Autonomic resource management for the Xen Hypervisor

Íñigo Goiri and Jordi Guitart
Universitat Politécnica de Catalunya
Barcelona, Spain
{igoiri,jguitart}@ac.upc.es

Abstract

Servers workload varies during time and they have to be able to support the maximum load rate during a day, hence, they are idle many time wating resources. A solution to avoid this situation is having different servers with different workloads in the same computer in an isolated environment and it can be achieved with virtualization.

Virtualization maximizes resources usage, nevertheless, virtual machines just competing for system resources is not a good solution in the real world. Providers needs to specify levels of service agreements with their users, like how many CPU percentage is guaranteed. In this case, introducing a manager that monitors resources usage and reallocates resources according to system load is necessary. In this article, how many overhead introduces an external manager respect native virtual machine scheduling will be studied.

If an application is underusing assigned resources, manager will reassign remaining resources between saturated applications taking into account SLAs.

Keywords: virtualization, Xen, service level agreements, performance analysis, resource manager

1. Introduction

Hosting providers hosts applications with different workloads that vary during the time. Thanks to consolidation of different servers in a single machine, server usage can be highly increased reducing costs and space.

Applications hosted in a server farm can have very different needs, for example, a transactional application has a different behavior to numerical applications [10]. Furthermore, application of the same type can have very different time patterns, for example, a web application that provides service to Asia has a behavior during a day as different as a European web page, hence they could share a physical machine complementing their loads.

Thanks to new virtualization technologies like paravirtualization, this machine sharing can be done in a smart way with full isolation and without losing too much performance. In the first part of this article, overhead and benefits of introducing virtualization instead of using a traditional machine will be evaluated.

In the real world, enterprises contract hosting services with different plans, a maximum bandwidth or a maximum amount of memory for instance. This idea introduces the concept of Service Level Agreement (SLA) [3] where the user can specify his resources requirements, for instance how many CPUs will he need, or the penalties if this agreement is broken.

This agreement is frequently used in e-business environments in order to provide quality of service to users. This can be seen as a contract that assures customers that they can get the service they pay for and will obligate the service provider to achieve its service promises. This idea has been evaluated in some environments using web-services [12] and this can take profit of virtualization features in order to provide promised quality of service and monitor it.

Current virtualization techniques applies scheduling algorithms that share resources between different domains getting a good resources usage degree. Nevertheless, virtualization scheduling applies generic policies without taking into account user requirements and makes a static resource allocation. For instance, a user can specify a SLA that can be easily achieved during a period making assigned resources becoming wasted, therefore, static resources allocation become a bad idea.

Introducing a manager that assigns these resources automatically [13] to hosted applications in the same physical machine taking into account the adquired SLA would increase system global usage giving a better service without breaking any agreement.

Obviously, introducing an external manager will decrease decrease performance obtained by native virtualization scheduling. This is due to the overhead introduced by the resource monitoring and reallocation in addition to native scheduling.

In this paper, an application that manages CPU in a Xen environment taking into account hosted applications

requirements has been developed and tested. This application takes advantage of Xen facilities to monitor resources usage and assignate them. Furthermore, it evaluates how wrong is introducing an external manager instead of just using native virtualization scheduler.

2. Virtualization

Computing in these days is becoming more and more powerful and this implies a resource underusage. Sharing these remaining resources between different virtual environments is a smart solution for this problem that can be achieved using virtualization techniques.

Virtualization allows abstract resources and isolate environments for different purposes. However, there is not just one way to virtualize a system, from complete hardware emulation to application virtualization [14]. In server consolidation a trade off between performance and system abstraction is needed.

The virtualization technology that achieves the right level of isolation and performance needed for this paper purpose is paravirtualization [11].

We also need managing resources between virtual environments in a smart way as well as monitoring domain behavior. This issue is crucial in order to achieve desired resource sharing. One of the products that implements paravirtualization and allows a high resource management level is Xen, which code is distributed under LGPL.

2.1. Xen

Xen [4] is a free open source hypervisor distributed under LGPL that allows a high usage degree and consolidation of servers created by XenSource. It provides mechanisms to manage resources, including CPU, memory and I/O. The key of Xen success is paravirtualization that allows obtaining a high performance level maintaining an appropiate level of isolation.

It is the quickest and safer virtualization infrastructure in this moment. Nevertheless, paravirtualization requires introducing some changes in the virtualized operating system but resulting in near native performance. Xen gives to the guest operating system an idealized hardware layer.

In a Xen environment a virtual server is just an operating system instance (called domain in the Xen environment) and its load is being executed on top of the Xen hypervisor. This instances accesses devices through the hypervisor, which shares resources with other virtualized OS and applications.

Overhead introduced by Xen hypervisor is less than 3.5% [7]. In addition, thanks to paravirtualization I/O operations are executed out of the hypervisor and shared between domains following resource sharing policies. Nevertheless, virtualized domains are fully isolated.

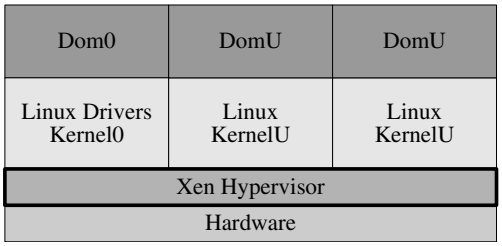

Figure 1. Simple Xen architecture

Xen also offers some tools like live migration, CPU scheduling and memory management combined with open source software advantages makes Xen a great alternative that allows having a full resources control.

2.1.1 Xen management

There are many ways to interact with Xen like XenStore or Xenstat. The first one is a great way to obtain information about system, it is an information storage space shared between domains that relies on XenBus. That is meant for configuration and status information rather than for large data transfers. Each domain gets its own path in the store, which is somewhat similar in spirit to procfs. When values are changed in the store, the appropriate drivers are notified.

The other mechanism to interact with Xen, Xenstat, is a library that allows configuring and obtaining information about Xen directly working with the Xen API. It allows setting the number of VCPUs associated with a domain when the domain is created, nevertheless, it can be modified dynamically during the execution allowing a high CPU management degree.

Another significant Xen feature in this paper is hypervisor CPU scheduling [5]. Every domain has assigned a given number of virtual CPUs that correspond to a physical CPU. This CPU allocation is done by the Xen hypervisor in a dynamic way, it assigns a CPU to a VCPU according to a set of policies that can be specified or tunned.

Xen supports different schedulers such as Round Robin, Credit and Earliest Deadline First Scheduling (sEDF). This schedulers can be configured by the system administrator according with his needs.

This paper will take profit of credit scheduling because it allows a simple and efficient policy tunning like changing priority or setting the maximum CPU amount. The credit scheduler is a proportional fair share CPU scheduler that supports SMP hosts. Is is the default scheduler and the other schedulers like SEDF and BVT are optional and is planned to remove them in next versions. This sched-

uler has a sorted queue that contains VCPU to be executed according with priorities and earned credits in the queue, thanks to this method is easy to define the maximum CPU capacity of a given domain.

3. Manager

This paper introduces a test manager that monitors and allocates resources to domains according with specified levels of services specified by the user. The user specifies how many CPU amount will he need and the application guarantees this level. Once the application starts running, the manager monitors the domain CPU usage and calculate if it is wasting assigned resources and in that case assigns these to overloaded domains.

This application tries to evaluate overhead and study different resource assignation possibilities

This resource usage checking is done every five seconds and therefore this reallocation. By the moment, this manager only takes into account processor, nevertheless, it could manager other resources like memory. Processor is one of the most relevant resources in this type of environment and it has direct relation with obtained performance.

Xen gives many resource management facilities, CPU related mechanisms includes VCPU assigning, CPU pinning and CPU maximum capacity. VCPU assignation allows defining how many CPUs will the domain see, therefore, how many CPUs can be used by the hosted application. Furthermore, CPU pinning allows specifying which physical CPUs can be used by VCPU, allowing specify CPU sharing. Last useful mechanism is setting maximum CPU capacity, having four CPUs we can assign from a 1 to a 400% (0 implies no limit).

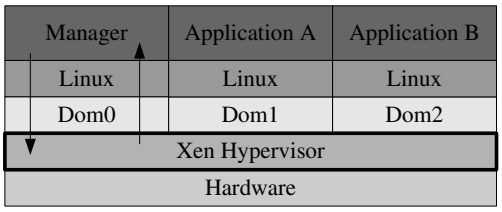

Figure 2. SLA manager architecture on Xen

Figure 2 shows how manager is implemented and its architecture. It runs on the domain 0 as a service that wakes up every five seconds for monitoring how many CPU is each application consuming. It utilizes Xenstore for monitoring application CPU utilization, Then manager evaluates if every virtual machine is achieving specified level of required service and if it is wasting resources, reassigns them to other applications.

In this paper, different techniques for CPU sharing will be tested such as VCPU assigning, CPU pinning and maximum capacity including different combinations of this mechanisms. This mechanisms are provided by the Xen management capabilities and the credit CPU scheduling policy.

4. Experimentation

In order to evaluate performance of the introduced manager, experiments will consists on three different scenarios. The generic scenario will consist on two servers running in a single machine using different techniques. The first one will run in a shared environment, the second one in a virtualized environment and the last one in a virtualized environment with a manager that manages resources between domains.

These servers (application servers) contained in different environments will be tested by load generators that will put them under different levels of workload, looking for checking some key situations that represents extreme situations.

In the two first experiments, tests will try to demonstrate that introducing a virtualized environment respect a shared environment introduces some benefits. Finally, some mechanisms to manage resources in a Xen domain such as setting maximum capacity or specifying CPU pinning will be analyzed in order to evaluate a possible introduction in a real environment.

4.1. Environment

The scenario simulates two different web pages (A and B) that receive different loads, allowing server consolidation. Experimentation environment is composed by two application server that simulates a typical auction website that are contained in the same machine using different techniques to consolidate server.

The application server will be a Tomcat, this is an open-source servlet container developed under the Apache license and its primary goal is to serve as a reference implementation of the Sun Servlet and JSP specifications, but it also aims to be a quality production servlet container too. Tomcat can work as a standalone server (serving both static and dynamic web content) or as a helper for a web server (serving only dynamic web content).

In this paper, we use the 5.0.19 version of Tomcat [**?**] as a standalone server that hosts a typical website. This Tomcat has been configured with the maximum number of HttpProcessors set to 100 and the connection persistence timeout is set to 10 seconds. The experimental application is composed by the 1.4.2 version of the RUBiS (Rice University Bidding System) [2] benchmark servlets for Tomcat. RUBiS implements the core functionality of an auction site: selling, browsing and bidding.

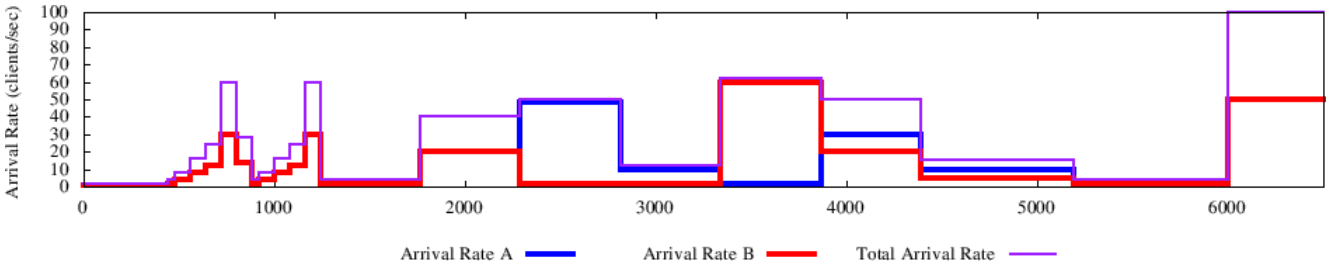

Figure 3. Application server input work load

This application will be stressed by two load generators, Httperf [1], that send requests to RUBiS following a typical user navigation patterns and measures its performance, according to a set of values like number of finished sessions, reply rate, etc.

Application servers will be installed on a Debian host machine composed by a 4-core Intel Xeon at 3.16GHz with 8 GB of RAM memory and due to this server characteristics, supported input loads by the server is too high to be generated by a single Httperf, and a way to join multiple load generators was needed. With this purpose an script that starts and stops Httperf in different machines and merge results was created. Thanks to this script a set of computers can generate high work loads on a server. These machines that run multiple instances of Httperf are two Intel Xeon Quad Cores.

Connections with the application server are secure relying on SSL and this implies a considerable CPU load for each connection. This overload collapses the systems if the number of requests is high enough, [9] explain this situation and how the application performance comes down when is achieved a certain level of workload.

In order to evaluate the application server, we need to know the maximum number of users allowed by the server before the performance comes down with a given number of CPUs. Server has been put down a fixed workload during 600 seconds for a different number of clients. Figure 4 shows obtained throughput with 1, 2, 3 and 4 CPUs.

Thanks to this information, we can know the number of clients needed to get the better performance according with the number of CPUs. Table 4.1 shows the number of clients allowed before the application level throughput starts decreasing.

The client workload generated by the web performance measurement tool has a variable input load throughout the run time, which is shown in figure 3, this displays the number of new clients per second that hit the server as a function of the time. Input load distribution has been chosen in order to represent the different processor requirements combinations when running in the hosting platform according to the values shown in figure 4.

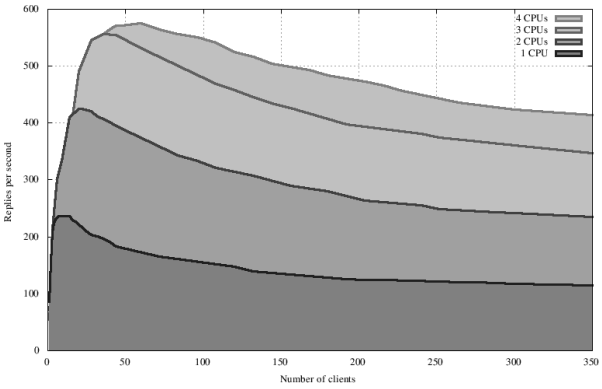

Figure 4. Tomcat scalability with different number of processors

Num. CPUs	Num. Clients	Replies per second
1	14	236.808
2	22	425.299
3	36	575.618
4	60	556.525

Table 1. Number of clients that overload the server and maximum achieved throughput before overloading

Generated load starts with two high peaks in a few time to warm-up application servers, then a low load is maintained during 500 seconds to stabilize servers. In this moment, a medium load starts in both applications and then application A grows and B becomes very low. After this moment, situation are swapped. Loads start decreasing gradually and finally both applications get a high load to stress both web applications.

Xen has been chosen to virtualize system and get system isolation because this virtualization product implement paravirtualization getting near native operating system performance and provides a complex resource management that

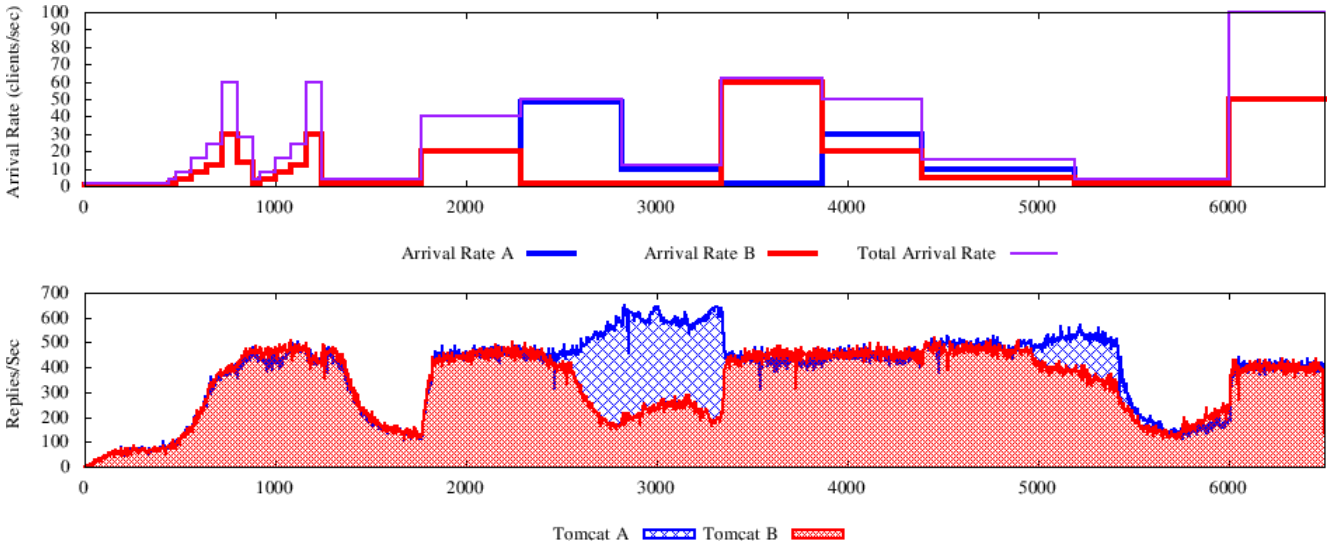

Figure 5. Applications throughput in a shared machine

allows easily implementing tools for managing domains. In addition, this product has a huge community and is becoming more and more popular everyday.

4.2. Traditiontal environment

This scenario simulates two application servers working on the same computer without any virtualization technique to isolate them causing a race for resources. These servers will run on a typical Linux system listening for connections on ports 8443 and 8444 respectively.

Figure 5 shows web application response time obtained by the two tested Tomcats (bottom graphic) according with the input work load (top graphic). In the response time graphic, we can observe that Tomcat is not very reactive to the input load and reducing input load it continues responding old clients that are in the queue.

Using a traditional environment, obtained application server performance is very high, it achieves peaks of 1000 replies per seconds. Nevertheless, system is not very reactive and one server load interferes the other one.

4.3. Virtualized environment

In this case, overhead of introducing a virtualized environment will be studied like following [15]. Each Tomcat is isolated in its own virtual machine providing server consolidation. We have a Debian host machine with Xen 3.0.4 and a 2.6.16.33-xen kernel that contains two virtual domains, each one with a Tomcat v5.0.19 running in a Ubuntu Dapper domain.

Both virtual machines have four virtual CPUs assigned dynamically by Xen between the four physical CPUs following credit scheduling.

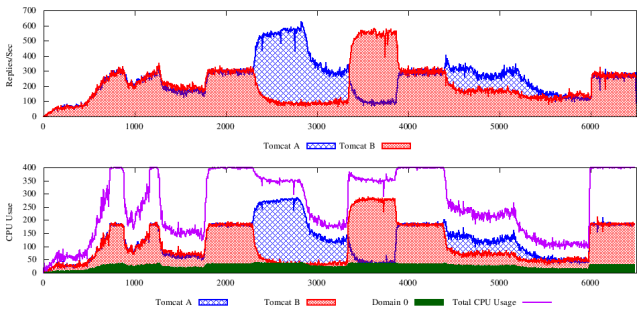

Figure 6. Applications throughput each one isolated in a domain

Figure 6 shows that using virtualization, obtained performance (top graphic) is really reactive to input workload. In addition, in peak point the number of replies per second obtained is so close to the obtained in the shared environment, close to 900 replies per second. In this case, obtained performance looks better and the number of finalized sessions has been improved.

In addition to the replies per second graphic, in this figure CPU usage by each application is shown at the bottom graphic. It shows that CPU usage is achieving the whole CPU usage in the moment of highest input load.

Taking into account, virtualization overhead (shown in green in the figure) obtained performance is so good (less

than a 5% of the CPU) and we can conclude that it can be a real alternative for achieving system consolidation with full isolation.

4.4. Simple SLA manager

This scenario introduces our manager allocated in the domain 0 that monitors and assigns resources to each virtual machine according to observed usage and specified SLA. In this paper, SLA will assign the same amount of resources to each domain, therefore, 200% to each Tomcat.

This version of the manager allocates full VCPUs and set maximum CPU capacities thanks to the Xen credit scheduler capabilities. Firstly, it get how many CPU amount is each application using and calculates if some application is wasting resources and gives remaining resources to overloaded domains conserving the SLA.

When it has calculated how many CPU amount will be assigned to the each domain, it assigns a number of VCPUs enough to achieve the assigned percentage of CPU and then sets the maximum capacity of each domain to the desired value.

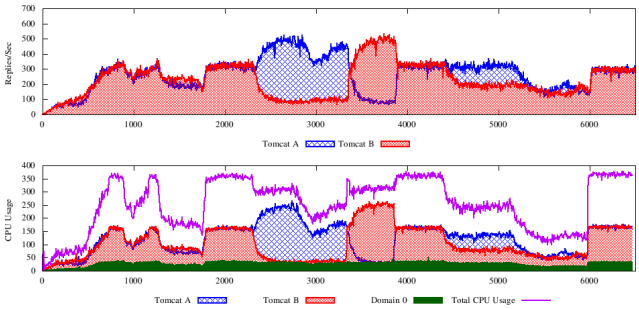

Figure 7. Applications throughput using the SLA manager in a virtual environment

Introducing our manager, obtained performance becomes lower than the other last cases (figure 7). Observing CPU usage, we can conclude that defining a maximum CPU usage, makes the highest CPU usage rate to be 350%, therefore, obtained replies per second never can achieve the same level than simple virtualized environment.

Using this manager, SLA support aim is reached with some lost of performance due to resource sharing in the moments of maximum load. Nevertheless, the major purpose of this manager is maintain the SLA specified by the use and figure 7 shows that when both applications need resources, they share the same amount of CPU carrying out the agreement.

4.5. SLA Manager using CPU pinning

In this scenario some improvements are introduced in the manager in order to evaluate new techniques. It smart allocates resources applying new techniques like CPU pinning forgetting about Xen native dynamic pinning.

Xen makes a dynamic pinning by itself distributing physical CPUs between virtual CPUs achieving an equitative global distribution. Distributing this resources taking into account domains needs, could give a bigger stability to the performance reducing context exchanges between domains.

It can be achieved using a manual pinning by the manager that calculates how many virtual CPUs would need a certain domain according with its usage and pin firsts virtual CPUs to a given physical CPU and assigns every physical CPU to the last one. This rescheduling is done every five seconds giving a better stability to the application.

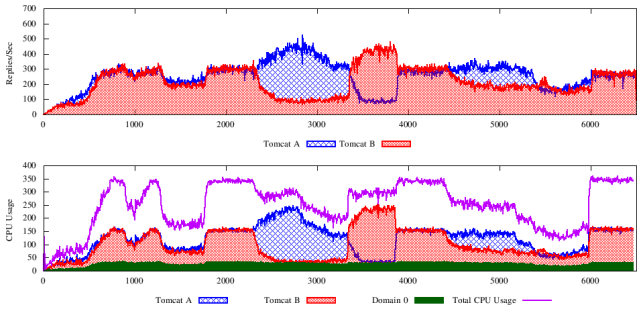

Figure 8. Applications throughput using the SLA manager with CPU pinning

Figure 8 shows obtained performance and the CPU load generated using the initial manager that pins CPUs. This pining is done according to application needs in addition to last mentioned policies. As is shown in the figure, this method obtains a little performance decreasement respect to the simple manager.

With this method the reactiveness won with a virtualized environment becomes little and it lost some performance. This is due to the big period of resource assignation respect the native scheduler. In addition, Xen works in a lower level implying a more efficient resource managing and a less overhead.

Therefore, using an external manager to do scheduling tasks is not a good idea that are done by the native scheduler. Furthermore, possible performance improvements introduced by a manager can't be compared with the overload introduced by an external application that has to go through several layers to monitor resources usage.

We can conclude that an external manager should not carry out scheduling tasks and just take care about service

level agreements assigning global resources and forgetting about internal issues.

4.6. Future trends

Service level agreement not only treats CPU available by an application, it specifies how many memory will be available or the uptime of the service. The last parameter is one of the advantages of a virtualized environment that supports mobility of system between different machines. Nevertheless, other parameters like memory has been not considered and can be managed by a manager in order to provide a more detailed SLA.

Memory management should be evaluated to decide if is reasonable to be introduced in a SLA manager tool. This feature has not a direct involvement in performance issues, however, it would provide much more capabilities to a SLA manager in a virtual environment. This idea could be extended to other resources opening new ways to specify SLAs by the user.

Some improvements in CPU managing can be done taking profit off Xen default scheduler capabilities like CPU weight domain definition. Thanks to this capability, a manager would provide some priorities between domains opening new ways to define these agreements.

Adding some kind of way to communicate the hosted applications in order to allow resource requesting and a better monitoring would be another way to improve performance of applications.

5. Conclusions

Introducing a virtualized environment with server consolidation purposes gives great results compared with a traditional operating system that contains two applications with different behavior during time.

In the first experiments, we have seen that Xen doesn't introduce a overhead bigger than a 5% of the overall system CPU consume. Therefore, it is a good alternative to be used to host different applications in a server giving great capabilities.

This type of performance analysis is a good way to evaluate an application behavior in a virtual domain. In addition, Xen allows extracting significant information about the domains like CPU usage, that allows making a more detailed analysis of the application performance and its behavior.

In the last part, the cost of introducing an external manager that manages service level agreement specified by the user has been studied. This is possible thanks to Xen because its resource managing capabilities. Assigning resources to each application supposes a considerable overhead. In addition, applying complicated resource limits is worse than just assigning whole CPUs.

However, introducing an external manager to carry out scheduling tasks instead of just taking proffit of native Xen scheduling capabilities to tune its parameters doesn't get better the system performance. Therefore, is better trust on Xen for scheduling purposes and just provide the manager with SLA management responsibilities.

Xen provides great tools and capabilities to support global resource management without taking into account low level details of the underlying system, hence, is better letting Xen do these tasks.

Finally, virtualization and paravirtualization specifically, allows new ways to manage resources opening new ways to solve problems like SLA agreement.

References

[1] Httperf. http://www.hpl.hp.com/research/linux/httperf.

[2] Rubis. http://rubis.objectweb.org/.

[3] The sla zone. http://www.sla-zone.co.uk.

[4] Xen. http://www.xensource.com.

[5] Xen scheduling. XenSource Wiki.

[6] I. M. Author. Some related article I wrote. *Some Fine Journal*, 99(7):1–100, January 1999.

[7] B. Clark. A moment of xen: Virtualize linux to test your apps. 2005.

[8] A. N. Expert. *A Book He Wrote*. His Publisher, Erewhon, NC, 1999.

[9] J. Guitart. *Performance Improvement of Multithreaded Java Applications Execution on Multiprocessor Systems*. PhD thesis, Computer Architecture Department, Technical University of Catalonia, 2005.

[10] E. V. Hensbergen. The effect of virtualization on os interference.

[11] B. G. Lamia Youseff, Rich Wolski and C. Krintz. Paravirtualization for hpc systems.

[12] A. S. Li-jie Jin, Vijay Machiraju. Analysis on service level agreement of web services.

[13] L. Simcox. Autonomic features of the ibm virtualization engine. 2004.

[14] T.-c. C. Susanta Nanda. A survey on virtualization technologies.

[15] P. P. S. S. Zhikui Wang, Xiaoyun Zhu. Capacity and performance overhead in dynamic resource allocation to virtual containers.