EEFSim: Energy Efficiency Simulator

Ferran Julià, Jordi Roldan, Ramon Nou, J. Oriol Fitó,
Alexandre Vaqué, Íñigo Goiri and Josep Lluis Berral

July 13, 2010

Abstract

Nowadays Cloud computing has emerged as one of the most promising computer paradigms. The idea of selling software as a service has promoted IT enterprises to bet for this new paradigm. Cloud computing aims to power the next generation data centers not only offering software as a service but virtual services like hardware, data storage capacity or application logic. The increasing use of Cloud-based applications will also increase the power dedicated to the data centers that support this Clouds.

Research in Cloud computing power saving techniques requires innovative solutions that have to be tested in real environments. It is difficult and expensive to set up suitable test-beds for large scale cluster applications. Simulation can fulfill the needs that we find in Cloud computing experimentation. A large data-center simulator can save lots of time and effort in Cloud investigation.

This paper presents the design and development process of a virtualized data-center simulator for Cloud computing research. It is able to reproduce the behavior of a real Cloud framework and the information that it offers of the execution makes it suitable for testing and investigation purposes. In order to better understand the development process we also present the analysis of real resource managing Cloud computing techniques that we have developed and helped us to understand which are the needs in a real Cloud and that also led us to the development of the presented simulator.

The final idea of this project was to understand the low level behaviors that rule the virtualized data-center in which the Cloud runs over, and to use all that experience in developing a simulator that permits us the improve resource scheduling techniques introducing innovative green policies.

1 Introduction

Cloud Computing is an emerging style of computing in which applications, data, and IT resources are provided to users as services over the Internet rather than being locally on the user's machine. Users can access these services anytime and anywhere, avoiding in this way hardware acquisition costs, software licenses or upgrades management, etc. These services are typically deployed and executed on third-party companies that act as *service providers*. The use of this technology is extending every day and now a days there are some examples of enterprises that are "selling" Cloud computing, like Amazon EC2 [1], Microsoft Azure [2], Google App Engine, and Aneka [3].

The scalability required for applications that are supposed to be executed in a Cloud implies having a real like system and that means a cluster of hundreds of different machines. Despite of the fact that it is very difficult to find such amount of hardware for research and development, most of times it will not be acceptable to have so many machines, with all the maintenance work and energy that involves only for testing, this kind of infrastructures are usually only used in production systems.

A simulator is a suitable and effective solution. It offers the possibility to reproduce our scheduling solutions in real-like data-centers and validate our solutions scalability saving lots of time and energy. A well designed simulator can provide results accurate enough for our proposals. And whats more, a simulator usually can run on a simple machine and can reduce the test time several times, so that would save lots of energy and time.

The simulator is based in a real Cloud framework named *emotive* [4]. It reproduces the internal behavior of the real framework and its modular design allows the use of real code and the possibility to run large tests in short time.

Virtualization's internal behaviors that this simulator implements and the power consumption results that have been compared and validated with real measures give this simulator unique characteristics needed for research in the Cloud computing power consumption area.

The simulator can calculate the CPU and power consumption values for each execution, which is a need in the development of power aware schedulers. It has been validated comparing its results with real ones obtaining very low error and it is being used at this moment in some resource scheduling research articles.

2 Background

In this section we give an overview to two important concepts which the simulator is based on: the underlying virtualization technology, and the virtualization management framework.

2.1 Xen's Virtual Machine Management

In order to better understand the challenges that involve the creation of a virtualized data-center simulator, in this section we will introduce some essential internals of virtualization systems. Notice that these mechanisms play a key paper in the structure as they govern the way real CPU is assigned to virtual CPUs and have a big influence on the overall behavior.

As the ultimate intention of this simulator is test the several parts of a real cloud system called (emotive [4]), and this framework is build on Xen [5] virtualization system, the simulator has to reproduce some of the Xen's internal mechanisms and low level behaviors.

Xen uses a virtualization technique called paravirtualization, this requires the partial modification of the guest operating system. The user level code is executed at ring 3 but Operating System code runs at ring 1, instead of ring 0 as usually, and Hypervisor runs at ring 0. As the guest O.S. is modified, some special operations call hypervisor instead of calling the hardware and is the hypervisor who manages the real resources and schedule them among the virtual hosts.

There is a special host in the system, called "dom0", which is always running at the host and that have some special operations that permit controlling the hypervisor. This special O.S. is in charge of creation, destruction and migration of virtual machines. Figure 1 shows the architecture of a Xen system with several virtual machines and "dom0" named there as "Management". In Xen usually the guest hosts or virtual machines are also named domains.

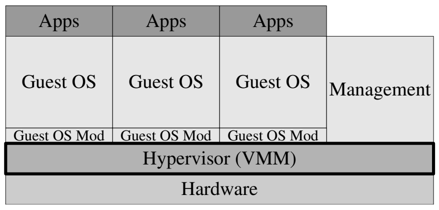

Figure 1: Paravirtualization Architecture

Although memory management is one of the most complex parts in paravirtualization for the simulation we do not have to reproduce all the mechanisms. We have to take into account which are the possibilities than Xen offers for memory management. For this case we have to consider one of the handicaps we noticed in [6], Xen allows variation of virtual hosts memory size but as the guest operating systems we use are Linux and it creates the memory structure at boot time, we can only reduce the amount of memory assigned to a virtual machine at boot but we never can grow it up.

In the real system we can overcome this setting higher levels of memory only at virtual machine boot time, but this also has problems when there are several guests running at the same host. For this reason during the simulation we will not change the memory assigned to the virtual machines although we have to take into account at scheduling time and we can not assign more memory than the real host have.

The simulation of the CPU scheduling behavior in Xen was the most difficult challenge. Xen allows very different configuration settings for CPU scheduling, virtual CPUs can be pinned to real CPUs or not, so you can configure which hosts share which CPUs, and you can also configure the way that this CPU is shared [7]. Although you can design a new scheduler, Xen have different ones implemented. The default one and the one that have better performance and accuracy (fits configured priorities and capabilities) is credit scheduler. In

[8] we can find a detailed view of all the schedulers delivered with Xen and a comparison among them.

The scheduler of our simulator is based on the Credit Scheduler [9]. The most important to notice from the credit scheduler is the Capabilities and Weight operation modes. With these parameters we can tune the scheduler behavior giving more ore less execution time or priority to the different virtual machines.

The scheduler assigns real CPU to virtual CPUs maintaining the same proportionality than have the “weight” parameter for the virtual machines. Each parameter is set at virtual machine level, so all virtual CPUs of the same virtual machine has the same weight parameter as it happens with the capability parameter. For example, if we have 2 virtual machines with 1 virtual CPU for each one running at the same host, if both have the same weight the real CPU will be shared at 50% for both, but if one have a weight value of 20 and the other of 80, the real CPU will be 20% for the former and 80% for the second. Notice that what is important here is the relation among the two weight values, nor the real value of them, instead of 20 and 80 values we could have 40 and 160 and the result would be the same 20%-80%.

The “Capability” parameter fixes the amount of real CPU that one virtual machine can use, so despite of what we calculated from the weight parameter if the real amount of CPU that is allocated for the virtual machine is grater than the capability the virtual machine will not have more real CPU than the capability.

With these two parameters we can adjust dynamically the real CPU that we allocate for each virtual machine thus control the behavior of our system. We could also use the “CPU pinned” ability of Xen but several test have shown that “pinning” virtual CPUs to real ones overcome in a notable loss of performance.

In Section 3 will see, in terms of CPU and power consumption, how the scheduler really schedules tasks in CPUs when there is less CPU consumption than CPU capacity.

Net management has also to be considered in the simulation, although we have not implemented by the moment because at this stage we are centered in the CPU and memory management, we will consider it in future works. For this reason we will not deep in the Xen’s network mechanisms.

2.2 EMOTIVE Cloud

In order to simplify virtualization management problems, EMOTIVE (Elastic Management of Tasks in Virtualized Environments) middleware [4] allows executing tasks and providing virtualized environments to the users without any extra effort in an efficient way. Actually, this is a virtualized environment manager which aims to provide VMs that fulfill the user requirements in terms of software and system capabilities.

EMOTIVE main feature is VM management with different scheduling policies. It can be also used as a cloud provider and is very easy to extend thanks to its modular Web Services architecture. In addition, EMOTIVE makes use of the Libvirt JAVA API which allows to use different virtualization technologies.

For example, it currently can use both Xen and KVM hypervisor, but in a close future it will be able to use VMWare ESX hypervisor, as well as support virtual LANs management.

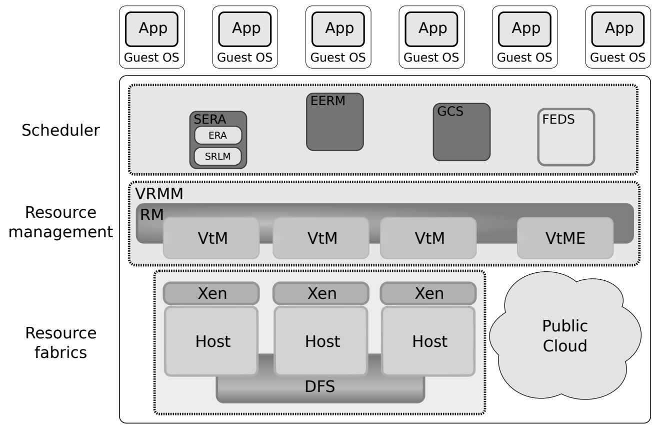

Figure 2: EMOTIVE Architecture

Figure 2 illustrates the EMOTIVE Cloud architecture, which is mainly composed by three different layers: the data infrastructure, the node management (VRMM), and the global Scheduler. The data infrastructure offers a distributed storage for supporting virtualization capabilities such as migration and checkpoint support, and it can use different kinds of storage. VRMM is in charge of creating and maintaining the whole virtual machine life cycle (create, destroy, migrate, etc) and tasks execution using JSDL. Finally, the scheduling layer is the responsible of distributing tasks and VMs among the physical nodes. Moreover, this framework has multiple schedulers with different policies and capabilities such as machine learning, prediction, economic, fault tolerance, semantic description, or SLA enforcement. In this sense, it can use a simplistic Round Robin, or a consolidation aware scheduling like Backfilling. This is achieved thanks to the usage of a common interface which allows developing new schedulers with different features and policies.

Furthermore, it also has the cpaability to use external resources, like from the public Cloud of Amazon EC2. This feature allows to be involved in a Cloud federation (insourcing/outsorcing) and create public, private and hybrids clouds.

From the simulator point of view, it is able to use the framework capabilities: VM management, different scheduling policies, and federation. In a nutshell, EMOTIVE is able to use real resources (VRMM) or can make use of the simulator presented (EEFSim).

3 Cloud Simulation Development

Development of scheduling techniques for power saving in large-scale clusters rises different handicaps, ones related with the decision taking problems and other related to the test and validation process. The supporting process requires the use of large amount of machine power and time that some times make the process itself not feasible. For these reason in such cases it is usual to use simulators that have the same behavior than the real infrastructure but that can save lots of time, power and effort while maintaining the levels of stringency in the results [10, 11].

We present a framework for evaluating the power efficiency of a virtualized data server. With this framework we can evaluate the effectiveness in both power consumption and the different scheduling techniques, which can take advantage of different capabilities such as migration of running tasks between nodes and considering the use of dynamic turn on/off, consolidation and virtual machine creation/migration costs.

Figure 3 shows the development cycle of the simulator. Firstly, different applications with different typologies and profiles and with different virtual machine configurations are executed (on a real, not simulated, machines) and their resource usage and power consumption monitored. Power usage is recorded using an external device which monitors the whole machine energy consumption. From these recordings, a model of the machine is built, which is then used to simulate a data center with many virtual machines. Validations are applied to refine the model and the simulator. Finally, the simulator is executed to provide the experimental data described here.

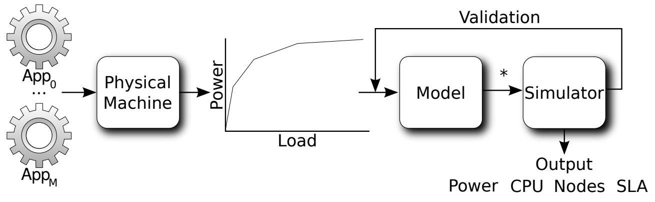

Figure 3: Simulator workflow

Our simulation technology is based on the OMNeT++ [12] platform. OMNeT++ is an object-oriented modular discrete event network simulation framework. OMNeT++ itself is not a simulator of anything concrete, but it rather provides infrastructure and tools for writing simulations. One of the fundamental ingredients of this infrastructure is a component architecture for simulation models. Models are assembled from reusable components termed modules. Well-written modules are truly reusable, and can be combined in various ways.Modules communicate through message passing.

The simulation uses similar techniques as seen in [13], but centered on power usage (measured in a real machine as seen in the environment section) and the scheduling of the different CPUs in the machines.

As in [13], the simulator does not try to simulate exact execution times working step-by-step, as this would be very time consuming. It is much simpler but enough for our purposes if we can predict the system's general behavior, in the long run, for the different scheduling techniques in large clusters, with respect to power usage. See [13] for details.

4 Architecture

We have designed a modular scheduler that makes easier to introduce new components or features. Our last intention was to test the most part of real cloud system code into the simulator, for this reason we tried to simulate as less as possible. In Figure 4 we can see the global simulator architecture. At first sight we can split the simulator en 3 parts. The workload generator, the "machine" or simulator core and the scheduler. Both workload and core are simulated using *OMNeT* simulator but we can plug-in any scheduler that follows the appropriate interface.

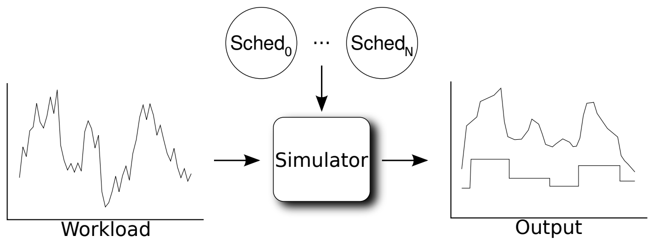

Figure 4: Simulator General Architecture.

The simulator core collects all the execution information and prints out the execution's power consumption, scheduling and resource usages statistics.

4.1 Workload Generator

Cloud computing is inevitably linked to the Quality of Service Concept. Emerging Cloud applications such as social networking, gaming portals, business applications, content delivery,... have different Quality of Service (QoS) requirements depending on interaction patterns and time, the way that the service provider offers a given QoS is sealed with a contract: Service Level Agreement (SLA). This contract specifies thinks like the amount of guaranteed memory, CPU or response time among others, and also establishes rewards and penalties for accomplishing or violating such agreements.

SLA creation and negotiation is a very complex process and it is out of the scope of this project, but it is important for us to take it into account in our simulator, for this reason we suppose that the contract negotiation is done before the job arrives at the system. As a consequence of the negotiation we obtain some parameters that are supposed to be estimated from the negotiation and that give us an idea of which are the requirements of the job, allowing us to have a more accurate resource reservation. In our simulator these values are CPU and memory that are supposed to be consumed by the job. So for each work we have specified an estimated value of CPU load and memory, that values are supposed to be derived from the SLA agreements.

The workload module reproduces the arrival of new jobs to the system following a previously generated trace. At the specified time the module simulates a new job arrival, sending a message, with all its characteristics that are supposed to be agree in the SLA negotiation process.

As we do not have real cloud traces we have adapted Grid traces, like the ones that can be obtained from Grid5000 [14]. These workloads specify the arrival time for each job and the type of job. As in a real cloud environment we can have very different kind of jobs, the simulator allows to have different types of jobs. At this moment, the platform is able to simulate HPC and Transactional jobs. In both cases the user have to specify the type of job and the CPU and memory that its supposed to be consumed. There is also the need to specify the load trace. This trance is specified in a separate file, see Figure 5. The simulator supports dynamic loads for each job, so we can trace real applications and add its behavior to the simulator. We can have several types of jobs configured and introduce several instances of the same job into the system. That makes more easy the creation of complex workloads. Notice that if the job sent is an HPC one the load trace will be expressed in terms of CPU in time but if it is a Transactional one the load will express requests/sec in time.

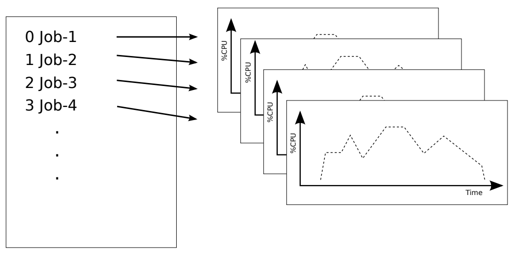

Figure 5: Workload.

4.2 Cloud Topology Modelling

As cloud data-centers grow up hardware heterogeneity also increases. These differences among hardware functionalities also is found at network level. Not all hosts in the data center have the same bandwith or are located at the same distance in time from each other. For this reason in the simulator we can define the network topology of the datacenter. This feature is used in the migration process. The network topoligy configuration file defines distances among machines in the same data center and distances among data centers. By this way we can know the cost in time of a migration between diferent machines that will vary in deppending on the choosen nodes. This information is also given to the scheduler who can take advantage of that at decision time.

4.3 Host Modelling

In a similar way that we can configure the workload, we can set the specification for each host. One of the most important characteristics in a Cloud is, probably, the heterogeneity, not only in the different type of jobs that it can execute but in the different kind of hardware that the Cloud runs on. May be this is the major difference between a data-center and a Cloud and our simulator is able to simulate that heterogeneity. We can configure the CPU amount, CPU frequency, memory and creation and migration characteristics for each host. The only condition is that all will have the same internal architecture, all hosts are supposed to have the same virtualization technology, Xen in this case. This is assumable because we simulate large workloads and regardless of the architecture the global behaviors of a virtualized systems are very similar. We do not simulate the very low level architecture for each type of machine, but we do simulate behaviors of the different machines. The model parameters are slightly different for each type so the simulation results are also different for each type. Although we do not simulate specifically each type of architecture, our approach is open enough to reproduce the global behavior of both of them, simply tuning the configuration parameters for each host.

As seen in Section 4.1, the workload generator module sends the jobs to scheduler's core module at the specified arrival time. Notice that although we have set the time at when the jobs must arrive into the system the simulator is able to speedup the global time. This makes the simulation go faster than real experimentation.

The simulator's core manages both, the connection with the scheduler and the host simulation. In Figure 6 we can see a picture of the HyperSched's internal architecture. The workload generator sends jobs to the Pluggable Scheduler who interacts with the real scheduler and then sends the job to the corresponding simulated machine. As we said, we can configure the number and the characteristics for each simulated host.

The simulator's communication with the scheduler is done using XML-RPC calls. The communication is one-way, always from simulator to scheduler, for reverse communication we have to use a polling mechanism in the simulator. Having bi-directional communication between the core and the scheduler would improve the overall simulation and give more freedom to the scheduler but it would require the creation of a XML-RPC server inside the OMNeT++ simulator which is a non feasible task because of structural problems of the platform.

Notice also that the scheduler must be "event driven", it cannot have any real time references or timers. The "real Clock" is driven by the simulator who accelerates or slows down according to its convenience.

One of the major challenges we had to face during the simulator's designs

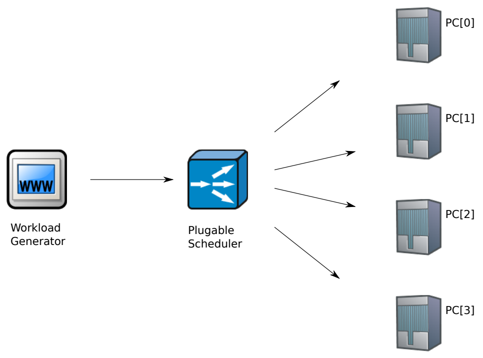

Figure 6: Simulator Modules

was the virtualization double scheduling. Xen hypervisor schedules the virtual CPUs (VCPUs) among the real CPUs and the virtual machines, at the same time, schedule the job's quantums among that VCPUs.

The solution proposed must be fast, scalable and provide all the features that real Xen offers and are used by ours schedulers, like migration and virtual machines allocation tuning. The design proposed is to simulate each virtual machine like it was real, this is allocating quantums of VCPUs to each job in the machine. At each quantum the the job's duration is decreased proportionally to the real CPU assigned to the virtual machine. The HyperScheduler, instead of having a queue and assign quantums of real CPU to each VCPUs as it is done in real Xen, it calculates the amount of CPU it has to be allocated to each virtual machine, considering capability and weigh parameters, and tells each virtual machine its corresponding value.

Every fixed period of time ("internal quantum") or each time a new virtual machine enters/exits the host, the HyperScheduler collects the %VCPU usage and using that values recalculates the real CPU assignation and the power consumption. Notice that the HyperScheduler needs to now the amount of CPU used in each virtual machine to share the surplus of CPU among the machines that are going to use it if there is any.

The proposed solution avoids a double-leveled scheduling that would slow down the overall simulation speed and generate hundreds of internal messages in the simulator which would be an impediment for scalability.

In Figure 7 we can see the internal architecture and the communication among the components corresponding to the CPU scheduling. This design is the same for all the hosts in the Cloud, although we can configure parameters like CPU, memory, creation and migration costs, boot time, and power consumption measures.

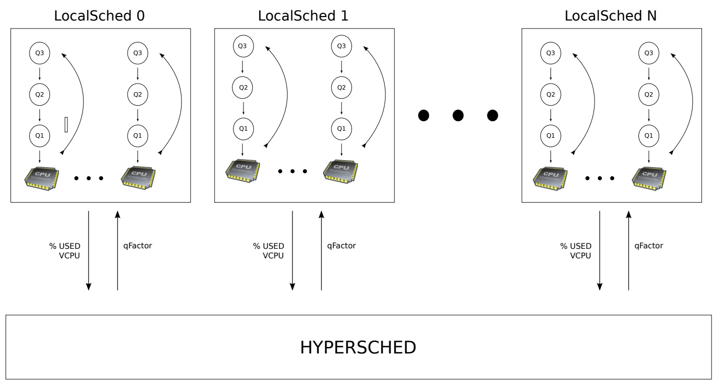

Figure 7: Virtualized Host Simulation Modules

4.4 Virtual Machine and Application Modelling

As we can see in Figure 7, HyperSched simulates the host and LocalSchedulers modules simulate the virtual machines. Each LocalScheduler simulates the job execution inside a virtual machine with different characteristics. That configuration parameters are specified by the Scheduler at the creation time of the virtual machine.

In order the unsderstand how we simulate the job execution we have to distinguish between the two types of jobs supported by the simulator: HPC and Transactional jobs.

For each HPC job we have a trace that indicates the CPU load at every second. At every second of execution we know the amount of CPU the job is consuming. At every quantum we check if the virtual machine have enough real CPU to fulfill the job's trace requirements and if that is the case we decrease the total duration of the job. Other ways, we decrease an amount of time proportionally to the real CPU assigned to the VM. This proportion factor of CPU is calculated by the HyperSched for all the virtual machines in the host. The amount of decreased time is also modified by the frequency scaling. The job trace it is supposed to be measured in a specific machine with a specific frequency, as we have this information in the job configuration parameters we increase or decrease the quantum size we decrease the duration. Using this mechanism we can simulate the jobs behaviors when changing frequencies. Notice that we are talking about HPC jobs which most of its running time is CPU time so their speed execution is proportional to the frequency variations.

Transactional have a more complex simulation algorithm. At this moment we have support for a Tomcat Web Application Server [15] simulation. When we want to simulate this type of application we have to send a job to the system with both the characteristics of the server and the workload. In this case, the workload is a trace of requests/second at every second of execution time. For transactional jobs the simulator have to calculate both the CPU consumption and the response time (R) that the server has at every moment. In order to do that, we get the requests/second at the current time and calculate the amount of CPU needed by a server, with the specified configuration, to serve that requests without getting overloaded. If there is enough CPU in the virtual machine to achieve that we have a low R, in other case we have to calculate, again using configuration parameters, the R that we would have for that requests/sec and the available CPU of the moment.

Although at this moment the system does not support having many jobs in one virtual machine the LocalSchedulers are able to do it. This is because at each host there is a special LocalScheduler, named domain0, that simulates the dom0 virtual machine in a real environment. This LocalSched simulated the creation, destruction, migration and checkpoint tasks in the host. As we have seen in our experiments this virtual machine has an extra overhead proportional to the number of virtual machines in the host, our simulator also takes this into account.

4.5 Power Consumption Modelling

Power consumption calculation is done using the CPU load values that we have at each host. We have studied the behavior of power consumption when we change CPU load in a machine and we have derived the behavior and introduced it into the simulator.

We measured the real power consumption in a host with any virtual machines. We used different workloads in a 4-CPU computer whose kernel included some energy efficiency policies. The power consumption of the machine was gathered using digital methods via an ammeter. In the past, analogical methods via oscilloscope where used, as seen in [16], but similar results are obtained with the ammeter method (however, instantaneous wattage is lost; we can only measure stable workloads). The resolution of the measurements is below 1 Watt. Figure 9 shows the system behavior; we can see that wattage increases with the workload (in a non constant slope), but that it is noticeable even in an idle machine, which is the main reason why we can gain by consolidation. This graph was included in the simulator, as part of the model. In other machines, power usage does not change with the load and is constant. These machines should be avoided because no wattage reduction can be obtain even adding energy efficiency capabilities to the kernel. It is important that idle wattage level should be decreased in the industry as it is one of the most used states and it is not energy efficient, as seen in [17].

Using these measures we can tune the simulator and get results as closer as possible to real ones. Although these measures tell us the amount of power as a function of the CPU consumption, we need to understand how tasks are scheduled in the host when there are several virtual machines.

Depending on how are the VCPUs scheduled we can have more or less power saving, for example, if we have 2 virtual machines with 1 virtual CPU each one and both are consuming 50% of that CPU, the Xen internal scheduler could allocate 50% of 2 different CPUs or 100% of a single CPU, that would cause no

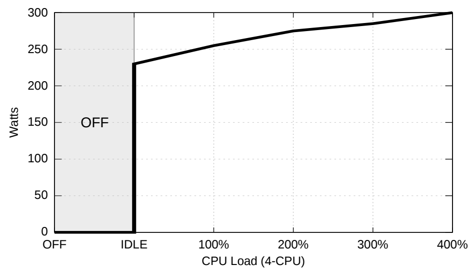

Figure 8: Power behavior of the target PC

difference in results but would vary the total power consumption. As we seen in Figure 9 there is not much difference in using one or two CPUs for a single machine, but in a cloud data-center we have hundreds of machines and this can be crucial for power saving.

As we expected from previous measurements, there is a minimum consumption when the machine is turned on. This is the idle machine consumption that for our machine is 230W, which corresponds to a machine with any virtual machines in it and executing nothing. We measured the power consumption having different number of virtual machines in the host and changing also the CPU consumption that each virtual machine did, the results obtained shows that there is no dependence in the number of virtual machines and in how they are configured. The only real dependence is with the total CPU consumed by the VM machines. Table 1 shows some of the obtained results. The first column is the number of virtual CPUs used by each virtual machine, the + sign indicates more than one virtual machine at the host. The second one shows the CPU consumed by each virtual machine and the last column the measured power for each configuration.

# VCPUs	CPU Usage	Power
1	100%	259 W
2	200%	273 W
3	300%	291 W
4	400%	304 W
2	100%	259 W
1 + 1	100% + 100%	273 W
1 + 2	100% + 200%	291 W
1 + 1 + 1 + 1	100% + 100% + 100% + 100%	403 W
1 + 1 + 1 + 1	0% + 0% + 0% + 0%	230 W

Table 1: Virtualized server power usage

Notice that the real CPU consumption is linked to the consumption in each virtual machine, so 100 virtual machines executing nothing produce 0 consumption, only the needed for having the host turned on which is 230W in our case.

The conclusion we can extract from data in Table 1 is that the real CPU usage is the sum of the CPU consumed by each virtual machines. This is not an obvious conclusion as Xen hypervisor schedules real CPUs among the running VCPUs and it could be possible that these VCPUs where distributed on all CPUs causing higher power consumption. As we can see in our system it has the same effect to have 2 virtual machines consuming 100% of CPU each one than having an application that consumes 200% of CPU.

Based on the information we obtained in Figure 9 and in Table 1 we adjusted our simulator to have a linear response in power. At every moment we know the amount of CPU that is consuming our host, so we can obtain the power that it would consume. As each type of host is different we can configure the different values from which we create the linear regression model and finally we obtain a $fpow = A \cdot CPU + B$ different for each host. That gives us a very good approximation to power consumption for each type of machine.

As we observed in our experiments that power consumption in real machines is not exact at all, our simulator introduces small deviations in the aforementioned formula to obtain a more realistic behavior.

4.6 Scheduler

As it has been presented in Section 2.2, EMOTIVE is able to support different schedulers and the presented simulator is able to use these schedulers. The simulator controls the whole timing of the system and manages the simulated tasks and resources. It informs the scheduler about the status of the VM (when they are submitted, finished, migrated…) and the hosts (informs about the available resources, if a local resource has been powered on, crashed…).

In order to support this communication, the simulator has a XML-RPC wrapper for the generic EMOTIVE schedulers which allows using them. This is done by informing the scheduler about the changes in the system and retrieving the decisions of the scheduler. This last communication is performed using an event queue where the scheduler leaves the decisions about the system (VM creation, migration or destruction, and the management of the nodes) and the simulator periodically checks this queue.

For example, executing a task in this environment is performed in the following way: the simulator firstly reads from the file which specifies the workload the tasks to be executed in the system. Once the time to submit a task has been accomplished, the simulator tells the scheduler it has a new a VM to create and this decides in which resource it will be started (local resource or external resource). At this point, the simulator gets this information and starts the VM in the given resource. Finally, when the task inside that VM has finished, it informs the scheduler about this and destroys the VM.

With all the system information, the scheduler does not only manages the VMs but it also can decide about when to power off resources or starting them again according to its policies.

4.7 Simulator output

One of the most valuable characteristics of the simulator is that we can obtain information at different levels and for all the hosts and executed jobs. By default the simulator shows result traces with information about: the Jobs creation, migration, destruction; the current powered on a powered off machines during the execution; total consumed power and CPU for each host and if we have Transactional jobs we also will obtain the response time of the server. Notice also that as we are simulating we can also customize the information we obtain. Each trace is timestamped so we can know exactly what is happening at every moment in the simulated system.

5 Experimentation

In this section we expose the two types of experimentation that validates our approach. Firstly we will compare the result produced by the simulator with the values obtained in a real execution i norder to validate the simulator architecture and behavior in both power and execution. Second we will present some scalability experiments to demonstrate that the simulator is able to represent very large cloud networks.

5.1 Power and Execution Validation

The simulator is validated comparing the results obtained with the same workload in a real data-center and in the simulator. In this case the workload used for validation has been designed for testing all possible situations we can find in real environments. We have measured the power consumption and the usage parameters in one host during 1300 seconds that lasts the validation workload and we also have simulated the same workload.

The results obtained where a total power consumed in the real experiment of 99.8 ± 1.8 Wh while the simulation estimated a total cost of 97.5 Wh which is an underestimation of 2,3%.

In Figure 9 we can see the power consumed in both the real an the simulation. A we can see in the figure the executions have not the exact sequence in CPU load, this is because the simulator cannot imitate the exact behavior of the real framework because of the simulator does not takes into account very low level details. That leds to slightly different instantaneous behaviors. That difference in CPU load is translated into a difference in the instantaneous CPU consumption, but not into the overall execution values. It is for that reason that altough the consumption can be slightly different during the execution the total results are almost equal and what is most important the global trend of the simulator is equal than the real system.

Although the simulator tries to reproduce the real software behavior, there are some unpredictable behaviors in real execution that can not be simulated. Notice also that the simulator pretends to obtain large scale execution results and its results are not applicable to short simulation. The ultimate idea of

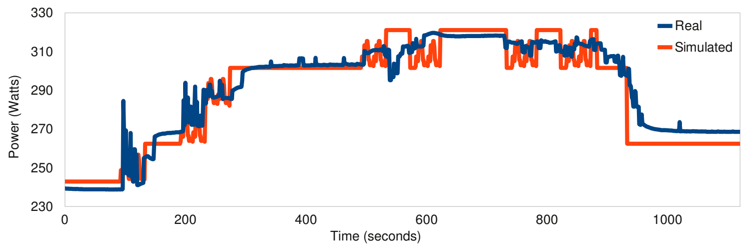

Figure 9: Execution validation

our project is test the goodness of some proposed solutions in scheduling and resource managing not to have a Cloud emulator. Figure 10 compares the distribution of load caused by a sample workload at each moment, we have the theoretical load calculated considering the creation/migration times and loads, the simulated and the real results. We can observe in that figure that the simulation and the real execution are slightly different, this is because of the non predictable behavior in the real environment and that will make that the results obtained in power and CPU load will not be exactly the same, even the simulator had no error. For that reason it is almost impossible to obtain error smaller that 2% when comparing the simulation with a real execution.

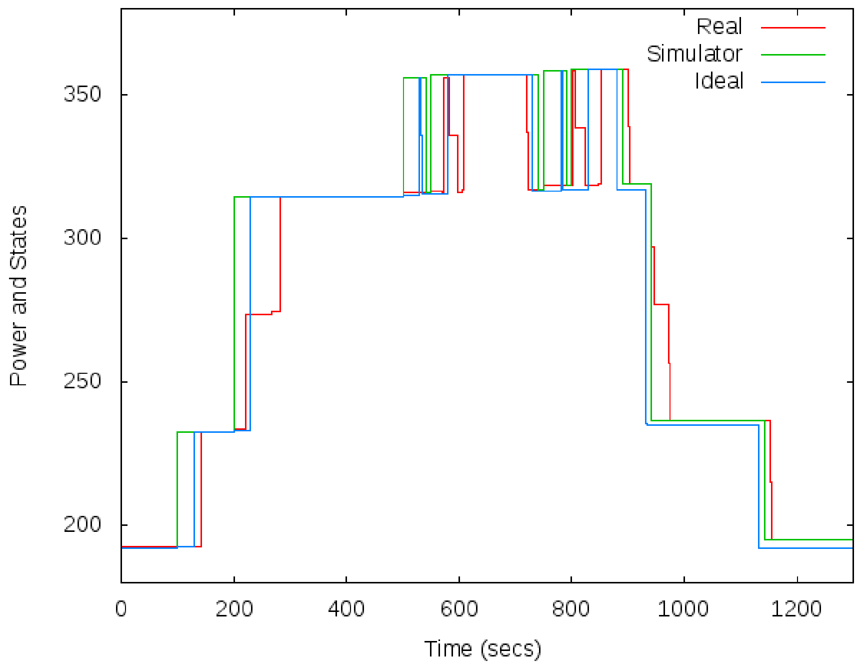

Figure 10: Simulator Model Validation in State

We consider satisfactory the results obtained in the validation experiment, the experiment has tested also all the possible situations we can find in a standard workload and we also believe that the 2,4% of error for a 1300 seconds test would be reduced in larger executions. The simulator is optimized to reproduce the overall behavior of the Cloud which will be much more accurate in larger workloads. Although the validation has been done in one host, due to foreign restrictions, if we can validate the results obtained for one host we can extrapolate these results to larger ones.

5.2 Scalability

In this section we present some scalability tests done to the simulator. We have done two kind of tests, the former shows the scalability of the simulator when we change the number of jobs we can handle in a single host. The results for this executing are exposed int table 2. As we can see the table shows the time duration of the experiment depending on the number of jobs we send to the host. Notice that all the jobs are sent to the host at the same time. For this reason we expect the simulation to last more as much jobs has to be executed, it will also occur in a real system.

# Jobs	Duration (secs)
1	44
10	45
100	62
1000	1283
10000	813600

Table 2: Scalability Test in 1 Host

Table 3 shows the results for the second scalability test. In this case we want to test the number of hosts that the simulator is able to handle and how this affects to the global system. To do such experiment we have made several executions changing the number of hosts in the system and the workload. We sent one job per host, using this technique we can ensure that the entire simulation is done in all the hosts. As we can see in the table the simulator is able to have simultaneously 1000 hosts running with 1 job at each one. The execution with 10000 of host is not supported by the moment due to the XML-RPC performance when connecting the scheduler with the simulator.

6 Conclusion

We have developed a simulator for cloud that gives us the possibility to validate different scheduling policies in very large clouds and using not common metrics like power. It is easy to use and simulates weeks of execution in a big data center in minutes which makes the development process easier and faster. In addition,

# Hosts/Jobs	Duration (secs)
1	47
10	143
100	1324
1000	15824
10000	-

Table 3: Scalability Test in Host number. There is one job per host.

thanks to its output metrics, it is easy to evaluate the power performance of a system by testing different solutions and approaches in a few minutes.Simulator has been proof to be totally worth when developing scheduling policies for a virtualized data center and take advantage of new features such as migration.

The simulator's design process has been done following the simulation development standards and we also have validated the results obtained from the simulator with results obtained from real execution experiments, achieving not only very accurate trend behaviours but measures that are very close to a real system.

The future work for the simulator is to upgrade the transactional features, giving more precision and configuration possiblitities to this type of aplications. We also want to add Input/Output metrics to de simulation.

References

[1] Amazon. Amazon elastic compute cloud. `http://www.amazon.com/ec2`.

[2] D. Chappell. Introducing the Azure Services Platform: An Early Look at Windows Azure, .NET Services, SQL Services, and Live Services, 2008.

[3] Aneka Software.
http://www.manjrasoft.com/products.html.

[4] Iigo Goiri, Jordi Guitart, and Jordi Torres. Elastic Management of Tasks in Virtualized Environments. In *Proccedings of the XX Jornadas de Paralelismo (JP 2009)*, pages 671–676, 2009.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, 2003.

[6] J. Ejarque, M. de Palol, I. Goiri, F. Julia, J. Guitart, R. Badia, and J. Torres. SLA-driven semantically-enhanced dynamic resource allocator for virtualized service providers. In *4th IEEE International Conference on e-Science (e-Science 2008), Indianapolis, USA*, 2008.

[7] S. Govindan, A.R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam. Xen and co.: communication-aware CPU scheduling for consolidated Xen-based hosting platforms. In *Proceedings of the 3rd international conference on Virtual execution environments*, page 136. ACM, 2007.

[8] L. Cherkasova, D. Gupta, and A. Vahdat. Comparison of the three CPU schedulers in Xen. *PERFORMANCE EVALUATION REVIEW*, 35(2):42, 2007.

[9] Xen Wiki Credit Scheduler. http://wiki.xensource.com/xenwiki/CreditScheduler.

[10] A. Vargas. Omnet++.

[11] D.A. Menascé and M. Bennani. On the use of performance models to design self-managing computer systems. In *CMG-CONFERENCE-*, volume 1, pages 1–10. Citeseer, 2003.

[12] Omnet, 2009. `http://www.omnet.org`.

[13] Ramon Nou, Samuel Kounev, Ferran Julià, and Jordi Torres. Autonomic QoS control in enterprise Grid environments using online simulation. *J. Syst. Softw.*, 82(3):486–502, 2009.

[14] The Grid Workloads Archive, 2009.
http://gwa.ewi.tudelft.nl.

[15] Apache Tomcat. http://tomcat.apache.org.

[16] Ramon Nou. Energy Efficiency: A case study. Technical Report UPC-DAC-RR-CAP-2009-14, Technical University of Catalonia (UPC) - Computer Architecture Department, 2009.

[17] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.