Fluminense Federal University

CARLOS ROBERTO DE OLIVEIRA JUNIOR

Experimentation with virtual machine migration and replication mechanisms

NITEROI

2011

CARLOS ROBERTO DE OLIVEIRA JUNIOR

Experimentation with virtual machine migration and replication mechanisms

Master thesis submitted to the Graduate School of Computation of Fluminense Federal University as a partial requirement for the degree of Master in Science. Topic Area: Parallel and Distributed Computing.

Supervisor:

ORLANDO GOMES LOQUES FILHO

Fluminense Federal University

NITEROI

2011

Experimentation with virtual machine migration and replication mechanisms

Carlos Roberto de Oliveira Junior

Master thesis submitted to the Graduate School of Computation of Fluminense Federal University as a partial requirement for the degree of Master in Science. Topic Area: Parallel and Distributed Computing.

Approved by:

Prof. Orlando Gomes Loques Filho, Ph.D. / IC-UFF
Chair

Prof. Julius Cesar Barreto Leite, Ph.D. / IC-UFF

Prof. Alexandre Sztajnberg, Ph.D. / IME-UERJ

Niteroi, September 2011.

"Commit thy works unto the LORD, and thy thoughts shall be established."

Proverbs 16:3

"It's time to start living the life you've imagined."

Henry James

Abstract

According to recent studies, 90% of the time an user is using a computer is spent on the Internet. As a result, many Internet applications have been developed targeting this public. Most part of these applications are multi-tier (e.g., e-commerce sites and social networks), consisting of application and database tiers, and several of them cannot be supported by a single physical machine.

In this work we use a typical virtualized server architecture to perform basic experiments using migration and replication mechanisms. In this architecture is assumed that both application and database tiers are implemented by virtual machines (VMs), which can be configured (i.e., added, removed or even dimensioned) dynamically. The architecture includes mechanisms for managing both application and database tiers, allowing the experimentation of different configuration management policies. To manage the applications in our architecture, a layer of proxies is interposed between clients and servers to intercept requests generated by clients. These proxies permit the load balancing in the different tiers. As we replicate VMs, both application and database tiers can be supported by different physical machines.

Using this architecture, we have carried out a set of experiments using the Rubis benchmark tool, that simulates a typical multi-tier application, the eBay site. In our experiments we evaluated the performance disruption incurred during the movement or replication of VMs. We also show in our experiments that using VM replication is possible to balance the workload among different servers in a cluster. Thus, the ability to dynamically distribute server workloads in a virtualized server environment allows for using the dynamic optimization model for power and performance management, proposed by our group, in this multi-tier architecture.

Keywords: Virtualized web clusters, Virtual machine migration, Virtual machine replication.

Shorthands

API	:	Application Programming Interface
CPU	:	Central Processing Unit
DVFS	:	Dynamic Voltage and Frequency Scaling
HTML	:	HyperText Markup Language
HTTP	:	Hyper-Text Transfer Protocol
IT	:	Information Technology
NFS	:	Network File System
PC	:	Personal Computer
PHP	:	PHP: Hypertext Preprocessor
QOS	:	Quality of Service
RAM	:	Random Access Memory
SLA	:	Service Level Agreement
TCO	:	Total Cost of Ownership
VCPU	:	Virtual Central Processing Unit
VM	:	Virtual Machine
VMM	:	Virtual Machine Monitor

Contents

List of Figures

Chapter 1

Introduction

In recent years we have seen an increase in demand for web content and services. According to Google [20], 90% of the time an user is using a computer is spent on the Internet. As a result, several Internet applications have been developed targeting this public. Most of these applications are multi-tier, consisting of application and database tiers (e.g., e-commerce sites and social networks).

In this context, to respond to such high demand, web content and service providers are building large data centers to support many different Internet applications and services. A data center is a large-scale distributed system that consists of hundreds or thousands of machines linked by a fast network. Servers in these data centers are used to store content and to process user's requests. The performance of a web server system plays an important role in the success of Internet related companies. Long response delays would frustrate users' interest in the interaction with the web server and they may give up browsing the web site. For e-commerce web sites, such degradation is especially harmful, as it may lead to significant loss in revenue and decrement in customer visits.

The growth in the number and size of data centers creates, at least, two more problems for enterprises: (1) it increases the Total Cost of Ownership (TCO) because more money is spent to buy computers and to pay IT professionals salaries. Moreover, according to [3], the costs associated with power and cooling could overtake hardware acquisition costs. Thus, requiring major investigation of techniques to improve the energy efficiency of their computing infrastructure [6, 18, 64]; (2) for many industries, data centers are one of the largest sources of CO2 emissions. As a group, their overall emissions are significant, in-scale with industries such as airlines. Data centers are expected to quadruple their CO2 emissions and to contribute to 30% of the world's carbon-dioxide emissions by 2020. In addition, it will surpass those CO2 emissions of the airline industry by 2020 [42]. Thus,

achieving power-efficient in today's Internet server systems is a fundamental concern.

Although there are high peak demands, the average server utilization remains low (the average resource utilization in traditional data centers range between 5-20% [22]), creating tremendous "waste" in terms of capital employed and energy used [42]. To increase resources utilization, server virtualization has been widely adopted in data centers around the world. Virtualization allows servers to run multiple independent application servers in a physical machine.

Instead of having a physical machine running a single application server. Each application server runs on the top of a Virtual Machine (VM). A physical server runs one or more VMs that dynamically share the underlying hardware resources leading to an increased resource utilization, energy savings and reduced TCO. It is recognized that the dynamic consolidation of application workloads based on virtualization techniques helps to increase server utilization, allowing to reduce the use of computer resources and the associated power demands [16, 39, 68, 72, 83]. Specifically, the ability to dynamically move application workloads around in a virtualized server environment enables some physical machines to be turned off in periods of low activity, and when the demand increases, it allows for bringing them up back and distributing the application workloads across them. Moreover, this on/off mechanism can be combined with DVFS (Dynamic Voltage and Frequency Scaling) - a technique that consists of varying the frequency and voltage of the microprocessor at runtime according to processing needs - to provide even better power and performance optimizations. This presents an efficient way of running a data center from a power management point of view [57].

Several Virtual Machine Monitors (VMMs), also known as hypervisors, like Xen [2] and VMware [21] have been developed to support server virtualization. They act as a layer between the virtual machine and the actual hardware. Each VM is subject to management operations such as creation, deletion, and migration between physical machines, as well as run-time resource allocation. These features enable resource sharing in arbitrary combinations between applications and physical servers and provide the means for efficient server consolidation.

The work presented in this Master dissertation is part of a project that aims to reduce power consumption using a dynamic optimization model for power and performance management of virtualized clusters [55, 58, 59, 60, 61]. However, in the scope of this work, we contribute describing and evaluating, through experiments, mechanisms that have been used by virtualized web clusters to increase resource utilization e.g., virtual

machine migration and replication. To perform experiments in this work, we have used an architecture that integrates mechanisms provided by Apache [75] and MySQL [47], that allowed the experimentation with a multi-tier application.

The remainder of this Master dissertation is organized as follows: In Chapter 2 we present the proposal of this work. In Chapter 3 we present a description of the testbed we have used to carried out experiments. The mechanisms used to perform virtual machine migration and replication are described in Chapter 4. Experimental evaluation is presented in Chapter 5. Related works are presented in Chapter 6. Finally, Chapter 7 presents conclusions and outlines directions for future work. In the Appendix A, we briefly describe the Xen hypervisor used in this work. In the Appendix B it is presented how the response time is measured in this work. We present the physical machines used in our experiments in the Appendix C.

Chapter 2

Proposal

Our goal in this work is to describe and evaluate, through experiments, mechanisms that allow applications to be supported by a virtualized environment. In the first phase of our work we have carried out experiments aiming to evaluate the migration mechanisms provided by VMMs and the VM replication in the application-tier. To perform these experiments, we used the architecture presented in Section 3.1, that had already been used by other researcher of our group. The results were published in [52] and are presented in Section 5.1. The httperf tool [25] was used in this phase for generating workload for the application.

As the architecture presented in Section 3.1 has no mechanisms to support the database-tier, in a second phase, we have integrated mechanisms to allow the experimentation with multi-tier applications. Thus, the contribution of this master dissertation consists on the integration of these mechanisms and the experiments we have carried out during this work. In the second phase of our experiments we used the RuBiS benchmark tool [67] to generate workload to the architecture presented in Section 3.2.

An important information for the experiments in this master dissertation is provided in Figure 2.1. When the CPU utilization of an application is low, the average response time is also low. This is expected since no time is spent queuing due to the presence of other requests. On the other hand, when the utilization is high, the response time goes up abruptly as the CPU utilization gets close to 400% (the maximum value for utilization is 400% because we are using quad-core machines and each core represents 100%). In order to meet fair response time requirements, we shall perform VM migration or replication before the machine saturates, dimensioned for playing safe as 300% of CPU utilization. This leaves an amount of 100% CPU capacity available to be used by the VM management

domain (Dom0) on the physical servers during the migration or replication activity, this utilization threshold was found suitable (because it keeps the request response time low) in our previous work [52], but other values could be used depending on specific performance requirements.

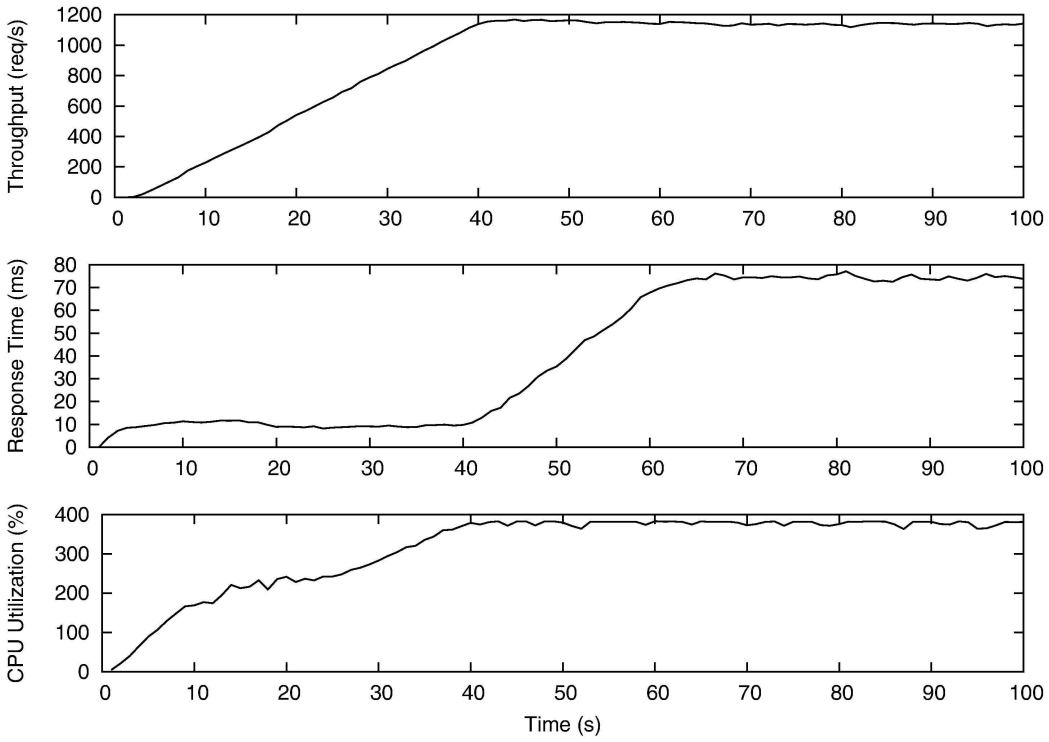

Figure 2.1: Relationship among throughput, response time, and CPU utilization

We use the CPU utilization as a metric to perform VM migration or replication since this is the metric used in the context of the project this master dissertation is inserted. However, in a real system, other metrics can be used to evaluate if a machine is saturated. As an example, authors in [29], use the response time. In addition, in a real system, it would be interest for an optimization model to use both migration and replication mechanisms. However, in this work we use no optimization model, thus we use migration or replication mechanisms separately according to the experiment goal.

To collect the CPU utilization we used a script, termed *Monitor*[1] in this master dissertation, that monitors the CPU utilization of the physical machines and VMs. The *Monitor* runs in the front-end physical machine and uses *XmlRpc* to communicate with the worker physical machines and collects the CPU utilization (for the Dom0 and the VMs) measured in the Dom0 of each worker machine. The Monitor runs in a loop and the CPU utilization is measured, using Xentop [27], each second.

[1]Implemented by Vinicius Petrucci in [56].

To perform migration or replication activities when the CPU utilization achieves the threshold of 300%, we used the OpenNebula toolkit [53] in the first phase of experiments. For the experiments in the second phase of our work, we implemented a script, termed *Manager* in this master dissertation, that runs in the front-end physical machine and uses *XmlRpc* to communicate with the worker physical machines. The Manager uses the Xen API to start or deallocate VMs in the cluster.

In the Manager, each worker is regarded as a data structure containing two boolean values *App* and *Db* that are set as *true* when an application or database virtual machine is started on the worker machine. The data structure also contains three *real* values, *CpuUtilization*, *CpuApp* and *CpuDb* assigned with the CPU utilization of the physical machine, CPU utilization of the application VM and CPU utilization of the database VM, respectively, collected by the Monitor. These measures are constantly updated in a loop. The Manager also uses an *active* list of the physical machines currently used to run VMs.

To balance the workload across the Apache servers, the *Manager* uses a module, that runs in the front-end machine, termed *frontend_module*[2]. The *frontend_module* is also used to collect the response time and throughput from the Apache. The *Manager* acts as follows. When a physical machine achieves 300% of CPU utilization, the Manager searchs for other physical machine in the active list with CPU utilization below than 150% to start a replica and balance the requests for the tier using more CPU cycles in the saturated physical machine. If there is no active worker with less than 150% of CPU utilization, the Manager searches for an active worker with less than 250% to balance requests. If there is no worker with CPU utilization below than 250%, the Manager adds a new worker machine to the active list and starts a VM replica in this new worker.

All application VMs start with the same weight in Apache. The load balancing is performed by increasing in one unit the weight in Apache for the VM running in the physical machine that has the lowest CPU utilization. Thus, weights are constantly adjusted in the loop while there is a physical machine achieving the threshold of 300% of CPU utilization. Note that the Manager only balance requests in the application-tier. For the load balancing in the database-tier, the MySQL Proxy balances read requests across slave databases and write requests are sent to the master database. To avoid bottleneck due to the master database, it is used a physical machine to run only the master database in our experiments. The Manager also uses a variable termed *CPUcluster* that represents the sum of

[2]Implemented by Vinicius Petrucci in [56].

the CPU utilization of all active worker machines. If $(CPUcluster/active - 1) < 270\%$ the Manager shutdown the VMs running in the physical machine with lowest CPU utilization and new requests are automatically send to the remaining worker machines. The value of 270% leaves a headroom for oscillations in the CPU utilization before achieving the threshold of 300%.

```
Manager algorithm

for (i = 1 to active)
if (worker[i].CpuUtilization >= 300) //worker is saturated
saturated = i  // number of the saturated worker

//check what is the bottleneck-tier
if (worker[saturated]. App = true) and (worker[saturated]. Db = true)
if (worker[saturated].CpuApp > worker[saturated].CpuDb)
saturatedTier = 1
else
saturatedTier = 2
     else
if (worker[saturated]. App = true)
saturatedTier = 1
else
saturatedTier = 2



for (k to active)
if (worker[k].CpuUtilization < 150) //start replica
start replica saturatedTier
if (saturatedTier = 1)
set weight for replica in Apache
worker[k].App = true
else
worker[k].Db = true

else
if (worker[k].CpuUtilization < 250) //balance requests
```

```
lessCpuUtilization = worker[saturated].CpuUtilization
index = saturated
for (m to active)     //find lowest CPU utilization
if (worker[m].App = true) //balances only the application-tier
if (worker[m].CpuUtilization < lessCpuUtilization)
lessCpuUtilization = worker[m].CpuUtilization
index = m
increase weight in 1 unit for replica in worker[index]
else
//add new physical machine
new = length (active) + 1
active <- worker[new]
                        start replica saturatedTier in worker[new]
if (saturatedTier = 1)
set weight for replica in Apache



for (j = 1 to active)
    CPUcluster = CPUcluster + worker[j].CpuUtilization

if ((CPUcluster / (length(active) -1) < 270) //deallocation of physical machines
lessCpuUtilization = worker[saturated].CpuUtilization
index = saturated
for (m to active)     //find lowest CPU utilization
if (worker[m].CpuUtilization < lessCpuUtilization)
lessCpuUtilization = worker[m].CpuUtilization
index = m
if (worker[index].App = true)
set as disable on Apache
shutdown App
if (worker[index].Db = true)
shutdown Db
takes worker[index] out of active
```

Chapter 3

Architecture and Experiment Environment

We have used two different architectures in this work. In this Chapter, we present the environment in which we have performed our experiments. The first architecture, presented in Subsection 3.1, is a basic architecture in which we evaluated migration and replication mechanisms. In a second phase, we have used the architecture presented in Section 3.2 targeted for multi-tier Internet applications. We also present the benchmark tool used for generating workload to the multi-tier architecture.

3.1 Basic cluster architecture

In previous works [55, 58, 59, 60, 61], our group have proposed an optimization solution for power and performance management in virtualized server clusters. The optimization deals with the problem of selecting at runtime a power-efficient configuration and a corresponding mapping of the multiple applications running on top of virtual machines to physical servers. The optimization decision also includes selecting the best voltage/frequency combination for each physical server, which can be imposed using DVFS (Dynamic Voltage and Frequency Scaling) support available in current processors. The optimization approach was experimented through simulations driven by real workload traces and achieved power savings of 47% compared to an uncontrolled system (used as baseline). However, in practice, a problem that arises in this context is that migration and replication activities in a virtualized environment may lead to disruption on the quality-of-service provided by the applications. For example, live migration mechanisms allow to make workload movements with a relatively short service downtime. However, the quality-of-service of the running applications are likely to be negatively affected during

the migration activities [82].

In the basic architecture presented in this Subsection, we consider a real virtualized cluster platform aimed at supporting the deployment of web applications. We used this architecture to carry out a set of experiments with different test scenarios, presented in Section 5.1, to evaluate the application behavior during the course of migration and replication actions. We also measure and analyze the disruption on the QoS (quality-of-service) provided by the applications, by means of server-side response time and throughput, during dynamic allocation operations of virtual machines in a server cluster. The response time of the web applications in the cluster is adopted as our main QoS metric since it is crucial for qualifying the end-user experience. In our experiments, we use Xen as the virtual machine manager and Apache servers for running the web applications.

Our target architecture (shown in Figure 3.1) consists of a cluster of replicated web servers. The cluster presents a single view to the clients through a *front-end* machine, which distributes incoming requests among the actual *servers* that run VMs and process the requests (also known as *workers*). These servers run CentOS Linux 5.4 [8] with Xen hypervisor enabled to support the execution of virtual machines.

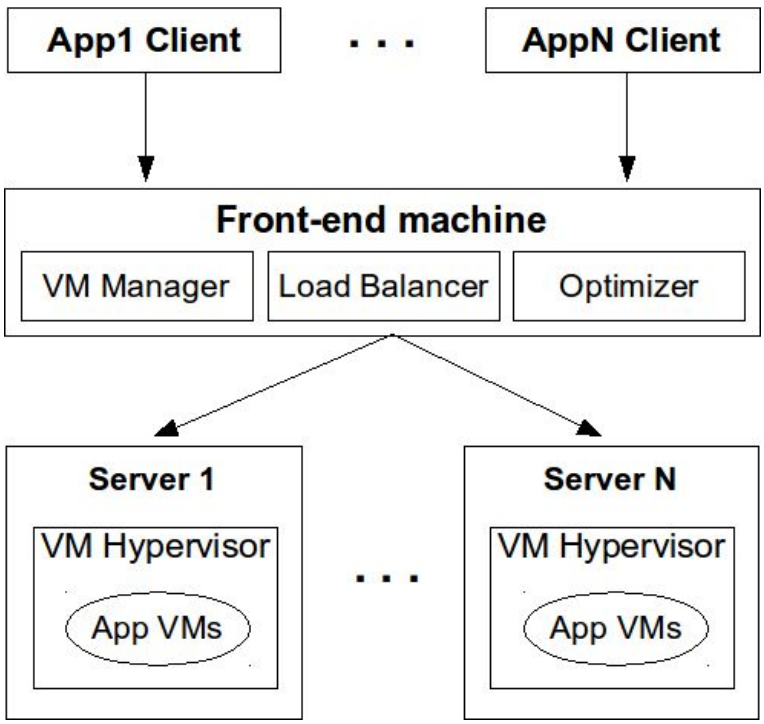

Figure 3.1: Server cluster architecture

The front-end machine is a key component in the architecture including three entities: (a) VM manager, (b) Load balancer, and (c) Optimizer. The `VM Manager` is

implemented using the OpenNebula toolkit [53] which enables the management of the VMs in the cluster, such as deployment, turning VMs on/off and monitoring. The `Load Balancer` implements a weighted round-robin scheduler strategy provided by the Apache's `mod_proxy_balancer` module [75]. Finally, the `Optimizer` is designed to monitor and configure the virtualized cluster. It consists of an external module implemented in Python [63] that relies on the primitives provided by the `VM Manager` and `Load Balancer` modules. The goal of the `Optimizer` is to dynamically configure the processors (using DVFS) and allocate the applications over the processor's cluster, in order to reduce power consumption, while meeting the application's performance requirements (see details of the overall optimization scheme in [61]). Note that this architecture was implemented by other researcher in our group. We use the architecture to perform the experiments presented in Section 5.1 and, in these experiments, we do not use the DVFS support.

3.1.1 Testbed for experiments in the basic architecture

The testbed platform used in the experiments presented in Section 5.1 is described in Figure 3.2. The web requests from the clients are redirected to the corresponding VMs that run the web servers on physical machines called *workers*. Each VM has a copy of a simple CPU-bound PHP script to characterize a web application. We define two different applications in the cluster, named App1 and App2. To generate the workload for each application, we use two machines with the httperf tool [25]. The load generator machines `Camburi` and `Cumulus` are physically connected via a gigabit switch. The worker machines `Maxwell` and `Edison` are connected via another gigabit switch. The front-end machine `Henry` has two gigabit network interfaces, each one connected to one of the switches.

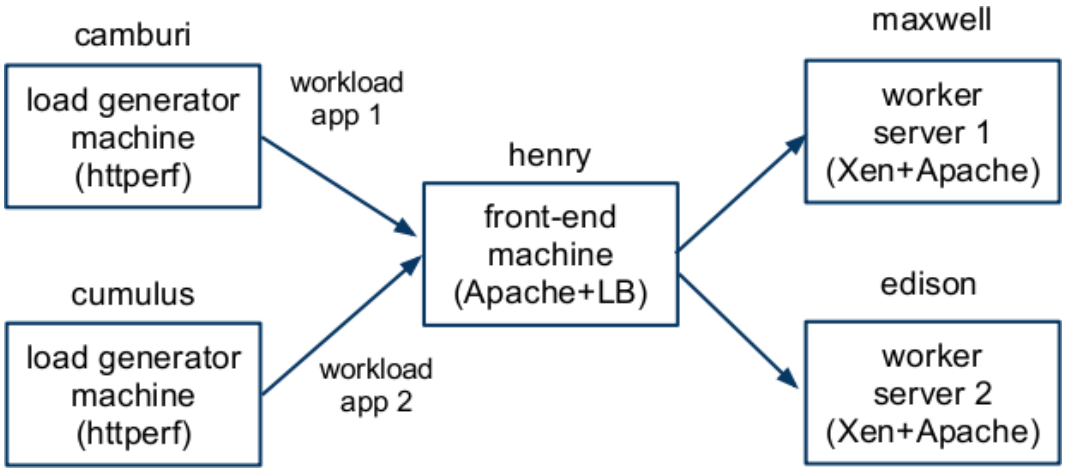

Figure 3.2: Basic cluster testbed setup

3.2 Multi-tier cluster architecture

After the experiments in which we evaluated the VM migration and replication costs, we have used the architecture, presented in Figure 3.3. In this architecture, each of the tiers may involve multiple servers running on multiple processors. That is, a request can be processed by one or more servers in each tier.

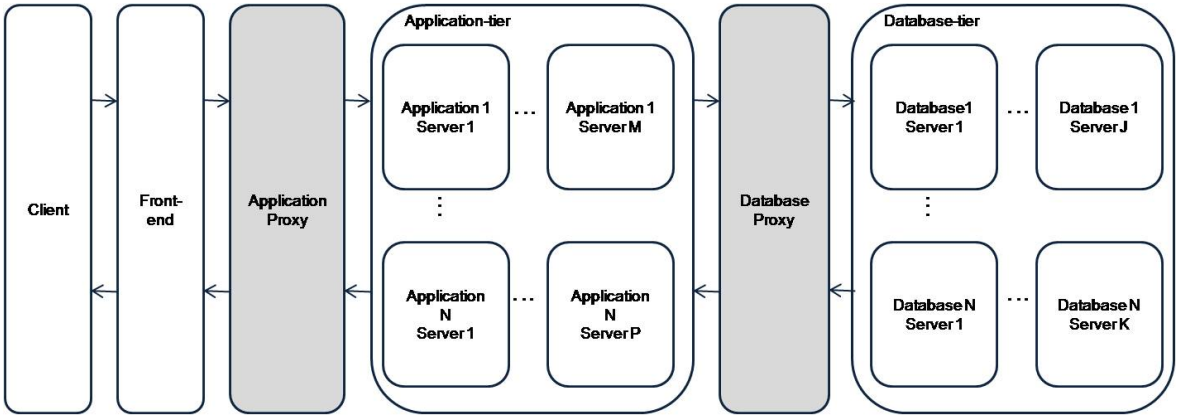

Figure 3.3: System architecture

Our architecture assumes that both application and database tiers are implemented by virtual machines (VMs), that can be configured (i.e., added, removed or even dimensioned) dynamically. Thus, resources can be allocated/deallocated for both application and database tiers. Specifically, our architecture uses the following technologies: Apache proxy [19] and MySQL proxy [46].

Apache is a freely available Web server that is distributed under an "open source" license. Because the source code is freely available, anyone can adapt the server for specific needs, and there is a large public library of Apache add-ons. MySQL is an open source relational database management system. It is based on the structured query language (SQL), which is used for adding, removing, and modifying information in the database. MySQL can be used for a variety of applications, but is most commonly found on Web servers [74]. A website that uses MySQL may include Web pages that access information from a database. These pages are often referred to as "dynamic", meaning the content of each page is generated from a database as the page loads.

3.2.1 Testbed for multi-tier applications

In this multi-tier architecture, the requests generated by clients are intercepted by the application proxy running in the front-end machine and redirected to an application

server. Each application server runs on top of a VM containing a copy of a RuBiS PHP version [67], that simulates an eBay site. An application server does not directly access a database server. A MySQL Proxy is interposed between applications and databases servers running on top of VMs. Each database server contains a copy of a MySQL database that stores tables containing information about users and items available in the web site. That is, the request is generated by the *Client* and received by the *Front-end* that directs it to the *Application proxy*. Thus, the request is sent to an *Application server*. The application server generates dynamic page content and query-language calls the database. Nevertheless, the query-language calls are intercepted by the *Database proxy* that redirects it to a *Database server*. Figure 3.4 presents the testbed used in our experiments. Note that in our experiments, for simplicity, a physical machine can run both application and database VMs. However, to optimize resources utilization, it would be interest to run application and database tiers in different physical machines. The physical machines configuration is presented in the Appendix C.

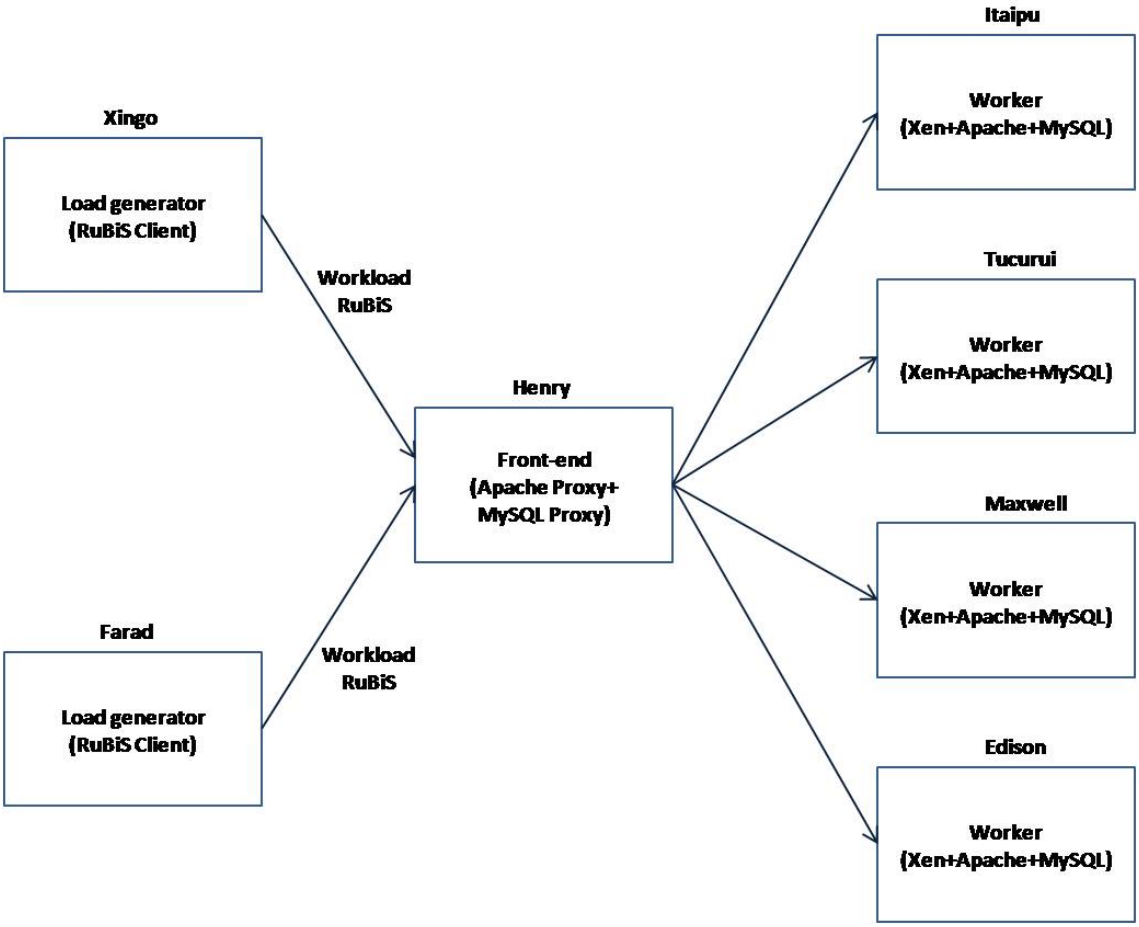

Figure 3.4: Cluster testbed setup

The front-end layer is not virtualized and performs low-latency functions such as

request forwarding and load balancing. One front-end tier can typically route requests to several servers of application and database tiers [40]. We assume that the physical machine in this layer has enough resources to perform its role and will not fail, although it could be replicated to scale and to achieve fault tolerance. In addition, the front-end runs two modules. The first, termed *Monitor*, that constantly monitors the CPU utilization of the worker machines and VMs. The second, termed *Manager*, that has a list of physical machines and VMs, and uses the information about CPU utilization, collected by the *Monitor*, to allocate/deallocate resources, i.e., to add or remove VMs, for both the application and database tiers. The Manager also uses weights available at the Apache Proxy to balance the amount of work to be done by the server in the application-tier. In the database-tier, the MySQL Proxy sends the write requests to the master database and the read requests are balanced between slaves databases, if there are more than one slave. If there is no active slave, all requests are directed to the master database. Our architecture is implemented in the physical machines as presented in Figure 3.5.

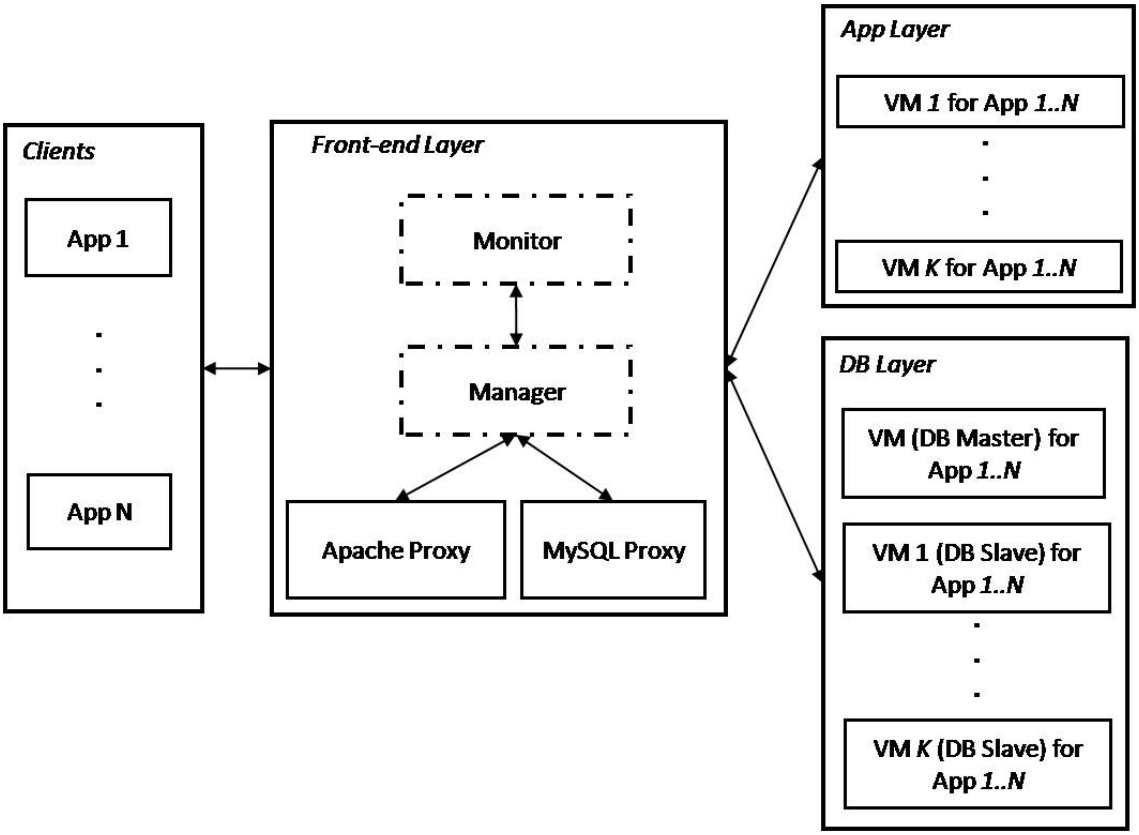

Figure 3.5: System implementation

3.3 Workload generation

To generate workload for our multi-tier architecture, we use the RuBiS benchmark tool [67]. RuBiS simulates eBay, an online auction and shopping web site in which people and businesses buy and sell a broad variety of goods and services worldwide [14]. Thus, clients perform read-only interactions with the website, and read-write interactions that modify the database. Several recent research works took advantage of the RuBiS benchmark, among them [9, 29, 34, 54, 70, 71, 77] which used the tool to evaluate their proposals.

In the context of a web site, a client is a customer who accesses a web site similar to eBay. For each customer, the client emulator opens a persistent HTTP connection to the Web server and closes it at the end of the session. During the course of a session, the customer makes some interactions, browsing the site. The auction site, simulated by RuBiS, defines 26 interactions that can be performed by clients. Among the most important ones are browsing items by category or region, bidding, buying or selling items, leaving comments on other users, and consulting one's own user page (known as myEbay on eBay). Browsing items also includes consulting the bid history and the seller's information.

The interactions a client does are determined by a state transition table that specifies the probability the client has to go to each available page from the page the customer is currently accessing. A think time between interactions and session time is generated from a negative exponential probability distribution. The RuBis benchmark defines two workload mixes: a browsing mix made up of pure read-only interactions and a bidding mix that includes 15% read-write interactions. In this work we used the bidding mix which is considered the most representative of an auction site workload. In the RuBis benchmark, workload variations on a web server are considered by varying the number of clients accessing the web site.

In the original RuBis benchmark tool, the number of clients is defined by the user before the experiment starts. The user needs also to assign three time parameters: *ramp up*, *session run time* and *ramp down*. The total time that will be taken to conduct the experiment will be given by the sum of these three parameters. The load will increase in the ramp up time until reaches the peak at the session run time. This peak will be maintained during the session run time until the ramp down starts, when the load decreases. The experiment finishes at the end of the ramp down time.

However, one can notice that Rubis is restricted to a semi-static number of clients which can only be increased and decreased following a ramp up/down style during a

given experiment. In order to add flexibility, in this work we have extended the RuBiS benchmark tool allowing it to produce arbitrary and more realistic benchmarks for server systems. Unlike the original RuBis benchmark, our extended version [51] does not uses the ramps values. The load variation occurs according to the number of clients and the experiment duration will be the time specified to simulate the input trace.

Our extended benchmark tool works as presented in Figure 3.6. An input trace is read and stored in a table. It is configured a period `P`, which is the time between the increment or decrement of client sessions. The given trace determines the set of clients to be simulated accessing the RuBis web site over the time. The extended benchmark starts with the number of clients according to the target of clients read from the input trace. After starting this number of clients the extended benchmark sleeps for a period `P`. The already started clients continue to run while no more clients are created or removed until the end of the period `P`. After each period P, we check how many clients are currently active in the system, and then we calculate how many clients should be added or removed to achieve the new target read from the input trace.

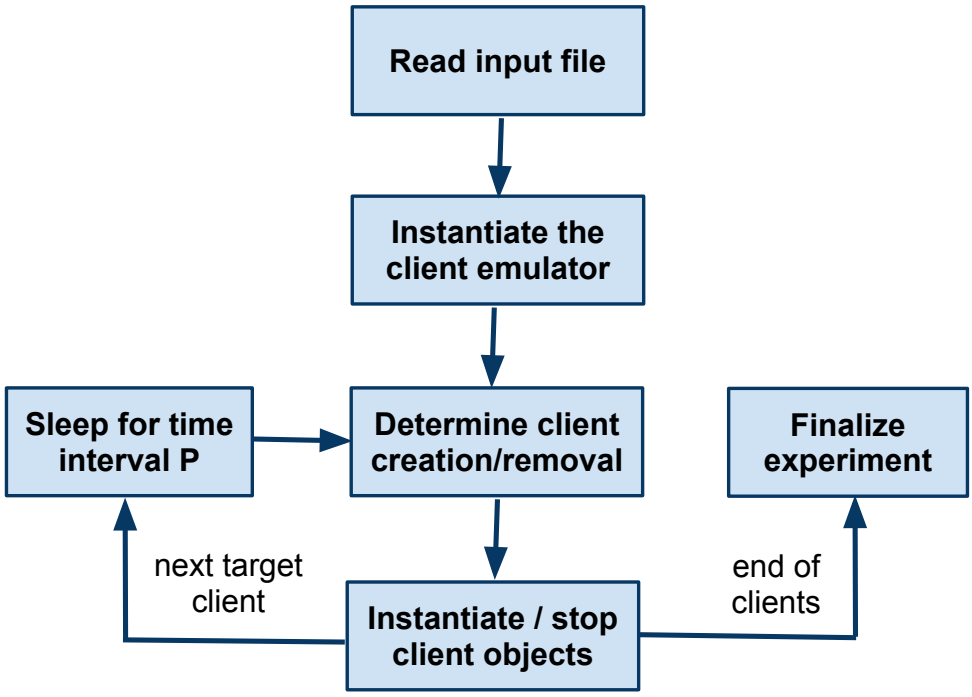

Figure 3.6: Loop of control for client management

3.4 Summary

In this Chapter we have presented the architectures used in the experiments described in Chapter 5. The first architecture, presented in Section 3.1, is used in the experiments presented in Section 5.1 and aims to provide a means to evaluate the disruption on the quality-of-service provided by applications during migration and replication activities. The second architecture, presented in Section 3.2, is used to perform basic experiments, presented in Section 5.2, with a multi-tier application. It was also described in this Chapter the benchmark tool used in our experiments.

Chapter 4

Implementation Mechanisms

In this Chapter we describe the *live migration* mechanism provided by VMMs, that is used in the experiments presented in Section 4.1. We also describe the mechanisms used in our experiments for VM replication both in the application and database tiers. We also describe the mechanisms used for load balancing between replicated servers in both application and database tiers.

4.1 VM migration

Migration of a virtual machine is simply moving the VM running on a physical machine to another physical machine [62]. Current VMMs offer two kinds of migration: cold and live migration. The difference between them is that the VM stops running during cold migration. Otherwise, the VM keeps running most of the time during live migration; actually, it stops only for a few milliseconds at Stage 3. The live migration stages are presented in Figure 4.1 [12].

Notice that cold migration does not have the Stage 2 as in live migration. Notice also that caches in hardware are not migrated [79], which can lead to cache misses in the target machine and impact application's performance when performing migrations. As pointed out by [82], both source and destination machines need extra CPU cycles for the pre-copying process during the migration activities. Moreover, an additional amount of network bandwidth is consumed.

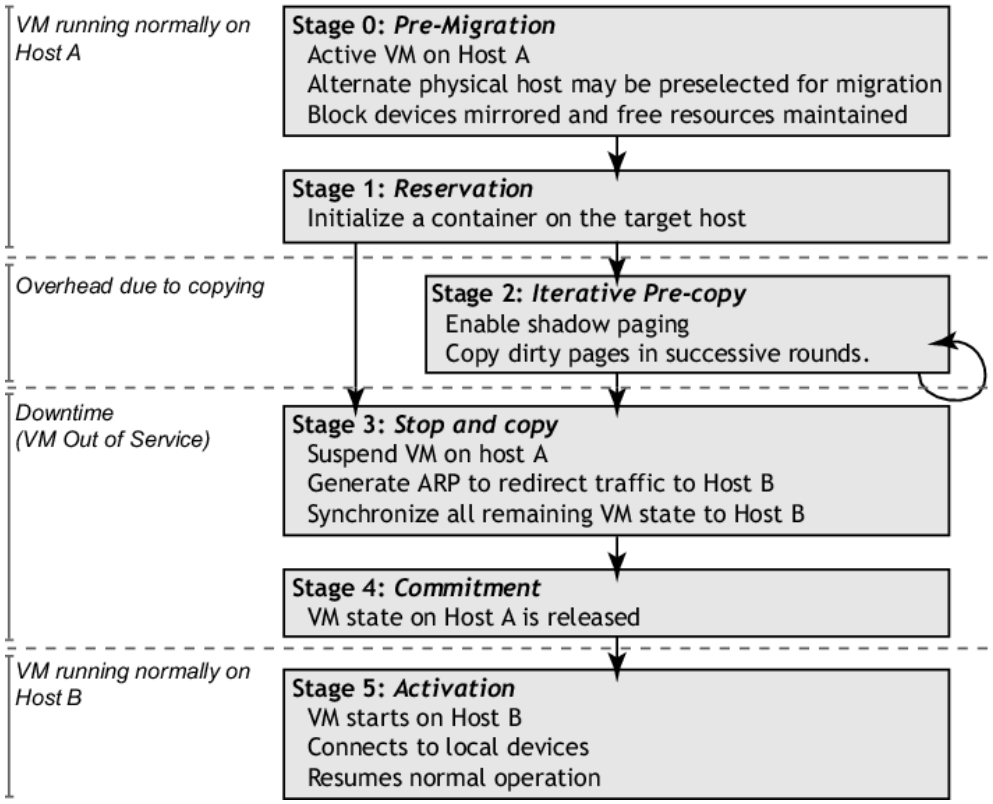

Figure 4.1: Migration timeline

4.2 VM Replication

VM replication would mean mirror the full state of a virtual machine real-time to another running virtual machine [15]. To implement the replication mechanism it is necessary to have a pool of servers able to provide the web-content. Server replication is achieved by balancing the incoming requests between the servers.

The applicability of replication is two-fold: first, for load balancing across the servers when an application cannot be supported by a single physical machine. In this case, servers remain active continuously and the load is balanced between them. Second, for moving a stateless server between physical machines avoiding the live migration mechanism and its costs [52]. The goal, in this case, is to maintain the QoS while the server is moved to the destination machine. In this case, the VM replication is achieved by redirecting small quantities of incoming requests until complete the redirection of all requests to the destination machine. It reduces the impact on the QoS due to cache misses during the load balancing. That is, in a reduced amount of redirected requests there is a reduced amount of cache misses when compared to a redirection of 100% of requests at once. In this work VM replication is used both for load balancing and because of the

reduced impact during replication activities.

4.3 Application-tier Replication

In our experiments an application has one or more VMs dedicated to the application-tier, that is, the application can be contained at a single VM or it can be replicated on top of more VMs. We used the Apache HTTP Server [75] for running the web application servers. Apache has a module termed *mod_proxy_balancer* [19] that is an extension for load balancing. The load-balancer runs in a front-end machine and redirects requests between the pool of VMs that act as back-end servers.

Weights are attributed to each back-end server. To do this, Apache provides the variable *lbfactor* that is how much traffic, in bytes, this worker will handle. This is a normalized value representing their "share" of the amount of work to be done, but instead of simply counting the number of requests, it is taken into account the amount of traffic this worker has seen. For example, if a balancer is configured as follows:

```
worker   a   b   c
lbfactor  1  2  1
```

Then we mean that we want b to process twice the amount of bytes than a or c should. It does not necessarily mean that b would handle twice as many requests, but it would process twice the I/O. Thus, the size of the request is applied to the weighting and selection algorithm.

When we need to replicate an application in a second back-end server, we start a VM replica at this server. At the moment this replica is ready for processing clients requests, its *status* on the load balancer manager is set up to *enable* and it is attributed a weight for this VM replica. Note that the weight is incremented in a loop, in small units until it reaches the target weight.

4.4 Database-tier Replication

There are several different database management systems. To simplify our task, we use MySQL, it provides two replication mechanisms: (1) Asynchronous replication [43], known as *replication* in MySQL, and (2) Synchronous replication, known as MySQL Cluster [44].

MySQL Cluster is a technology that enables clustering of in-memory databases in a shared-nothing system. MySQL Cluster is designed not to have any single point of failure. In a shared-nothing system, each component is expected to have its own memory and disk, and the use of shared storage mechanisms such as NFS (Network File System) is not supported [45]. We do not use MySQL Cluster because we use a NFS in our experiments and it is not supported in MySQL Cluster [45].

The asynchronous replication, used in our experiments, enables data from one MySQL database server (the master) to be replicated to one or more MySQL database servers (the slaves). Replication is based on the master server keeping track of all changes to its databases (updates, deletes, and so on) in its binary log. The binary log serves as a written record of all events that modify the database from the moment the server was started. Each slave that connects to the master requests a copy of the binary log. That is, it pulls the data from the master, rather than the master pushing the data to the slave. The slave also replays the events from the binary log that it receives. Thus, tables are created or their structure modified, and data is inserted, deleted, and updated according to the changes that were originally made on the master [49]. This has the effect of repeating the original changes just as they were made on the master. Note that before the experiment starts, it is created a snapshot of the data in the master database, using the *mysqldump* tool [48], a backup program that can be used to dump a database for backup or transfer to another MySQL server. We then import this data into the slaves before starting the experiment. Thus, when the slave connects to the master it needs only to read the binary log from the master and to execute the events in the binary log on the slave's local database.

Because each slave is independent, the replaying of the changes from the master's binary log occurs independently of each slave that is connected to the master. In addition, because each slave receives a copy of the binary log only by requesting it from the master, the slave is able to read and update the copy of the database at its own pace and can start and stop the replication process at will without affecting the ability to update to the latest database status on either the master or slave side. In this environment, all writes and updates must take place on the master server. Reads, however, may take place on one or more slaves. This model can improve the performance of writes (since the master is dedicated to updates), while increasing read speed across the slaves.

4.4.1 MySQL Proxy

In our multi-tier architecture, each application has a master database to handle the write queries and one or more slaves to handle the read queries. The load balancing in the database layer uses the MySQL Proxy [46], that is a module provided by MySQL that communicates over the network using the MySQL network protocol and provides communication between one or more MySQL servers and one or more MySQL clients. Thus, it makes possible to direct applications requests to the proxy that schedules them between a pool of available application databases. It avoids the effort for changing the application's code.

MySQL Proxy interposes itself between the server and clients, passing queries from the clients to the MySQL Server and returning the responses from the MySQL Server to the appropriate client. MySQL Proxy should be populated with the list of available MySQL servers to use when distributing work. The MySQL Proxy distributes connections from clients to each server. Distribution is handled by a count for the number of connections distributed to each server. New connections are automatically sent to the server with the lowest count. If MySQL Proxy identifies that the slave is lagging behind the master for its replication threads, then the slave is automatically taken out of the list of available servers. Work will therefore be distributed to other MySQL servers within the slave replication group. If the replication delay decreases to an acceptable level, then the slave will be brought back in to the list of available MySQL servers [50].

In addition to the basic pass-through configuration, the MySQL Proxy is also capable of monitoring and altering the communication between the client and the server. Additional decisions about the routing of incoming connections to MySQL servers can be made based on the replication status.

4.5 Summary

The mechanisms described in this Chapter will be used in the experiments to allow the application replication and load balancing in the different tiers.

Chapter 5

Experimental evaluation

In this Chapter we present a set of experiments that we have performed in our testbed (described in Chapter 3). The experiments performed in Section 5.1 aim to evaluate the migration mechanisms provided by VMMs and the replication mechanism. Thus, we measure and analyze the disruption on the QoS (quality-of-service) provided by the applications, in terms of server-side response time and throughput, during dynamic allocation of virtual machines in a server cluster. Section 5.2 consists of those experiments performed in the multi-tier architecture. The goal is to use the mechanisms described in Chapter 4 to perform basic experiments using a three-tier (front-end/application/database) application. The response time is measured in our experiments as described in the Appendix B.

5.1 Dynamic allocation's costs

We performed a set of experiments in our testbed (described in Chapter 3) aiming to evaluate the application behavior during the course of migration and replication actions. We measure and analyze the disruptive impact on the QoS (quality-of-service) provided by the applications, by means of server-side response time and throughput, during dynamic allocation operations of virtual machines in a server cluster. In the first step, we used the Apache Benchmark (ab) [75] to measure the maximum number of requests per second that our physical machines can handle. We found that our worker machines (`maxwell` and `edison`) achieved a maximum of 1145 requests/sec for a typical PHP script web request with an average processing time of 6 milliseconds.

In the next step, we allocated two virtual machines (VMs) to run on the `maxwell` machine. Each VM had 256 MB of RAM, running an Apache 2.2 over Debian 4.0.

Since our applications are CPU-bound, this memory capacity was found suitable in our experiments in [52]. Note that the quantity of memory a VM is using may impact on how much time is needed to complete a migration [23], thus requiring major investigation in a future work.

The first VM runs the application `App1` and it uses 120% of the total CPU resources (considering a quad-core machine). The second VM, which runs the application `App2`, starts using 40% of the CPU resources. Then, we increase the `App2` workload demand until both VMs for `App1` and `App2` (along with `Dom0`) are using 300% of the physical CPU resources. After such a condition occurs we perform the actions described in the following experimental scenarios in order to maintain quality-of-service requirements. We ran experiments with three different scenarios: (a) cold migration, (b) live migration, and (c) replication. Each of the experiments showed similar results for repeated executions.

5.1.1 Cold migration

In this scenario, the cold migration mechanism is applied to move the `App2` VM to a physical machine with spare capacity (that is, from `maxwell` machine to `edison` machine). We observe that in the cold migration, the `App2` VM stops during the migration. Figure 5.1 shows the throughput, response time and CPU utilization for both App1 and App2 VMs during the course of the experiment. The experiments have approximately 10 minutes in duration.

We show that this kind of migration cannot be used in a soft real-time system because the VM being migrated stops during the course of migration. This is explicitly shown at the throughput curve of the application `App2`, which was migrated and then stopped for approximately 10 seconds. During the course of this kind of migration, the slowdown in the service was 100% because the execution of `App2` had been completely suspended and both response time and throughput measurements dropped to zero. In all scenarios, the drop in the throughput shows the instant in which the VM movement was performed. After the migration phase, the response time varied considerably reaching up to 8 seconds.

5.1.2 Live migration

In this scenario, we use live migration to move a VM to a new server machine without service interruption [12]. Besides not stopping the service during migration, we still need to maintain an acceptable quality-of-service in terms of application's response time. The

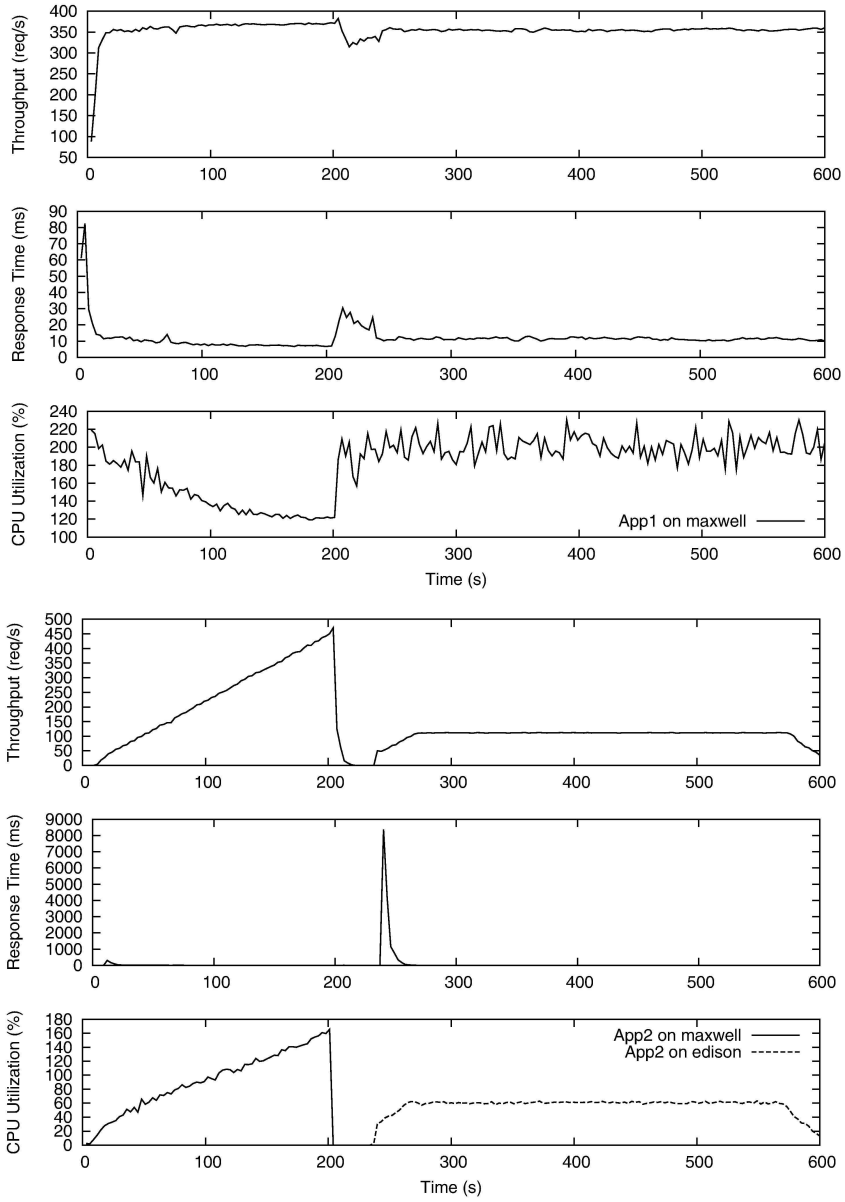

Figure 5.1: Execution of the *cold migration* scenario: App1 (top) and App2 (bottom)

goal of the experiment (shown in Figure 5.2) was to evaluate the impact of applying the live migration mechanism.

As would be expected, unlike cold migration, we observe that in live migration the VM is not paused during the migration. In Figure 5.2, the application `App2` was migrated with no noticeable interruption to the service. However, we noticed that the response time for `App2` increased substantially during the course of migration. For instance, the response time measured for `App2` raised from 11 milliseconds to 300 milliseconds on average for a period of 3 seconds. The throughput measurement was also affected by the migration. We also notice that `App1` was slightly affected when migration was performed. The slowdown in the App2 throughput was 61.5% (from 414.1 req/s to 159.5 req/s). We can note that

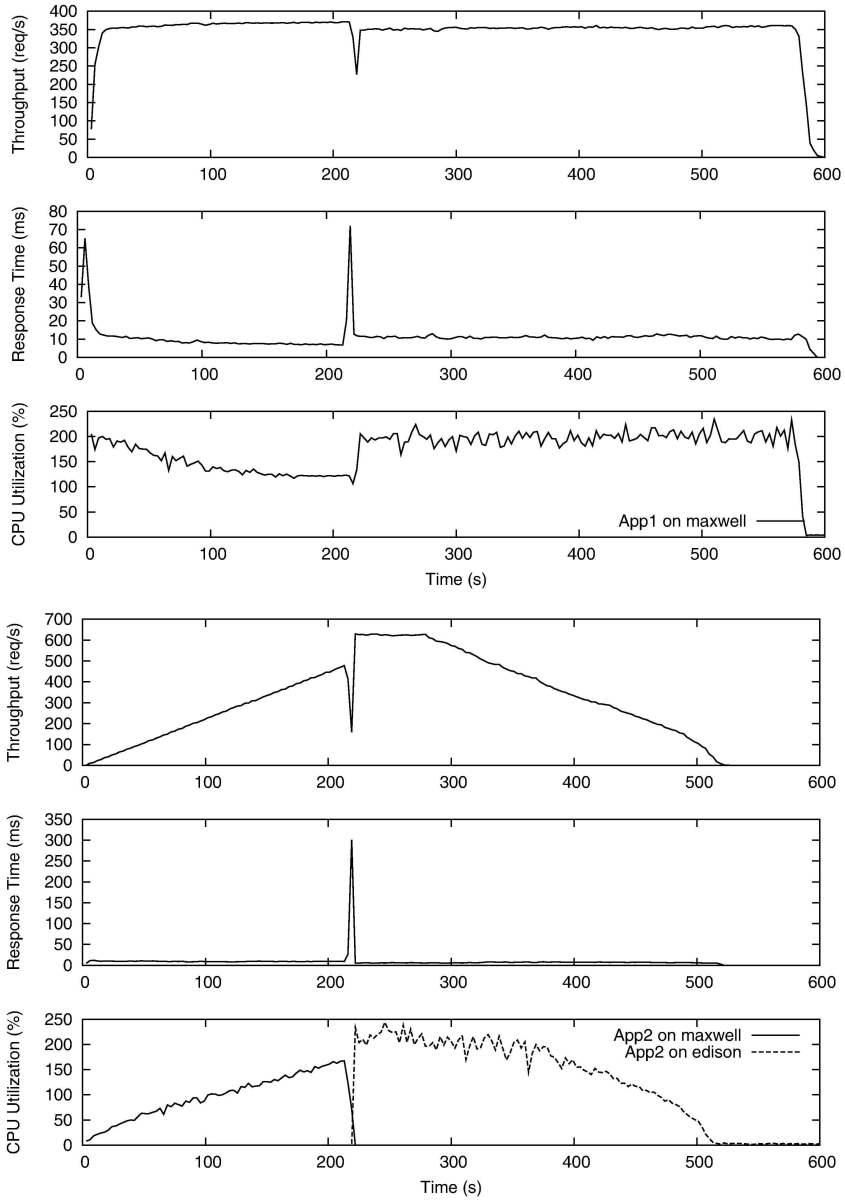

Figure 5.2: Execution of the *live migration* scenario: App1 (top) and App2 (bottom)

the disruptions observed when performing dynamic changes through live migration last a short time and are basically unavoidable.

5.1.3 Replication

We also have investigated an alternative approach using replication to help minimize these disruptive impacts in the QoS of the applications. In this scenario, we consider creating and deploying a new VM replica for the application `App2` on the destination server. At the moment the new replica is ready for processing the client requests, we start redirecting the requests to this new replica.

The goal of the experiment for this scenario (see Figure 5.3) is to evaluate the response time impact compared to the live migration scenario. Specifically, we have measured the response time (and throughput) during the replication process to identify potential practical issues such as the time delay to boot a new VM and the stabilization time of the load balancer when transferring requests to the new replica.

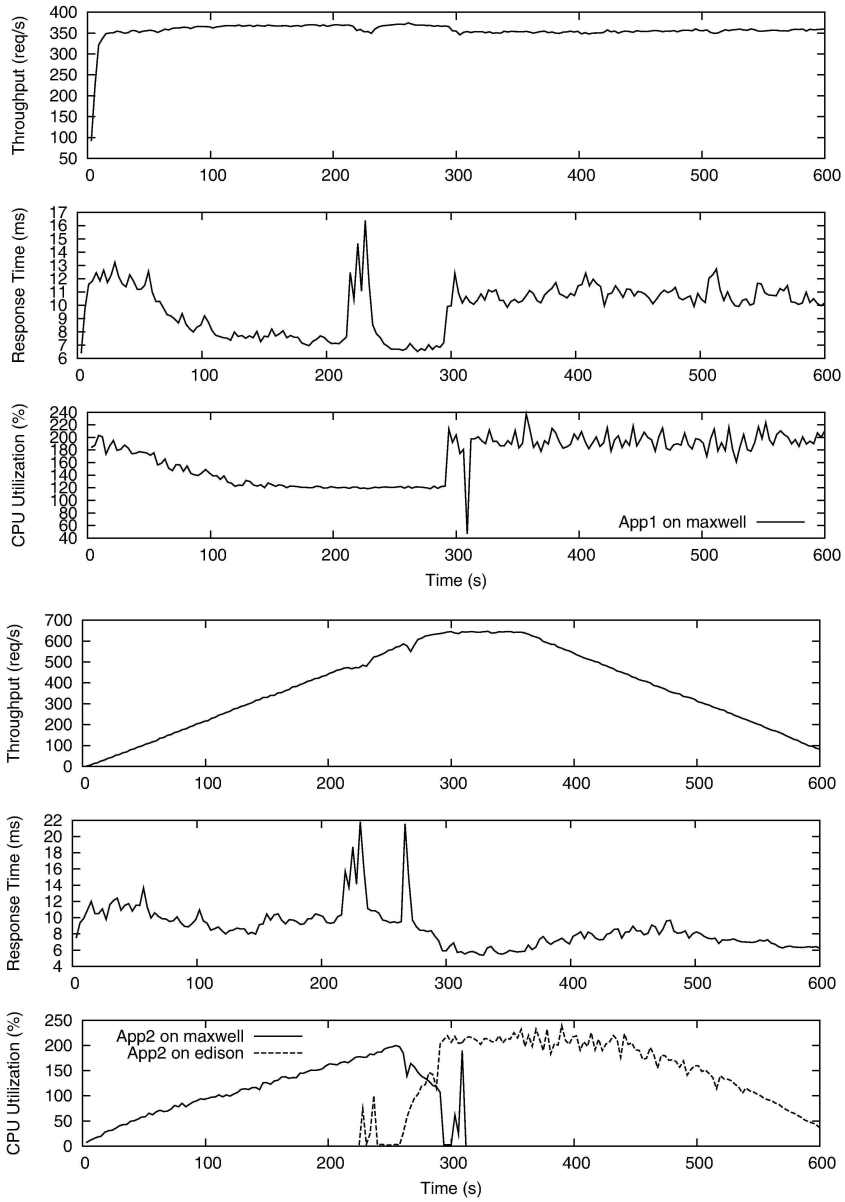

Figure 5.3: Execution of the *replication* scenario: App1 (top) and App2 (bottom)

The use of replication shows improvements compared to the live migration, for example, by analyzing the decrease observed in the throughput in Figure 5.3, contrasting it to Figure 5.2. Specifically, the execution behavior of both applications `App1` and `App2` was found more stable when using replication in comparison to live migration. For example, the response time observed for `App2`, which was replicated in another server machine,

increased from 10 milliseconds to 22 milliseconds. In addition, the throughput observed had a very slightly drop from 473 req/s to 467 req/s. We can also observe that `App1` was less affected when replication was performed instead of migration.

The basic steps for replication consists of (1) booting a new VM replica and (2) redirecting the requests to the new replica. According to our experiments, in Section 5.1, the time needed to boot a VM is in between 25 and 40 seconds[1], which may be a bit longer than 10 seconds, on average, observed in the live migration in Scenario 2. In this experiment, when the physical machine *Maxwell* achieves 300% of CPU utilization, we boot the new VM replica in the physical machine *Edison*.

The phase of redirecting requests for the new replica raised an implementation issue that needs to be addressed. We have observed that if all the current requests were abruptly redirected to the new VM replica it would take a long time to get both throughput and response time stable. The sticking point is that Apache has a single control process responsible for launching child processes (daemons) which listen for connections and serve their requests when they arrive. To tackle this redirecting bottleneck, we used a configurable mechanism termed "spare servers" [75]; setting the Apache configuration to maintain a suitable set of idle server daemons, which standby ready to serve incoming requests[2]. In this way, clients do not need to wait a long time for a new child processes to be forked before their requests can be served. Moreover, redirecting the requests at a slower rate we achieved further reduction of the server settling time. In our experiments, we redirected 10% of the requests each time, until 100% of the requests were redirected to the VM replica.

5.1.4 Section Summary

In this Section, we have presented a virtualized server environment targeted for dynamic deployment and allocation of VMs to physical machines. Our goal was to carry out experiments to evaluate the performance impact in terms of response time and throughput of applications during the course of VM migration and replication.

The replication steps involved starting a VM replica in the target host and redirecting requests to the new VM replica. Our results showed that by using replication we can minimize some performance disruption in the response time and throughput incurred

[1]For the experiments performed in Section 5.2 we use the Ubuntu Linux 10.04 that boots in 10s.

[2]This can increase power consumption but in this experiment we are interested in maintaining the QoS metrics only.

during migration. Finally, the evaluation described in this Section will help with the implementation of the dynamic optimization model and strategy for power and performance management of virtualized web clusters [61], proposed by other researcher in our group.

5.2 Management of multi-tier architecture

In this Section, we use the architecture described in Section 3.2 to perform experiments using a multi-tier application. Our goal is to carry out basic experiments using the replication mechanisms for both application and database tiers. The capability to replicate both application and database tiers will facilitate the use of the dynamic optimization model for power and performance management, proposed by our group in [55, 58, 59, 60, 61][3]. In these experiments, we also aim to use the replication mechanisms to keep the CPU utilization under a given threshold.

In Subsection 5.2.1 we briefly relate approaches employed by several authors concerned to resource provisioning and an experiment is performed to evaluate one of these approaches. In Subsection 5.2.2, it is evaluated the CPU utilization maintenance during an experiment that uses the replication mechanisms described in Chapter 4. We also aim to show in this experiment that is possible to balance the workload among the servers in the cluster, thus allowing the use of the dynamic optimization model for power and performance management, proposed by our group. In Subsection 5.2.3 it is suggested to control the memory (in addition to the CPU) during experiments. In Subsection 5.2.4 we provide a brief description of how to improve energy-efficiency in a system using DFVS and a simple experiment is performed aiming to evaluate the difference in power consumption between a server using DFVS and without using it.

5.2.1 Provisioning a multi-tier application

Dynamic provisioning of resources, allocation and deallocation of servers, for replicated applications has been studied by many researchers in the context of single-tier applications, of which clustered servers are the most common example.

As observed in [77], most authors focus only on the common bottlenecked tier, allocating new resources only for the most compute-intensive tier of the application [80]. In the web site implemented by RuBiS, the application-tier is the most compute intensive tier.

[3]Currently this optimization model is used, by other researcher in our group, in the architecture described in Section 3.1.

Aiming to evaluate the application behavior, we performed the experiment, presented in Figure 5.4, in which we only use the replication mechanisms to allocate new resources for the application-tier. Our goal is to evaluate if a fixed resource allocation for one of the tiers can cause disruptions to the application. In the experiment, we used one load generator machine (Xingo) and three physical machines (*Itaipu*, *Tucurui* and *Maxwell*) each of them running a VM (*Application*, *Database* and *App Replica*, respectively).

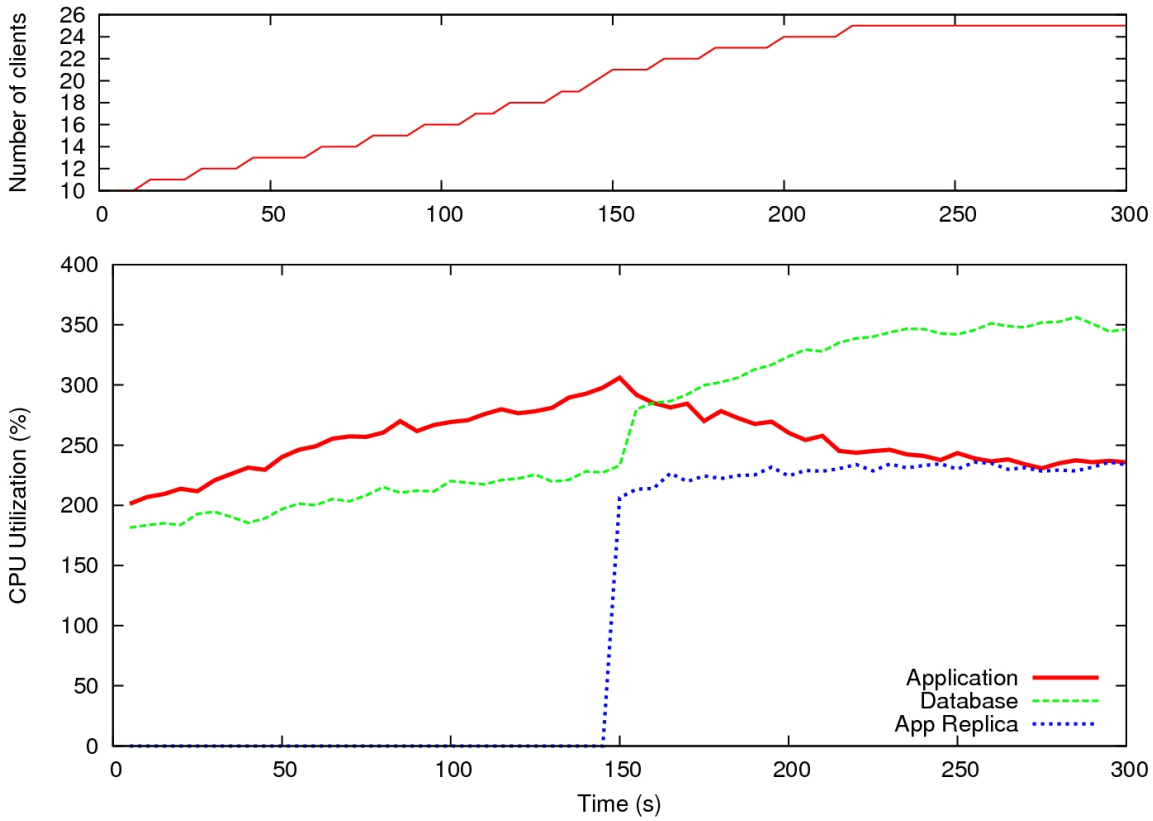

Figure 5.4: Resource provisioning

In this experiment, the number of clients increases and at the moment the physical machine (*Itaipu*) running the application server reaches 300% of CPU utilization, an application replica was started in another physical machine (*Maxwell*). Even though plenty of capacity was available at the database-tier, the increasing of newly arriving sessions cause the physical machine (*Tucurui*) running the database to reach 300% of CPU utilization. As shown in Figure 5.4, the database-tier gets overloaded while the application-tier consisting of two servers has free CPU cycles. Thus, focusing only on the the commonly bottlenecked tier is not adequate, since the bottleneck will eventually shift to other tiers.

A straightforward extension is to employ single-tier provisioning methods at each tier of the multi-tier application. This enables VM provisioning decisions to be made

independently at each tier based on local observations. The idea is to provision additional servers at a tier when the CPU utilization reaches a given threshold. Thus, we use the replication mechanisms for both application and database tiers, preventing degradation due to bottleneck in specific tiers.

5.2.2 Replication mechanisms

In this Subsection we perform an experiment in which we use the replication mechanisms, described in Chapter 4. Our goal is to maintain the CPU utilization of the physical machines under the threshold of 300% of CPU utilization. To do this, in the experiment, we vary the number of accesses to the web site leading to the necessity to use VM replicas in both tiers. In the experiment, we increased the workload in steps, thus resources were allocated to handle the incoming user requests. After the allocation of our four worker machines (Itaipu, Tucurui, Maxwell and Edison), the workload decreased in steps and resources were deallocated. It is shown, in Figure 5.5, the allocation of the physical machines, the number of clients and the response time during the experiment. The virtual machines allocation/deallocation is presented in Figure 5.6.

The experiment starts with one load generator machine (*Xingo*), one application VM (running on *Itaipu*) and one master database VM (running on *Tucurui*). At time=60s, the workload increased. Thus, a new worker machine (*Maxwell*) was added to the cluster[4] and the *Application VM2* was started on it. The increase in requests saturated the database-tier, and Tucurui (database master along with Dom0) achieved the threshold of 300% of CPU utilization. To handle the increase in the amount of user requests, a database slave (Slave1) was started at time=200s. As Maxwell had free CPU cycles, the database Slave1 was started on it. At this moment, all read requests were sent to the database Slave1 while the write requests were sent to the Master database, reducing the CPU utilization on the Master database.

To provide a new workload increase, at time=250s, we used a second load generator machine (*Farad*) and a new worker machine (*Edison*) was added to the cluster. A third application VM, *Application VM3*, was started on it aiming to generate more user requests to the web site. At time=300s, Maxwell, that was running the database Slave1 (and thus handling the read requests) achieved the threshold of 300% of CPU utilization. To increase the capacity to handle read requests, the *database Slave2* was started on Edison. At this

[4]In this text, "*added to the cluster*" does not means turn the physical machine on. Actually, all physical machines remain turned on during all the experiment. Thus, "added to the cluster" means start using the physical machine to run one or more VMs.

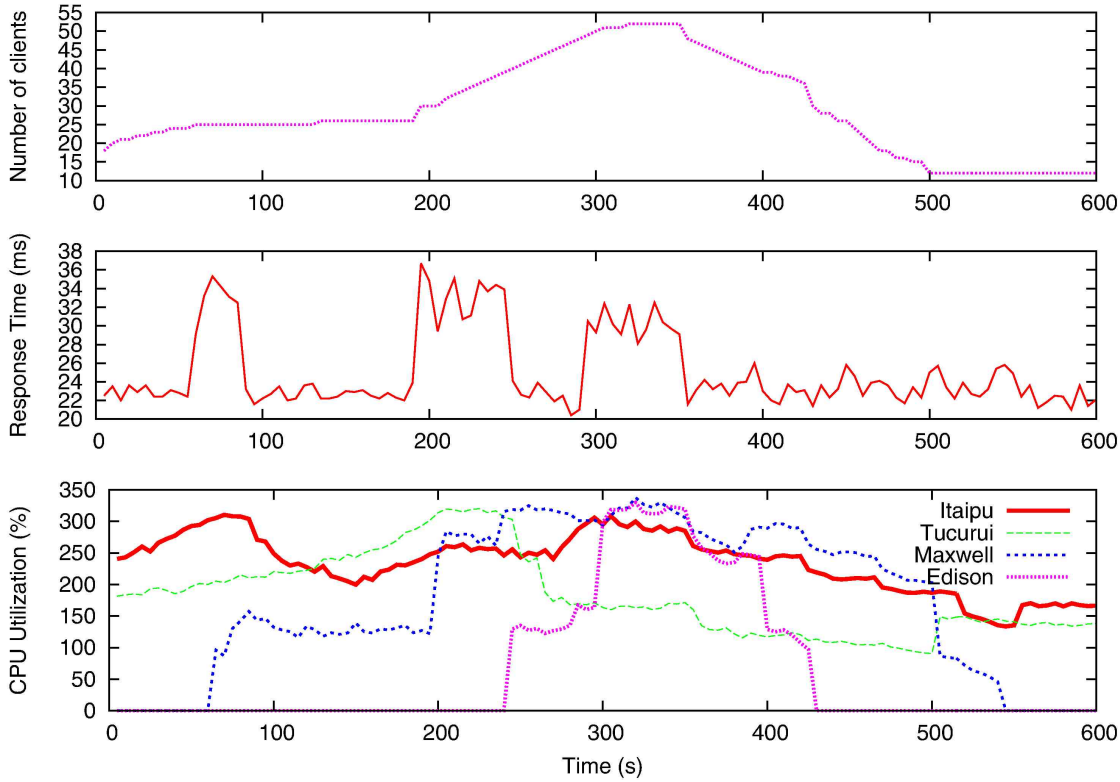

Figure 5.5: Physical machines allocation

moment, read requests were balanced between Slave1 and Slave2.

After the workload reach a peak of 52 clients at time around 320s, the workload decreased and the first virtual machine (*Slave2*) is deallocated at time=400s. At time=425s *Application VM3* was deallocated and *Edison* was removed from the cluster[5]. As the workload continued to decrease, the database Slave1 was deallocated at time around 500s and the Application VM2 was deallocated at time=550s. Thus, Maxwell was removed from the cluster at time=550s and the experiment finished at time=600s.

We observed in this experiment that resources can be allocated/deallocated in our architecture for both the application and the database tiers by using the replication mechanisms. Using these mechanisms, we maintained the CPU utilization under a given threshold (300%) during the experiment. In addition, the replication mechanisms allow the optimization model for power and performance management of virtualized clusters,

[5]In this text, "*removed from the cluster*" means that the physical machine is no longer used during the rest of the experiment.

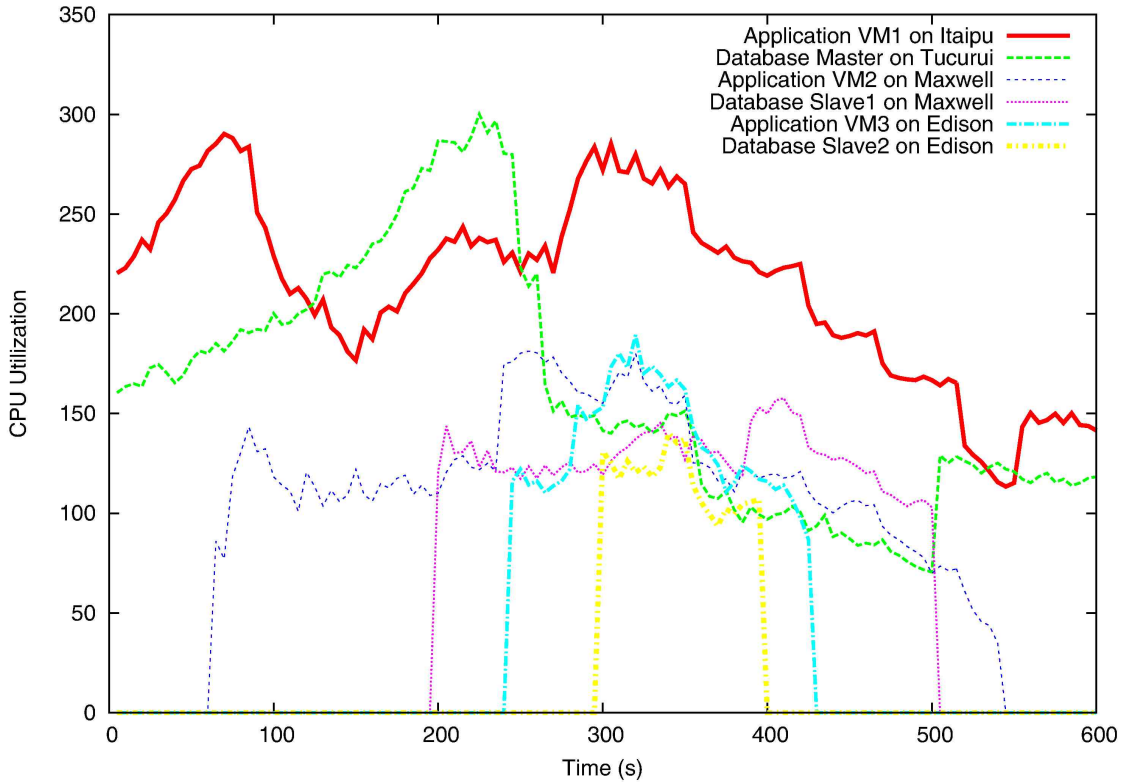

Figure 5.6: Adding/Removing resources to keep CPU utilization

proposed by our group, to be used in this architecture. Note that for a large system, a single master can became a bottleneck. Thus, would be interesting to change the MySQL Master-Slave replication to another database replication policy that supports multiple master databases.

5.2.3 Keeping CPU utilization under a threshold

In order to meet fair response time requirements, it is important to keep the CPU utilization under a given threshold, as presented in Figure 2.1. The experiments in this Subsection aim to show that is possible to keep the CPU utilization under a given threshold by using different mechanisms. We focus on CPU utilization in this work since CPU is often the key resource in determining the performance of multi-tier applications [9]. As described in previous researches, guarantying the CPU utilization under a certain level, one can guarantee the response time of applications.

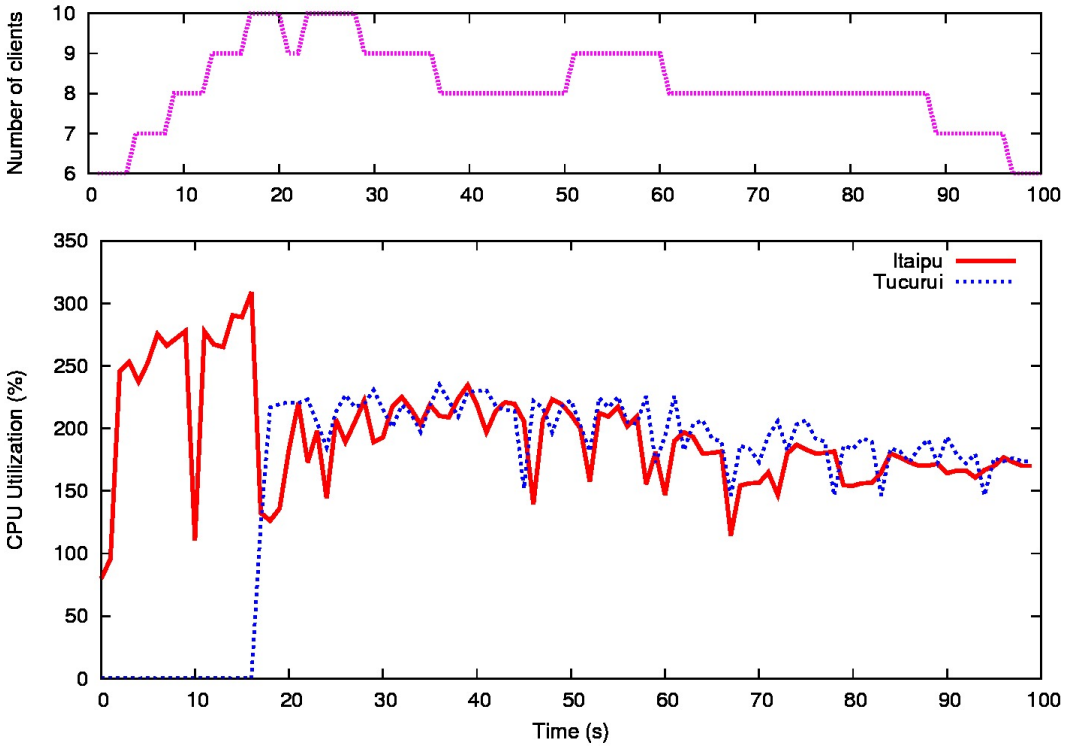

Figure 5.7: Keeping CPU utilization by relocating a VM

To maintain the CPU utilization under the threshold of 300% in the experiment presented in Figure 5.7, we used the live migration mechanism to migrate a VM to another physical machine when the CPU utilization on the first physical machine achieved the given threshold. In this simple experiment, both application and database VMs start running in the same physical machine (*Itaipu*) and when the physical machine is almost overloaded, the database VM was live migrated to another physical machine (*Tucurui*) and the CPU utilization in the first physical machine (*Itaipu*) decreased. In the experiment presented in Figure 5.7, the Application VM had 256Mb of RAM, that was found suitable for our experiments since our application is CPU-bound, and the database VM had 512 Mb of RAM.

However, in addition to the live migration mechanism used in this experiment, and the replication mechanism described in this work, other approaches can be required to maintain CPU utilization under a given threshold, thus, preventing resource relocations. It is known that, when the memory capacity increases, it is possible to store more data in the buffer preventing accesses to the disk. We were interested in evaluating if increasing the memory capacity for the individual VMs (application or database) it would be possible to decrease the CPU utilization.

We repeated the experiment presented in Figure 5.7, increasing the memory available

for the application VM to 1 Gb of RAM. The database VM had 512 Mb of RAM. In this memory configuration, the physical machine has not achieved the given threshold and was not necessary to relocate a VM to another physical machine.

As shown in Figure 5.8, Itaipu supported both application and database VMs under the CPU threshold and Tucurui remained unused during the experiment. Thus, we found that using RuBiS, if we increase the memory available for the application-tier, the application will achieve more cache hits and it will allow the application to handle client's requests with a small amount of accesses to the database. In our experiments, it was not possible to reduce the CPU utilization by increasing the memory available for the database.

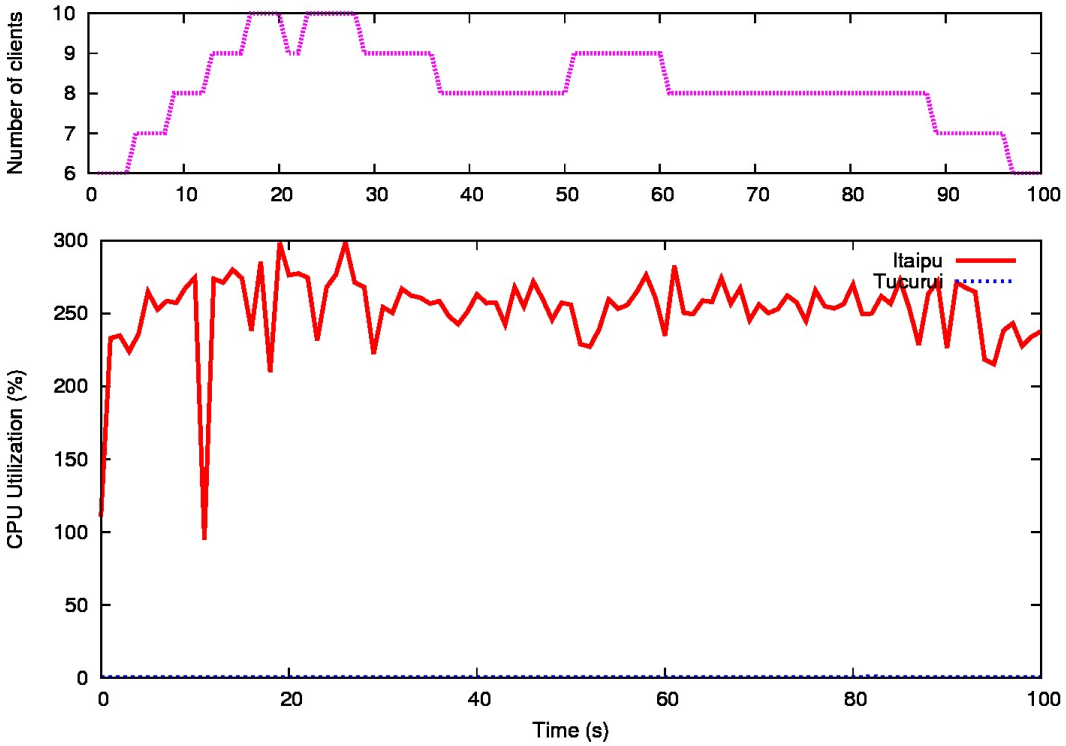

Figure 5.8: Reducing CPU utilization by increasing Application's memory

As observed in [22], runtime re-allocation of memory among multiple VMs has not been widely studied. They found that as the memory allocation becomes smaller, the total CPU consumption goes up, mainly due to the extra paging activities by the guest OS. They found that the mean response time increases sharply as the memory utilization goes beyond 90% because when memory utilization is high, the guest operating system experiences significant memory pressure and starts reclaiming memory by paging a portion of the application memory to disk [7]. This will lead to a higher number of page faults when the application tries to access the main memory, resulting in higher latency for the

application. Thus, the investigation on how to dynamically allocate memory resources, poses a great challenge for future research.

5.2.4 Improving energy efficiency in virtualized environments

As briefly described in Chapter 1, the virtualization technology leads to an increased resource utilization and energy savings. However, to improve energy efficiency in our virtualized multi-tier architecture, we intend to use the dynamic frequency and voltage scaling (DVFS) [30] technique, that is a technique whereby the frequency of a microprocessor can be automatically adjusted "on-the-fly", either to conserve power or to reduce the amount of heat generated by the chip [84]. It can be used to decrease energy and cooling costs for lightly loaded machines.

Current processor architectures provide the DVFS capability. Both Intel and AMD have their supports for DVFS technologies, SpeedStep [28] and PowerNow! [1], respectively. DVFS reduces the number of instructions a processor can issue in a given amount of time, thus reducing performance.

We present in this Subsection an experiment in which we aim to evaluate the difference in power consumption between a server using DFVS and without using it. Thus, we measured the CPU frequency during an experiment in which we increase the number of clients accessing the web site running on the physical machine *Itaipu*. We have also measured the CPU utilization during the experiment. Note that the CPU utilization measured accounts for the sum of utilization of all virtual machines running in the physical server. We performed the experiment in two scenarios. A scenario in which the DVFS is used, and a second scenario in which the DVFS is not used.

In the experiment presented in Figure 5.9, we use the DVFS according to an on-demand policy. That is, the CPU frequency increases or decreases scaling the CPU capacity to handle the current workload. To do this, we used the On-demand Governor [26] available in the Linux operating system.

As observed in Figure 5.9, to support until 10 clients, the physical server achieved a higher CPU utilization in the scenario in which we used the DVFS. It was due to the frequencies used by the DVFS that tries to use the lowest possible CPU frequency to handle client requests. When the machine was overloaded (from 11 to 20 clients) the highest CPU frequency was used in both scenarios. The power consumption can be expressed as $P \approx f.V^2$, where f is the CPU frequency and V is the CPU voltage. Thus,

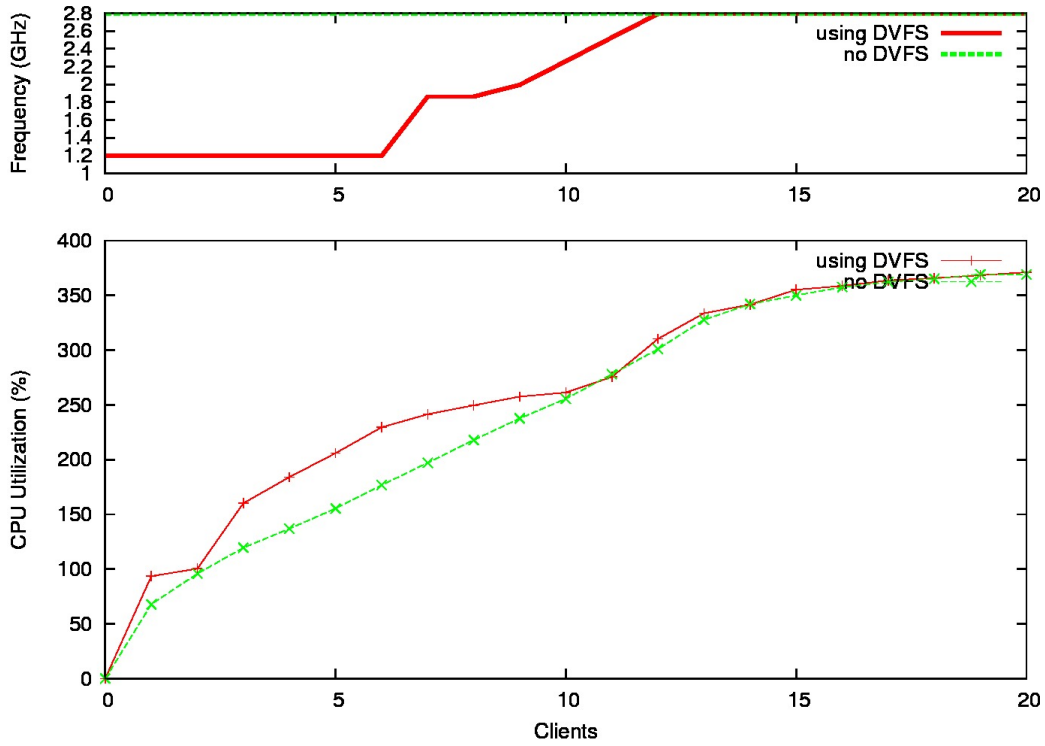

Figure 5.9: Comparison between a server using DVFS and without using it

the scenario in which the DVFS was not used, achieved a higher power consumption as it used the highest CPU frequency during the experiment. Hence, encouraging the use of the DVFS technique that has been shown to reduce energy consumption by about 10%-30% [24, 35, 65].

Moreover, the investigation on how to apply a cluster-level scheme to turn off unneeded servers poses a great challenge for future research as it can increase power savings to 40%-80% [4, 31]. One should note that the process to turn on a server can take a few minutes. Consequently, when using schemes to turn servers on/off, it is worthwhile to use a forecasting algorithm to predict the workload like in [69].

Thus, as a future work, we aim to adopt the dynamic optimization model for power and performance management of virtualized clusters presented in [55, 58, 59, 60, 61] in our multi-tier architecture. Currently, the optimization model is used, by other researcher in our group, only in the basic cluster architecture presented in this work. The optimization model deals with the problem of determining the most power efficient configuration (i.e., which servers must be active and their respective CPU frequencies) that can handle a certain set of application workloads [55].

5.2.5 Section Summary

In this Section, we have performed experiments using the mechanisms described in Chapter 4. Using these mechanisms the workload can be balanced among the servers in the cluster. Thus, helping the use of the optimization model [55, 58, 59, 60, 61], proposed by our group, that determines the most power efficient configuration (i.e., which servers must be active and their respective CPU frequencies) that can handle time-varying workloads for different applications in the cluster. It is also possible to keep the CPU utilization under a given threshold, thus maintaining the QoS metrics.

5.3 Summary

In this Chapter, we have performed experiments in which we evaluated the migration mechanisms provided by VMMs and the replication mechanism. The results showed that by using replication we can minimize some performance disruption in the response time and throughput incurred during migration. We have also performed basic experiments using a typical multi-tier architecture. Our goal was to provide an architecture in which both application and database tiers can be replicated, facilitating the use of the dynamic optimization model for power and performance management proposed by our group [55, 58, 59, 60, 61].

Chapter 6

Related work

Virtualization has achieved renewed interest in the last years. This technology has some features that can bring benefits in many areas, e.g., it can be applied aiming to achieve energy savings, to reduce the TCO, to isolate applications, etc. In this Chapter, we present some works in which authors apply virtual machine migration and replication mechanisms to support the execution of applications in virtualized environments. Note that, different from these works, we do not have an optimization model to dynamically allocate resources. Thus, we are not interested in compete with these works.

Authors in [29] aim to maintain the response time for multi-tier applications. To achieve this goal, they use VM replication in both application and database tiers, like us. However, there are two basic differences between our work and their work. (1) They dynamically manage resources based on response time requirements. When the response time indicates saturation, the system scales the application-tier or the database-tier, according to the bottlenecked tier. We differ from them, as we perform resource allocations when physical machines achieve a CPU utilization threshold.

Authors in [9] use an analytical model to capture the relationship between the application's high level goals (e.g., application's response time) and lower level goals (e.g., CPU share assigned to application and database tiers). For example, a component profile captures the relationship between an Apache web server's mean service rate and the CPU share allocated to it. Their static resource allocation takes into account the change in the number of users only, and assumes a fixed transaction mix. However, a more practical approach must handle workload changes in both the number of users and transaction mix. As pointed out in [13], a guideline regarding resource utilization for practitioners in real world is to keep peak utilization of resources, such as CPU, under a given threshold. As shown in [73], service times remain relatively stable when the CPU is not saturated.

Unlike [9], authors in [85] use a dynamic resource allocation. They apply a regression-based approach to provide a solution for capacity planning and resource provisioning of multi-tier applications under changing workloads conditions. They use a framework to approximate CPU demands of transactions on a given hardware. To capture the site behavior across time, they collect the application server access log observing a number of different transactions over monitoring windows of fixed length T. They use these observations over the current monitoring window to allocate resources for the next monitoring window. A limitation in their approach is that the resource allocation for the next monitoring window should not be applied to a very different workload mix compared to the mix it was derived from. They also observed in their experiments that the approximation of CPU demands at the application-tier is of higher accuracy than that at the database-tier. This reflects a higher variance in the CPU service time at the database-tier for the same transaction type.

To both track the CPU utilization of virtualized servers and to guide their resource allocations, authors in [33, 34] implemented a resource management scheme that integrates the Kalman filter [32] into feedback controllers to dynamically allocate CPU resources to multi-tier virtualized applications. The allocation problem was formulated as a CPU utilization tracking problem where a controller aims to track and maintain the CPU allocation under a given threshold. Their system has 3 controllers: The first controller dynamically allocates CPU resources to individual VMs. A second controller adjusts the allocations of all the VMs of a multi-tier application. The third controller collectively allocates CPU resources to all VMs of an application and it self-configures its parameters and self-adapts to diverse workload conditions. Each controller allocates CPU resources to VMs based solely on resource utilization observations and the application performance model. Each VM is regarded as a black-box which can host an application tier or a whole application. To ensure that no VM can use any more CPU time that it has been allocated, they used the *cap* mechanism [10] provided by Xen's CPU scheduler [11]. An important contribution of [33, 34] is that they developed a zero-configuration mechanism to detect and adapt to workload conditions without any advance information.

Authors in [38, 39] use a different approach for resource allocation. They use two different services, termed *gold* and *silver*. The revenue generated by each service is specified via a pricing scheme or SLA that relates the achieved response time to a dollar value that the client is willing to pay. Thus, resources are allocated for both services aiming to maximize the profit generated by this system by minimizing both the power consumption and SLA violations. To achieve this objective, a controller decides the number of physical

and virtual machines to allocate to each service where the VMs and their hosts are turned on or off according to the workload demand, and the CPU share to allocate to each VM.

Based on the assumption that Internet applications tend to see dynamically varying workloads that contain long-term variations such as time-of-day effects as well as short-term fluctuations due to flash crowds, authors in [77] use a queuing model to determine how much resources to allocate to each tier of the application and a combination of predictive and reactive methods that determine when to provision these resources, both at large and small time scales.

It is presented in [5] a proposal that aims to address the problem of resources allocation in data centers that run VMs from different customers. In this work it is used a dynamic provisioning technique for a cluster-based virtualized multi-tier application to satisfy the requirement of given response time for customers. They employ a hybrid queuing model to determine the number of virtual machines at each tier in a virtualized application. They adopted an optimization model aiming to minimize the total number of VMs while satisfying the customer average response time constraint.

Authors in [78] emphasize the importance of taking migration cost into account during consolidation actions. They proposed pMapper, an application placement controller that dynamically places application servers aiming to minimize power consumption while meeting performance guarantees. A Migration Manager estimates the cost of moving a VM from a given placement to a new placement. They observed that the impact of migration depends on the VM characteristics. Hence, the cost of each live migration can be computed a priori. The migration cost is determined by estimating the impact of migration on application throughput and consequent revenue loss, computed as per the SLA. When possible pMapper resizes the VM to reduce its migration costs.

Aiming to avoid what they referred to as "server and storage sprawl", i.e., many underutilized servers with heterogeneous storage elements, authors in [37] perform resource allocation according to a proposed algorithm for migrating VMs. During the run time, if requirements like response time and throughput of an application are violated, it is often because of factors such as high CPU utilization and high memory usage of the server where it is hosted. To detect and resolve application performance problems in a data center, they collect measurements for each server and when the workload increases or when demand is low, they use live migration of VMs to maximize performance or to consolidate servers. If any of the VMs report a SLA violation (e.g., high response time) it is performed a dynamic re-allocation of VM(s) from the hosting physical machine to

another physical machine such that the SLA is restored.

The results of the experiments performed in this work can be applied, in the context of the works related in this Chapter, to manage resources in a virtualized server environment.

Chapter 7

Conclusion

In this Master dissertation we have described and evaluated migration and replication mechanisms that have been used in virtualized datacenters. These mechanisms helps datacenters to increase resources utilization, reduce the power consumption and the TCO. In this work we have used these mechanisms separately, however, they can be combined aiming to optimize resources utilization. In this work we have also integrated these mechanisms in an architecture that permits the execution of multi-tier applications. This architecture was used to perform part of the experiments presented in this work and can be used in future experiments by other researchers in our group.

7.1 Contributions

The contribution of this Master dissertation consists in the description and integration of mechanisms that permit virtual machine migration and replication. These mechanisms can be applied in web clusters since they permit an application to be supported by different physical machines and also permit the load balancing in the different tiers. We have combined these mechanisms in an architecture that we have used to perform experiments using a multi-tier application. This architecture can be used in future experiments. Moreover, we have adapted the RUBiS benchmark tool to simulate different load variations when compared to the original RUBiS benchmark tool that simulates only a ramp up/down. This adapted benchmark tool can also be used in future experiments.

7.2 Future Work

- For future experiments, we recommend the use of more applications in the system e.g., the RUBBoS benchmark tool [66]. RUBBoS is a bulletin board benchmark modeled after an online news forum like Slashdot. As RUBBoS places high load on the database-tier [41], several recent research works which aim to stress the database-tier took advantage of the RUBBoS benchmark tool, among them [38, 41, 77]. In this case, would be interest to adopt the response time as a metric to perform migration or replication activities. We also recommend to aggregate different workload generators in a single tool aiming to automate experiments.

- In order to optimize resources utilization, we recommend to use an optimization model, like the proposed by our group [55, 58, 59, 60, 61], to combine migration and replication activities. This would lead to energy savings while maintaining QoS metrics.

- Although we have used the MySQL proxy as a mechanism for load balancing between replicated databases, the load balancing in our experiments was performed according to an algorithm in which write requests are sent to the master database and read requests are distributed between slaves databases. However, in a large-scale system, this policy can impose a limitation as the capacity of the entire system will be limited by a single master database. Thus, in a large system we recommend to replicate the master database and to use different load balancing algorithms.

- As observed in [22], runtime re-allocation of memory among multiple VMs has not been widely studied. Thus, the investigation on how to dynamically allocate memory resources, poses a great challenge for future research.

References

[1] AMD. Amd powernow! technology. http://www.amd.com/us/products/technologies/amd-powernow-technology/Pages/amd-powernow-technology.aspx, 2010.

[2] Paul Barham et al. Xen and the art of virtualization. In *SOSP'03*, pp. 164–177. ACM, 2003.

[3] Luiz André Barroso. The price of performance. *ACM Queue*, 3(7):48–53, 2005.

[4] Luciano Bertini, Julius C. B. Leite e Daniel Mossé. Power optimization for dynamic configuration in heterogeneous web server clusters. *J. Syst. Softw.*, 83:585–598, April 2010.

[5] Jing Bi, Zhiliang Zhu, Ruixiong Tian e Qingbo Wang. Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. *Cloud Computing, IEEE International Conference on*, 0:370–377, 2010.

[6] Ricardo Bianchini e Ram Rajamony. Power and energy management for server systems. *Computer*, 37(11):68–74, 2004.

[7] Daniel P. Bovet e Marco Cesati Ph. *Understanding the Linux Kernel, Third Edition*. O'Reilly Media, 3 edio, November 2005.

[8] CentOS Project. The community enterprise operating system. http://www.centos.org/, 2010.

[9] Yuan Chen, Subu Iyer, Xue Liu, Dejan Milojicic e Akhil Sahai. Translating service level objectives to lower level policies for multi-tier services. *Cluster Computing*, 11(3):299–311, 2008.

[10] Citrix Systems. Credit-based cpu scheduler. http://wiki.xensource.com/xenwiki/CreditScheduler, 2010.

[11] Citrix Systems. Xen scheduling. http://wiki.xensource.com/xenwiki/Scheduling, 2010.

[12] Christopher Clark et al. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation*, pp. 273–286, 2005.

[13] Adrian Cockcroft e Walker. *Capacity Planning for Internet Services*. Prentice Hall Professional Technical Reference, 2001.

[14] eBay. ebay. http://www.ebay.com/, 2010.

[15] Egon Bianchet. Virtual machine replication. http://www.krisbuytaert.be/blog/node/478, 2010.

[16] E. N. Elnozahy, Michael Kistler e Ramakrishnan Rajamony. Energy-efficient server clusters. In *Proceedings of the 2nd international conference on Power-aware computer systems*, PACS'02, pp. 179–197, Berlin, Heidelberg, 2003. Springer-Verlag.

[17] Engineeting Statistics Handbook. Single exponential smoothing. http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc431.htm, 2010.

[18] Xiaobo Fan, Wolf-Dietrich Weber e Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pp. 13–23, New York, NY, USA, 2007. ACM.

[19] The Apache Software Foundation. Apache module mod_proxy_balancer. http://httpd.apache.org/docs/2.1/mod/mod_proxy_balancer.html, 2010.

[20] Google. Introducing the google chrome os. http://googleblog.blogspot.com/2009/07/introducing-google-chrome-os.html, 2010.

[21] Edward L. Haletky. *VMware ESX Server in the Enterprise: Planning and Securing Virtualization Servers*.

[22] Jin Heo, Xiaoyun Zhu, Pradeep Padala e Zhikui Wang. Memory overbooking and dynamic control of xen virtual machines in consolidated environments. In *Proceedings of the 11th IFIP/IEEE international conference on Symposium on Integrated Network Management*, IM'09, pp. 630–637, Piscataway, NJ, USA, 2009. IEEE Press.

[23] Fabien Hermenier et al. Cluster-wide context switch of virtualized jobs. Technical Report 6929, INRIA, 2009.

[24] Tibor Horvath, Tarek Abdelzaher, Kevin Skadron e Xue Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Trans. Comput.*, 56:444–458, April 2007.

[25] HP Labs. Httperf homepage. http://www.hpl.hp.com/research/linux/httperf/, 2010.

[26] IBM. The ondemand governor. http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaai/cpufreq/TheOndemandGovernor.htm, 2010.

[27] IBM. xentop(1) - linux man page. http://linux.die.net/man/1/xentop, 2010.

[28] Intel. Enhanced intel speedstep technology. http://www.intel.com/cd/channel/reseller/asmo-na/eng/203838.htm, 2010.

[29] Waheed Iqbal, Matthew N. Dailey, David Carrera e Paul Janecek. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, November 2010.

[30] iwebdevel. Define notions: What is cpu throttling/processor throttling? http://iwebdevel.com/2009/09/05/define-notions-what-is-cpu-throttling-processor-throttling-shared-hosting/, 2010.

[31] X.Liu T.Abdelzaher J.Heo, D.Henriksson. Integrating adaptive components: An emerging challenge in performance-adaptive systems and a server farm case-study. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, RTSS '07, pp. 227–238, Washington, DC, USA, 2007. IEEE Computer Society.

[32] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[33] Evangelia Kalyvianaki. *Resource provisioning for virtualized server applications*. PhD thesis, Cambridge University, 2008.

[34] Evangelia Kalyvianaki, Themistoklis Charalambous e Steven Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *ICAC '09: Proceedings of the 6th international conference on Autonomic computing*, pp. 117–126, New York, NY, USA, 2009. ACM.

[35] Jeffrey O. Kephart, Hoi Chan, Rajarshi Das, David W. Levine, Gerald Tesauro, Freeman Rawson e Charles Lefurgy. Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In *Proceedings of the Fourth International Conference on Autonomic Computing*, pp. 24–, Washington, DC, USA, 2007. IEEE Computer Society.

[36] Nick Kew. *The Apache Modules Book: Application Development with Apache*. Prentice Hall, 2007.

[37] G. Khanna, K. Beaty, G. Kar e A. Kochut. Application performance management in virtualized server environments. pp. 373 –381, apr. 2006.

[38] Dara Kusic. *Combined power and performance management of virtualized computing environments using limited lookahead control*. PhD thesis, Drexel University, 2008.

[39] Dara Kusic, Jeffrey O. Kephart, James E. Hanson, Nagarajan Kandasamy e Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009.

[40] Julius C.B. Leite, Dara M. Kusic, Daniel Mossé e Luciano Bertini. Stochastic approximation control of power and tardiness in a three-tier web-hosting cluster. In *Proceeding of the 7th international conference on Autonomic computing*, ICAC '10, pp. 41–50, New York, NY, USA, 2010. ACM.

[41] Simon Malkowski, Deepal Jayasinghe, Markus Hedwig, Junhee Park, Yasuhiko Kanemasa e Calton Pu. Empirical analysis of database server scalability using an n-tier benchmark with read-intensive workload. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pp. 1680–1687, New York, NY, USA, 2010. ACM.

[42] McKinsey, Company. Revolutionizing data center energy efficiency. `http://uptimeinstitute.org`, 2010.

[43] MySQL. Mysql 5.1 reference manual: Chapter 16. replication. `http://dev.mysql.com/doc/refman/5.1/en/replication.html`, 2010.

[44] MySQL. Mysql 5.1 reference manual: Chapter 17. mysql cluster ndb 6.x/7.x. http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster.html, 2010.

[45] MySQL. Mysql cluster overview. http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-overview.html, 2010.

[46] MySQL. Mysql proxy. http://dev.mysql.com/doc/refman/5.0/en/mysql-proxy.html, 2010.

[47] MySQL. Mysql: The world's most popular open source database. http://www.mysql.com/?bydis_dis_index=1, 2010.

[48] MySQL. mysqldump a database backup program. http://dev.mysql.com/doc/refman/5.0/en/mysqldump.html, 2010.

[49] MySQL. Replication implementation. http://dev.mysql.com/doc/refman/5.1/en/replication-implementation.html, 2010.

[50] MySQL AB. A mysql load balancer.

[51] Carlos Oliveira et al. Leveraging real-world web traces for server benchmarking. Technical report, UFF, 2010.

[52] Carlos Oliveira, Vinicius Petrucci e Orlando Loques. Impact of server dynamic allocation on the response time for energy-efficient virtualized web clusters. In *WTR 2010*, may 2010.

[53] OpenNebula. The open source toolkit for cloud computing. http://opennebula.org/, 2010.

[54] Pradeep Padala, Xiaoyun Zhu, Zhikui Wang, Sharad Singhal, Kang G. Shin, Pradeep Padala, Xiaoyun Zhu, Zhikui Wang, Sharad Singhal e Kang G. Shin. Performance evaluation of virtualization technologies for server consolidation. Technical report, 2007.

[55] V. Petrucci, E. V. Carrera, O. Loques, J. Leite e D. Mossé. Optimized management of power and performance for virtualized heterogeneous server clusters. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid (CCGrid'11)*, 2011.

[56] Vinicius Petrucci. A framework for supporting dynamic adaptation of power-aware web server clusters, 2008.

[57] Vinicius Petrucci. Dynamic optimization of power and performance for virtualized server clusters. Technical report, UFF, 2010.

[58] Vinicius Petrucci et al. Dynamic optimization of power and performance for virtualized server clusters. Technical Report RT-05/09, UFF, 2009.

[59] Vinicius Petrucci, Orlando Loques e Daniel Mossé. A dynamic configuration model for power-efficient virtualized server clusters. In *11th Brazillian Workshop on Real-Time and Embedded Systems (WTR)*, 2009.

[60] Vinicius Petrucci, Orlando Loques e Daniel Mossé. A framework for dynamic adaptation of power-aware server clusters. In *SAC '09: Proceedings of the 24th ACM Symposium on Applied Computing*. ACM, 2009.

[61] Vinicius Petrucci, Orlando Loques e Daniel Mossé. A dynamic optimization model for power and performance management of virtualized clusters. In *1st Int'l Conf. on Energy-Efficient Computing and Networking. In cooperation with SIGCOMM*. ACM, 2010.

[62] Pradeep Padala's blog. Understanding live migration of virtual machines. `http://ppadala.net/blog/2010/06/understanding-live-migration-of-virtual-machines/`, 2010.

[63] Python Software Foundation. Python programming language. `http://www.python.org/`, 2010.

[64] Parthasarathy Ranganathan. Recipe for efficiency: principles of power-aware computing. *Commun. ACM*, 53(4):60–67, 2010.

[65] Parthasarathy Ranganathan, Phil Leech, David Irwin e Jeffrey Chase. Ensemble-level power management for dense blade servers. In *Proceedings of the 33rd annual international symposium on Computer Architecture*, ISCA '06, pp. 66–77, Washington, DC, USA, 2006. IEEE Computer Society.

[66] RUBBOS. Rubbos: Bulletin board benchmark. `http://jmob.ow2.org/rubbos.html`, 2010.

[67] RUBiS. Rubis: Rice university bidding system. `http://rubis.ow2.org/`, 2010.

[68] Cosmin Rusu, Alexandre Ferreira, Claudio Scordino e Aaron Watson. Energy-efficient real-time heterogeneous server clusters. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 418–428, Washington, DC, USA, 2006. IEEE Computer Society.

[69] Carlos Santana, J. C. B. Leite e Daniel Mossé. Load forecasting applied to soft real-time web clusters. In *SAC '10: Proceedings of the 2010 ACM Symposium on Applied Computing*, pp. 346–350, New York, NY, USA, 2010. ACM.

[70] Saeed Sharifian, Seyed A. Motamedi e Mohammad K. Akbari. A content-based load balancing algorithm with admission control for cluster web servers. *Future Generation Computer Systems*, 24(8):775 – 787, 2008.

[71] Gokul Soundararajan, Cristiana Amza e Ashvin Goel. Database replication policies for dynamic content applications. In *EuroSys '06: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pp. 89–102, New York, NY, USA, 2006. ACM.

[72] Shekhar Srikantaiah, Aman Kansal e Feng Zhao. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, pp. 10–10, Berkeley, CA, USA, 2008. USENIX Association.

[73] Christopher Stewart, Terence Kelly e Alex Zhang. Exploiting nonstationarity for performance prediction. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pp. 31–44, New York, NY, USA, 2007. ACM.

[74] TechTerms.com. Mysql. `http://www.techterms.com/definition/mysql`, 2010.

[75] The Apache Software Foundation. Apache HTTP server version 2.2. `http://httpd.apache.org/docs/2.2/`, 2010.

[76] Tim Abels, Puneet Dhawan and Balasubramanian Chandrasekaran. An overview of xen virtualization. `http://www.dell.com/downloads/global/power/ps3q05-20050191-Abels.pdf`, 2010.

[77] B. Urgaonkar, P. Shenoy, A. Chandra e P. Goyal. Dynamic provisioning of multi-tier internet applications. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pp. 217 –228, 2005.

[78] Akshat Verma, Puneet Ahuja e Anindya Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pp. 243–264, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[79] Akshat Verma et al. pMapper: power and migration cost aware application placement in virtualized systems. In *Middleware'08*, pp. 243–264, 2008.

[80] Daniel Villela, Prashant Pradhan e Dan Rubenstein. Provisioning servers in the application tier for e-commerce systems. *ACM Trans. Internet Technol.*, 7, February 2007.

[81] VMware. A performance comparison of hypervisors. `www.cc.iitd.ernet.in/misc/cloud/hypervisor\_performance.pdf`, 2010.

[82] William Voorsluys, James Broberg, Srikumar Venugopal e Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *CloudCom '09: Proceedings of the 1st International Conference on Cloud Computing*, pp. 254–265, Berlin, Heidelberg, 2009. Springer-Verlag.

[83] Yefu Wang, Xiaorui Wang, Ming Chen e Xiaoyun Zhu. Power-efficient response time guarantees for virtualized enterprise servers. In *Proceedings of the 2008 Real-Time Systems Symposium*, pp. 303–312, Washington, DC, USA, 2008. IEEE Computer Society.

[84] Wikipedia. Dynamic frequency scaling. `http://en.wikipedia.org/wiki/Dynamic\_frequency\_scaling`, 2010.

[85] Qi Zhang, Ludmila Cherkasova, Ningfang Mi e Evgenia Smirni. A regression-based analytic model for capacity planning of multi-tier applications. *Cluster Computing*, 11(3):197–211, 2008.

APPENDIX A – Xen hypervisor

Virtualization allows servers to run multiple independent application servers in a physical machine. Instead of having a physical machine running a single application server. There are different approaches to implement virtualization, between them *full virtualization* and *para-virtualization*. Full virtualization is designed to provide total abstraction of the underlying physical system and creates a complete virtual system in which the guest operating systems (OS) can execute. No modification is required in the guest OS or application; the guest OS or application is not aware of the virtualized environment so they have the capability to execute on the VM just as they would on a physical machine.

However, full virtualization may incur a performance penalty. The VMM must provide the VM with an image of an entire system, including virtual BIOS, virtual memory space, and virtual devices [76]. The VMM also must create and maintain data structures for the virtual components, such as a shadow memory page table. These data structures must be updated for every corresponding access by the VMs [76]. In contrast, para-virtualization requires modifications to the guest OSs that are running on the VMs. As a result, the guest OSs are aware that they are executing on a VM, allowing for near-native performance.

To support server virtualization in our system we have used the Xen VMM (also known as *hypervisor*). It provides the basic layer of interaction between running operating systems and hardware resources. Based on x86 para-virtualization, it creates different execution environments, the VMs or domains in Xen terminology. A VMM provides a low-overhead virtualization platform [81] and supports execution of heterogeneous operating systems with minimal modifications required within the operating system kernel [33].

In a virtualized environment, CPU, memory, and I/O components need to be virtualized. Xen is designed to enable para-virtualization of all three hardware components. Specifically for the CPU, focused in this work, the Intel x86 architecture provides four

levels of privilege modes. These modes, or rings, are numbered 0 to 3, with 0 being the most privileged. In a non-virtualized system, the OS executes at ring 0 and the applications at ring 3. Rings 1 and 2 are typically not used. In Xen para-virtualization, the VMM executes at ring 0, the guest OS at ring 1, and the applications at ring 3. This approach helps to ensure that the VMM possesses the highest privilege, while the guest OS executes in a higher privileged mode than the applications and is isolated from the applications. Privileged instructions issued by the guest OS are verified and executed by the VMM.

In a Xen system, the `Domain0` or `Dom0` is the first domain launched when the system is booted. The management tools for controlling the Xen virtualization platform reside in dom0. It can be used to create, delete, migrate and pause all other regular guest domains. In addition, access to resources and permission settings on VMs are administered through dom0. A regular guest domain is called a `DomU` or unprivileged domain. `Dom0` is scheduled like `DomUs`. If a domain has only one VCPU, it can be executed in one processor or core at a time. Each domain may have one or more virtual CPUs (VCPUs) which run on physical CPUs. In our experiments, each VM has four VCPUs since our worker machines are quad-core. Xen has another feature termed "cap" [10] that can be used to control the maximum percentage of CPU a domain can use, even if there are free CPU cycles. This mechanism is used by several authors, among them [22, 34], and allows better performance isolation among multiple VMs, preventing an application from draining the CPU capacity.

APPENDIX B – Response time measurement

The response time considered in the experiments in Chapter 5 is related to the server side. Thus, the response time is defined by the difference between the time a response is generated and the moment the server has accepted the associated request. To obtain the response time for the web applications we have implemented a new Apache module that collects the time information between these two moments using pre-defined hooks provided by the Apache Module API [36]. The hooks used to measure the response time are: `post_read_request` and `log_transaction`. The `post_read_request` phase allows our module to store the moment a request was accepted by Apache and the `log_transaction` phase allows it to store the moment a response was sent back to the client. The difference between these values gives the response time.

To smooth out high short-term fluctuations in measurements readings, we have integrated a filter procedure in our Apache module based on a single exponential moving average [17]. Specifically, the filter computes the next value, S_t, by summing the product of the smoothing constant α ($0 < \alpha < 1$) with the new value (X_t), and the product of $(1 - \alpha)$ times the previous average, as follows: $S_t = \alpha * X_t + (1 - \alpha) * S_{t-1}$. Values of α close to 1.0 have less smoothing effect and give greater weight to recent changes in the data, while values of α close to 0.0 have a greater smoothing effect and are less responsive to recent changes. Some techniques may be used to optimize the value of α, such as using the Marquardt procedure to find the value of α that minimizes the mean of the squared errors (MSE) [17]. In the filter module, we have used $\alpha = 0.5$ as the default smoothing factor; based on our experiments this value was found suitable.

APPENDIX C – Physical machines configuration

We present in Table 1 the physical machines used in the experiments. All machines share a NFS (Network File System) storage mounted in the front-end to store the VM images. The NFS does not causes disruption to the proxies running in the front-end since the NFS is not CPU-bound.

Henry	AMD Athlon 64 3500+	3GB RAM	Front-end
Camburi	Intel Pentium 4 2.80GHz	1GB RAM	Load generator
Cumulus	Intel Pentium 4 2.80GHz	1GB RAM	Load generator
Xingo	Intel Core i7 CPU 2.67GHz	8GB RAM	Load generator
Farad	Intel Core2 Duo CPU 3.00GHz	6GB RAM	Load generator
Itaipu	Intel Core i5 CPU 2.67GHz	8GB RAM	Worker
Tucurui	Intel Core i5 CPU 2.67GHz	8GB RAM	Worker
Maxwell	Intel Core i5 CPU 2.67GHz	8GB RAM	Worker
Edison	Intel Core i7 CPU 2.67GHz	8GB RAM	Worker

Table C.1: Physical Machines Configuration