INFORMATICA, 2013, Vol. 24, No. 3, 357–380
© 2013 *Vilnius University*

Cloud Scheduling Optimization: A Reactive Model to Enable Dynamic Deployment of Virtual Machines Instantiations

Nik BESSIS[1], Stelios SOTIRIADIS[2]*, Fatos XHAFA[3]
Eleana ASIMAKOPOULOU[1]

[1]*Department of Business Computing and Law*
University of Derby, Kedleston Road, Derby, DE22 1GB, UK
[2]*Intelligent Systems Laboratory (InteLLigence)*
Department of Electronic & Computer Engineering,
Technical University of Crete (TUC), Chania, Crete, GR-73100, Greece
[3]*Departament de Llenguatges i Sistemes Informàtics*
Universitat Politècnica de Catalunya, Campus Nord, Ed. Omega,
C/Jordi Girona 1-3, 08034 Barcelona, Spain
e-mail: n.bessis@derby.ac.uk, s.sotiriadis@intelligence.tuc.gr, fatos@lsi.upc.edu, eleana.asimakopoulou@googlemail.com

Received: March 2012; accepted: February 2013

Abstract. This study proposes a model for supporting the decision making process of the cloud policy 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 orchestration, 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 is prosperus as it offers optimized job execution performance.

Key words: cloud computing, virtual machine 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 have been encapsulated in application execution requests posed by the end-users, and this is identical to the overall scheduling problem in Grid

*Corresponding author.

systems (Xhafa and Abraham, 2010). Hence, 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 because of 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 preconfigured and already executed VMs. Specifically, by using forking processing method (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. Motivation

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. In general, the challenge in managing virtualized environments efficiently is directly related with scheduling (Frachtenberg *et al.*, 2008). 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 (Frachtenberg *et al.*, 2008). 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 (Frachtenberg *et al.*, 2008). The notion of the dynamically virtualized setting is also presented in literature (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. It has been stated 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 (Diaz *et al.*, 2011).

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. This has been highlighted in past (Foster and Kesselman, 2004) 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. Similarly, it has been emphasized the need for customized tools and service in a Cloud (Younge *et al.*, 2011). 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

This section presents 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. Various works (Lagar-Cavilla *et al.*, 2007) 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. Other work (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. Similarly, a different study (Sapuntzakis *et al.*, 2002) uses a lazy copy-on-reference solution. Using secondary storage devices, allow the copy of processes among VMs by achieving a low-bandwidth consumption model. At last, in other work authors perform an experiment to virtualize a large number of nodes for mimicking an experiment (Gong *et al.*, 2010). Their results show that requires a significant amount of time to instantiate the whole network of nodes.

A migration mechanism is presented in literature (Zhang *et al.*, 2000) 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 (Bustos, 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. To continue, the Zap (Osman *et al.*, 2002) is 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. Next, the sandpiper (Wood *et al.*, 2012) is 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. Finally, a storage migration-scheduling algorithm has been implemented (Zhang *et al.*, 2011) 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, our work address the problem of low latency replication of VMs (Lagar-Cavilla *et al.*, 2011). The authors (Lagar-Cavilla *et al.*, 2009) 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 ms).

In this work our 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 extend the application of the forking method (Lagar-Cavilla *et al.*, 2009) in a cloud environment with the assumption that VM migration is less than 1000 ms. Based on this, we present our VM instantiation analysis by exploring and contrasting job execution performances when using static and dynamic VM 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*

Here, 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 $D = [d_1, d_2, \dots, d_n]$ that each of these contains various hosts $H = [h_1, h_2, \dots, h_n]$. Each host generates a number of VMs $VM = [VM_1, VM_2, \dots, VM_n]$ and each VM accepts a set of cloudlets $C = [c_1, c_2, \dots, c_n]$ 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 $\forall VM_{hn} \in V$ there is a specific configuration setting that contains the VM characteristics. These are the the name of the VM_{VNM}, the VM size VM_S, the RAM VM_{RAM}, the available million of instructions per second (MIPS) VM_{MIPS}, the bandwidth speed VM_{BW}, the number of processing cores VM_{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 $\forall H_h \in H$ there is a configuration setting that contains the host characteristics. These are the host id_{HID}, 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 Sections 5.3 and 5.4.

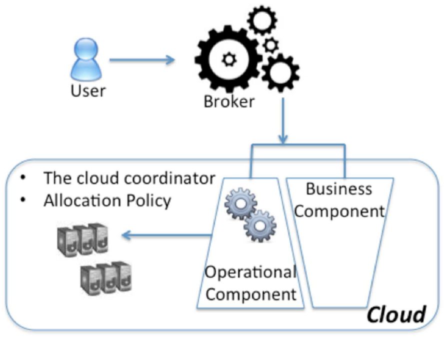

Fig. 1. The cloud exchange model.

Thus, we define as cloud capacity for cloudlet execution the augmentation of physical and virtual resources as given by formula (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 datacenter, as $h(\beta)$ the datacenter host and as $VM(\gamma)$ the host VM.

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

During the cloud life cycle the cloudlets submitted by the users through a broker to the VMs for execution. In particular, the job distribution is based on the message exchanging optimization (MEO) model (Bessis *et al.*, 2012c). This allows a sophisticated dipersal of messages in order to minimize the latency of the setting. 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). The brokering model (Sotiriadis *et al.*, 2012c) where we have detailed the topologies and interactions of brokers and meta-brokers. 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 Service Level Agreement (SLA) communication process involving payments and debts, which are decided prior to the service submission and scheduling.

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:

- The cloud contains one or more datacenters, which are the resource providers.
- 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.
- The cloud deploys VMs either by scratch or by VM migration etc. and informs the broker for the level of availability.
- The cloud accepts the cloudlets from the broker and initializes the cloudlet submission phase.
- 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:

- 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.
- 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 (first-come-first-served local scheduler).

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. Specifically, this aims to explore static and reactive dynamic deployment of VMs instantiations. For that reason we utilize a forking processing method (Lagar-Cavilla *et al.*, 2009), in order to discover the performance of the VMs average execution time. It should be mentioned that in the forking case, the generation happens by inheritance of the state of the parent thread that creates 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. The algorithms presented in this section demonstrate that procedure.

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. Algorithm 1 demonstrates the cloudlet initialization phase that includes the cloudlet lenght,

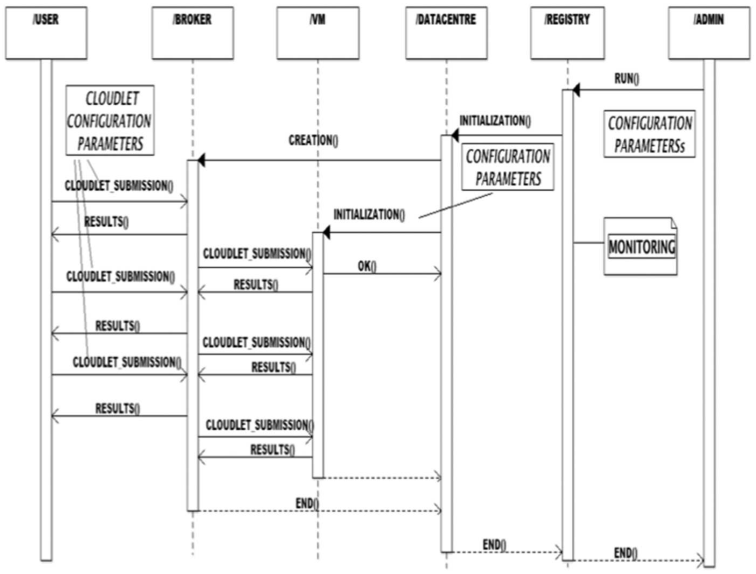

Fig. 2. The sequence diagram of the cloud life-cycle.

```
Algorithm 1: The cloudlet initialization phase
input  : c_LIST: The container of cloudlets (in the form of an array)
         c_L: The length of the cloudlet
         c_FS: The cloudlet filesize
         c_OF: The cloudlet output size
         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
         c_UM: The utilization model (defined for experimental purposes)
output: The cloudlet formation

for $c_c = [i, i{+}{+}, x]$ do
    $\mathrm{c}_c\ [i_1] \leftarrow (i, \mathrm{c}_L, \mathrm{c}_{PEs}, \mathrm{c}_{FS}, \mathrm{c}_{OF}, \mathrm{c}_{UM})$
    $\mathrm{c}_c\ [i_2] \leftarrow \mathrm{user}_{ID}\ (i)$
    $\mathrm{c}_c\ [i_3] \leftarrow \mathrm{c}_{LIST}\ [i]$
end
```

number of coes (processing elements – PEs), the number of cloudlets and the utilization model. Specifically, the last one implies that the hosts wil be utilized if there is computational power available.

Algorithm 2 demonstrates the configuration setup of the VM deployment.

```
Algorithm 2: The VM initialization phase
input  : $\mathrm{VM}_{LIST}$: The container of VMs (in the form of an array)
         $\mathrm{VM}_{S}$: The size of the VM
         $\mathrm{VM}_{PES}$: The number of CPU cores (PEs)
         $\mathrm{VM}_{RAM}$: The RAM size
         $\mathrm{VM}_{MIPS}$: The available MIPS
         $\mathrm{VM}_{BW}$: The bandwidth speed
         $\mathrm{VM}_{NAME}$: The VM name
         $\mathrm{VM}_{y}$:The VM (in the form of an array of sub-VMs)
         $\mathrm{user}_{ID}$: The user ID
         $y$: Number of VMs
         $\mathrm{VM}_{SP}$: The VM allocation policy (VM scheduling)
output: The VM formation
for $VM_H = [i, i{+}{+}, y]$ do
    $\mathrm{VM}_{VM}\ [i_1] \leftarrow (i, \mathrm{VM}_S, \mathrm{VM}_{PES}, \mathrm{VM}_{RAM}, \mathrm{VM}_{MIPS}, \mathrm{VM}_{BW}, \mathrm{VM}_{NAME}, \mathrm{VM}_{SP})$
    $\mathrm{VM}_{VM}\ [i_2] \leftarrow \mathrm{VM}_{LIST}\ [i]$
end
```

```
Algorithm 3: The datacenter initialization phase
input  : $\mathrm{host}_{LIST}$: The container of hosts (in the form of an array)
         $\mathrm{host}_{ID}$: The host ID
         $\mathrm{host}_{PEs}$: The number of CPU cores (PEs)
         $\mathrm{host}_{RAM}$: The RAM size
         $\mathrm{host}_{ST}$: The storage size
         $\mathrm{host}_{MIPS}$: The available MIPS
         $\mathrm{host}_{BW}$: The bandwidth speed
         $\mathrm{host}_{H}$: The current host
         $z$: The number of hosts
         $\mathrm{host}_{SP}$: The VM allocation policy (VM scheduling)
         $\mathrm{host}_{PP}$: The host provisioining policy (host scheduling)
output: The datacenter hosts initizalization
for $host_H = [i, i{+}{+}, z]$ do
    $\mathrm{host}_H\ [i_1] \leftarrow (i, \mathrm{host}_S, \mathrm{host}_{\mathrm{ID}}, \mathrm{host}_{PEs}, \mathrm{host}_{RAM}, \mathrm{host}_{ST}, \mathrm{host}_{MIPS}, \mathrm{host}_{BW},$
    $\mathrm{host}_{SP}, \mathrm{host}_{PP})$
    $\mathrm{host}_H\ [i_2] \leftarrow \mathrm{host}_{LIST}\ [i]$
end
```

Algorithm 3 illustrates the datacenter initialization phase which includes the host formation. It contains information regarding the host PEs, RAM, MIPS, bandwidth etc. as well as the number of VMs to be instantiated along with the VM allocation policy.

Algorithm 4: The cloudlet submission and execution phase

```
input : $\mathrm{dc}_B$: The datacenter broker
        $\mathrm{dc}_{ID}$: The datacenter ID
        $\mathrm{VM}_{LIST}$: The VM instantiated list
        $\mathrm{c}_{LIST}$: The cloudlet instantiated list
        $x$: Number of cloudlets
        $y$: Number of VMs
        HostAlloctionPolicy: The VM scheduling policy
        VMAllocationPolicy: The cloudlet scheduling policy
        $\mathrm{create}_{VM}$: The VM creation function
        $\mathrm{create}_C$: The cloudlet instantiation function
        $\mathrm{get}_C$: The function to get cloudlets
        $\mathrm{sumbit}_C$: The submission function
        run: A function to indicate the execution of policies
output: The cloudlet submission and execution

$\mathrm{dc}_B(\mathrm{dc}_{ID})$
$\mathrm{VM}_{LIST}[] \leftarrow \mathrm{create}_{VM}(\mathrm{dc}_B, y)$
$\mathrm{c}_{LIST}[] \leftarrow \mathrm{create}_c(\mathrm{dc}_B, x)$
for all $host_H$ $[i]$ do
    run ($\mathrm{host}_{BW}$, $\mathrm{host}_{SP}$, $\mathrm{host}_{PP}$)
    HostAllocationPolicy($\mathrm{host}_i$)
    for all $VM_{VM}$ $[i]$ do
        run ($\mathrm{VM}_{SP}$)
        VMAllocationPolicy($\mathrm{VM}_i$)
        for all $c_C$ $[i]$ do
            run ($\mathrm{c}_{UM}$)
        end
    end
end
```

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. Specifically, it includes the datacenter broker and ID description, the VM instantiated list, the number of cloudlets and VMs, the host allocation policy as well as operations for creating, submitng and executing requests.

Within the timeframe of Fig. 2, that represents the cloud life cycle by incorporating the interactions between Algorithms 1, 2, 3 and 4, 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 as happened within the hosts of the cloud datacetner. The next sections contain the configuration of the simulation environment, and the discussion of the static and dynamic VMs instantiation cases. We firsly explore the results of the cloud simulator with a variation of the cloudlet input number and secondly, we present the dynamic reactive workload deployment case in whcih generation of VMs based on VMs migrations from a pool of available VMs.

5. Performance Modelling 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 based on the CloudSim (Calheiros *et al.*, 2011), a framework for modeling and simulating clouds and their services. 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. In addition, we have utilized functionalities implemented in the SimIC (Sotiriadis *et al.*, 2013a) such as the deployment of VMs. In particalar, the SimIC aims of achieving interoperability, flexibility and service elasticity while at the same time introducing the notion of heterogeneity of multiple clouds configurations. Based on that we configure our experiment to contain the characteristics of Table 1.

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 formula (2). The variable I represents an integer number greater than 0.

$$\mathit{TotalSimulationTime} = \sum_{set=1}^{n} \mathit{FinishSimulatorTime}.\mathit{Cloudlet}[i]. \tag{2}$$

In addition we use the average execution time that a cloudlet set requires to be executed as given from formula (3). The variable named as *Cl* denotes a typical cloudlet.

$$\mathit{AverageExecutionTime}(\mathit{ClSet}) = \left(\sum_{set=1}^{n} \mathit{Cl}.\mathit{FinishTime}[i]\right)/\mathit{ClCount}[i]. \tag{3}$$

Table 1
The cloud experiment configuration of hosts, VMs and cloudlets.

Host	VM	Cloudlet
PEs: 1	PEs: 1	PEs: 1
Ram: 2048 (mb)	Ram: 1048 (mb)	Ram: 1000 (mb)
Storage: 1 000 000 (mb)	Size: 1000 (mb)	File size: 100 (mb)
MIPS: 1000	MIPS: 150	Output size: 100 (mb)
Bandwidth: 10 000 (mb/s)	VM name: Xen	User ID: 1
Allocation Policy: In time	Allocation Policy: In time	Utilization Model: Full

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. Both solutions are implemented in the Cloudsim simulation framwork which is an alternative for utilizing resources and evaluating static and dynamic hypothesis. Specifically, static presents the experimental analysis of the exploration of the testbed for identifying the performance of the hosts, VMs and cloudlets for certain job variations.

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 Lab, 2012).

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 (Buyya *et al.*, 2010). 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, Fig. 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.

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

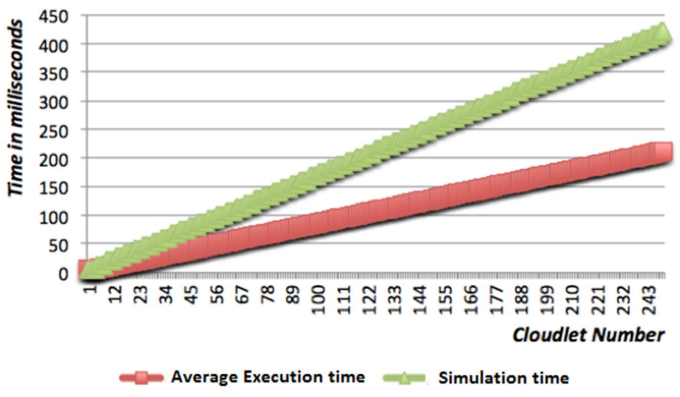

Fig. 3. Static job executions for 1 to 250 cloudlets in non-reactive deployment 1 to 50.

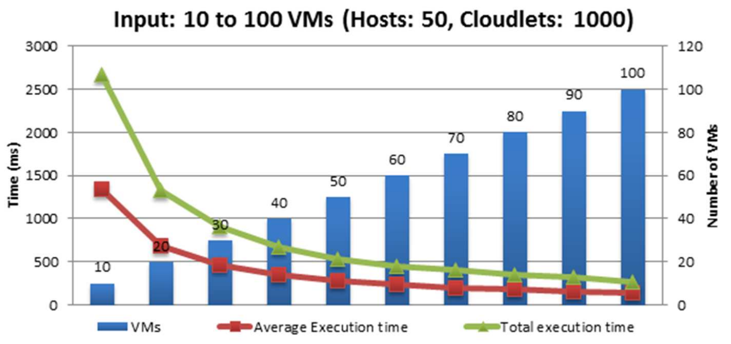

Fig. 4. Static deployment of VM instantiation performance for input submission of 1000 cloudlets in non-reactive setting.

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. 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.

In Fig. 6, 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. 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 (ms).

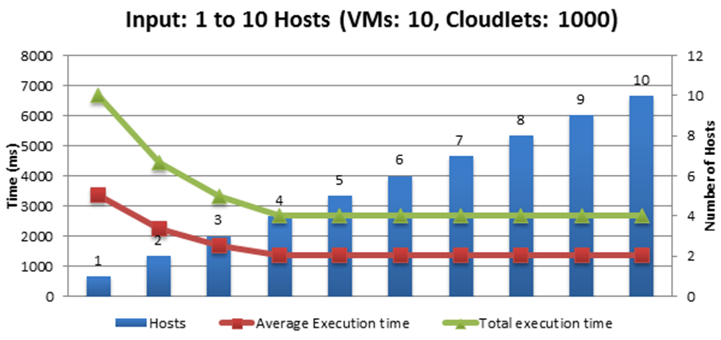

Fig. 5. Static deployment of VM instantiation performance for input submission of 1000 cloudlets in non-reactive setting.

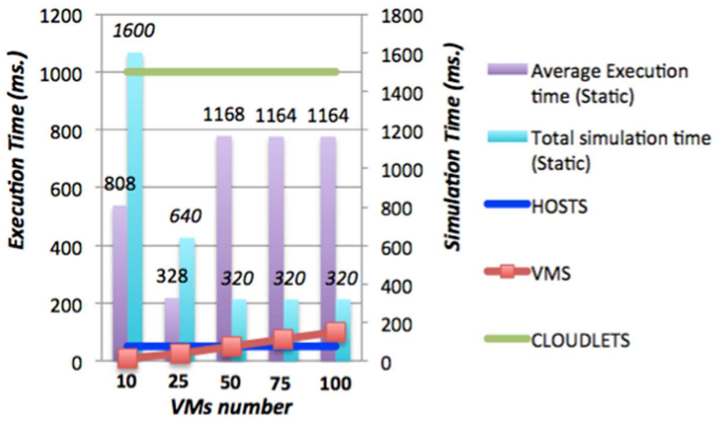

Fig. 6. The static and non-reactive benchmark results (average simulation and total execution time).

Table 2
The comparison of average execution times and simulation times fro static and dynamic cases.

Hosts	50	50	50	50	50
Cloudlets	1000	1000	1000	1000	1000
VMs	10	25	50	75	100
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. *The Static VM 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

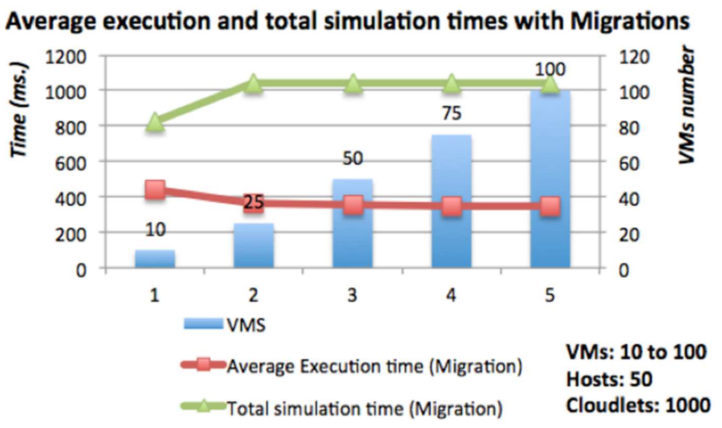

Fig. 7. Average execution and simulation times of 1000 cloudlets with dynamic migration of reactive VMs instantiation.

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. 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 (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 formula (4).

$$MigrationDelay_{VMi} = \frac{VM_{RAMi}}{VM_{BWi}} + (f_{VMi} \times \mathrm{const}_{VMi}). \tag{4}$$

We extend the formula of (Cloudsim Lab, 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} (Lagar-Cavilla *et al.*, 2009) multiplied by a constant variable constVMi to control the rate of latency. For example in this experiment we set the fi in 1000 (ms.) and the consti 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.

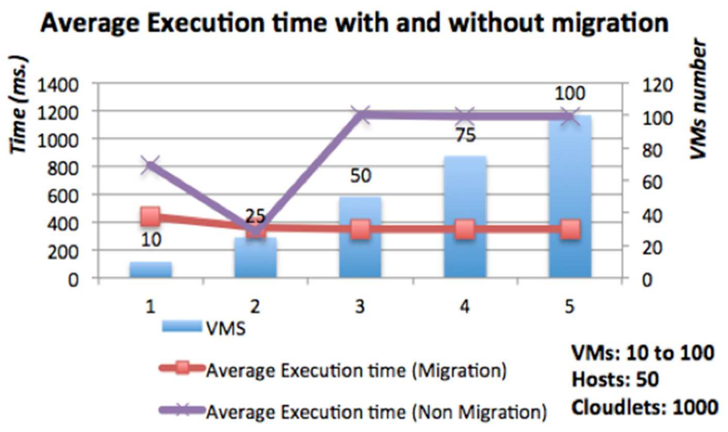

Fig. 8. Average execution time of 1000 cloudlets with the dynamic migration of reactive VMs instantiation in comparison with the static case of Fig. 6.

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 400 ms. However, the total simulation time 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 1000 ms. Figure 8 demonstrates the results of the average execution time by comparing the static benchmarks (as presented in Fig. 6) and the dynamic instantiation for the same VMs variation (10 to 100).

Specifically, Fig. 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 100 ms.) 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, 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 1000 ms.

Figure 10 demonstrates the indicators of the number of VM migrations, the SLA violations and the energy consumption according to the model of (Beloglazov and Buyya, 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).

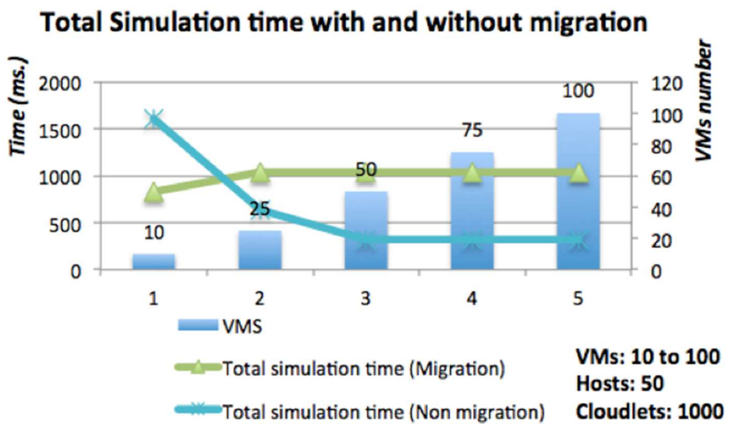

Fig. 9. Total simulation time of 1000 cloudlets with the dynamic migration of reactive VMs instantiation in comparison with the static and non-reactive case of Fig. 6.

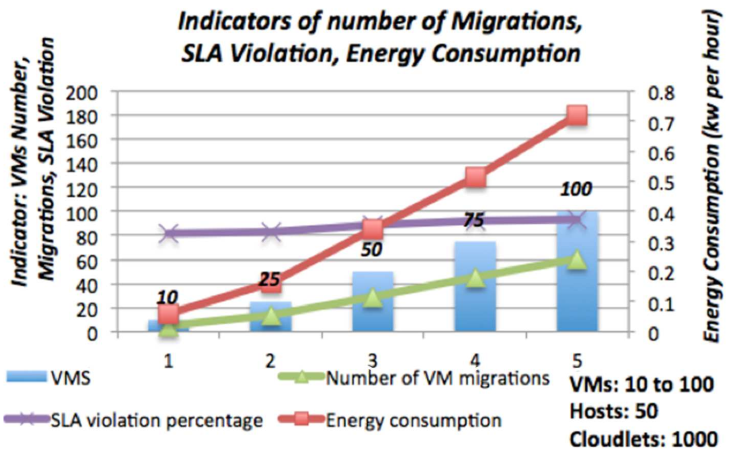

Fig. 10. Number of migrations, SLA violation and energy consumption for reactive VM instantiation.

In Fig. 10, the indicators related with VM migration 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 and Buyya, 2010), 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 371 ms. while in static case is 926 ms. In contrary, the average simulation time in static case is 640 ms. while in the dynamic case is 997 ms. Thus, a future challenge is to identify ways of minimizing the simulation time in order to reach the average levels of static instantiation 640 ms.

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 have been presentes because of the optimization of the average execution time of the cloudlets, which represents the running time of the jobs. The next section illustrates various scenarios in which the dynamic solution could be applied. The common aim of each one is to enhance the agility and flexibility of the cloud by dynamically instantiating VMs.

6. Implications of Deploying Dynamic VMs Instantiations

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

- 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.
- 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.
- 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.
- 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.

- 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. Such scenarios are presented in literature (Beloglazov and Buyya, 2010), 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 CloudSim 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 to increase the quality of service levels from the users perspective, the overall service execution time levels (represented by the average execution time) 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 (Sotiriadis *et al.*, 2012b), a setting for exchanging cloudlets and VMs among interoperable clouds for improving the quality of service levels.

Finally, the work will aim to the inter-cloud model. The last one comes to expand cloud capabilities in terms of quality of provisioned services and elasticity. In particular this the Inter-Cloud Meta-Scheduling (ICMS) (Sotiriadis *et al.*, 2013b) model presents an adaptive environment that allows efficient service distribution among interoperable clouds. In such setting the effective VM migration will enhance the inter-cloud performance by allowing the exchanging of VMs for dynamic situations.

References

Beloglazov, A., 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, Article 4, 6 pp.

Bessis, N., Sotiriadis, S., Pop, F. and Cristea, V. (2012). An architectural strategy for meta-scheduling in inter-clouds. In: *1st International Workshop on Inter-Clouds and Collective Intelligence (iCCI-2012) in Conjunction with the 26th IEEE International Conference on Advanced Information Networking and Applications (AINA-2012)*, Fukuoka, Japan, 26–29 March 2012, ISBN 978-1-7695-4652-0, pp. 1184–1189.

Bessis, N., Sotiriadis, S., Cristea, V. Pop, F. (2011). Modelling requirements for enabling meta-scheduling in InterCloud and inter-enterprises. In: *Proceedings of the 3rd International Conference on Intelligent Networking and Collaborative Systems (INCoS 2011)*, IEEE CSP.

Bessis, N., Sotiriadis, S., Cristea, V., Pop, F. (2012). Meta-scheduling issues in interoperable HPCs, grids and clouds. *International Journal of Web and Grid Services*, (8)2, 153–172.

Bessis, N., Sotiriadis, S., Cristea, V., Pop, F. (2012). Optimizing the energy efficiency of message exchanging for service distribution in interoperable infrastructures. In: *4th IEEE International Conference on Intelligent Networking and Collaborative Systems (INCoS-2012)*, 19–21 September 2012, Bucharest, Romania, ISBN: 978-0-7695-4808-1, pp. 105–112.

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., Calheiros, R.N. (2010). InterCloud: utility-oriented federation of cloud computing environments for scaling of application services. In: *Proc. of Conference on Algorithms and Architectures for Parallel Processing*. LNCS, Vol. 6081. Springer, Berlin, pp. 13–31.

Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R. (2011). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1) 23–50.

Carolan, J., Gaede, S. (2009). *Introduction to Cloud Computing Architecture*. White paper 1st edition. Available by Sun Microsystem, Inc.

Cloudsim, The Clouds Lab (2012). *Cloudsim: a framework for modeling and simulation of cloud computing infrastructures and services*. Available at `http://www.cloudbus.org/cloudsim/`. Accessed 28/02/2012.

Diaz, J., Laszewski, G., Wang, F., Younge, A.J., Fox, G.C. (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, pp. 560–564.

Foster, I., Kesselman, C. (2004). In: Nabrzyski, J., Schopf, J.M., Weglarz J. (Eds.). *The Grid in a Nutshell. In Grid Resource Management*, Kluwer Academic, Norwell, pp. 3–13.

Frachtenberg, E., Schwiegelshohn, U. (2007). New challenges of parallel job scheduling. In: Frachtenberg, E., Schwiegelshohn, U. (Eds.), *Proceedings of the 13th International Conference on Job Scheduling Strategies for Parallel Processing (JSSPP'07)*. Springer, Berlin, pp. 1–23.

Gong, C., Liu, J., Zhang, Q., Chen, H., Gong, Z. (2010). The characteristics of cloud computing. In: *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., Lepreau, J. (2008). Large-scale virtualization in the Emulab network testbed. In: *Proceedings of the USENIX Annual Technical Conference*, Boston, MA.

Lagar-Cavilla, H.A, Tolia, N., De Lara, E., Satyanarayanan, M., O'Hallaron, D. (2007). Interactive resource-intensive applications made easy. In: Campbell, R.H., Cerqueira R. (Eds.), *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware (Middleware '07)*. Springer, New York, pp. 143–163.

Lagar-Cavilla, H.A, Whitney, J.A, Bryant, R., Patchin, P., Brudno, M., De Lara, E., Rumble, S.M., Satyanarayanan, M., Scannell, A. (2011). SnowFlock: virtual machine cloning as a first-class cloud primitive. *ACM Transactions on Computer Systems* 29(1), Article 2 (2011), 45 pp.

Lagar-Cavilla, H.A., Whitney, J.A., Scannell, A.M., Patchin, P., Rumble, S.M., De Lara, E., Brudno, M., 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, pp. 1–12.

Osman, S., Subhraveti, D., Su, G., Nieh, J. (2002). The design and implementation of Zap: a system for migrating computing environments. *SIGOPS Operating Systems Review*, 36(SI), 361–376.

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

Sapuntzakis, C.P., Chandra, R., Pfaff, B., Chow, J., Lam, M.S., Rosenblum, M. (2002). Optimizing the migration of virtual computers. *SIGOPS Operating Systems Review*, 36(SI), 377–390.

Sotiriadis, S., Bessis, N. Xhafa, F., Antonopoulos, N. (2012a). From meta-computing to interoperable infrastructures: a review of meta-schedulers for HPC, grid and cloud. In: *26th IEEE International Conference on Advanced Information Networking and Applications (AINA-2012)*, Fukuoka, Japan, 26–29 March 2012, ISBN 978-1-7695-4651-3, pp. 874–883.

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

Sotiriadis, S., Bessis, N., Antonopoulos, N. (2012c). Decentralized meta-brokers for inter-cloud: Modeling brokering coordinators for interoperable resource management. In: *4th IEEE International Conference on Intelligent Networking and Collaborative Systems (INCoS-2012)*, Chongqing, 29–31 May 2012, ISBN 978-1-4673-0024-7/10, pp. 2475–2481.

Sotiriadis, S., Bessis, N., Antonopoulos, A., Anjum, A. (2013a). SimIC: designing a new inter-cloud simulation platform for integrating large-scale resource management. In: *27th IEEE International Conference on Advanced Information Networking and Applications (AINA-2013)*, 25–28 March, Barcelona. IEEE Computer Society, Washington, pp. 90–97.

Sotiriadis, S., Bessis, N., Kuonen, P., Antonopoulos, A. (2013b). The inter-cloud meta-scheduling (ICMS) framework. In: 27th IEEE International Conference on Advanced Information Networking and Applications (AINA-2013), 25–28 March, Barcelona. IEEE Computer Society, Washington, pp. 64–73.

Vrable, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A., Voelker, G., Savage, A. (2005). Scalability, fidelity and containment in the Potemkin virtual honeyfarm. *SIGOPS Operating Systems Review*, 39(5), 148–162.

Wood, T., Shenoy, P., Venkataramani, A., Yousif, M. (2007). Black-box and gray-box strategies for virtual machine migration. In: *Proceedings of the USENIX Conference on Networked Systems Design Implementation (NSDI'07)*. USENIX Association, Berkeley, 17 pp.

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

Zayas, E. (1987). Attacking the process migration bottleneck. *SIGOPS Operating Systems Review*, 21(5) 13–24.

Zhang, Y., Franke, H., Moreira, J.E., Sivasubramaniam, A. (2000). The impact of migration on parallel job scheduling for distributed systems. In: Bode, A., Ludwig, T. II, Karl, W., Wismuller, R. (Eds.). *Proceedings of the 6th International Euro-Par Conference on Parallel Processing (Euro-Par '00)*. Springer, London, pp. 242–251.

Zheng, J., Eugene Ng, T.S., 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, pp. 133–144.

Younge, A.J., Henschel, R., Brown, J.T., Von Laszewski, G., Qiu, J., Fox., G.C. (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, pp. 9–16.

N. Bessis is a Head of Distributed and Intelligent Systems (DISYS) research group, a full Professor and a Chair of Computer Science in the School of Computing and Mathematics at University of Derby, UK. He is also an academic member in the Department of Computer Science and Technology at University of Bedfordshire (UK). His research interest is related to dynamic distributed environments such as grids, interclouds, crowds and internet of things. He is involved in and leading a number of funded research and commercial projects in these areas. Prof. Bessis has published over 180 papers, won 3 best paper awards and is the editor of 7 books and the Editor-in-Chief of the International Journal of Distributed Systems and Technologies (IJDST). In addition, Prof. Bessis is a regular reviewer and has served several times as a keynote speaker, conferences/workshops/track chair, associate editor, session chair and scientific program committee member.

S. Sotiriadis is a research collaborator of the Intelligent Systems Laboratory (InteLLigence) of the Department of Electronic & Computer Engineering, Technical University of Crete (TUC) located in Chania. His research interests are related with Future Internet (FI) application design, FI Architectures, inter-cloud and cloud computing and in general with large scale distributed infrastructures. Other areas of interest include artificial intelligence and interoperable systems and multithreaded simulation infrastructures. Stelios Sotiriadis has published over 45 papers in conference proceedings and international journals as well as he is program committee member and peer-reviewer of international conferences and journals.

F. Xhafa holds a PhD in Computer Science from the Department of Languages and Informatics Systems (LSI) of the Technical University of Catalonia (UPC), Barcelona, Spain. He was a Visiting Professor at the Department of Computer Science and Information Systems, Birkbeck, University of London, UK (2009/2010) and a Research Associate at College of Information Science and Technology, Drexel University, Philadelphia, USA (2004/2005). Dr. Xhafa holds a permanent position of Professor Titular at the Department of LSI, UPC (Spain). His research interests include parallel and distributed algorithms, combinatorial optimization, approximation and meta-heuristics, networking and distributed computing, Grid and P2P computing. Dr. Xhafa has widely published in peer reviewed international journals, conferences/workshops, book chapters and edited books and proceedings in the field. Dr. Xhafa has an extensive editorial and reviewing service. He is Editor in Chief of the International Journal of Space-based and Situated Computing, and International Journal of Grid and Utility Computing, Inderscience Pubs. Dr. Xhafa is actively participating in the organization of international conferences.

E. Assimakopoulou holds a first degree (University of Luton, UK), an MA in Architecture (University of Westminster, UK) and a PhD in Managing Natural Disasters using Grid Technology (Loughborough University, UK). She is a Visiting Lecturer at the Department of Computer Science and Technology at the University of Bedfordshire, UK and currently, a Visiting Researcher at the Distributed and Intelligent Systems (DISYS) research group, University of Derby, UK. She is an editor of a book, conference track/workshops chair and a regular reviewer in several international conferences and journals. Her research interests include emergency management, response and planning for disasters, business continuity, construction and risk management, resource management in distributed environments (such as grids, clouds and inter-clouds) and also advanced ICT methods (such as grid, clouds, crowd and other forms of applicable collaborative and distributed technologies) for disaster management. Eleana has published over 40 refereed works in these areas.

Debesų kompiuterijos planavimo optimizavimas: reaguojantis modelis virtualiosioms mašinoms dinamiškai diegti

Nik BESSIS, Stelios SOTIRIADIS, Fatos XHAFA, Eleana ASIMAKOPOULOU

Šis tyrimas siūlo sprendimų priėmimo proceso modelį virtualiosioms mašinoms diegti debesų kompiuterijos aplinkoje. Mes tiriame dvi veiklos laiko metrikų optimizavimo konfigūracijas: statišką, kurioje virtualiosios mašinos yra sukuriamos pagal debesų kompiuterijos orkestravimą, ir dinamišką, kurioje naudojant migravimą virtualiosios mašinos yra reaguojančiai adaptuojamos pagal darbų pateikimus. Mes integruojame abu sprendimus tame pačiame simuliatoriuje, kad galėtume matuoti įvairių virtualiųjų mašinų, darbų ir kompiuterių kombinacijų veiklą vidutiniu vykdymo ir viso modeliavimo laikais. Galime daryti išvadą, kad dinamiška konfigūracija yra perspektyvi, nes įgalina optimizuotą darbų vykdymo veiklą.