; each set of clients has a duration of 60secs giving an experiment of length 540secs.

The results can be seen in Figure 5 and Figure 6. Figure 5(a) shows that automatic sizing allows requesting CPU resources taking into account the dynamics of the observed requests. Therefore, it is achieved that the resources assigned to the VM can grow or shrink by tracing platform-independent metrics (http requests).

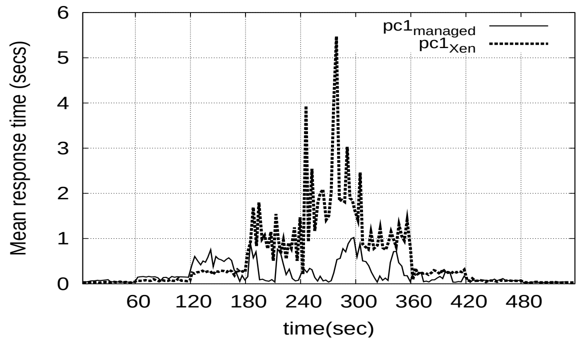
We achieve to trace accuracy http metrics that allow us to keep track of the pending requests in the system. Figure 6(b) shows the queue length of the mean pending requests observed during the last controller period.

We observe that, when managed, the queue of pending requests is less than when there is not admission control. Finally, the admission control applies the requested policy in order to meet the slo_3. Figure 6(a) compares the mean response times of the web application server as seen by the end-user. When managed, the response times are kept below 1 second.

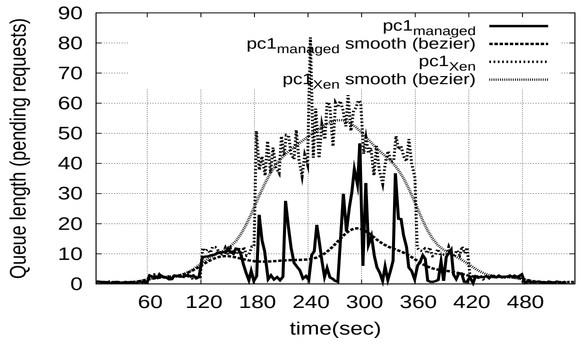
IV. Related work

Dongyan Xu *et al.* [9], [10] identified the following challenges that arise in realizing the vision of an *autonomic virtual environment adaptation* in a multi-domain share infrastructure:

1) *Live adaptation mechanisms*: The need to support application-transparent adaptation of Virtual Distributed Enviroments (VDEs). VMs supports runtime resource re-allocation and VM migration within LANs but, a multi-domain infrastructure needs live migration across networks domains without pausing or checkpointing the application. The solution has to meet two requirements: VMs need to retain the same IP address and remain connected to each other and migration mechanism cannot relay on NFS.
2) *Logistic service for VM migration*: Consisting of distributed *depots*. A depot is part of a infrastructure domain, in it, VM images are assembled using either local or transfered “parts”. Optimization problem: how to compute a distributed schedule for VM parts delivery and assembly so that all VMs will be ready in their destination hosts no later than a certain deadline?.
3) *Adaptation decision making*: Mechanisms for monitoring, controling, and adjusting resource allocations and



(a) Observed end-user response times



(b) Observed rps in-process

Fig. 6: Comparing traced metrics of an application web server

locations of VDEs. The identified problems are related to find out when an application needs more resources to perform well (or better), how to conciliate when adaptation affects the virtual environments sharing the same resources?, and migration issues, i.e. decide which virtual environment and where should it go by solving tradeoffs between resource availability and overhead.

4) *Adaptation shepherding*: a intelligent component that takes decisions (“justify and approve”) regarding adaptation requests, as a mechanism for preventing the abuse of adaptations.

The factors that drive the adaptation of *VDEs* are: availability of infrastructure resources that are dynamic and heterogeneous, and (2) the changing resources needs of the applications that run in a virtual environments.

Paul Ruth *et al.* [11] presents VioCluster, virtualization for dynamic computational domains. The problem is that each computational domain (e.g. cluster) faces the conflict between dynamic workload and static capacity. An opportunity to arises to resolve this conflict by dynamically adapting the capacity of clusters by borrowing idle machines of peer domains. Authors introduce the concept of *virtual computation domains* (or “*virtual domains*” for short) which allow a cluster to dynamically grow and shrink based on resource demand. VioCluster uses both machine and network virtualization techniques to logically move machines between virtual domains.

The HPC research community is particularly interested in using VMs. However, the main concerns widely discussed are the overhead caused by the virtualization layer, and the security [12], [13], [14]. On the other hand, large-scale scenarios such as HPC will benefit from fine-grain management tools to assign CPU resources.

The work of Kephart *et al.* [15] discusses the importance of self-management systems in the context of autonomic computing. These systems accept high-level objectives from administrators and apply self management policies.

Policy-based QoS control and learning have been proposed in non-VM contexts. Solutions based on QoS guarantees have been discussed using control theory [16], [17], online analytic performance models [18], regression-based analytic models [19], and statistical inference [20]. Some applications of these approaches are dynamic provisioning [21] and energy conservation [22]. With our proposal, we aim to provide a framework to meet the low-level VM requirements for the dynamic workload of the hosted application.

V. Conclusion

We have presented the conceptual design and theoretical foundation of a Quality of Service App-SLO manager which is in charge of managing the application goals.

The proposal has a set of definitions that captures the properties for the management of two types of SLOs: slo_2 expressed in the app-rate metric transactions (or requests) per second, and slo_3 expressed in the web metric response times. Additionally, the manager supports an admission control mechanism for the management of net-oriented applications.

Through experiments, we have presented results for different types of workloads: a math parallel application and a web-based application. We evaluated the management of two service level objectives: application rate, and response times.

The results show that the component is able to concurrently manage mixed workloads with their specific application’s objectives at different levels with mixed workloads.

Future work includes extending the capabilities of the proposal to support distributed applications in Cloud Computing environments.

Acknowledgment

This work is supported in part by the Computer Architecture Department of the Technical University of Catalonia, the Ministry of Education of Mexico, the program PROMEP of the Public Education Secretary under Contract PROMEP/103.5/10/7336, and the University of Colima.

References

[1] L. Cherkasova, D. Gupta, and A. Vahdat, “When virtual is harder than real: Resource allocation challenges in virtual machine based IT environments,” Hewlett Packard Laboratories, Tech. Rep. HPL-2007-25, Feb. 20 2007. [Online]. Available: http://www.hpl.hp.com/techreports/2007/HPL-2007-25.pdf

[2] F. Rodríguez-Haro, F. Freitag, and L. Navarro, "Enhancing virtual environments with qos aware resource management," *Annals of Telecommunications*, vol. 64, pp. 289–303, 2009, 10.1007/s12243-009-0106-1. [Online]. Available: http://dx.doi.org/10.1007/s12243-009-0106-1

[3] VMWare, "Understanding Full Virtualization, Paravirtualization, and Hardware Assist. Whitepaper," 2007, http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf.

[4] M. MSDN, "Hyper-V Architecture," 2008, http://msdn.microsoft.com/en-us/library/cc768520.aspx.

[5] "NIST/SEMATECH e-handbook of statistical methods," 2006, url-http://www.itl.nist.gov/div898/handbook/.

[6] R. F. Nau, "Introduction to ARIMA: nonseasonal models," 2005. [Online]. Available: http://www.duke.edu/~rnau/411arim.htm

[7] J. Bollinger, "Bollinger bands," 2012, http://en.wikipedia.org/wiki/Bollinger_bands.

[8] Siege, "An http regression testing and benchmarking utility," 2012. [Online]. Available: http://www.joedog.org/JoeDog/Siege

[9] D. Xu, P. Ruth, J. Rhee, R. Kennell, and S. Goasguen, "Short paper: Autonomic adaptation of virtual distributed environments in a multi-domain infrastructure," in *15th IEEE International Symposium on High Performance Distributed Computing (HPDC'06)*, June 2006, pp. 317–320.

[10] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen, "Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure," in *IEEE International Conference on Autonomic Computing, 2006. ICAC '06*, 2006, pp. 5–14.

[11] P. Ruth, P. Mcgachey, and D. Xu, "Viocluster: Virtualization for dynamic computational domains," *Proceedings of the IEEE International Conference on Cluster Computing (Cluster'05)*, 2005.

[12] L. Youseff, R. Wolski, B. C. Gorda, and C. Krintz, "Paravirtualization for hpc systems." in *ISPA Workshops*, 2006, pp. 474–486.

[13] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis, "Virtualization for high-performance computing," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 2, pp. 8–11, 2006.

[14] W. Huang, J. Liu, B. Abali, and D. K. Panda, "A case for high performance computing with virtual machines," in *ICS '06: Proceedings of the 20th annual international conference on Supercomputing*. New York, NY, USA: ACM Press, 2006, pp. 125–134.

[15] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[16] T. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, Y. T. A. Lu, J. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback performance control in software services," *Control Systems Magazine, IEEE*, vol. 23, no. 3, pp. 74–90, 2003.

[17] J. L. Hellerstein, "Challenges in control engineering of computing systems," IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Research Report RC23159 (W0309-091), sep 2003.

[18] D. A. Menascé, M. N. Bennani, and H. Ruan, "On the use of online analytic performance models in self-managing and self-organizing computer systems," in *Self-star Properties in Complex Information Systems*, ser. Lecture Notes in Computer Science, Ö. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. P. A. van Moorsel, and M. van Steen, Eds., vol. 3460. Springer, 2005, pp. 128–142. [Online]. Available: http://dx.doi.org/10.1007/11428589_9

[19] Q. Zhang, L. Cherkasova, and E. Smirni, "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," in *ICAC '07: Proceedings of the Fourth International Conference on Autonomic Computing*. IEEE Computer Society, 2007, p. 27.

[20] L. Bertini, J. C. B. Leite, and D. Mosse, "Statistical qos guarantee and energy-efficiency in web server clusters," in *ECRTS '07: Proceedings of the 19th Euromicro Conference on Real-Time Systems*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 83–92.

[21] B. Urgaonkar and A. Chandra, "Dynamic provisioning of multi-tier internet applications," in *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 217–228.

[22] C.-H. Tsai, K. G. Shin, J. Reumann, and S. Singhal, "Online web cluster capacity estimation and its application to energy conservation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 932–945, 2007.