# Impact of server dynamic allocation on the response time for energy-efficient virtualized web clusters

**Carlos Oliveira, Vinicius Petrucci, Orlando Loques**

Universidade Federal Fluminense (UFF), Niteroi, RJ, Brazil

{cjunior,vpetrucci,loques}@ic.uff.br

***Abstract.*** *Virtualization has been widely adopted in data centers around the world for improving resource usage efficiency; particularly helping to make these computing environments more energy-efficient. Server virtualization allows for on-demand allocation (using either migration or replication) of virtual machines (VMs), which run the web applications and services, to physical servers. In this paper, we measure and analyze the disruptive impact 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. 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 results show that VM replication with workload balancing may lead to reduced disruption impact on the QoS measures when compared to VM migration.*

## 1. Introduction

Virtualization has been widely adopted in data centers around the world for improving resource usage efficiency; particularly helping to make these computing environments more energy-efficient. Several virtual machine monitors or hypervisors have been developed to support virtualization, e.g., VMware [Sugerman et al. 2001] and Xen [Barham et al. 2003]. The key idea is that server virtualization allows for on-demand configuration (either by migration or replication) of virtual machines (VMs), which run the web applications and services, to physical servers. Concentrating applications and services in a smaller number of servers, using this capability, helps to increase resource utilization, allowing to reduce the use of computer resources and the associated power demands.

In previous works [Petrucci et al. 2009, Petrucci et al. 2010], we 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. We have experimented our optimization approach through simulations driven by real workload traces. 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 [Voorsluys et al. 2009].

In this work, we consider a real virtualized cluster platform aimed at supporting the deployment of web applications. We carry out a set of experiments with different test scenarios 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. 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 results show that VM replication with workload balancing may lead to reduced disruption impact on the QoS compared to VM migration.

The paper is organized as follows. The description of our virtualized cluster and testbed is presented in Section 2. In Section 3, we present some experiments for evaluating the response time impact during virtual machine migration and replication. Section 4 summarizes related works and Section 5 concludes the paper.

## 2. Virtualized cluster description

### 2.1. Architecture

Our target architecture (shown in Figure 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 process the requests (also known as *workers*). These servers run CentOS Linux 5.4 with Xen hypervisor enabled to support the execution of virtual machines.

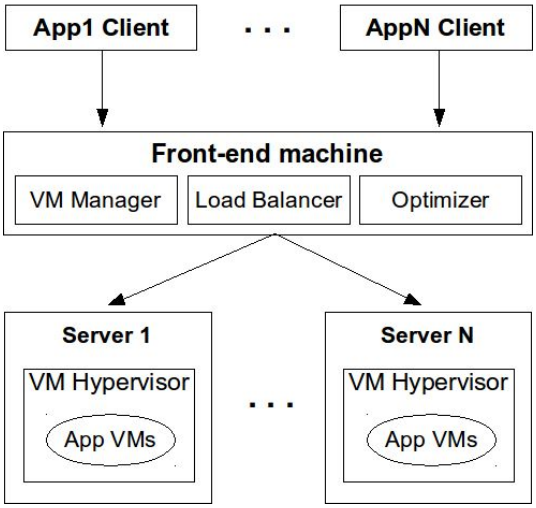


**Figure 1. Server cluster architecture**

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 imple-

mented using the OpenNebula toolkit [OpenNebula 2010] which enables the management of the VMs in the cluster, such as deployment and monitoring. The `Load Balancer` implements a weighted round-robin scheduler strategy provided by the Apache's `mod_proxy_balancer` module [The Apache Software Foundation 2010]. Finally, the `Optimizer` is designed to monitor and configure the virtualized cluster. It consists of an external module implemented in Python 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 [Petrucci et al. 2010]).

### 2.2. Testbed

The testbed platform used to implement the proposed architecture is described in Figure 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. The load generator machines `camburi` and `cumulus` (two Intel Pentium 4 2.80GHz, 1GB RAM, Ubuntu Linux 9.04) are physically connected via a gigabit switch. The worker machines `maxwell` (Intel Core i5 2.67GHz, 8GB RAM, CentOS 5.4) and `edison` (Intel Core i7 CPU 2.67GHz, 8GB RAM, CentOS 5.4) are connected via another gigabit switch. The front-end machine `henry` (AMD Athlon 64 3500+, 3GB RAM, CentOS 5.4) has two gigabit network interfaces, each one connected to one of the switches. All machines share an NFS (Network File System) storage mounted in the front-end to store the VM images.

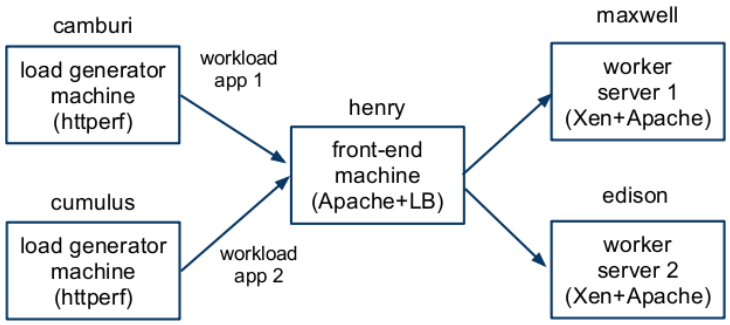


**Figure 2. Cluster testbed setup**

### 2.3. Response time measurement

The response time considered in this work 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 [Kew 2007]. 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 [Engineeting Statistics Handbook 2010]. Specifically, the filter computes the next value, $S_t$, by summing the product of the smoothing constant $\alpha$ ($0 < \alpha < 1$) 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 $\alpha$ close to $1.0$ have less smoothing effect and give greater weight to recent changes in the data, while values of $\alpha$ 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 $\alpha$, such as using the Marquardt procedure to find the value of $\alpha$ that minimizes the mean of the squared errors (MSE) [Engineeting Statistics Handbook 2010]. In the filter module, we have used $\alpha = 0.5$ as the default smoothing factor; based on our experiments this value was found suitable.

### 2.4. Xen hypervisor

In a Xen system, the virtual machines are termed *domains*. The `Domain0` or `Dom0` is the first domain launched when the system is booted. It can be used to create and configure all other regular guest domains. 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 we use a quad-core architecture. Xen has another feature called "cap" that can be used to control the maximum percentage of CPU a domain can use, even if there are free CPU cycles. This may be useful if one wants to control how the Xen schedules the domains to the physical CPUs.

The Xen hypervisor offers two kinds of migration: cold and live migration. The difference between them is that on cold migration the VM stops running during migration. Otherwise, on live migration the VM keeps running most of the time; actually, it stops only for a few milliseconds at Stage 3. The live migration stages are listed below [Clark et al. 2005]:

- *Stage 0 (Pre-Migration)*: Alternate physical host may be preselected for migration. Block devices mirrored and free resources maintained;
- *Stage 1 (Reservation)*: Initialize a container on the target host;
- *Stage 2 (Iterative Pre-copy)*: Enable shadow paging. Copy dirty pages in successive rounds;
- *Stage 3 (Stop and copy)*: Suspend VM on source host. Generate ARP to redirect traffic to target host. Synchronize all remaining VM state to target host;
- *Stage 4 (Commitment)*: VM state on target host is released;
- *Stage 5 (Activation)*: VM starts on target host. Connects to local devices. Resumes normal operation.

Notice that cold migration does not have the Stage 2 as in live migration. Notice also that caches in hardware are not migrated [Verma et al. 2008], which can lead to

cache misses in the target machine and impact application's performance when performing migrations. As pointed out by [Voorsluys et al. 2009], both migration activities need extra CPU cycles for the pre-copying process which are consumed on both source and destination servers. Moreover, an additional amount of network bandwidth is consumed as well, which may affect the quality-of-service in the cluster. A third available option is VM replication, which means creating a new VM (application instance) from a stored image, instead of migrating an already running one (see Section 3). It is worth mentioning that our optimization model includes the possibility of running replicated servers for fulfilling the resources required by an application at a given operational stage.

## 3. Experiments

We performed a set of experiments in our testbed (described in Section 2.2). In the first step, we used the Apache Benchmark (ab) [The Apache Software Foundation 2010] 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.

As shown in Figure 3, 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 activities.

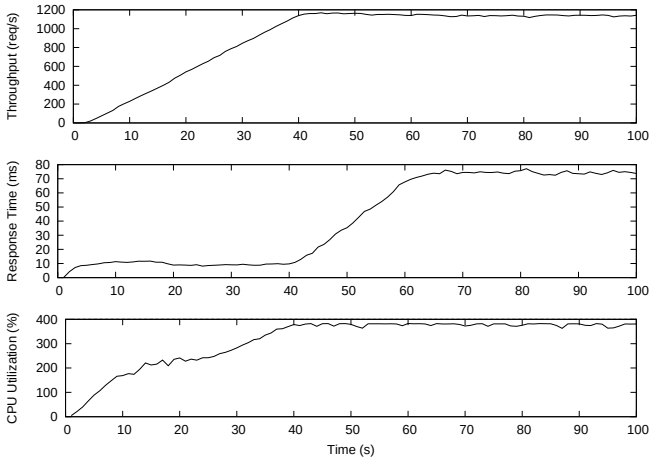


**Figure 3. Relationship among throughput, response time, and CPU utilization**

In the next step, we allocate two virtual machines (VMs) to run on the `maxwell` machine. Each VM has 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 for our experiments. Notice that the quantity of memory a VM is using may impact on how much time

is needed to complete a migration [Hermenier et al. 2009]. We plan to evaluate this issue in future experiments.

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 consider increasing 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.

### 3.1. Scenario 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). As expected, we observe that in the cold migration, the VM stops during the migration. Figure 4 shows the throughput, response time and CPU utilization for both 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.

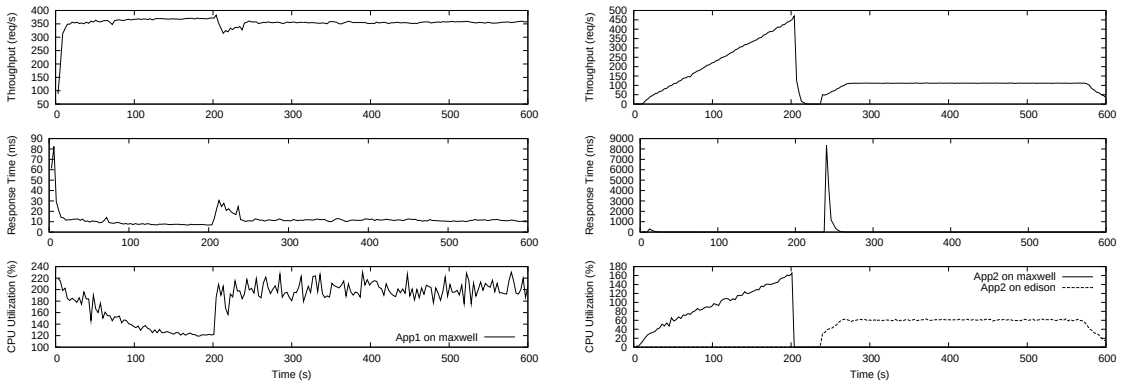


**Figure 4. Execution of the *cold migration* scenario: App1 (left) and App2 (right)**

### 3.2. Scenario 2: Live migration

In this scenario, we use live migration to move a VM to a new server machine without service interruption [Clark et al. 2005]. 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 goal of the experiment (shown in Figure 5) was to evaluate the impact of applying the live migration mechanism.

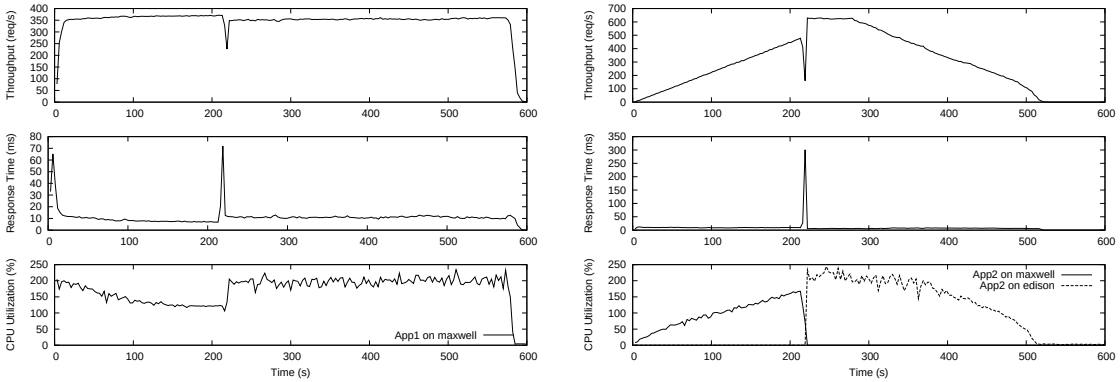


**Figure 5. Execution of the *live migration* scenario: App1 (left) and App2 (right)**

As would be expected, unlike cold migration, we observe that in live migration the VM is not paused during the migration. In Figure 5, the application `App2` was migrated with no 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 even `App1` was slightly affected when migration was performed. The slowdown in the throughput was 61.5% (from 414.1 req/s to 159.5 req/s). We can note that the disruptions observed when performing dynamic changes through live migration last a short time and are basically unavoidable.

### 3.3. Scenario 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. We may then either shutdown or keep running in the origin physical server the old VM replica. The first option was adopted in the experiment. The last option may be adopted to provide a high capacity server, summing the replicas resources, to support an application with high resource demands.

The goal of the experiment for this scenario (see Figure 6) is to evaluate the response time impact compared to the live migration scenario presented in Section 3.2. 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.

The use of replication shows improvements compared to the live migration, for example, by analyzing the decrease observed in the throughput in Figure 6, contrasting it to Figure 5. 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,

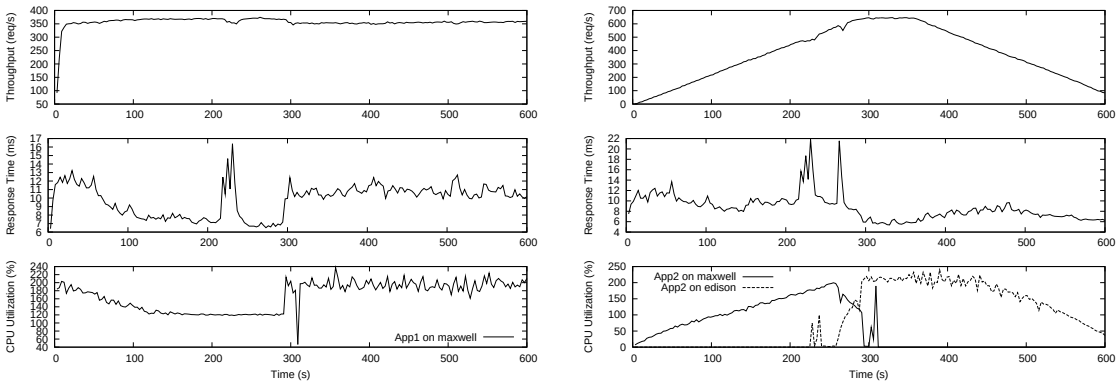


**Figure 6. Execution of the *replication* scenario: App1 (left) and App2 (right)**

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 emphasize 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. The time needed to boot a VM is in between 25 and 40 seconds, which may be a bit longer than 10 seconds, on average, observed in the live migration in Scenario 2 (see Section 3.2). We argue that if the replication scheme is able to take advantage of prediction techniques to anticipate the booting process considering typical load patterns, such as proposed in [Dinda and O'Hallaron 2000], the time delay for booting a new VM may be minimized. We may also boot the new replica on the target machine a few seconds earlier to have it running and ready at the moment necessary for using the replicated VM.

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" [The Apache Software Foundation 2010]; setting the Apache configuration to maintain a suitable set of idle server daemons, which standby ready to serve incoming requests. 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 this experiment, we redirected 10% of the requests each time, until 100% of the requests were redirected to the VM replica.

## 4. Related Work

A heuristic algorithm for server consolidation is available [Khanna et al. 2006], but it does not take into account the cost of migrating virtual machines from one physical machine to another. Another approach [Wang et al. 2008] presents a two-layer control architecture aimed at providing power-efficient real-time guarantees for virtualized computing environments. The work relies on a control theory based framework, but does

not addresses live migration in a multiple server context. In [Kusic et al. 2009], a dynamic resource provisioning framework is developed based on lookahead control. A power-aware migration framework for virtualized HPC (High-performance computing) applications, which accounts for migration costs during virtual machine reconfigurations, is presented in [Verma et al. 2008]. As in our approach, it relies on virtualization techniques used for dynamic consolidation, although the application domains are different. In [Voorsluys et al. 2009], the authors quantify the effect of VM live migrations in the performance of social networking websites. The overall objective of their experiments is to quantify slowdown and downtime experienced by the application when VM migrations are performed in the middle of a run.

## 5. Conclusion and future work

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 incurred during migration. Finally, the evaluation described in this work will help us to implement our dynamic optimization model and strategy for power and performance management of virtualized web clusters [Petrucci et al. 2010].

As for future work, we plan to develop additional experiments with state-aware applications considering another layer as database. To address this, we intend to use Rubis [RUBiS 2010] multi-tier benchmark. In the replication process, the VM in the source server was turned off. We would like to investigate if it would be valuable while maintaining this application on the source server too for load balancing proposes. And, if so, we would like to identify what part of the application workload would be allocated both in the source and target physical machines.

## References

Barham, P. et al. (2003). Xen and the art of virtualization. In *SOSP'03*, pages 164–177. ACM.

Clark, C. et al. (2005). Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation*, pages 273–286.

Dinda, P. A. and O'Hallaron, D. R. (2000). Host load prediction using linear models. *Cluster Computing*, 3(4):265–280.

Engineeting Statistics Handbook (2010). Single exponential smoothing. http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc431.htm.

Hermenier, F. et al. (2009). Cluster-wide context switch of virtualized jobs. Technical Report 6929, INRIA.

Kew, N. (2007). *The Apache Modules Book: Application Development with Apache*. Prentice Hall.

Khanna, G. et al. (2006). Application performance management in virtualized server environments. *10th IEEE/IFIP Network Operations and Management Symposium*, pages 373–381.

Kusic, D. et al. (2009). Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15.

OpenNebula (2010). The open source toolkit for cloud computing. http://opennebula.org/.

Petrucci, V., Loques, O., and Mossé, D. (2009). A dynamic configuration model for power-efficient virtualized server clusters. In *11th Brazillian Workshop on Real-Time and Embedded Systems (WTR)*.

Petrucci, V., Loques, O., and Mossé, D. (2010). 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 (to appear).

RUBiS (2010). Rubis: Rice university bidding system. http://rubis.ow2.org/.

Sugerman, J. et al. (2001). Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. *USENIX Annual Technical Conference*.

The Apache Software Foundation (2010). Apache HTTP server version 2.2. http://httpd.apache.org/docs/2.2/.

Verma, A. et al. (2008). pMapper: power and migration cost aware application placement in virtualized systems. In *Middleware'08*, pages 243–264.

Voorsluys, W., Broberg, J., Venugopal, S., and Buyya, R. (2009). Cost of virtual machine live migration in clouds: A performance evaluation. In *CloudCom '09: Proceedings of the 1st International Conference on Cloud Computing*, pages 254–265, Berlin, Heidelberg. Springer-Verlag.

Wang, Y. et al. (2008). Power-efficient response time guarantees for virtualized enterprise servers. In *RTSS'08*, pages 303–312.