Cloud virtual machine scheduling: Identifying issues in modelling the cloud virtual machine instantiation

Stelios Sotiriadis[1], Nik Bessis[1], Fatos Xhafa[2], Nick Antonopoulos[1]
[1]School of Computing & Maths, University of Derby, Derby, United Kingdom
[2]Departament de Llenguatges i Sistemes Informàtics,
Universitat Politècnica de Catalunya, Barcelona, Spain
[1](s.sotiriadis, n.bessis, n.antonopoulos)@derby.ac.uk, [2]fatos@lsi.upc.edu

***Abstract*— Cloud computing provides an efficient and flexible mean for various services to meet the diverse and escalating needs of IT end-users. It offers novel functionalities including the utilisation of remote services in addition to the virtualization technology. The last one offers an efficient method to harness the cloud power by fragmenting a cloud physical host in small manageable virtual portions. As a norm, the virtualized parts are generated by the cloud provider administrator through the hypervisor based on a generic need for various services. However, several obstacles arise from this generalised and static approach. In this paper we study and propose a model for instantiating dynamically virtual machines in relation to the current jobs submission input. Following, we simulate a virtualised cloud environment in order to evaluate the model's dynamic-ness by measuring the correlation and analogy of virtual machines to hosts for certain job variations. This will allow us to measure the deviation of the execution time of various VMs instantiations per job length.**

Keywords: Cloud, Virtualization, Virtual Machine instantiation, Static and Dynamic Virtual Machine scheduling

I. INTRODUCTION

Over the recent years, cloud computing has been emerged as one of the most important IT infrastructures for delivering computational services. These are in the form of virtualized hardware and software that delivered to the end-users via the public internet. In the generic form the cloud paradigm, is a marketing term which offers low charge computational resources with cost analogous to the actual consumption (CPU, memory, etc.) of the user. It also includes major characteristics of the functionality of various evolving technologies such as grid, utility and virtualisation computing. On the one hand, starting with the grid computing part, cloud utilizes a federation of computer resources – analogous to distributed systems – that belong to the same administrative domain with the purpose of addressing the users' requests. On the other hand, utility computing, offers the mean for measuring computing power and storage; and packaging resources in order to be distributed as metered service in a lower cost.

At last, the virtualization technology refers to the creation and orchestration of small virtual computational chunks in the form of an abstract computing platform; thus hiding the physical complexity and its characteristics from the end-users. By using this technology, cloud computing could easily manage and control the whole power by consolidating various small virtual machines to integrate hosts with different features. The great advantage of this technique is that multiple physical servers could be replaced by one larger that manage the integration of pseudo-machines for increasing the utilization of the resources based on the generated needs. Likewise, virtual machines (VMs) could be easily orchestrated and relocated to a different physical host, if desired, in a situation of high system utilization or a disaster case. In addition, provisioning of services could be increased as various virtual platforms (OS and software) could be instantiated at regular intervals in the form of heterogeneous settings.

Based on that discussion, in this paper we take the advantage of virtualization to study the effectiveness of VM instantiation based on static and dynamic circumstances. In addition we propose two models for generating cloud VMs based in those two cases. Firstly, the static case in which virtual machines are generated according to requirements posed by hosts and secondly the dynamic case in which virtual machines are generated according to a recorded variation of the number of jobs submitted in previous job delegations within the cloud datacentre from the datacentre end-users.

The actual solutions will be able to be applied initially to a single cloud and at a later stage to an extended interoperable cloud setting namely as InterCloud. In general, the last term aims to expand the cloud capabilities in terms of hosted services with the aim of achieving a wider distribution of resources, yet by retaining global resource utilization equilibrium among various resource pools [6]. Thus in this work we integrate the solution for measuring the performance of tasks, in terms of the deviation of the average execution time of jobs implemented in a simulation platform (CloudSim [refs]). We further test our environment by simulating the infrastructure in terms of the number of hosts and processing elements (PEs) of a real grid system by utilizing a hybrid grid workload trace of the AuverGrid project [refs]. The target is to evaluate the stability of the model for achieving a constant optimisation of the range of the average execution time. By allowing virtual machines to be auto-generated and terminated according to a coefficient value corresponding to the bulk of tasks we will identify the effect of various analogies of virtual machines instantiation by hosts along with the number of jobs that enter the cloud datacentre.

To conclude, the next section (2) presents a literature analysis study of the theoretical issues of the virtualized services. The rest of the paper is organised as follows, in section 3 we discuss the virtual machine layered structure methodology, in section 4 the cloud platform simulation environment, in section 5 the static and the dynamic approaches and the generated simulation results from both solutions (section 6) along with the comparison and the critical discussion of the produced simulated datasets. At last, the research study completes with a discussion of the concluding remarks and the future work directions of our research.

II. Review of Related Approaches

As discussed in previous section, the term virtualization refers to the deployment of virtual hosts instead of physical ones for splitting the computational power of the underlying infrastructure. The fundamental idea is that the actual machine (called host machine) generates and orchestrates various virtual machines (called guest machines). The terms host and guests distinguish the software that is executed in the virtual machines. In addition, the host machine contains software for creating and controlling the virtual parts that called hypervisor. The last one controls and allows multiple isolated guests to run concurrently within the same host machine.

In virtual cloud environments, the hypervisor plays a vital role in the whole service management procedure. In general, authors in [1] suggest that two types of hypervisors exist, the Type 1; native or bare metal hypervisor that is executed within the physical computer for hosting guests, and the Type 2; hosted hypervisors that execute guests as applications on an unmodified commodity operating system. Examples of Type 1 are the Kernel-based Virtual Machines (KVM) and Xen [refs], while examples of Type 2 include the VMWare Server and Workstation, Parallels Workstations and Oracle VM VirtualBox [refs]. In any case of Type 1 or Type 2, developers make use of the hypervisor software for developing and deploying their services (hardware or software) based upon the generic needs of the customers or the company’s leasing target, by always aiming to scalability and flexibility of lightweight solutions.

At a first glance, the most common used hypervisors are the Xen [3] and the KVM [4] which both are under the GNU general public licence. Authors in [2] compare both solutions and discuss that Xen project has been released earlier in 2003 and has been included various Linux distributions, while is also the base hypervisor for Citrix Enterprise solution and Amazon EC2. In contrast, KVM, is has been released in 2007 [refs]; it introduced a new way to manage virtual machines, that has been proven to be quite efficient while at the same time particularly lightweight as presented in [4]. All these years various studies have compared both hypervisors and authors in [5] suggest that in the case of comparable performance Xen scalability properties outperforming KVM. Nevertheless, the choice for one hypervisor or the other can depend on performance, flexibility of use, and elasticity of requested services as well as strategic considerations [2] of the cloud provider.

To this extend, all these years numerous cloud management solutions have been developed in order to deploy manageable virtual parts of physical cloud hosts. Though, a general appreciation of the technical issues in cloud hypervisors is not the scope of this study itself. In contrast, the study aims to explore the means that affect the performance of a cloud hypervisor (either Xen or KVM) within an InterCloud environment. In such case one of the most important design issues for an inter-collaborative cloud is the resource scheduling strategy with respect to its local cloud data-centre scheduling plan. Specifically, the approach implies that a local data-centre should participate in the on-demand resource selection process at both local (intra-) and global (inter-) scale as well as manage the resource selection, demand allocation and queuing of user tasks at a local level by considering the characteristics of the actual system (centralized or decentralized) as well as the requirements of the desired scenario. Thus, in this work we take the view of virtualization by highlighting the job scheduling perspective. For this reason we will design virtual machines based on two requirements, firstly the generic needs (e.g. virtual machines are generated analogous to hosts) and secondly the job input records. The last one implies that VMs are generated at runtime based on the appreciation of jobs for computational resources.

Similar works in this field that aim to a dynamically changed of development and deployment of VMs include the VM SnowFlock technique that enables the implementation of numerous patterns in cloning guests. Specifically, [7] presents a study of VM forking solutions. These include the sandboxing technique from the security viewpoint, the enabling of parallel computation of tasks in VMs, the load handling to instantiate new VMs on-demand based on sudden variations of workload. Finally, the opportunistic job utilization allows the utilizing of unused CPU cycles with short jobs. An analogous solution to VM fork is the VM migration [8] that aims to improve manageability, performance and fault tolerance of systems. In this case the incentive that justifies the VM orchestration is to balance the system load by migrating overloaded VMs from the one server to another. However, the VM migration will be addressed as an additional issue in our future works. This is because in this work we study the efficient virtual machine orchestration based on various job submissions initially within a single and at a later stage in a InterCloud environment.

The last term, InterCloud, has been emphasised by the leading vendors in cloud services area such as HP, Intel, Yahoo, etc. [refs]. It is noticeable that their state-of-the-art efforts have led to the establishment of a federation of collaborated clouds with joint initiatives. In contrast to aforementioned works, our vision of IntreCloud includes an inter-cooperative infrastructure of inter-enterprises as introduced in [refs]. To conclude, in this section we have described virtualization issues for identifying current approaches. It should be mentioned that we will base our design in a Type 2 hypervisor as discussed in the next section 3 in which we present layered structure for developing and deploying virtual machines.

III. The Cloud Virtual Machine Layered Structure

The term virtualization in cloud computing refers on the execution of multiple operating systems in small manageable chunks concurrently, thus generating guests which are the cloud virtual hosts or nodes. Although the virtualization technology is not relatively new, the concept of an efficient scheduling could be affected from a dynamic VM reasoning is an emerging challenge [9]. Specifically, the last authors suggest that scheduling research for virtualization is still in its infancy; however underlines the various areas to explore in this subject. In real world cloud environments, applications are installed in VMs and their ends-users request for utilizing specific software. For example, a cloud could host an OS for a user; however, the cloud hypervisor is unaware of the exactly demanded workload, thus the VM generated in an opportunistic method of an average request. In addition, in the case of an InterCloud environment, the distributed hypervisors of the cloud providers usually function with no coordination and knowledge sharing [9] and this affects the overall scheduling procedure. To this extend in this work we detail the scheduling issue of a single cloud that at later stage of the research will be extended into an InterCloud environment. Thus, the primary goal of this work is to study the feasibility of VM instantiation when the solution is based in static and dynamic scheduling decisions made by the cloud hypervisor. Specifically, we study the scheduling behaviour when user demands for executing specific jobs arrive in a cloud datacentre.

Before discussing that, it is essential to demonstrate the overall cloud and InterCloud layers that could affect the overall scheduling decisions. Figure 1 demonstrates the layered structure of a typical cloud environment.

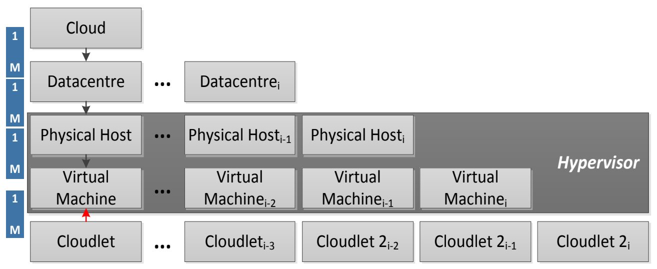

FIGURE 1: THE CLOUD COMPUTING LAYERED STRUCTURE

Specifically, each cloud environment contains one or more datacentres (DCs), thus the cardinality is one (cloud) to many (DCs). Then each DC could contain one or more physical machines (hosts) thus the relationship is again one (DC) to many (hosts). Finally each host could generate one or more VMs that are controlled by the hypervisor software which is responsible for generating, partitioning and instantiating VMs based on the posed requirements of the cloud administrator. Finally, various jobs (cloudlets) could be executed within one virtual machine thus the cardinality in this case is many (cloudlets) to one (VM). It should be mentioned that in this work we assume that each cloudlet could be the smallest part of large job submitted in the environment in the form of a parallel processing job.

In the case of management of the overall virtual machine development, the hypervisor plays an important role as controls the OS and the deployment of applications within the VM. It should be mentioned that the hypervisor is located among the physical host and the virtual machine layer of the layered structure as illustrated in figure 1. As discussed in section 2, there are two basic types of hypervisors, the Type 1: bare-metal and Type 2: hosted. In figure 2 we demonstrate the Type 1 hypervisor that is located beneath the host hardware layer [refs].

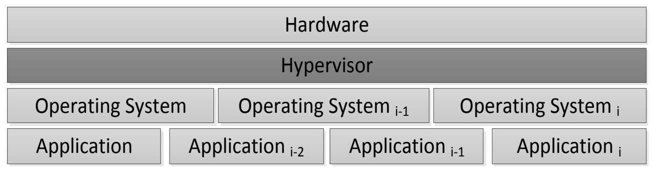

FIGURE 2: THE TYPE 1: BARE-METAL HYPERVISOR STRUCTURE

In contrast, figure 2 demonstrates the Type 2 hypervisor that is placed as software beneath the OS layer of the hardware [refs].

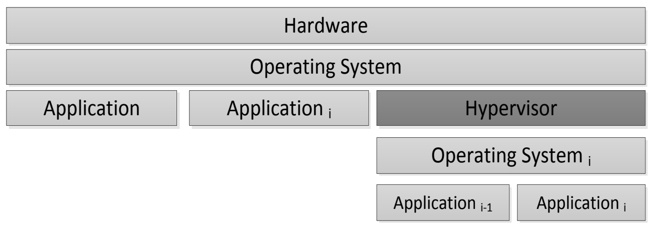

FIGURE 3: THE TYPE 2: HOSTED HYPERVISOR STRUCTURE

In general both aforementioned types of hypervisors could be utilised by our solution, however Type 2 layered structure is similar to the one of the experimental setting, and thus, we will base our design in this type.

Having said that, herein, we present a study of considering two different VM instantiation parameters within a cloud environment (and eventually an InterCloud) as follows:

a) In the first case we utilise a static and predefined hypervisor decision for the number of VMs within the system. The result for selection is based on an opportunistic decision of the number of the VMs within the datacentres. In addition, the VMs are generated prior to the job scheduling phase.

b) In the second case we utilise a dynamic hypervisor decision for the number of VMs instantiation based on the amount of the jobs to be submitted. In this situation, the VMs are generated in analogy with the number of the jobs and their computational workload.

Unambiguously, on the one hand, both solutions could address the same cloud requirements. This is to achieve a well-organized and immaculate workload handling in efficient amount of time. Moreover, both cases offer a high degree of heterogeneity by either instantiating randomly chosen VM platforms (in the static case) or required platforms (in the dynamic case). On the other hand, some advantages the one are implications of the other. This is clear in the event of power management, in which the dynamic VM instantiation case can play an important role in power saving. However, security issues could be raised as the static environment is repeatedly functioned and considered less complex in structure.

To conclude in this section we have discussed the cloud virtual machine layered structure by analysing the functionality of each part. In the next section we define the experimental platform, the appropriate metrics and the simulation configuration for developing VMs in static and dynamic job submissions. In the following sections we integrate our model and we perform the experiments for identifying benchmarks of each approach.

IV. The Virtual Machine Instantiation Model

Herein, we present a model for instantiating VMs within a cloud environment analogous to the AuverGrid [refs] grid system. Specifically, at the first stage we demonstrate the standard static method of instantiating the VMs based on the host requirements. This is an opportunistic decision of the self-interested motives of the hypervisor for creating VMs, e.g. to develop a huge number of VMs for handling all requests, however by suffering in computational power. Another example is to deploy different kinds of VMs with various OS for minimizing heterogeneity issues; however this could lead to a significant number of idle machines. To this extend, we present a model for developing VMs by the hypervisor in Type 2 systems, by identifying dynamic and run-time issues on the job submission phase. In the next section we present the static model and the dynamic models along with their pseudo-codes and their simulation results within the CloudSim.

A. The static VM instantiation model

The static VM instantiation model assumes that the decision of the VMs deployment is taken prior to the job submission phase. Thus, when the jobs arrive in the resource pool, the schedulers (either local or meta-) select the appropriate resource (virtual or physical) for scheduling the tasks. In our case each VM has a local scheduler that queues and executes the jobs in a first come first serve (FCFS) manner.

Static VM instantiation Job distribution algorithm

```
Require:  $Jobs_{num}$, $Wa_i$ : the initial jobs number of the workload archive
          $Host_i$: the physical host
          $Req_{node}$: the requested node
          $Pool_{host}$: the physical resource list
          $Pool_{VMs}$: the virtual resource list
          $Res_{VM}$: the responder virtual resource (the guest)
          $Res_{LRMS}$: the responder LRMS
          $Res_{queue}$: the responder queue list
          $Job_{desc}$: job description in requested processing elements, estimation execution time
          $Message_{jobAllocation}$: the job allocation requested message
          $Message_{informative}$: the information on job delegation message
          $Message_{jobDelegation}$: the job execution request
          $Message_{results}$: the job delegated job results come directly from the remote centralised scheduler
          $Del_{list}$: A vector with a list of accepted delegated resources
          $OpportunisticCriterion_i$: The opportunistic execution criterion
Require:  Hypervisor(), Send message(), Get message(), Set criterion()
1:  **for** $Host_i = \{i, i{+}{+}, n\} \in Pool_{host}$ **do**
2:        Hypervisor ($Pool_{host}$, $Pool_{VMs}$, $OpportunisticCriterion_i$) **accepts** $Req_{node}$
3:        **for** $Jobs_{num} = \{y, y{+}{+}, y\} \in Wai$ **do**
4:              **for all** $Res_{LRMS} \in Pool_{VMs}$ **do**
5:                    Send MessagejobAllocation($Job_{desc}$) **to** $Res_{LRMS}$, $Res_{queue}$
6:                    Set criterion($Criterion_i$)
7:                    Get $Message_{informative}$
8:                    $Del_{list} \leftarrow Res_{LRMS}$
9:                    $Del_{list++}$
10:             **end for**
11:       **end for**
12:       for all $Res_{meta\text{-}scheduler} \in Del_{list}$ **do**
13:             Send MessagejobDelegation ($Job_{desc}$) **to** $Res_{LRMS}$, $Res_{queue}$
14:             Get $Message_{results}$
15:       **end for**
16:       **if** $Del_{list} = \emptyset$ **then goto step 1**
17:       **end if**
18: **end for**
```

The above algorithm demonstrates the job distribution within a static setting of a cloud hypervisor for instantiating VMs based on an opportunistic decision for always having a sufficient number of virtual resources.

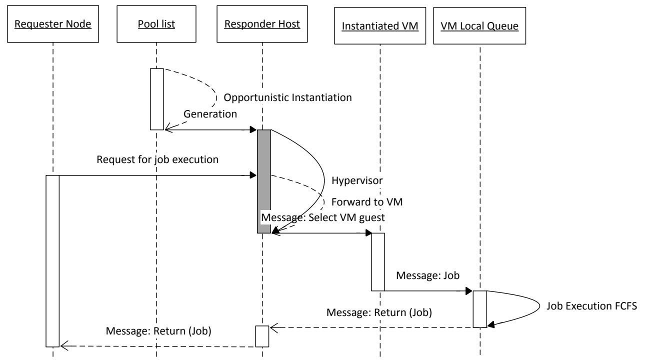

FIGURE 4: THE SEQUENCE DIAGRAM OF THE STATIC VM INSTANTIATION

Specifically, each host of the pool list generates a number of VMs according to the opportunistic instantiation criterion. The newly deployed virtual resources are ready for job execution as they already have a local queue in FCFS fashion. When a new job arrives in the queue the request is directly send to the physical resource, in which the hypervisor has already instantiated the VMs. The forwarded message for job execution goes to the guest VM node and places the job(s) to its local queue named as Local Resource Management System (LRMS). Finally, the executed job returned back to the requester node through the physical host. Figure 4 demonstrates the sequence model of the aforementioned procedure.

B. The dynamic VM instantiation model

The dynamic instantiation allows VMs to be generated on demand based on the current job input from the analysis of previous job delegations. This is to say that the workload affects directly the number of the VMs and the computational virtual resources deployed by the hypervisor. Basically, the requester node asks for job execution directly from the pool list of the physical resources. Then the hypervisor of each physical machine generates the number of the VMs to be deployed for the specific job requirements. The generation is happens by forking, which is a way of generating child VMs from parents by only copying the state of the thread within a multithreading environment. After that, the job sends to the instantiated VM(s) in their LRMS. Then the results are sending back to the requester node through the responder host. In parallel the responder host gets a notification message of job completion. This is to say that the developed VMs will be terminated.

The following algorithm demonstrates the job distribution within a dynamic setting of a cloud hypervisor for instantiating VMs based on a criterion analogous to the number of jobs and the required computational power of the workload archive. In addition, the pseudo-code includes a functionality of meta-scheduling for advanced and more complex scheduling cases. Specifically, each resource has a meta-scheduler that is responsible for coordinating the local queue (LRMS). Thus, a new layer has been added to delegate messages from the LRMS of the VM to the responder host.

Dynamic VM instantiation Job distribution algorithm

```
Require:  Jobs_num, Wa_i : the initial jobs number of the workload archive
          Jobs_counter: a variable to store the count of the jobs
          Jobs_characteristics: a variable to store the characteristics of the jobs
          Job_PEs: a variable to store the PEs of the job workload archive
          coefficient_i: a coefficient variable with regards to the jobs total number
          VM_num: the number of VMs
          VM_characteristics: the computational characteristics of VMs
          Host_i: the physical host
          Req_node: the requested node
          Pool_host: the physical resource list
          Pool_VMs: the virtual resource list
          Res_VM: the responder virtual resource (the guest)
          Res_LRMS: the responder LRMS
          Res_queue: the responder queue list
          Job_desc: job description in requested processing elements, estimation execution time
          Message_jobAllocation: the job allocation requested message
          Message_informative: the information on job delegation message
          Message_jobDelegation: the job execution request
          Message_results: the job delegated job results come directly from the remote centralised scheduler
          Del_list: A vector with a list of accepted delegated resources
Require:  Hypervisor(), Terminate(), Send message(), Get message(), Set criterion()
1:  for $Jobs_{num} = \{y, y{+}{+}, y\} \in Wai$ do
2:      $Jobs_{counter} \leftarrow Jobs_{num}$
3:      $VM_{num} \leftarrow Jobs_{counter} \cdot coefficient_i$
4:      $VM_{characteristics} \leftarrow Job_{PEs} \cdot coefficient_i$
5:      for $Host_i = \{i, i{+}{+}, n\} \in Pool_{host}$ do
6:      Hypervisor ($Pool_{host}$, $Pool_{VMs}$, $VM_{num}$, $VM_{characteristics}$) accepts $Req_{node}$
7:          for all $Res_{LRMS} \in Pool_{VMs}$ do
8:                  Send MessagejobAllocation($Job_{desc}$) to $Res_{LRMS}$, $Res_{queue}$
9:                  Set criterion($Criterion_i$)
10:                 Get $Message_{informative}$
11:                 $Del_{list} \leftarrow Res_{LRMS}$
12:                 $Del_{list++}$
13:                 Terminate($Pool_{VMs}$);
14:         end for
15:     end for
16:     for all $Res_{meta\text{-}scheduler} \in Del_{list}$ do
17:         Send MessagejobDelegation ($Job_{desc}$) to $Res_{LRMS}$, $Res_{queue}$
18:         Get $Message_{results}$
19:     end for
20:     if $Del_{list} = \emptyset$ then goto step 1
21:     end if
22: end for
```

The following figure 5 illustrates the procedure of the dynamic instantiation of VMs by incorporating the VM queue. Here, it should be mentioned that the VM queue first implements the meta-scheduling behaviour to coordinate the LRMS.

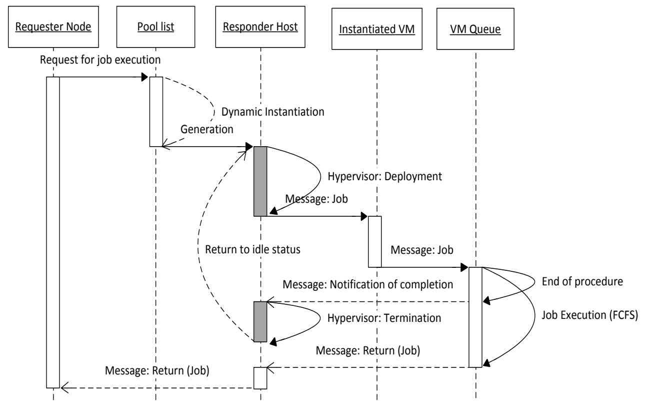

FIGURE 5: THE SEQUENCE DIAGRAM OF THE DYNAMIC VM INSTANTIATION

To conclude, in this section we have modelled the VM instantiation procedure in the cases of a) the static environment – that meant to be not affected from the job input file, and b) the dynamic VM instantiation that contains the on-demand creation and termination of VMs –that meant to be function in parallel with the number and the characteristics of the job workload input file. Accordingly, the next section presents the simulation experiment and the metrics to be used as benchmarks for each of the two circumstances.

V. The Simulation Configuration

In our experiment we utilize a simulation-based tool for designing the actual cloud infrastructure as an alternative to a real testbed system. Specifically, the whole experiment is configured to be run in CloudSim (version 2.1) as a way to investigate our hypothesis without being concerned of the lower level of technical details. By using this simulation setting we develop a cloud that consists of various datacentres and VMs that execute numerous cloudlets. So, our solution is based on developing a VM orchestration policy identical to a real-world hypervisor.

The policy integrates the static and dynamic circumstances of VMs instantiation when simulating a real grid workload traces. Specifically, we develop a hybrid workload trace configuration identical to the grid workload achieve (GWA) of the AuverGrid project. The last one is a grid platform that consists of five remote clusters that are composed from physical machines with dual 3GHz Pentium-IV Xeon running Linux OS. In our experimental platform environment we simulate one cloud identical to the AuverGrid project that contains five datacentres with the same host characteristics. Table 1 demonstrates the AuverGrid list of resources.

Table 1: The AuverGrid cluster configuration (include PEs as the number of CPUs and machines)

Resource ID	Cluster Name	CPU Number	Rating	Machines
14	clrlcgce032	186	1	93
6	clrlcgce010	112	1	56
10	clrlcgce021	84	1	42
22	obc4	56	1	28
18	iut153	38	1	19
Total available MIPs		**476**		

Then we run various experiments by generating a hybrid workload trace of the same amount of jobs (6000) with the same characteristics from the GWA file of the AuverGrid project. For comparing various simulation experiments we use as a metric the average deviation of the finish execution time.

VI. Simulation & Comparison

In this section we present the experimental analysis of a single cloud for certain job and VMs variations in the case of a static and dynamically instantiation setting.

A. *Static VM Instantiation configuration*

Specifically the experiment specification which has been implemented in CloudSim contains 5 hosts identical to the AuverGrid configuration. Then, we explore the behaviour of a hybrid job input file (identical to the 10% of the AuverGrid workload ≈ 600) when enters the cloud datacentre and instantiates a number of VMs. Specifically, we test the environment when each host instantiates 1, 2, 4, 6, 8 and 10 VM(s). In table 2 we illustrate the aforementioned specification.

Table 2: The cloud datacentre specification

Host Number	Vm Number	Cloudlet Workload Percentage	DCs Number	Average Deviation of execution time
1	1	10%	1	4808
1	2	10%	1	2408
1	4	10%	1	302
1	6	10%	1	202
1	8	10%	1	152
1	10	10%	1	122

The metric used for this experiment is the deviation of the average execution time as given from the formulae 1.

In table 3 we present the VM cloud specification, including the hybrid job inputs along with the datacentre parameters and the scheduling policy.

$$\text{AvgExTime(cloudlet)} = \sum_{i}^{n} (\text{ExTime}(LRMS_{cloudlet}) \cdot \frac{1}{count(\text{cloudlet})} \quad (1)$$

Table 3: The VM and Datacentre experiment parameters

VM parameters	Datacenter parameters
Size: 10000 (image size in MB) Ram: 512 (vm memory) Mips: 250 mips (millions of instructions per second) Bw: 1000 PesNumber: 1 (number of CPUs)	Policy: FCFS provision Mips: 476 Machine: 1 Dual Core Ram: 2048 Storage: 1000000 Bw: 10000

Figure 6 contains a graph to demonstrate the average deviation of execution time for the same job submission and various VMs instantiations. It should be mentioned that VMs are generated prior to the job submission.

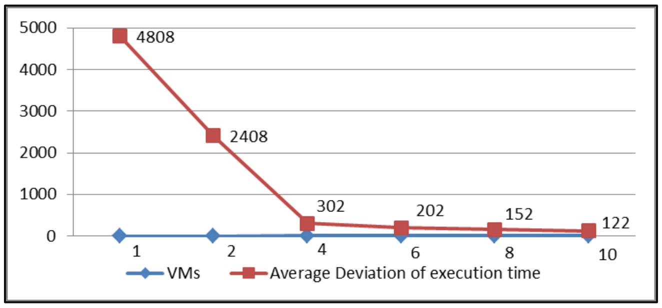

FIGURE 6: THE AVERAGE DEVIATION OF EXECUTION TIME PER VMS

It is apparent from the graph that if the number of VMs increased the average execution time is decreased significantly. Thus, in this case of the specific job input (with the precise cloudlet length) the more VMs are instantiated from the hypervisor the better overall performance gained. However, due to the experiment specification the highest number of concurrent VMs to be auto generated at the primary stage is a fixed number (tested to 10) because if the number gest bigger the cloud failed to execute cloudlets as there is no enough physical resources for the jobs.

For testing the behaviour of the same environment in heaviest job submissions we have developed a function to control the coefficient values of each characteristic that affects the overall job weight. In formulae 2 the *cl* denotes the current cloudlet, the *l* denotes the cloudlet length, the *fs* is the filesize, the *of* is the output file and the *pes* is the processing elements required from the job.

$$f(cl) = l * C_l + fs * C_{fs} + of * C_{of} + pes * C_{pes} \quad (2)$$

In table 4 we present the VM instantiation within the same cloud of table 2 for assessing the behaviour of the specification when the coefficient value of the cloudlet length changed to four times and eight times bigger respectively. This is to demonstrate the average execution time when heaviest jobs (initial length grew by 4 and 8 times) arrive in the queue.

Table 4: The cloud datacentre specification

VMs	**Average Deviation of execution time C_l= 1**	**Average Deviation of execution time C_l = 4**	**Average Deviation of execution time C_l = 8**
1	4808	19712.8	38944.8
2	2408	9872.8	19504.8
4	302	1238.2	2446.2
6	202	828.2	1636.2
8	152	623.2	1231.2
10	122	500.2	988.2

Similarly, figure 7 illustrates the deviation of the execution time for various coefficient values of the cloudlet length.

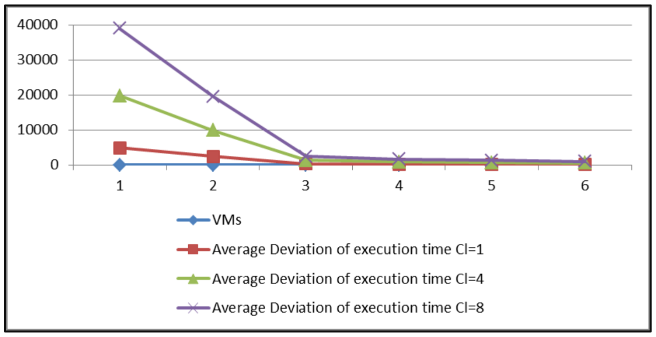

FIGURE 7: THE AVERAGE DEVIATION OF EXECUTION TIME PER VMS

However, as the number of VMs that are instantiated from the hypervisor gets bigger the amount of idle VMs increased significantly as well. In that case the computational power is spreader in low performance VMs that execute jobs slower, thus penalised the overall cloud performance. Figure 8 demonstrates the aforementioned situation.

As a conclusion, an important challenge is to identify the analogy of the variation of the VMs instantiation for various job submissions including low and heavy workloads. To this extend, the next section presents the empirical study of the dynamic instantiation of VMs in correlation with the cloudlet lengths with respect to the overall delay of the hypervisor for deploying VMs on demand.

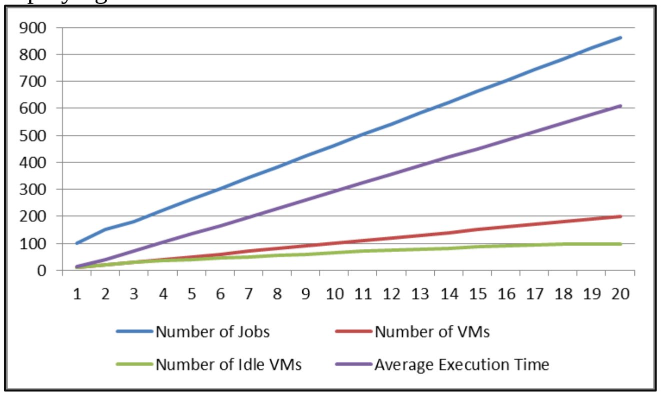

FIGURE 8: THE COMPARISON OF THE NUMBER OF JOBS, VMS, IDLE VMS AND AVERAGE EXECUTION TIME

B. *Dynamic VM Instantiation configuration*

The dynamic instantiation case offers a major advantage; the ability to actively deploy a number of VMs after considering previous job submissions with regards to the current input, for improving the overall performance of the scheduling. However, in reality the instantiation of newly installed VMs is a slow operation, which typically takes “minutes” (measured by Amazon elastic services 2 –EC2 [refs]) and presented by [7]. The result is that the runtime overhead is getting significant high when creating new VMs on the fly upon request. Thus, for reducing that time we suggest that in such situations developers could utilize a more complex technique used in Unix-like systems named as “process fork” [refs]. This is to say that new VMs are cloned from old ones that exist within a pool of standard deployed machines, rather than build VMs by scratch. Specifically, the actual procedure is executed in a multithreading environment in which threads are created based on the current demands.

In this section we present a dynamic configuration of the VMs instantiation by assuming that the hypervisor takes that decision during the cloud run-time. In practice, we measure the delay time by implementing an event within the cloud broker of the CloudSim simulator that decides when new VMs are required. The monitoring of the whole environment is currently happened empirically by observating task executions and documented the overall performance of the simulation. In this way, new VMs are instantiated within the broker class and a request to the cloud datacentre (and its hypervisor) sent in the form of periodically generic events.

Based on that, herein we suggest the configuration parameters that directly impact the quality (in terms of computational power) and the quantity of the VM instantiation. In detail, the simulation parameters are the following:

a) The number of cloudlets (jobs).

b) The length (size) of each cloudlet in terms of the image size in MB as allocated within a cloud VM.

c) The cloudlet function (2) that contains the relation of cloudlet characteristics (e.g. the filesize, outputsize, and

Pes number) with regards to the coefficient values of each one.

Taking the above parameters into account we have designed a component for supporting the decision of the number of VMs to be generated (by forking) when certain job variations happens. Figure 9 demonstrates the comparison of the number of Jobs, VMs, idle VMs and the deviation of the average execution time when the job input affects the VMs instantiation decision.

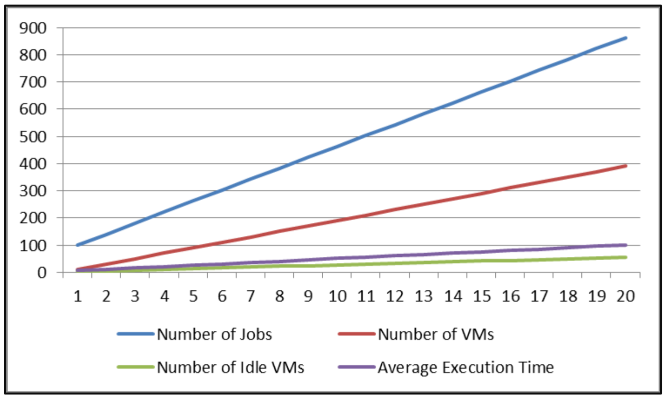

FIGURE 9: THE COMPARISON OF THE NUMBER OF JOBS, VMS, IDLE VMS AND AVERAGE EXECUTION TIME

In the setting of figure 9, we integrate our solution by incorporating a function that is analogous to the parameters as discussed in a, b, and c of the previous list. Specifically, this analogy is given by the formulae (3) where · denotes the operator.

$$\text{VMnum} = \text{JobNum} \cdot \text{JobLength} \cdot \text{coefNum} \cdot \text{coefLength} \quad (3)$$

Finally, when compare figure 8 and figure 9 we conclude that the number of idle VMs and the overall execution time have been optimised better when we control the VMs instantiation process.

VII. CONCLUSION AND FUTURE WORK

To conclude in this paper we have highlighted the need for controlling the VMs instantiation and the significant incentives gained, in terms of the optimization of the average execution time, when this model is adapted. By initially presenting a literature study of the theoretical issues of the virtualized services we have discussed the virtual machine layered structure and the static and the dynamic approaches. Finally through an experimental analysis we have presented the cloudlet-oriented configuration parameters that affect directly the VM instantiation decision of the hypervisor component. The next research step includes the identification of similar ways to improve the VM dynamic instantiation such as VM migration and the integration the solution within the simulation setting. This includes the realisation of the hypervisor component within the infrastructure along with the brokering implementation to monitor simulation times more efficiently.

VIII. ACKNOWLEDGEMENT

The authors would like to thank the AuverGrid project team for the source of the workload archive.

IX. REFERENCES

[1] Li, P., and Toderick, L. W. 2010. Cloud in cloud: approaches and implementations. In Proceedings of the 2010 ACM conference on Information technology education (SIGITE '10). ACM, New York, NY, USA, pp. 105-110

[2] Cerbelaud, D., Garg, S., and Huylebroeck, J. 2009. Opening the clouds: qualitative overview of the state-of-the-art open source VM-based cloud management platforms. In Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware (Middleware '09). Springer-Verlag New York, Inc., New York, NY, USA, , Article 22 , 8 pages

[3] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A.. 2003. Xen and the art of virtualization. In SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, pages 164–177, New York, NY, USA

[4] Habib., I. 2008. Virtualization with kvm. Linux J., 2008(166):8

[5] Matthews, J. N., Deshane, T., Shepherd, Z., Ben-Yehudah, M., Shah, A., and Rao. B., Quantitative comparison of xen and kvm. In Xen Summit, June

[6] Sotiriadis, S., Bessis, N., and Antonopoulos, N., Towards inter-cloud schedulers: A survey of meta-scheduling approaches, Sixth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Barcelona, Spain, Oct 26-28 2011

[7] Lagar-Cavilla, H. A., Whitney, J. A., Scannell, A. M., Patchin, P., Rumble, M. 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, 1-12.

[8] Liu, H., Xu, C-Z., Jin, H., Gong, J., and Liao., X. 2011. Performance and energy modeling for live migration of virtual machines. In Proceedings of the 20th international symposium on High performance distributed computing (HPDC '11). ACM, New York, NY, USA, 171-182.

[9] Frachtenberg, E., Schwiegelshohn, U. 2007. Job Scheduling Strategies for Parallel Processing, 12th International Workshop, JSSPP 2006, Saint-Malo, France, June 26, 2006, Revised Selected Papers Springer

[10] 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), Volume: 6081/2010, Issue: LNCS 6081, Publisher: Springer, Pages: 13-31

[11] Kessler, C. 2004. A practical access to the theory of parallel algorithms. SIGCSE Bull. 36, 1 (March 2004), 397-401.

[12] Bessis, N., Sotiriadis, S., Cristea, V., Pop, F., Towards inter-cloud schedulers: Modelling Requirements for Enabling Meta-Scheduling in Inter-Clouds and Inter-Enterprises, Third International Conference on Intelligent Networking and Collaborative Systems (INCOS 2011) , Nov 30 - Dec 2 2011, Fukuoka, Japan.

[13] Bessis, N., Sotiriadis, S., Cristea, V., and Pop, F., Towards inter-cloud schedulers: Modelling Requirements for Enabling Meta-Scheduling in Inter-Clouds and Inter-Enterprises, Third International Conference on Intelligent Networking and Collaborative Systems (INCOS 2011), Nov 30 - Dec 2 2011, Fukuoka, Japan .

[14] Xhafa, F., and Abraham, A., Computational models and heuristic methods for Grid scheduling problems, Future Generation Computer Systems, Volume 26, Issue 4, April 2010, Pages 608-621, ISSN 0167-739X

[15] The AuverGrid team, http://www.auvergrid.fr/, Accessed in 16/11/2011