Design of a Flexible and Scalable Hypervisor Module for Simulating Cloud Computing Environments

A. Núñez[1], G. G. Castañé[1], J. L. Vázquez-Poletti[2], A. C. Caminero[3], J. Carretero[1] and I. M. Llorente[2]

[1]Dept. de Informática. Universidad Carlos III de Madrid, Spain.
{anunez,gabriel,jcarretero}@arcos.inf.uc3m.es
[2]Dept. de Arquitectura de Computadores y Automática. Universidad Complutense de Madrid, Spain
jlvazquez@fdi.ucm.es,llorente@dacya.ucm.es
[3]Dept. de Sistemas de Comunicación y Control. Universidad Nacional de Educación a Distancia, Spain
accaminero@scc.uned.es

***Abstract*—Cloud computing is a paradigm which allows a flexible and dynamic provision of resources in order to solve problems pertaining to a great variety of domains. When considering public cloud infrastructures, those which follow a *pay-as-you-go* philosophy, it is clear that optimizing usage costs results in the main problem to solve. For this reason, simulation tools have become a strong ally when determining the best infrastructure setup. In this contribution, we introduce the hypervisor module, which is a key component of iCanCloud, our simulation platform. Definition of brokering policies is explained in the context of iCanCloud and two novel policies are introduced.**

I. Introduction

Nowadays, cloud computing systems are increasing their role due to the fast (r)evolution of computer networks and communication technologies. A very clear proof of this fact is that very important companies like Amazon, Google, Dell, IBM, and Microsoft are investing billions of dollars in order to provide their own cloud solutions [1].

The entire architecture of cloud computing systems lies in the concept of virtualization, which consists on the abstraction of the physical resources offered to users, who have the illusion to use the physical resources transparently. In fact, due to cloud systems decouples resources from their users with the purpose to provide high levels of scalability and flexibility, new challenges arise to balance trade-offs between *costs* and *performance*.

At present there are mainly two cloud infrastructure types. On the one hand, a *private cloud* is a system where the user's institution maintains the physical infrastructure where the VMs will be executed. These cloud infrastructures can be built using virtualization technologies like Nimbus [2], OpenNebula [3] or Eucalyptus [4]. On the other hand, the cloud service can be outsourced by paying each deployed VM per unit of time basis - this being called *public cloud*. Some examples of public clouds are ElasticHosts[1] and Amazon's Elastic Compute Cloud[2].

[1]http://www.elastichosts.com/
[2]http://aws.amazon.com/ec2/

One of the main features of those systems is the ability to horizontally scale computing resources with the purpose to satisfy the user's demand. This concept basically consists on the ability which users can execute their applications within an infrastructure that is able to scale up resources as demand for the application increases, and scale down when demand decreases. However, this level of scalability and flexibility does not guarantee that the resources requested by users provide a favorable functionality.

When using a public cloud infrastructure and in order to alleviate the burden to optimize costs depending of the tasks to be executed by users in the cloud, it is necessary to use tools that provide to users an estimation of those costs. This necessity is the result of a redefinition of the brokering concept by adding a new level of complexity to the scheduling problem, which has been studied in other computing paradigms such as cluster or grid. Being cloud computing a layer which floats around the other paradigms, resource provisioning becomes this new level of complexity. In order to tackle this new complexity level, some metrics have been developed [5].

Due to the lack of open source cloud computing software platforms that support horizontal scaling, and the cost required to execute experiments using a wide spectrum of configurations, the use of modelling and simulation tools targeted to cloud computing becomes a very useful an feasible option. In this work we propose a new hypervisor module, which is integrated into a cloud infrastructure simulation platform. The main feature of this proposal is the high level of flexibility provided to use custom scheduling policies, depending on the user requirements such as time, cost and resource utilization. This simulation platform is named iCanCloud and is currently available at http://www.icancloudsim.org/.

Our contribution is threefold:

1) We have designed and implemented a flexible hypervisor module that provides an easy method for integrating new cloud brokering policies.
2) We show how iCanCloud can be used in practice to

implement any cloud brokering policies.

3) We propose new brokering policies targeted to public cloud infrastructures and based on classic scheduling heuristics.

The rest of the paper is structured as follows: Section II presents related work in simulations in computer science; Section III describes a brief overview of the iCanCloud simulation platform; Section IV presents the design of our proposed hypervisor module; Section V describes two novel cloud brokering policies and their implementation within the iCanCloud simulator. Finally, Section VI draws conclusions and suggests guidelines for future research.

II. Related Work

Simulations have been widely used in different fields of computer science over the years. Focusing on Cloud Computing, [6] introduces a model for characterizing the usage of resources pertaining to a private cloud infrastructure. However and to the authors' knowledge, the only complete tool that can simulate a real cloud system is CloudSim [7]. Although CloudSim was presented very recently [8], several research articles have been published presenting results obtained with it [9] [10] [11] [12].

The work [13] describes a discrete event simulation model that is used to explore the relationship between the horizontal scaling profile configurations and the functionality of the cloud model. Initial results show that both a state-aware load distribution algorithm and the parameters that dictate the elasticity of the horizontal scaling ability are essential to achieve high rates of utilization. Through modelling and simulation, this work presents both a framework and initial results to further explore the cloud model.

Regarding hypervisors, a number of them exist in practice, among others Xen [14], KVM [15], VMWare Player[3], or VirtualBox[4]. They provide isolation between the virtual machines and the physical resources running them, taking the needed provision actions. In this contribution we introduce the way iCanCloud simulates this behavior.

III. Overview of iCanCloud

The ever-increasing complexity of computing systems has made simulators a very important choice for designing and analyzing large and complex architectures. In the field of cloud computing, simulators become especially useful for calculating the trade-offs between cost and performance in *pay-as-you-go* environments. Hence, in this work we have used a simulation platform for modelling and simulating large cloud environments, named *iCanCloud*, which represent both actual and non-existent cloud computing architectures. The main aim of this tool, is to estimate the trade-offs between cost and performance of a given application executed in a specific hardware, and then provide users with useful information about such costs.

[3]http://www.vmware.com/products/player/

[4]http://www.virtualbox.org/

The iCanCloud simulation platform has been built on the top of SIMCAN simulation platform [16]. Figure 1 shows a layered schema and the relations between iCanCloud and the rest of the simulation utilities. Thence, the models for simulating the hardware parts of the cloud environment and the application models belong to those modules included in the repository of SIMCAN.

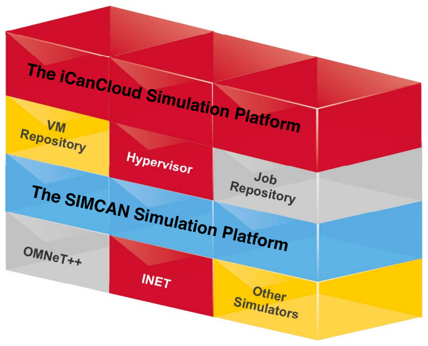

Fig. 1. Basic layered architecture of iCanCloud.

Developing simulations of cloud systems is a task that involves both the simulation of the computer system and the simulation of the applications that run on it. Each user has his own agenda regarding which applications have to be simulated. A good simulation platform should supply the user with a good set of features that ease the development of these applications. Therefore, the iCanCloud simulation platform provides a POSIX-based API and an adapted MPI library for modelling and simulating applications. Also, several methods for modelling applications can be used in iCanCloud: using traces of real applications; using a state graph; and programming new applications directly in the simulation platform.

However, the key innovation of this simulation framework lies in a modular and flexible design. Figure 2 shows the UML 2.3 class diagram of the iCanCloud simulation platform. This model is split in two different parts. On the one hand, dark grey squares represent the hardware part of the cloud environment, where those modules belong to the SIMCAN simulation platform. In this simulation platform, a physical node is defined by the configuration of the 4 basic systems: computing, memory, storage and network.

On the other hand, the light grey squares represent those modules in charge of managing the cloud system. The VM class acts as a link between the physical resources of the cloud environment, such as nodes and networks, and the resources used by users, giving them an illusion of using directly the physical resources to execute the corresponding jobs. Thus, depending of the configuration of each VM, those are mapped to the physical nodes existent in the cloud system model. The main module of this section is the Hypervisor, which is the center piece of the system and the genesis of this work.

The Hypervisor module is the master key of the iCanCloud simulation core. Basically, this module is in charge of managing the virtual machines and executing the jobs defined by users. In order to accomplish this task, the Hypervisor can be fully configured and customized depending of the pursued goals to be fulfilled.

IV. Basic architecture of proposed hypervisor

The main goal of this module is to let users schedule a set of jobs in cloud systems using their own brokering policies, such as cost, execution time and resource allocation. In order to fulfill this requirement, the proposed hypervisor contains a very intuitive API. Thus, new customizable schedulers can be implemented using the functions provided by this API (see listing 1).

The method *get_job_list()* obtains a list of all submitted jobs; *get_job (jobID)* obtains the object associated with the identifier *jobID*; similarly, *get_vm (vmID)* returns the object corresponding to the virtual machine *vmID*; *insert_running_q (userID, jobID, vmID)* inserts the job *jobID*, submitted by the user *userID*, to be executed in the virtual machine *vmID*; *run_job (vmID, jobID)* executes the job *jobID* in the virtual machine *vmID*; move_Rq_to_Fq (jobID) moves the job *jobID* from the running queue to the finish queue; Finally, job_has_finished (jobID, results) indicates that the job *jobID* has finish with the feedback *results*.

```
jobList get_job_list ();
jobObject get_job (jobID);
vmObject get_vm (vmID);
void insert_running_q (userID, jobID, vmID);
void run_job (vmID, jobID);
void move_Rq_to_Fq (jobID);
void job_has_finished (jobID, results);
```

Listing 1. Functions provided by the API of the hypervisor.

Similarly, the scheduler module has to implement an interface in order to provide a right communication with the hypervisor. Basically, this interface consists of two functions that are shown in listing 2.

The method *select_job()*, selects the next job to be executed; similarly, *select_vm()* selects the virtual machine to execute next job; *job_has_been_inserted()* indicates that a new job has been submitted by a user; Finally, *job_has_finished(jobID)* indicates that job jobID has finished its execution.

```
jobID select_job ();
vmID select_vm ();
void job_has_been_inserted ();
void job_has_finished (jobID);
```

Listing 2. Functions of the scheduler's interface.

Figure 3 shows the basic schema of the proposed hypervisor module. Basically, the main parts of the hypervisor model are:

- Waiting queue: This queue handles jobs that have been submitted to the system, but don't have started their execution.
- Running queue: This queue handles jobs that are currently running on any virtual machine.
- Finished queue: This queue handles jobs that have finished their execution.
- Scheduler: This module is in charge of selecting the jobs to be executed in the idle virtual machines.
- Set of VM: This set contains a list of all those VM that have been started up in the cloud system.

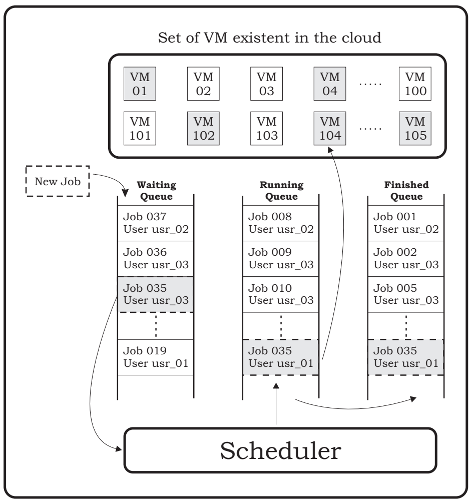

Fig. 3. Basic architecture of proposed hypervisor.

The key module of the proposed hypervisor is the scheduler module. The underlying idea of this module is to select the next job to be executed in the cloud, and the virtual machine instance where that job will be executed. Thus, new scheduler modules can be easily integrated into the proposed hypervisor just by implementing the methods shown in listing 2. Hence, the scheduler may use methods provided by the hypervisor API (see listing 1), and the relevant information of each job submitted to the system. This information is basically the number of instructions to be executed, the size of the data to be read, and the output data to be written, which can be retrieved from each job object.

Figure 4 shows a sequence diagram of a typical use case, since a user send a new job to the cloud system, until that job is done and the hypervisor collects the corresponding results.

V. Proof Case: Implementation of provisioning and scheduling algorithms

To illustrate the usefulness of this work, two provisioning and scheduling algorithms (namely, Cloud Min-Min and Cloud Max-Min) are implemented using iCanCloud. The algorithms are divided in two main phases:

1) Provisioning, where a specific type and number of EC2 instances are deployed.
2) Scheduling, where tasks are matched with available resources.

In order to perform the provisioning, [5] introduced a metric named Cost per Performance (C/P). This metric relates the time it takes to execute a number of tasks in Amazon EC2, with the cost required to hire the machines that produce this execution time. This way, the best infrastructure setup would be that which produced the lowest C/P value.

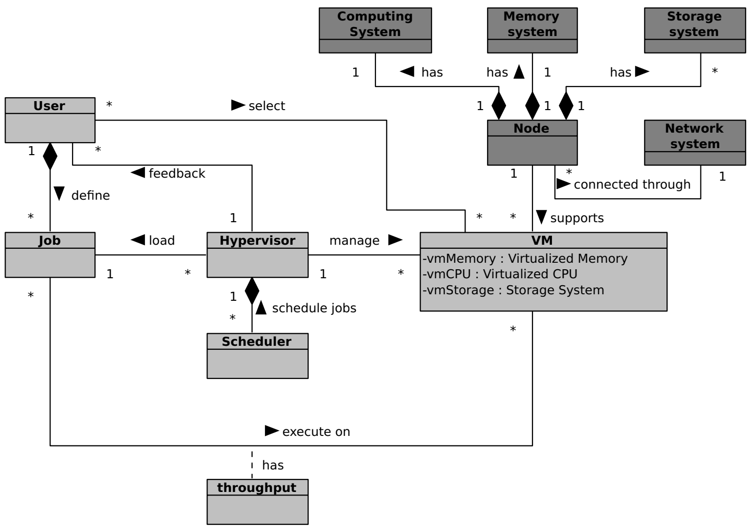

Fig. 2. UML class diagram of iCanCloud simulation platform.

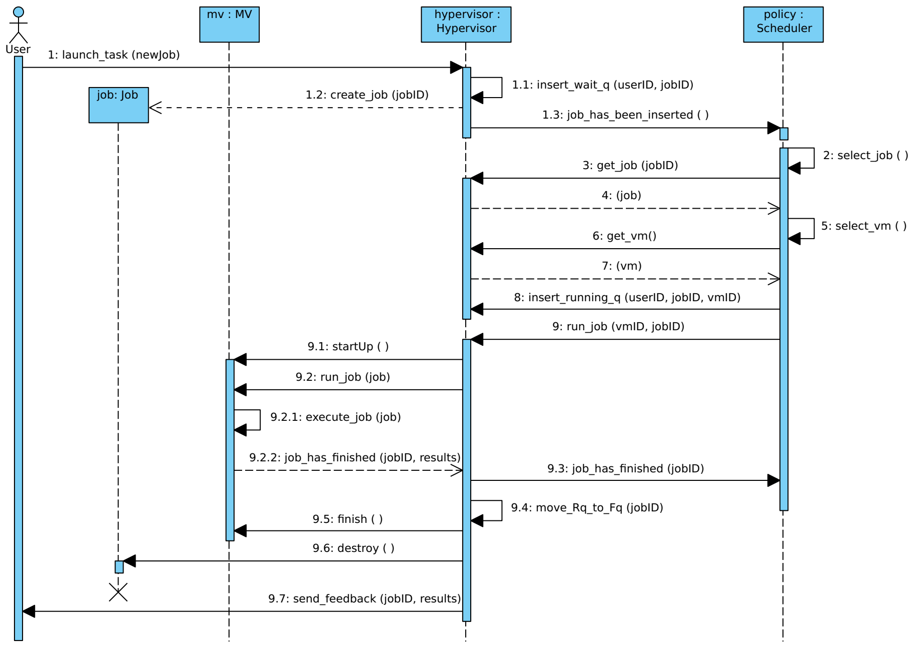

Fig. 4. Sequence diagram of the proposed hypervisor model.

A. *Cloud Min-Min*

In the Cloud Min-Min algorithm, the n tasks with shortest execution time are taken for obtaining its best virtual infrastructure deployment. A sweeping is made with different values of n to achieve the best value for the C/P metric. In the case of having similar values, decision turns to less execution time or to less cost. Task scheduling follows the same procedure as the original Min-Min algorithm [17].

The pseudocode of this heuristic shown in Algorithms 1, 2 and 3. Initially, an array of execution times is created as Algorithms 1 depicts, which uses the job size in millions of instructions (MI) and VMs power in millions of instructions per second (MIPS).

After that, Algorithm 2 describes the resource provisioning phase. As the resource provisioning consists in choosing the best configuration, a loop is made for using a different number of j tasks from the total tasks N to be scheduled (line 11). Then, temporal arrays containing the execution times of tasks in each VM instance are created, to be used during the execution of the algorithm (line 13).

Next, another loop creates a vector (P_v) for each instance type v with the j shortest tasks (line 19) and removes the shortest task from the array of execution times of the given v (line 20). As can be seen in line 19, each P_v is ordered from

Algorithm 1 Creation of arrays of execution times.

```
1: Let $N$ = the number of tasks.
2: Let $x$ = the number of instance types of VM.
3: Let $T^v$ = the task execution time vector for the instance type $v$.
4: Let $jobID$ = the identificator for a job.
5: Let $v$ = the identificator for a VM.
6: Let $vm$ = a VM.
7: Let $job$ = a job.
8: $jobList = get\_job\_list()$
9: for $jobID = 0$ to $jobList.size() - 1$ do
10:    for $v = 0$ to $x - 1$ do
11:       $job = get\_job(jobID)$
12:       $vm = get\_vm(v)$
13:       $T^v(jobID)$ = $job.getMI()/vm.getMIPS()$
14:    end for
15: end for
```

the lowest execution time (held at $P_v[0]$) to the highest (held at $P_v[j-1]$). This line is one of the few differences between the Cloud Min-Min algorithm and Cloud Max-Min.

Then, a temporal CP metric is calculated. This loop is performed to set the number of instances for each instance type provided by Amazon EC2 so that the j tasks are executed in parallel (line 25). This followed by the calculation of C/P metric, which is obtained for each instance type (line 26). Being the smallest value for this metric the best one, it may happen that it is achieved by several combinations of n instances of type x. At this point with equal C/P values, the setup is chosen considering the shortest execution time or the lowest cost (line 30). If there is only one configuration of instance types yielding the minimum C/P value, it is chosen as the C/P metric for this j (line 32).

At the end of the provisioning main loop, there will be a $CP[j]$ vector that contains the best C/P values from the previous iterations – this is, for each number of tasks j considered. To conclude this algorithm, the infrastructure with n machines of type x with the minimum C/P value is then instantiated (line 35).

Finally, the scheduling phase depicted by Algorithm 3 is performed. Initially, the jobs with the n lowest execution times are submitted to the n VMs (line 9). If there are more jobs than VMs, jobs are submitted to VMs as they become idle from the executions of previous jobs (line 16).

B. Cloud Max-Min

This algorithm is similar to Cloud Min-Min presented above, and is not presented in detail for space limitations. The only differences are (1) in Algorithm 2, line 19, in which the job with the lowest execution time is inserted at the end of P_v array; and (2) in Algorithm 3, lines 10 and 18, which choose the job with the highest execution time.

Algorithm 2 Resource provisioning phase.

```
1: Let $N$ = the number of tasks.
2: Let $x$ = the number of instance types of VM.
3: Let $T^v$ = the task execution time vector for the instance type $v$.
4: Let $V_{v,n}$ = the set containing $n$ virtual machines of a $v$ instance type.
5: Let $j$ = the number of tasks used each loop in the resource provisioning phase.
6: Let $CP$ = the vector that keeps the $C/P$ metric for each $j$ for all the instance types.
7: Let $CP_{temp}$ = the temporary vector that keeps the $C/P$ metric for each instance type for a given $j$.
8: Let $P_v$ = the vector that keeps the execution time for the $j$ tasks with minimum execution time for the instance type $v$.
9: Let $itype$ = the instance type of VMs chosen.
10: Let $n$ = the number of VMs chosen.
11: for $j = N$ downto 1 do
12:    {Creation of temporal arrays of execution times}
13:    for $v = 0$ to $x - 1$ do
14:       $T^v_{temp}$ = $T^v$
15:    end for
16:    {Creation of $P_v$ arrays}
17:    for $k = 0$ to $j$ do
18:       for $v = 0$ to $x - 1$ do
19:          insert($P_v[k]$, min($T^v_{temp}$))
20:          remove($T^v_{temp}$, min($T^v_{temp}$))
21:       end for
22:    end for
23:    {Calculate the temporal CP metric}
24:    for $v = 0$ to $x - 1$ do
25:       setN ($j$, $v$)
26:       $CP_{temp}[v]$ = time($V_{v,n}$) $\times$ cost($V_{v,n}$)
27:    end for
28:    {Finally, calculate the CP metric}
29:    if $|$min($CP_{temp}$)$| > 1$ then
30:       $CP[j]$ = min(time($V_{v,n}$)) or min(cost($V_{v,n}$))
31:    else
32:       $CP[j]$ = min($CP_{temp}$)
33:    end if
34: end for
35: createInstances($V_{itype,n}$) producing min($CP$)
```

VI. Conclusions

The *cloud* has emerged as a new paradigm that allows the creation of dynamic and flexible computing platforms. Considering that it may work in a *pay-as-you-go* basis, the development of techniques to minize budget requirements are more than a need. To this end, the use of simulation tools becomes a key.

This paper introduces the hypervisor module of iCanCloud simulator [18]. This module allows the implementation of brokering policies easily. This paper also introduces two

Algorithm 3 Task scheduling phase.

```
1: Let $minJobID =$ the identificator for the job with the lowest execution time.
2: Let $job =$ the job with the lowest execution time.
3: Let $V_{v,n} =$ the set containing $n$ virtual machines of a $v$ instance type.
4: Let $itype =$ the instance type of VMs chosen.
5: Let $T^{itype} =$ the task execution time vector for the instance type $itype$.
6: Let $n =$ the number of VMs chosen.
7: Let $vmID =$ the identificator of a VM.
8: {Initially, submit the first $n$ jobs to the $n$ VMs}
9: for $vmID = 0$ to $n$ do
10:    $minJobID$ = getMin($T^{itype}$)
11:    $job$ = $get\_job(minJobID)$
12:    $run\_job(vmID, job)$
13:    $vmID++$
14: end for
15: {If there are more jobs than VMs, submit jobs to VMs as VMs become idle}
16: while $getJobList().size > 0$ do
17:    if $checkIdleVM(vmID) ==$ true then
18:       $minJobID$ = getMin($T^{itype}$)
19:       $job$ = $get\_job(minJobID)$
20:       $run\_job(vmID, job)$
21:    end if
22:    $vmID = select\_vm()$
23: end while
```

brokering policies named *Cloud Min-Min* and *Cloud Max-Min*, which are based on the well-known policies called Min-Min and Max-Min [17].

Regarding future work, authors plan to extend the iCanCloud framework to make the simulator parallel. This way, one single experiment can be executed spanning more than one machine, which allows larger experiments to be conducted.

SOFTWARE AVAILABILITY

The iCanCloud simulator is Open Source (GNU General Public License version 3) and available at the following website:
http://www.icancloudsim.org/

ACKNOWLEDGMENTS

This work has been partially funded by the Spanish Ministry of Science and Innovation under the grant TIN2010-16497, and it was supported by the following projects: MEDIANET (Comunidad de Madrid S2009/TIC-1468) and HPCcloud (MICINN TIN2009-07146).

REFERENCES

[1] The Economist, A Special Report on Corporate IT, Web page at http://www.economist.com/specialReports/showsurvey.cfm?issue=20081025.1st October, 2008.

[2] I. T. Foster, T. Freeman, K. Keahey, D. Scheftner, B. Sotomayor, and X. Zhang, "Virtual clusters for grid communities," in *Proc. of the Sixth Intl. Symposium on Cluster Computing and the Grid (CCGRID)*, Singapore, 2006.

[3] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual Infrastructure Management in Private and Hybrid Clouds," *IEEE Internet Computing*, vol. 13, pp. 14–22, 2009.

[4] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proc. of the Sixth Intl. Symposium on Cluster Computing and the Grid (CCGRID)*, Shanghai, China, 2009.

[5] J. L. Vazquez-Poletti, G. Barderas, I. M. Llorente, and P. Romero, "A Model for Efficient Onboard Actualization of an Instrumental Cyclogram for the Mars MetNet Mission on a Public Cloud Infrastructure," in *Proc. of PARA: State of the Art in Scientific and Parallel Computing*. Reykjavik, Iceland: Lecture Notes in Computer Science, in press, 2010.

[6] B. Sotomayor, K. Keahey, and I. Foster, "Combining batch execution and leasing using virtual machines," in *Proceedings of the 17th international symposium on High performance distributed computing*, ser. HPDC '08. New York, NY, USA: ACM, 2008, pp. 87–96. [Online]. Available: http://doi.acm.org/10.1145/1383422.1383434

[7] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience (in press, accepted on June 14, 2010)*.

[8] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities," in *Proc. of the 7th High Performance Computing and Simulation Conference (HPCS)*, Kingston, Canada, 2009.

[9] K. H. Kim, A. Beloglazov, and R. Buyya, "Power-aware provisioning of cloud resources for real-time services," in *Proc. of the 7th Intl. Workshop on Middleware for Grids, Clouds and e-Science*, Urbana Champaign, Illinois, USA, 2009.

[10] R. N. Calheiros, R. Buyya, and C. A. F. De Rose, "A heuristic for mapping virtual machines and links in emulation testbeds," in *Proc. of the Intl. Conference on Parallel Processing (ICPP)*, Vienna, Austria, 2009.

[11] R. N. Calheiros, R. Buyya, and C. A. De Rose, "Building an automated and self-configurable emulation testbed for grid applications," *Software: Practice and Experience*, vol. 40, no. 5, pp. 405–429, 2010.

[12] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges," in *Proc of the Intl. Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Las Vegas, USA, 2010.

[13] J. Idziorek, "Discrete event simulation model for analysis of horizontal scaling in the cloud computing model," in *Proceedings of the 2010 Winter Simulation Conference*, 2010, pp. 3004–3014.

[14] 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," *SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[15] M. J. Hammel, "Managing KVM deployments with Virt-Manager," *Linux Journal*, vol. 2011, no. 201, pp. 1–7, Jan. 2011.

[16] A. Núñez, J. Fernández, J. D. Garcia, F. Garcia, and J. Carretero, "New techniques for simulating high performance MPI applications on large storage networks," *Journal of Supercomputing*, vol. 51, no. 1, pp. 40–57, 2010.

[17] R. F. Freund, M. Gherrity, S. L. Ambrosius, M. Campbell, M. Halderman, D. A. Hensgen, E. G. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," in *Proc. of the 7th Heterogeneous Computing Workshop (HCW)*, Orlando, USA, 1998.

[18] A. Nunez, J. L. Vazquez-Poletti, A. C. Caminero, J. Carretero, and I. M. Llorente, "Design of a new cloud computing simulation platform," in *Proc. of the 11th Intl. Conference on Computational Science and Its Applications (ICCSA)*, Santander, Spain, 2011.