Cloud scheduling optimization: A reactive model to enable dynamic deployment of virtual machines instantiations

Stelios Sotiriadis, Nik Bessis, Fatos Xhafa, Carsten Maple

Abstract: Herein we propose a model for supporting the decision process of the cloud hosts for the deployment of virtual machines in cloud environments. We explore two configurations; the static case in which virtual machines are generated according to the cloud hosts, and the dynamic case in which virtual machines are reactively adapted according to the job submissions, using migration, for optimizing performance time metrics. We integrate both solutions in the same simulator for measuring the performance of various combinations of virtual machines, jobs and hosts in terms of the average execution and total simulation time. We conclude that the dynamic configuration offers improved optimization of the job execution performance.

Keywords: Cloud computing, Virtual Machines scheduling, Virtual Machine migration, Virtual Machine dynamic deployment

1 Introduction

Over the recent years, the cloud paradigm emerged as one of the most important IT infrastructures for delivering services via the public Internet. This based on the clouds' great capability to offer virtualized configuration (computational power along with the required software) (Bessis et al., 2011). In such environments, massive computing capacity resides at a remote space and could be delivered in the form of software, hardware, or developers platform. These offered services are identical to job submissions that have been encapsulated in application execution requests and posed by the end-users, and this is identical to the overall scheduling problem in Grid systems (Xhafa and Abraham, 2010). Hence, at a first glance, cloud computing share similar fundamental elements with other large scale and distributed computing paradigms, e.g. clusters and grids (Bessis et al., 2012b). To this extend, this work adapts the cloud definition (Carolan and Gaede, 2009) that suggests that cloud is about "services that are encapsulated, have an API, and are available over the network".

This inclusive vision allows us to focus on how to orchestrate the cloud service distribution, rather than aim to the deployment of the underlying infrastructure. Usually, the application tasks are submitted to a cloud datacenter through the user interface namely also as broker. The latter acts on behalf of users and is responsible for the communication between cloud low-level organization and users. The tasks that arrive in the cloud are forwarded for execution within the cloud datacenter. This virtualizes required computational power and software in small easy manageable chunks namely as Virtual Machines (VMs). By the use of virtualization paradigm cloud provides the ability to deliver virtual versions of hosts (nodes) by increasing the scalability of the overall setting. In addition, the cloud could contain a lower number of physical machines and servers, and this reduces the management time, maintenance labor and failures handling (Gong et al., 2010).

Similarly, one of the advantages of VMs is that physical space utilization could be augmented within a cloud datacenter by virtualization (Bessis et al., 2012b). However, this decision (taken by the hosts) for the most appropriate deployment of VM instantiation method is a challenging task. This is to say that selecting VMs generation

strategy could be affected by various options. For example, a cloud could initialize a huge number of a small scale VMs to handle more requests, and this will penalize the overall performance. On the contrary, low number of VMs for increasing performance, could conclude with job latencies due to non-availability.
In this work we suggest that the choice for selecting the VM generation strategy could be based on reactively adaptive decisions taken by the hosts and based on historical, current and expected job input. Specifically, by analyzing the characteristics of the cloud jobs (cloudlets) we determine a more efficient VM instantiation model. This includes an initial appreciation of the job features, e.g. the number of submitted cloudlets, the physical resources demanded and the scheduling allocation policy (Bessis et al., 2012a). Although this practice seems more efficient in terms of enhancing the quality of service levels, yet it compromises the overall performance due to the waiting time for VM deployment. Thus cloud hosts need a sufficient amount of interval time delay of configuring VMs from the very beginning (Lagar-Cavilla et al., 2011). It should be mentioned, that a VM configuration typically includes the time for specifying physical resources (CPU, RAM and storage space) and operating system (OS) installation along with the required software set up. Thus, in the case of a large number of job submissions, the time needed to analyze the jobs and produce the appropriate number of VMs sufficiently increases the interval time for service management. Instantiating new VMs is a slow operation that usually includes minutes e.g. in EC2 (Lagar-Cavilla et al., 2007).
A convenient method to propagate VMs in hosts is by deploying from a pool of pre-configured and already executed VMs. Specifically, by using forking processing method presented in Lagar-Cavilla et al., 2009, we explore the performance of the VMs average execution time. In forking, the generation happens by inheritance of the state of the parent thread and creating a distinct child thread within a multithreaded environment. After forking, the VMs are migrated from the pool and placed to available hosts for accepting job submissions.
In this work we explore static and reactive dynamic deployment of VMs instantiation. First, we discuss the motivation (section 2) and the related works (section 3) for identifying relevant features. The rest of the paper is organized as follows. We discuss the cloud parameters and model the algorithms of the VM instantiation (section 4). Then, we present the configuration specification (section 5.1) and we analyze performance results from various metrics as extracted from a simulated static environment for certain job variations to explore benchmarks (section 5.2). Next, we integrate the dynamic reactive instantiation using the forking method and simulate the VM migration within the same environment of the static case (section 5.3). Finally, we brief applicable scenarios for dynamic VM instantiation (section 6) and conclude in section 7.

2 Motivations

There are several evolving motivations that cloud computing encompasses in the area of VM deployment. These are mostly related with the association of virtualization paradigm to the large size of clouds and the decision process on deploying VMs. Frachtenber and Schwiegelshohn, 2007, present that the challenge in managing virtualized environments efficiently is directly related with scheduling. They describe that host VMs often operate with no coordination and knowledge of each other scheduling issues. The view is that scheduling will play an important role in virtualized settings where performance and utilization matter (Frachtenber and Schwiegelshohn, 2007). Eventually, novel works should aim to the direction of orchestrating the overall process of scheduling in twofold, firstly guests (VMs) within hosts and secondly scheduling inside the guests OS.

Although, there are still no strong literature directions in virtualization scheduling, the characterization of dynamic allocation of customized VM images is of increasingly importance due to the large variety of services and multi-hosted environments (Frachtenber and Schwiegelshohn, 2007).
One of the works that utilize the notion of the dynamically virtualized setting is the one of Diaz et al., 2011. The authors present the Future Grid, a testbed to perform experiments in HPC, grids and clouds. Within it, developers use VMs for not simply storing a guest image but rather focus on the way an image is created through a method known as templating". Diaz et al. 2011, have stated it that "it is possible at any time to regenerate an image based on the template that is used to install the software stack onto a bare operating system. In this way, the development of a customized image repository not only provides functional advantages, but it also provides structural advantages aimed to increase efficient use of the storage resources".
Thus, it is vital to consider new ways of deploying VMs for increasing the quantity of provided services, while at the same time hiding the low level functionality from the end-users. Foster and Kesselman, (2004), highlight that users should be able to invoke any desired operation, however, without regarding how the instances are implemented within the environment. They continue that on the lower level various implementation directions could be applied including stochastic or reactive virtualization of services. Younge et al., 2011, have emphasized the need for customized tools and service in a Cloud. Particularly, they imply that in order to meet user needs an easy to customize environment is required, and virtualization emerges this capability. In this 'settings service' implementation is a requested operation and sandboxed in services that may be virtual provided through an interface, which in turn may be virtualized as well (Foster and Kesselman, 2004). This implies that more levels of virtualization might be applied when the service integration process taking place.
With these in mind, our motivation and contribution is to explore and contrast whether virtual machines that are generated according to the cloud host equals or else the execution performance as if virtual machines were reactively adapted according to the job submissions.

3 Related Works

Herein the related works are discussed within the area of VM deployment. In general the term virtualization refers to the organization of virtual chunks of the physical hosts for splitting the computational power. Usually, the cloud administrator configures VMs in a custom way to meet user requirements. However, as the number of users increased, it becomes apparent that a more automatic way needed for VMs creation. The work of Lagar-Cavilla et al., 2007, and Lagar-Cavilla et al., 2011 discuss that the current solutions include substantial labor effort of the cloud administrator. In this way resources could be remain in idle state, as it is difficult for a developer to take highly level decisions when a large number of services request execution. For addressing this shortage, various solutions have been developed that mainly aim to the VM life cycle provisioning.
Initially, those include the migration of ready states of VMs among hosts for achieving automation of VMs scheduling. There are two generic classifications of migration procedures in this area, called process migration and live migration. Process migration is an old studied approach, which includes the procedure of transferring a process from one machine (host resource) to another (remote resource). Live migration, conversely, provides the capability to move VMs from one machine to another while processes still

running. Process and live migration share advantages and drawbacks, however a detailed appreciation of these methods is beyond the aim of this study.

Thus, initially, the "copy on reference" solution (Zayas, 1987) is an old way to process migration and was the basic for developing the memory-on-demand method (Lagar-Cavilla et al., 2007). Using this method, VMs are cloned on reference by duplicating their state. This raises questions regarding the state transfer mechanism and the level of transparency. The work of Vrable et al., 2005 illustrates a more advanced solution that uses the copy-on-write technique. In this way VMs are cloned from a static template that does not provide migration among VMs to different hosts. Other works (Sapuntzakis et al., 2002; Lagar-Cavilla et al., 2007) use a lazy copy-on-reference solution. Using secondary storage devices, allow the copy of processes among VMs by achieving a low-bandwidth consumption model. Hibler et al., 2008 perform an experiment to virtualize a large number of nodes for mimicking an experiment. Their results show that requires a significant amount of time to instantiate the whole network of nodes.

Zhang et al., 2000 present a migration mechanism that first vacates tasks from node to node and then re-instantiate those to the target set. The authors suggest that migration when combined with backfilling can be beneficial in terms of utilization. However, results show that the maximum utilization does not change from the whole system perspective. A different solution presented in Bustos et al., 2003 is based on percentage loading at node. The solution uses a proactive library for the migration of jobs, and a multicast channel to coordinate the nodes. The experimental analysis shows low resource utilization at a medium average throughput. Osman et al., 2003 present the Zap, a system to allow process migration of domains called pods. However, this model has limited success primarily because of the difficulty of encapsulating the state of a running application. Wood et al., 2007 present the sandpiper, a system that initiates migration using automation of monitoring tasks and by detecting hotspots it determines a new mapping of physical to virtual resources. However, this system does not support replication services automation. Zheng et al., 2011 discuss a storage migration-scheduling algorithm for improving storage and I/O performance during migration. The benefits include the minimization of extra traffic, and the efficient handling of large image sizes.

Different from all the aforementioned solutions, Lagar-Cavilla et al., 2009 addresses the problem of low latency replication of VMs. The authors suggest that by using the VM fork abstraction clones could be easily instantiated and transferred as replicas to the same or different hosts. Furthermore, they present a solution called SnowFlock that offers significant advantages in VM cloning. Firstly, it decreases the time that needs for a clone to be instantiated by only copying the state critical state along with the VM memory image, and secondly, by modifying the guest kernel minimizes the bandwidth transfer. Conclusively, the authors prove through their experimental analysis that their solution overcomes concerns from aforementioned works in terms of minimized interval time of VM cloning (less than 1000 millisecond).

In this work the focus is on exploring the VM instantiation from the perspective of different job submissions, so we do not aim to explore the migration low-level infrastructure. In our work, we apply the forking method as presented in Lagar-Cavilla et al., 2009 and design our model based on their assumption that VM migration is less than 1000 milliseconds. Based on this, we present our VM instantiation analysis by exploring and contrasting job execution performances when using static (virtual machines that are generated according to the cloud host) and dynamic (virtual machines are reactively adapted according to the job submissions) cases.

4 Designing the cloud exchanging model

A key feature in optimizing a cloud performance is the level of gratification that the setting could offer to the users' job submissions (Bessis et al., 2012b). In this section we take the user view and we explore the VM instantiation that is correlated to job characteristics. Let us first define the terms of static and reactive dynamic deployment for VMs instantiation.

- As static case we define the deployment of VMs in which there is a fixed number of VMs that are instantiated by the hosts. Specifically, static VMs are established from the cloud administrator and are not drawn up from the users queries and requests for service executions.
- As dynamic case we define the vigorous deployment of VMs in which instantiation is based on different criteria and may vary on time. This is to say that VMs are generated based either on the number of jobs enter the environment. In addition, VMs could be migrated as a way of further enhancing the cloud performance.

Based on that we present the cloud life cycle for a cloudlet submission. The rationality behind this decision is to explore a static cloud and extract performance results by exploring the setting for certain job variations and various VMs number. Thus, we first analyze the VM instantiation process by modeling the algorithms of the cloud component. Then, we integrate algorithms within a simulator for extracting results in static and dynamic cases. The dynamic VMs are reactively adapted to auto-migration utilizing the VM forking concept.

4.1 The VM instantiation conceptual analysis

Herein, we analyze the VM instantiation concept by discussing the typical cloudlet life cycle within a cloud datacenter. Specifically, we assume that there is one cloud with various datacenters $D = \{d_1, d_2, \ldots, d_n\}$ that each of these contains various hosts $H = \{h_1, h_2, \ldots, h_n\}$. Each host generates a number of VMs $V_h = \{ v_{h1}, v_{h2}, \ldots, v_{hn} \}$ and each VM accepts a set of cloudlets $C = \{c_1, c_2, \ldots, c_n\}$ for job execution. For all cloudlets $\forall c_n \in C$ there is a specific configuration setting that contains the cloudlet characteristics. These are the required number of processors $c_{PES,}$ the length c_L, the filesize c_{FS} (input), the output size c_{OS}. Each cloudlet is associated with a utilization model for experimentation, related either with a stochastic or a standard utilization threshold level. This study incorporates both solutions to explore multiple threshold performance. In particular, the static case implies that the utilization model attempts to execute all cloudlets by achieving utilization threshold 100%. In dynamic case we test the utilization threshold in 80% thus we assume that the rest 20% is utilized by the migration mechanism. These are extensively discussed in section 5.

Similarly to cloudlets, for all VMs $\forall v_{hn} \in V$ there is a specific configuration setting that contains the VM characteristics. These are the the name of the VM v_{NM} , theVM size v_S, the RAM v_{RAM}, the available million of instructions per second (MIPS) $v_{MIPS,}$ the bandwidth speed v_{BW}, the number of processing cores v_{PES}. In addition, each VM is

associated with a scheduling policy for cloudlets execution. These are extensively discussed in section 5.2.

As said previously each cloud contains h_h hosts and for all $\forall h_h \in H$ there is a configuration setting that contains the host characteristics. These are the host id h_{ID}, the host mips h_{MIPS}, the number of cores h_{PES}, the RAM size h_{RAM}, the storage h_{ST}, and the bandwidth speed h_{BW}. In addition each host is associated with a scheduling policy that controls the allocation of VMs to hosts. This is to control the provisioning of physical resources for VM instantiation (e.g. provisioning of PEs, RAM etc.). Different host provisioning policies are extensively discussed in section 5.3 and 5.4.

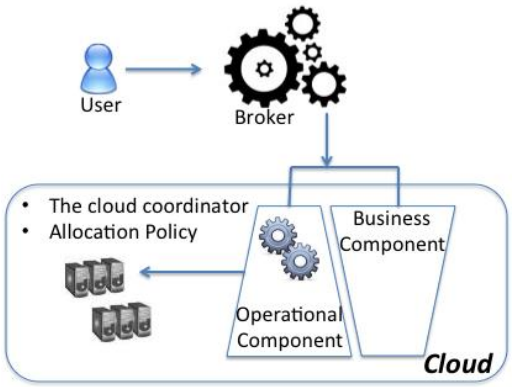
Thus, we define as cloud capacity for cloudlet execution the augmentation of physical and virtual resources as given by formulae (1). Specifically, the sum of cloud capacity is equal to the product of the sums of datacenter, hosts and VMs number. As $d(\alpha)$ is denoted the current datacetner, and as $h(\beta)$ the current dacetner host.

$$cloud_{capacity} = \sum_{\alpha=1}^{d_d} d(\alpha) \times \sum_{\beta=1}^{h_h} h(\beta) \times \sum_{\gamma=1}^{VM_v} VM(\gamma) \quad (1)$$

During the cloud life cycle the cloudlets submitted by the users through a broker to the VMs for execution. It should be mentioned that these jobs are identical to a real cloud application abstraction, thus we assume that the cloudlets could represent either a single job or the smallest manageable chunk of a large parallel job submission.

Figure 1 shows the cloud exchange model. Specifically, a user interacts with the broker for demanding service executions. The latter acts on behalf of the user and requests from the environment specific resources in the form of resource capacity (Rodero et al., 2010). In the cloud setting, the general management platform offers the operational and business functionalities for responding to the user request. Specifically, various processes take place within both components e.g. operational management involves security control, fault tolerance management, and eventually scheduling coordination. The operational management is also responsible for the core middleware functionalities including VM orchestration via the hypervisor. The hypervisor is a piece of software for controlling the VM deployment. The business functionalities, on the other hand, include the SLA communication process involving payments and debts, which are decided prior to the service submission and scheduling.

Figure 1: The cloud exchange model



During the exchange phase, cloudlets are submitted to the VMs through the broker for job executions. The users do not have any knowledge on that, as the process is organized by the broker who is responsible for binding cloudlets to VMs. Together various other components initialize communication within the datacenter. It should be mentioned that each a cloud registry monitors the whole procedure and the communication among users,

broker and datacenters. A finer detail of the most important functionalities of the cloud life cycle will allow us to develop the interactions among key cloud components and identify the VMs scheduling issues. These are based on the study of Calheiros et al., 2011 and listed bellow:

a) The cloud contains one or more datacenters, which are the resource providers.
b) The cloud is supplied with a cloud broker that is responsible for the cloudlet submissions to the hosts. It should be mentioned that additional brokering functionalities could be applied e.g. to submit cloudlets to VMs based on specific rules.
c) The cloud deploys VMs either by scratch or by VM migration etc. and informs the broker for the level of availability.
d) The cloud accepts the cloudlets from the broker and initializes the cloudlet submission phase.
e) The cloudlet execution happens by the allocation policies as defined within the hosts and VMs.

Hence, typically in a cloud environment job scheduling is a two-fold decision (Sotiriadis et al., 2012a) that includes the following:

a) The host level scheduling that incorporates the way in which VMs allocated within the host namely as the host allocation policy. This applies to the sharing of resource in a time and space manner.
b) The VM level scheduling that integrates the local job allocation policy (local resource management system). This includes the way in which jobs are executed in the local queue. In our setting the local queue is organized in a first-come-first-served manner.

4.2 Modeling the algorithmic structure

This section demonstrates the algorithms of the cloud run-time phases including components such as cloudlets, VMs and datacenters along with the job submission and execution phases. Using these algorithms, we integrate our solution to model a cloud setting that explores the optimization methods for VM deployment. The algorithm 1 demonstrates the cloudlet initialization phase.

Algorithm 1: The cloudlet initialization phase

Require: c_{LIST} []: The container of cloudlets (in the form of an array)
c_L: The length of the cloudlet
c_{FL}: The cloudlet filesize
c_{FL}: The cloudlet outputsize
c_{PEs}: The number of cores (PEs)
c_c []: The cloudlet (in the form of an array of subcloudlets)
$user_{ID}$: The user id
x: cloudlet number
Precondition: c_{UM}: The utilization model (defined within the cloudlet for experimental purposes)

1: **for** c_c = {i, i++, x} **do**
2: c_c [i] → (i, c_L, c_{PEs}, c_{FL}, c_{FL}, c_{UM})
3: c_c [i] → $user_{ID}$ (i)
4: c_c [i] → c_{LIST} [i]
5: **end for**

Algorithm 2 demonstrates the configuration setup of the VM deployment.

Algorithm 2: The VM inititalization phase

Require: vm_{LIST} []: The container of VMs (in the form of an array)
vm_S: The size of the VM
vm_{PEs}: The number of CPU cores (PEs)

vm_{RAM}: The RAM size
vm_{mips}: The available mips
vm_{BW}: The bandwidth speed
vm_{NAME}: The VM name
vm_v []: The VM (in the form of an array of subcloudlets)
$user_{ID}$: The user id
y: VM number
vm_{SP}: The VM allocation policy

Precondition:

1: **for** vm_v= {i, i++, y} **do**
2: vm_V [i] → (i, vm_S, vm_{PEs}, vm_{RAM}, vm_{mips}, vm_{BW}, vm_{NAME}, vm_{SP})
3: vm_c [i] → vm_{LIST} [i]
4: **end for**

Algorithm 3 illustrates the datacenter initialization phase.

Algorithm 3: The DC inititalization phase

Require: $host_{LIST}$ []: The container of hosts (in the form of an array)
$host_{ID}$: The host id
$host_{PEs}$: The number of CPU cores (PEs)
$host_{RAM}$: The RAM size
$host_{ST}$: The storage size
$host_{mips}$: The available mips
$host_{BW}$: The bandwidth speed
$host_H$: The current host
y: the number of hosts

Precondition: $host_{SP}$: The host allocation policy of VMs
$host_{PP}$: The host provisioning policy of physical resources

1: **for** $host_H$= {i, i++, y} **do**
2: $host_H$ [i] → (i, $host_S$, $host_{ID}$, $host_{PEs}$, $host_{RAM}$, $host_{ST}$, $host_{mips}$, $host_{BW}$, $host_{SP}$, $host_{PP}$)
3: $host_H$ [i] → $host_{LIST}$ [i]
4: **end for**

Algorithm 4 demonstrates the cloudlet submission phase that comprises the preconditions of initialization phase (algorithm 1) and the host allocation policy included at the host level. It demonstrates also the cloudlet execution phase based on the precondition of the cloudlet submission phase (algorithm 1). This includes the scheduling that happens at the local level within the VM.

Algorithm 4: The cloudlet submission and execution phase

Require: b_B: The datacenter broker
dc_{ID}: The datacenter id

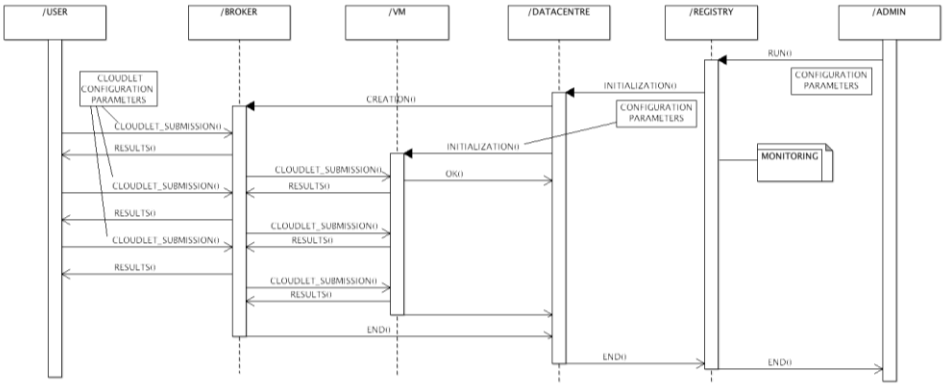
Precondition: HostAlloctionPolicy: The VM scheduling policy
VMAllocationPolicy: The cloudlet scheduling policy
$create_{VM}$: The VM creation function
$create_c$: The cloudlet creation function
get_c: The function to get cloudlets
$submit_c$: The submission function
run: A function to indicate the execution of policies
Algorithm 1, Algorithm 2, and Algorithm 3

1: b_B (dc_{ID})
2: vm_{LIST} []→ $create_{VM}$ (b_B, y)
3: c_{LIST}[]→$create_C$ (b_B, x)
4: get_c → b_B
3: **for all** $host_H$ [i] **do**
4: run (hostBW, hostSP, hostPP)
5: HostAllocationPolicy($host_i$)
6: **for all** vm_V [i] **do**
7: run(vm_{SP})
8: VMAllocationPolicy(VM_i)
7: **for all** c_c [i] **do**

6: run(c_{UM})
8: **end for**
9: **end for**
10: **end for**

Figure 2 demonstrates the sequence diagram of the aforementioned discussion. Specifically, it includes the initialization phase in which the cloud administrator configures the host of the datacenter, and the VMs that deployed within. Then the broker accepts the cloudlet submission from the user and redirects jobs to the VMs for execution.

Figure 2: The sequence diagram of a typical cloud life cycle incorporating the interactions between algorithms 1, 2, 3 and 4



Within the timeframe various allocation policies and utilization strategies are taking place. Specifically, the diagram contains two actors, the end-user who submits the cloudlets, and the administrator who is responsible for initializing components. This concludes the discussion of the VM instantiation model. The next sections contain the configuration of the simulation environment, and the discussion of the static and dynamic VMs instantiation cases.

5 Performance modeling of the VM instantiation

Herein we integrate the model of a typical cloud by using the algorithms of section 4.2 within a simulation setting.

5.1 The experiment configuration

The simulation environment is developed using the Cloudsim version 3.0 (Calheiros et al., 2011; Buyya et al., 2010), a framework for modeling and simulating clouds and their services. Specifically, Cloudsim allows control of a) large scale clouds, b) datacenters, brokers and scheduling policies in a self-contained fashion, c) adaptability of the virtualization technology for creating multiple virtualization services, and d) flexibility of the processing cores to switch between time and space shared allocation policies. Based on that we configure our experiment to contain the characteristics of table 1.

Table 1: The experiment configuration

Host		**VM**		**Cloudlet**	
PEs:	1	PEs:	1	PEs	1
RAM:	2048	RAM:	1024	Length:	1000

Storage:	1000000	Size:	1000	File size:	100
Mips:	1000	Mips:	150	Output size:	100
Bandwidth:	10000	VM name:	Xen	Utilization model:	Full Utilization
Allocation Policy:	Simple	Allocation Policy:	Space shared		

As selected metrics we utilize the total simulation time that represents the time that the cloud requires to execute a bunch of cloudlets as given by the formulae (2). The variable I represents an integer number greater than 0.

$$TotalSimulationTime = \sum_{set=i}^{n} FinishClock.cloudlet[i] \quad (2)$$

In addition we use the average execution time that a cloudlet set requires to be executed as given from formulae (3).

$$AverageExecutionTime(CloudletSet) = {\sum_{set=i}^{n} cloudlet.clock[i]}\Big/{Count[i]} \quad (3)$$

Next, we explore the VM instantiation performance which is analogous to a combination of the number of cloudlets, VMs and hosts. After, we integrate the dynamic case that contains the VM migration specification and the experimental analysis.

5.2 The static VM performance

We first explore the cloud simulator with a variation of the cloudlet input number. Here, it should be mentioned that the VM allocation policy includes the scheduling of cloudlets to VMs in a twofold mean as presented in (Calheiros et al., 2011). Firstly the space sharing policy in which cloudlets are placed in the queue when there are free PEs (number of cores) available. Secondly, the time-sharing policy indicates that at any given time multiple cloudlets could be allocated within the cores of a VM. The same policies could be applied in the case of the VM to hosts' allocation policy. This means that VMs can either queued in space or in time with respect to the cores installed in the host. In this experiment, we have utilized the time-sharing policy for both cloudlet and VM scheduling, thus multi-VMs instantiation within the host cores. It should be mentioned that queuing in hosts and VMs allocation happens in a first-come-first-serve scheduling (Cloudsim, 2012; Buyya et al., 2010).

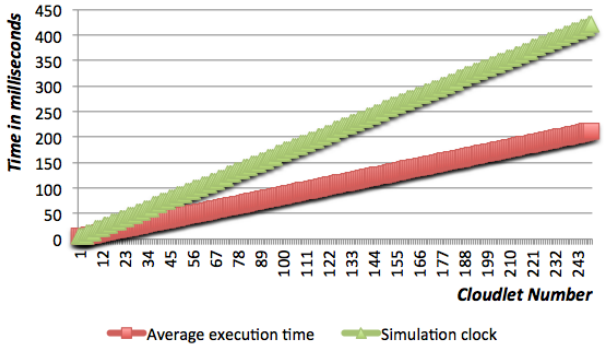
With respect to the utilization model, the static case selects VMs in a stochastic manner. That means that cloudlets are submitted to VMs randomly and aim to reach a 100% utilization level. Finally, the provisioning policies of physical resources (CPU, RAM, etc.) are executed in order to provide the best guaranteed service; this is to allocate a resource whenever it is available (Calheiros et al, 2011).

Specifically, the cloudlets number varied from 1 to 250, while the VMs number is fixed to 50. The whole setting runs within 50 hosts of one datacenter. Figure 3 shows the behavior of the simulator in terms of average execution time and simulation time.

The results show that the average execution and the simulation times are constantly increased in a linear way. In addition, the average execution time is increased with a lower rate than the simulation runtime. This means that in the case of a high peak workload the average execution time will stay in rational levels, however, the whole simulation time that represents the complete service time will be huge. This is because of the operations happened within the simulated environment e.g. due to communication latencies. Specifically, figure 3 illustrates that for a big cloudlet submission input the total simulation time (simulation clock) indicator (the total cloud service execution time) raise

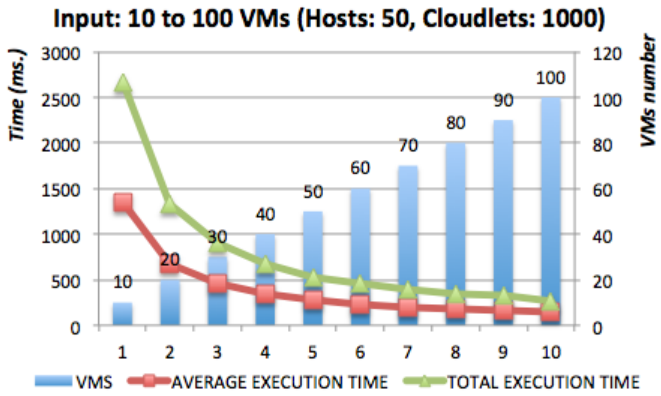
in a higher rate than the average execution time. This could cause significant problems in the case of a huge workload submission.

Figure 3: Static job executions for 1 to 250 cloudlets in non-reactive deployment 1 to 50 VMs instantiations in 50 hosts



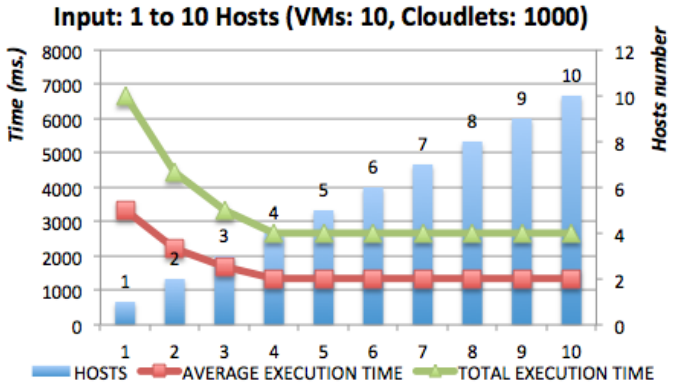
In a different experiment, we execute 10 to 100 VMs with a fixed number of 1000 cloudlets within an environment of 50 hosts in one datacenter. Figure 4 demonstrates the performance of the simulator when the specification is set to the values of table 1. It is apparent that as the number of VMs increases with constant values of hosts and cloudlets the average execution and simulation times decrease significantly. Especially, for VMs number greater than 50 the values increase with a lower rate. In the case of 100 VMs the total execution time decreases dramatically and almost becomes identical to the average execution time. This means that for the specific configuration, and with the VMs number greater than 100 the system reaches a steady state.

Figure 4: Static deployment of VM instantiation performance for input submission of 1000 cloudlets within a non-reactive setting of 10 to 100 VMs with a fixed setting of 50 hosts



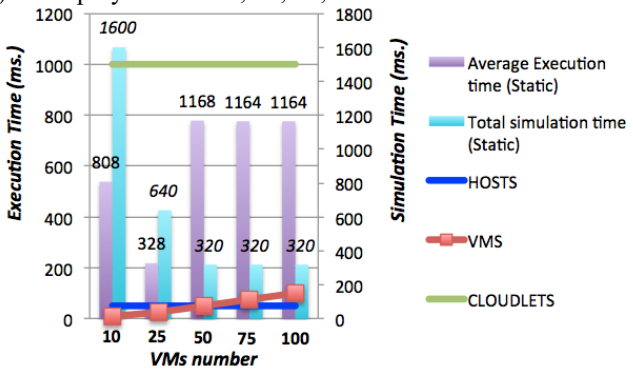
In the next experiment we monitor the performance of 1 VM when executed in 1 to 20 hosts. Figure 5 shows the performance of the simulation for the specification of Table 1.

Figure 5: Static hosts performance for input submission of 1000 cloudlets within a non-reactive setting of 1 to 10 hosts with fixed deployment of 10 VMs within each host



In figure 5, as the hosts number varied the job input of 1000 cloudlets that run in 10 VMs shows a decrease trend in the average execution and simulation times. In the case of hosts number greater than 5 the rate comes in a steady state for both metrics. To conclude, the static setting presents the experimental analysis of the exploration of the testbed for identifying the performance of the hosts, VMs and cloudlets for certain job variations. Figure 6 combines all static case results in a way that can be contrasted with the results from the dynamic case to be discussed in section 5.3. The specification includes the allocation of 10, 25, 50, 75 and 100 VMs within a fixed environment of 50 hosts. The job input includes the allocation of 1000 cloudlets.

Figure 6: The static and non-reactive benchmark results (average simulation and total execution time) for deployment of 10, 25, 50, 75 and 100 VMs in fixed setting of 50 hosts



We present the experimental configuration and results from static and dynamic cases in table 2 implemented in Cloudsim. The next section presents figures that illustrate those values. Execution times are in milliseconds.

Table 2: The simulation results for static and dynamic cases

Hosts	50	50	50	50	50
VMs	10	25	50	75	100
Cloudlets	1000	1000	1000	1000	1000
Average Execution time (Static)	808	328	1168	1164	1164
Total simulation time (Static)	1600	640	320	320	320
Average Execution time (Dynamic)	437	364	355	351	348
Total simulation time (Dynamic)	824.6	1039.6	1039.6	1039.6	1039.6

5.3 Dynamic deployment of reactive VMs instantiation performance

This section presents the dynamic reactive workload deployment case that generates VMs based on VMs migrations from a pool of available VMs. For modeling this functionality we assume that the cloud administrator has previously configured a set of VMs. Utilization is based on migration of VMs within a physical space of the same host (Calheiros et al, 2011). Specifically, migration utilization policy selects a host with the least computational power due to utilization increase caused by the VM allocation (B. Thus, in the experiment we set the utilization threshold to 80%, so the system tries to keep the host utilization (CPU) under the specific utilization threshold. The rest 20% of the utilization consumed by the migration operations. In this case the migration includes a duplication of the actual VM by using the forking method of Lagar-Cavilla et al., 2009 as described in section 2. Specifically, each time a cloudlet submitted to the broker for execution, the datacenter offers an additional functionality that allows VMs to be migrated rather than created from the beginning. For experimental purposes we did not implement the VM forking solution for VM duplication, however, we have defined the delay of migration by formulae (4).

$$MigrationDelay_{VMi} = \frac{VM_{RAMi}}{VM_{BWi}} + (f_{VMi} \times const_{VMi}) \quad (4)$$

We extend the formulae of Cloudsim, (2012) and we measure the delay of the division of the VM_{RAM} by the bandwidth speed VM_{BW} in addition to the result of a coefficient value that represents the extra delay. This corresponds to forking latency time f_{VMi} as illustrated in Lagar-Cavilla et al., 2009 multiplied by a constant variable $const_{VMi}$ to control the rate of latency. For example in this experiment we set the f_i *in 1000 (ms.) and the* $const_i$ in 10, that means that the delay is actually 10 times greater (10^4). This happens because we want to perform the experiment with a worst-case scenario. By performing migration of tasks in a simulated forking environment, we allow VM instantiation in a dynamic case. This is to say that when there is no availability in terms of computational power, new VMs are generated from a virtual resource pool to handle the demanded workload.
Figure 7 presents the performance of the testbed by measuring the average execution and simulation times when dynamic instantiation occurs. The specification includes the execution of 1000 cloudlets when the VM numbers are varied from 10 to 100.

Figure 7: Average execution and simulation times of 1000 cloudlets with dynamic migration of reactive VMs instantiation (comparison for deployment of VMs 10 to 100 in 50 hosts)

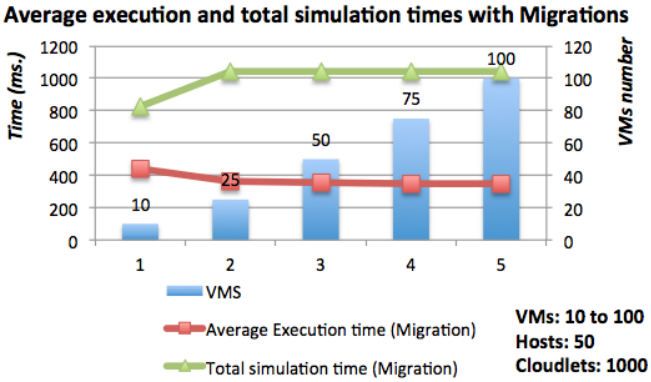
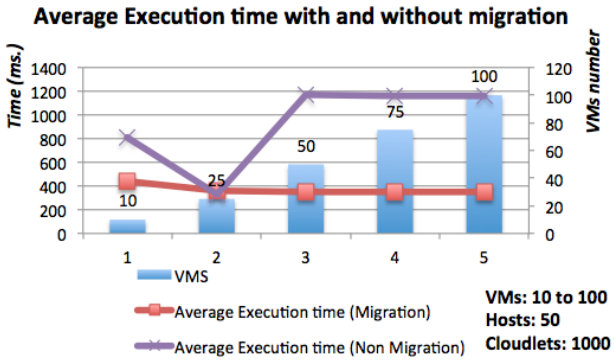


Figure 7 results show the average execution decreases while the simulation time increases. In addition, when the VMs number is greater than 25 the system goes to a stable state with cloudlets execution to be under 400ms. However, the simulation time (the total system run) is increased significantly. For VMs number greater than 25 the testbed offers a stable state in which execution of 1000 cloudlets happens in less than 1000ms.

Figure 8 demonstrates the results of the average execution time by comparing the static benchmarks (as presented in figure 6) and the dynamic instantiation for the same VMs variation (10 to 100). The rest of the configuration parameters remain the same.

Figure 8: Average execution time of 1000 cloudlets with the dynamic migration of reactive VMs instantiation in comparison with the static case of figure 6 (comparison for deployment of 10 to 100 VMs in 50 hosts)



Specifically, figure 8, compares the average execution time with and without migration. It is clear that the dynamic case with migration outperforms the static solution. Specifically, for VMs number greater than 50 the static solution gets stable (under 100ms.) while the dynamic case for VMs number greater than 25 it offers a stable state with execution time under 40 ms. A unique situation is the case of 25 VMs, in which the solution offers the same results for both cases.

Figure 9 shows the results of the total simulation time by comparing the static and the dynamic instantiation for the same VMs variation (10 to 100) with the same configuration.

Figure 9: Total simulation time of 1000 cloudlets with the dynamic migration of reactive VMs instantiation in comparison with the static and non-reactive case of figure 6 (comparison for deployment of 10 to 100 VMs in 50 hosts)

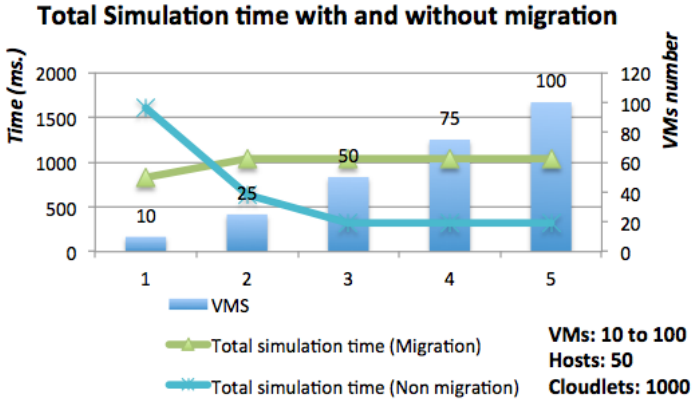
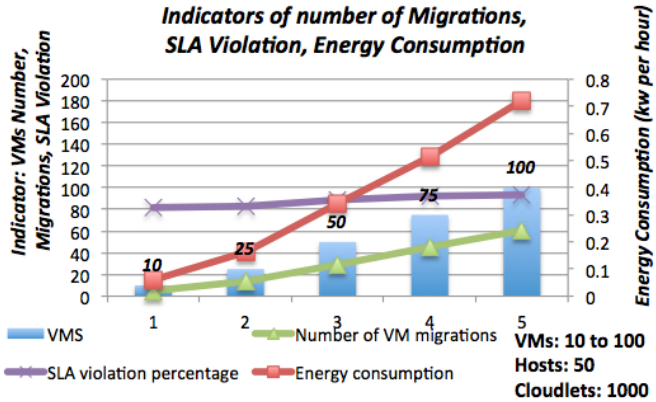


Figure 9, demonstrates the testbed performance with regards to the total simulation time. In this case the non-migration solution outperforms the reactive because of the latencies happens due to VM migrations. It should be mentioned that the initial appreciation that delay is set to ten times higher than the non-migration is the reason for the high delay numbers. However, when the number of VMs is greater than 25 the solution is getting in steady execution level under 1000ms.

Figure 10 demonstrates the indicators of the number of VM migrations, the SLA violations and the energy consumption according to the model of Beloglazov et al., 2010 in the reactive dynamic migration case when the VMs are varied from 10 to 100. For experimental purposes the determination of SLA violations is measured by the difference between total requested and allocated mips divided by the total requested mips (Calheiros et al. 2011).

Figure 10: Number of migrations, SLA violation and energy consumption for reactive VM instantiation of propagating 10 to 100 VMs in 50 hosts



Figure's 10 indicator show that for higher number of VMs the number of VM migrations is increased. This happens because the experiment aims always to achieve a better distribution of cloudlets among VMs. In addition, the number of SLAs violations, which is related with the requested and allocation mips, is slightly increase. At last, the energy consumption (measured by the model of (Beloglazov et al., 2011), increases significantly due to the extra computational power needed by the datacenter for migrating VMs.

To conclude, by comparing static and dynamic reactive cases we could argue that a dynamic VM deployment causes a higher number of energy consumption because of the higher total simulation time, however, the average execution time of cloudlets has been well optimized. Specifically, the average value of the average execution times in dynamic case is 371ms. while in static case is 926ms. In contrary, the average simulation time in static case is 640ms. while in the dynamic case is 997ms. Thus, a future challenge is to identify ways of minimizing the simulation time in order to reach the average levels of static instantiation 640ms. Nevertheless, if the perspective is from the view of the user, by reactively utilizing dynamic migration of VMs instantiation, a better quality of service level could be achieved because of the optimization of the average execution time of the cloudlets, which represents the running time of the jobs.

6 Implications of deploying dynamic VMs instantiations

The following five scenarios reflect the benefits of the dynamic deployment of VMs instantiation.

1. Service consolidation and isolation: Over the years it will be common to organize services from multiple providers to be collaborative. Similarly, IT infrastructure can be seen as a large established service. A flexible solution includes the deployment of VMs in hosts by sandboxing the required workload into small virtual chunks. However, this static view arises questions regarding the agility in serving the different levels of workload requests. A more advanced solution is the dynamic deployment of VMs instantiations. As presented in sections 5.2 and 5.3 the performance metrics of times present an improved performance in executed those requests. In other words, the benefit by consolidating workloads will be increased by migrating VMs within the environment. Similar to consolidation, isolation defines that application services could be separated from each other. With reactive VMs management undesired interactions and conflicts could be eliminated.
2. Security and consistency of applications: This implies the creation of VMs for each of the user application. Thus, VMs are generated according to the requirements of the user in a reactive manner, rather than in a static deployment case. Again, the dynamic case will be able to control the security and reliability based on the SLAs signed among providers and consumers. As presented in section 5.3 the SLA violations in the simulation environment is increased with a slower rate for large number of VMs and cloudlets.
3. Testing of applications: In the case of testing, virtualization allows the concurrent use of products when implemented in different virtual machines. This is a very handy solution for customers that require developing their own applications and for administrators as well. In advance, the reactive deployment of VMs instantiations will make testing more efficient as migration will allow the faster execution of heavy testing situations to remote idle hosts.
4. Disaster management: As datacenter conditions change due to unforeseen situations, migration is the key to move workloads in real time. With this, users do not expect to suffer from any significant interruptions to the service availability. Critical case in disaster management includes the orchestration of the environment to act in a predictable manner by prioritizing tasks when a disaster incidence occurs. This includes the movement of active VMs to idle or spare datacenter hosts within a convenient time.

5. Energy consumption: The effects of migrating VMs in a cloud environment offer significant advantages like resource distribution and energy consolidation. However, due to migration the consumption of energy is increased; thus a challenge is to find scenarios in order to optimize the power consumption. Huang et al., 2011 discuss such a scenario and present that in the case of a power consolidation setting, the power overhead of migration is much less than without consolidation.

These scenarios are different in terms of the perspective of the decision maker for VMs deployment. However, the common overall aim to enhance the agility and flexibility of the cloud by dynamically instantiating VMs. For example, in scenario 1 workloads are sandboxed in VMs on demand from the perspective of the computational power, while scenario 2 involves application services from the user perspective. Eventually, we consider that dynamic and reactive deployment of VMs will play an important role in the deployment of VMs instantiations, especially when large-scale clouds occur.

7 Conclusions

In this work we present the VM instantiation analysis in a cloud environment. To this extend, we have compared the static and dynamic instantiations (with VM migration) by using Clousim as the simulation testbed. The analysis presented herein shows that in dynamically reactive model of VMs instantiation the average execution times using VM migration outperforms the result found in the static case. However, this compromises the overall system time (simulation time) and the energy consumption levels. This is because we initially set the delay of VM migration to the highest delay levels. Nevertheless, as the primary view of the paper was the quality of service levels from the users' perspective, the overall service execution time (represented by the average execution time) levels have been optimized for the specific configuration.

Opportunities for further research include extending the functionality of the allocation policies and the utilization model in order to achieve decrease simulation times as well as decrease energy consumption levels. A future direction is to incorporate cloud datacenters and allow tasks to be migrated between different hosts belonging to various datacenters. Challenges to this direction include the implementation of the InterCloud environment (Buyya et al., 2010), a setting for exchanging cloudlets and VMs among interoperable clouds for improving the quality of service levels.

8 References

A. Lagar-Cavilla, H. Tolia, N., De Lara, E., Satyanarayanan, M., and O'Hallaron, D. (2007*). Interactive Resource- Intensive Applications Made Easy*. In Proceedings of the *8th International Middleware Conference, Newport Beach, CA*.

A. Lagar-Cavilla, H., A. Whitney, J., Bryant, R., Patchin, P., Brudno, M., De Lara, E., M. Rumble, S., Satyanarayanan, M., and Scannell, A. (2011). *SnowFlock: Virtual Machine Cloning as a First-Class Cloud Primitive*. *ACM Trans. Comput. Syst.* 29, 1, Article 2 (February 2011), 45 pages.

A. Lagar-Cavilla, H., A. Whitney, J., M. Scannell, A., Patchin, P., M. Rumble, S., De Lara, E., Brudno, M., and Satyanarayanan, M. (2009). *SnowFlock: rapid virtual machine cloning for cloud computing*. In Proceedings of the *4th ACM European conference on Computer systems (EuroSys '09)*. ACM, New York, NY, USA, pp. 1-12.

Beloglazov, A., and Buyya. B. (2010). *Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers*. In Proceedings of the *8th International Workshop on Middleware for Grids, Clouds and e-Science (MGC '10)*, ACM, New York, NY, USA, Article 4, 6 pages

Bessis, N., Sotiriadis, S., Cristea V. and Pop, F. (2012). *An architectural strategy for meta-scheduling in InterCloud*. Proceedings of *first International Workshop on InterCloud and Collective Intelligence (iCCI-2012), in conjunction with the 26th IEEE International Conference on Advanced Information Networking and Applications (AINA 2012)*. Fukuoka, Japan, March 26-29 2012 {To appear}

Bessis, N., Sotiriadis, S., Cristea V. and Pop, F. (2011). *Modelling Requirements for Enabling Meta-Scheduling in InterCloud and Inter-Enterprises*, Proceedings of the *3rd IEEE International Conference on Intelligent Networking and Collaborative Systems (INCoS 2011)*, Fukuoka, Japan, pp. 149-156

Bessis, N., Sotiriadis, S., Xhafa, F., Pop, F. and Cristea, V. (2012). *Meta-scheduling Issues in Interoperable HPCs, Grids and Clouds*, *International Journal of Web and Grid Services*, (8):2.

Bustos, J. (2003). *Robin hood: An active objects load balancing mechanism for intranet*. In Proceedings of the *Workshop de Sistemas Distribuidos y Paralelismo*, Chile

Buyya, R., Ranjan, R. and Calheiros, R. N. (2010). *InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. Algorithms and Architectures for Parallel Processing* (2010), Vol. 6081/2010, Issue: LNCS 6081, Springer, pp. 13-31

Calheiros, R., N., Ranjan, R., Beloglazov, A., De Rose, A. F., C. and Buyya, R. (2011). *CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*, Softw. Pract. Exper, Vol.41, No.1, pp. 23-50

Carolan, J. and Gaede, S. (2009). *Introduction to cloud computing architecture, White paper 1st edition*, Sun Microsystem, inc.

Gong, C., Liu, J., Zhang, Q., Chen, H. and Gong, Z. (2010). *The Characteristics of Cloud Computing*, *Proceedings of the 39th International Conference on Parallel Processing Workshops (ICPPW)*, pp. 275-279

Hibler, M., Ricci, R., Stoller, L., Duerig, J., Guruprasad S., Stack, T., Webb, K., and Lepreau, J. (2008). *Large-scale Vir- tualization in the Emulab Network Testbed.* In Proceedings of the *USENIX Annual Technical Conference, Boston, MA*.

Osman, S., Subhraveti, D., Su, G., and Nieh, J. (2002). *The design and implementation of Zap: a system for migrating computing environments*. *SIGOPS Oper. Syst. Rev. 36*, SI (December 2002), pp. 361-376.

Rodero, I., Guim, F, Corbalan, J, Fong, L. and Sadjadi, S.M. (2010). *Grid broker selection strategies using aggregated resource information*,*Future Generation Computer Systems*, pp.72-86

Sapuntzakis, C. P., Chandra, R., Pfaff, B., Chow, J., Lam, M. S., and Rosenblum, M. (2002). *Optimizing the Mi- gration of Virtual Computers*. In Proceedings of the *5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA

Sotiriadis, A., Bessis N. and Antonopoulos, N. (2012). *From meta-computing to interoperable infrastructures: A literature review of schedulers and simulators*, *Conference on Advanced Information Networking and Applications (AINA 2012)*, Fukuoka, Japan, March 26-29 2012 {to appear}

Sotiriadis, S., Bessis N. and Antonopoulos, N. (2012). *Towards InterCloud schedulers: A survey of meta-scheduling approaches*, Proceedings of the *6th International Conference on P2P, Parallel, Grid, cloud and Internet Computing (3PGCIC 2011)*, Barcelona, Spain, pp. 59-66

Cloudsim: A Framework For Modeling And Simulation Of Cloud Computing Infrastructures And Services, Available at: http://www.cloudbus.org/cloudsim/, Accessed

Vrable, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A., Voelker, G., and Savage, A. (2005). *Scalability, Fidelity and Containment in the Potemkin Virtual Honeyfarm*. In Proceedings of the *20th Symposium on Operating Systems Principles (SOSP)*, Brighton, UK.

Wood, T., P. Shenoy, A. Venkataramani and M. Yousif. (2007). *Black-box and Gray-box Strategies for Virtual Machine Migration*. In Proceedings of the *Fourth Symposium on Networked System Design and Implementation (NSDI '07)*

Xhafa, F. and Abraham, A. (2010). *Computational models and heuristic methods for Grid scheduling problems*, *Future Generation Computer Systems*, Vol. 26, Issue 4, April 2010, pp. 608-621, ISSN 0167-739X

Zayas, E. (1987). *Attacking the Process Migration Bottle- neck*. In Proceedings of the *11th Symposium on Operating System Principles (SOSP)*, Austin, TX.

Zhang, Y., Franke, H., E. Moreira, J., and Sivasubramaniam, A. (2000). *The Impact of Migration on Parallel Job Scheduling for Distributed Systems*. In Proceedings of the *6th International Euro-Par Conference on Parallel Processing (Euro-Par '00), Arndt Bode, Thomas Ludwig, II, Wolfgang Karl, and Roland Wismuller (Eds.)*. Springer-Verlag, London, UK, pp. 242-251.

Zheng, J., S Eugene Ng, T., and Sripanidkulchai, K. (2011). *Workload-aware live storage migration for clouds*. In Proceedings of the 7th *ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE '11).* ACM, New York, NY, USA, pp. 133-144.

Frachtenberg, E., and Schwiegelshohn, U. (2007). New challenges of parallel job scheduling. In Proceedings of the 13th international conference on Job scheduling strategies for parallel processing (JSSPP'07), Eitan Frachtenberg and Uwe Schwiegelshohn (Eds.). Springer-Verlag, Berlin, Heidelberg, 1-23.

Diaz, J., V. Laszewski, G., Wang, F., J. Younge, A., and Fox, G. (2011). FutureGrid Image Repository: A Generic Catalog and Storage System for Heterogeneous Virtual Machine Images. In Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CLOUDCOM '11). IEEE Computer Society, Washington, DC, USA, 560-564.

Ian Foster and Carl Kesselman. 2004. The Grid in a nutshell. In Grid resource management, Jarek Nabrzyski, Jennifer M. Schopf, and Jan Weglarz (Eds.). Kluwer Academic Publishers, Norwell, MA, USA 3-13.

Andrew J. Younge, Robert Henschel, James T. Brown, Gregor von Laszewski, Judy Qiu, and Geoffrey C. Fox. 2011. Analysis of Virtualization Technologies for High Performance Computing Environments. In Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing (CLOUD '11). IEEE Computer Society, Washington, DC, USA, 9-16.

Huang, Q., Gao, F., Wang, R., and Qi., Z. (2011). Power Consumption of Virtual Machine Live Migration in Clouds. In Proceedings of the 2011 Third International Conference on Communications and Mobile Computing (CMC '11). IEEE Computer Society, Washington, DC, USA, 122-125.