On the Evaluation of VM Provisioning Time in Cloud Platforms for Mission-Critical Infrastructures

Gabriella Carrozza, Luigi Battaglia, Vittorio Manetti
SESM s.c.a.r.l.
Via Circumvallazione Esterna - 80014
Giugliano in Campania (Naples), Italy
Email: gcarrozza, lbattaglia, vmanetti {@sesm.it}

Antonio Marotta, Roberto Canonico, Stefano Avallone
University of Naples Federico II
Dipartimento di Elettrica e Tecnologie dell'Informazione
Via Claudio, 21 - 80125 - Napoli, Italy
Email: antonio.marotta, roberto.canonico, stavallo {@unina.it}

***Abstract*—Cloud Computing has risen a great interest over the last years as it represents an enabling technology for flexible and ubiquitous access over the network to a set of shared computing resources. This work comes from an industrial experience aiming at exploiting the cloud potential for virtualizing complex infrastructures such as an entire Air Traffic Center (ATC), a clear example of complex SoS (System of Systems). The use of virtualization is convenient because industry can leverage in-house test-beds to perform distributed testing campaigns in pre-operational phases or to design new automatic fail-over mechanisms for fully distributed systems. In order to realize such mitigation and recovery techniques in the ATC field, indeed, a cloud platform is required to guarantee a low VM provisioning time with the objective of minimizing the service disruption. In this perspective, after having introduced the principal concepts and factors of the provisioning time, we propose a deep analysis and comparison for two different Infrastructure-as-a-Service (IaaS) platforms, namely OpenStack and OpenNebula (using KVM as hypervisor), that were selected through a preliminary scouting phase.**

I. Introduction

Cloud Computing has risen a great interest over the last years as it represents an enabling technology for flexible and ubiquitous access over the network to a set of shared computing resources. Cloud paradigm is known to be considered according to a service and a deployment model. In the first case the classification is based on the abstraction level of the resources the users are able to access over the cloud. In this work we only focus on the lowest level of abstraction, which is also referred to as Infrastructure-as-a-Service, in which clients are given the chance to access resources without being conscious of the underlying physical infrastructure. Instead, the discrimination based on the deployment mode is fundamentally derived by the level of exposure to the global Internet: there can be public, private or hybrid clouds. Despite all the advantages that come from the application of the cloud paradigm, there is some reluctance because of the difficulty in guaranteeing the quality-oriented requirements expressed by the client. This is due to the fact that performances are often dependent on the used configuration and on the existence of different concurrent workloads. Furthermore, because plenty of solutions exist, it is often hard to select one of the available platforms, since it is also difficult to identify the key indicators for benchmarking cloud-based services in a standard way. We started our work on Cloud Computing with the aim of exploiting its potential to build a virtualized environment for complex and mission-critical systems. Indeed, Air Traffic Control (ATC) systems are very demanding and software-intensive: they are safety critical, highly distributed and hard real-time. In the ATC field, ATC centers belonging to the same system are often deployed over different cities in a given country, either for fault tolerance purposes and remote connection needs at country level. For this reason, cloud can be considered as an enabling technology for setting up an extended enterprise private platform and connecting geographically distributed ATC centers for dependability purposes, e.g., by realizing a fail-over configuration among centers, in order to increase the overall system availability. The fail-over techniques we have in mind [1] rely on live migration which, in turn, is enabled by using a shared storage. In particular, we used a storage configuration in which both the image and the instance datastores are exported to the nodes belonging to the infrastructure through the Network File System [2] (NFS) protocol. In the design of back-up and recovery mechanisms for high performance, reliability and ultra-high availability demanding systems, a cloud platform which guarantees minimum impact on performance and service disruption is needed. Therefore we decided to draw a comparison between different open-source IaaS Cloud Computing platforms in terms of the *provisioning time* of Virtual Machines, i.e., the time needed by the cloud platform to create and activate a new VM. We also evaluated the stress of the physical nodes hosting the VMs in terms of CPU utilization and memory occupation.

The rest of the paper is structured as follows. In section II the related work is discussed, while in section III the motivations that led us to select OpenStack and OpenNebula are presented. In section IV the concept of provisioning time is explained, by also stressing out the factors it is affected by. In section V the deployment of a virtual machine in the two platforms is illustrated, whereas in section VI the experimental campaign is presented and the results of the evaluation are debated. In the last section some conclusions and the future work are drawn.

II. Related Work

Different works use or propose different models and benchmarking techniques to address the problem of evaluating Cloud

Computing IaaS platforms. Semploski *et al.* [3] provide a critical comparison between three open-source projects that offer Infrastructure-as-a-Service, namely OpenNebula, Eucalyptus [4] and Nimbus [5]. After having illustrated how they realize the spawning of a new VM and the differences between them, the authors present a raw level comparison according to the provided features. Anyway the paper only considers functional and qualitative characteristics as discriminating factors, such as customizability, security and network-related issues.

In [6] a process for implementing a custom benchmark, useful to understand the performance of various cloud IaaS offerings (in a pay-per-use model), is presented. The authors face the lack of precise indicators to assess the performance of the same VM running on different platforms and the inapplicability of hardware benchmarking. They show the results of the proposed benchmarking suite of applications by considering two instance types (a cheap and an expensive configuration) and some factors, such as the data-center location and the provided performance level both in a short-term and a long term utilization.

In [7] a performance comparison between Eucalyptus and OpenNebula is conducted. The workload consists of the software packages needed to run a web application very similar to Wikipedia [8] web service and a benchmarking application which emulates an online stock system. They attempted to compare the platforms in terms of:

- provisioning time of VMs;
- elapsed time of batch processing;
- throughput of web transactions;

by considering several configurations differing for the location of VMs images (local disk or NFS storage) and their allocation type (using dense or sparse files). The experiments prove that the best results for the three parameters are not always achieved with the same configuration. Anyway they consider a start-up time that also includes the booting phase of the VM, which we do not take into account; besides they do not show an evaluation of the single steps composing the provisioning time.

In [9], the authors argue that classical benchmarking techniques, like TPC-W, are not suitable for the characteristics of the cloud platforms, which lie in scalability, pay-per-use model and fault tolerance. For the purpose of addressing the lack of a standard benchmarking technique for Cloud Computing, they suggest their static metrics to assess the dynamism and the elasticity which are typical of the cloud paradigm. However no implementation of the proposed model is given.

Ming *et al.* [10] propose an analysis of the VM *start time*, which they consider as the time needed to find an allocation for the virtual appliance and to copy, boot and configure all the required resources. The authors compare the start time of three different cloud providers (Rackspace [11], Amazon EC2 [12] and Microsoft Azure [13]). The experiments aim at evaluating how the start time is influenced by different parameters, such as the date and time, the size of the image, the virtual hardware template and the data center location. For example, they demonstrate in their findings that the start time is linearly dependent with the OS size and it is also related to the instance type, while it is quite constant when requesting a pool of virtual resources.

SMICloud [14] is a framework based on the publicly available service measurement indexes for comparing and ranking cloud providers according to the service level agreed with the clients and the perceived QoS. The framework receives the client's requests along with the application deployment requirements. By monitoring the performance of the cloud services, it is able to compute the ranking factors for all the providers which are listed in a service catalogue. After describing the quantifiable metrics which are of interests for IaaS Cloud Computing, they solve the problem of ranking services by adopting the Analytic Hierarchical Process [15] technique, which is often applied in the context of Multiple Criteria Decision Making (MCDM) [16].

VMmark [17] is a benchmark to assess the stress load of the physical nodes' resources, such as RAM, CPU utilization and I/O load, by deploying a set of VMs on top of VMware [18] technology. Also Casazza *et al.* [19], in their vConsolidate project, suggest a benchmarking for evaluating the performance of different workloads consolidated in cloud hosting servers.

To the best of our knowledge, so far none of the works in the literature has proposed a comparison among open-source IaaS platforms with a deep analysis of the factors composing the provisioning time.

III. IaaS Platforms Under Evaluation

In the initial scouting phase we established a group of key features on which we based the selection of a restricted number of platforms to consider in the experimental campaign. The key requirements we took into account are:

- full open-source license;
- the support for KVM [20] technology, since we decided to use this hypervisor;
- the support for OpenvSwitch [21] technology for handling the creation of the virtual network interfaces;
- a strong community and then
  - bug reporting;
  - continuous sw maintenance;
  - updating process and new releases;
- fully downloadable packages without limitations;
- industrial support and existing experiences.

According to these features we are confident that OpenStack and OpenNebula are the open-source cloud projects that best respond to our requirements.

OpenStack is an open Cloud Computing platform, whose objective is to offer the IaaS paradigm through a collection of different services which can be deployed in a fully distributed and scalable fashion. It started as a project under the name of OpenStack Austin in 2010 and it has been characterized by different releases (the latest stable one is called Havana).

OpenStack offers a number of services which exchange messages through an asynchronous communication queue with a callback mechanism. Among them, we can mention:

- the authentication and authorization module (Keystone);
- the compute service which handles the life-cycle of the virtual machine from its creation till its deletion (Nova);
- the VM images repository (Glance);
- the volume service for creating and attaching new volumes to a running VM (Cinder);
- the storage service (Swift);
- the virtual networking module (Quantum);
- the web interface exposed to the end-users (Dashboard).

OpenNebula (ONE) was born as a research project in 2005 with the main purpose of designing and implementing an efficient and scalable management platform for virtual machines on large-scale distributed infrastructures. The ONE platform assumes that the underlying physical infrastructure is built up in line with the classical cluster-like architecture, where the basic components are: front-end, hosts and datastores. The front-end node is where the ONE installation has been accomplished: it is responsible for the execution of the centralized ONE services (monitoring & management, scheduling, web interface, and so on). Hosts are the hypervisor-enabled nodes providing the resources needed by the running VMs, while datastores are the storage areas holding the VM disk images.

IV. Performance Indicators: Provisioning and Scheduling Time

The Cloud Service Measurement Index Consortium (CSMIC) [22] has identified a set of business oriented key performance indicators (KPIs) that can be exploited in order to compare all cloud services (IaaS, PaaS, SaaS, etc.). The framework includes both critical business and technical features and it has a hierarchical structure: in the top level there are seven categories, each with a certain number of attributes. As argued in section I, in our investigation we do only focus on two different categories, namely agility and performance. Elasticity is among the attributes included in the former category and it is crucial to the performance analysis of cloud platforms: it can be interpreted as the ability of a service to adapt itself to the changes and the requirements issued by the applicant. It is also attached to the scalability of the service, that is a measure of the increase in its performance when additional resources are granted to the system. By having in mind the characteristics of the IaaS cloud paradigm and its application in the context of mission-critical infrastructures, we evaluated elasticity through the *provisioning interval*. In general the provisioning interval can be roughly described as the time needed to activate/deactivate a generic resource. More in details, the provisioning time can be viewed as a service response time: it is the time interval starting with the request for a specific service and ending with its availability. In our perspective, the service consists in the creation of a virtual machine in the cloud platform, according to a well-known hardware template. Thus, we can consider an average response time for the creation of the virtual appliance as follows:

$$\frac{1}{N}\sum_{k=1}^{N} T_k \tag{1}$$

where, considered N subsequent requests of the same VM, T_K stands for the time between the client's request and the instant in which the resource is considered in an *active* status by the monitoring process of the platform. This is a state in which the VM operating system was copied in the instances folder and its creation ended successfully, even if the VM is still in the booting phase, so it is not ready to serve the first user's request.

A. Analysis of the Provisioning Time

The time needed to deploy VM k can be divided in three different terms, as explained in the following formula:

$$T_k = T_k^r + T_k^s + T_k^e \tag{2}$$

where

1) T_k^r takes into account the issue of the creation request by using the APIs of the platforms and instant in which the command starts to be processed.
2) T_k^s is the time it takes for the platforms to select one of the available physical hosts with enough resources to host VM k. This time is dependent on the complexity of the scheduling algorithm implemented by the platform and on the number of the configured compute hosts. A general description of the scheduling algorithms used in the platforms is presented in sections V A-B.
3) T_k^e embodies the processing time including all the tasks necessary to spawn and configure the virtual appliance (sections V A-B). It can vary according to the type of storage which has been chosen for holding the VM images: all the IaaS platforms allow to use different kinds of storage solutions. It has been proved in [7] that different configurations, like for example local storage or NFS, can lead to diverse provisioning times. This factor is also influenced by the network configuration of the virtual appliance.

In order to obtain the provisioning and the scheduling times a Java-based application was developed and deployed on the controller node. Basically, the application exploits the APIs of the platform to perform the following tasks:

- constructing one of the virtual hardware flavors which were taken into consideration (the details of the flavors are described in section VI B), according to a binary string that represents all the configuration parameters;
- requesting a new VM with the selected service offering;
- obtaining the status of the deployed VM;
- computing the time between the issue of the creation request and the ready status response.

B. CPU and RAM Stress

In order to compare the performance of the considered cloud platforms, we monitored the CPU and RAM utilization of

the compute hosts, hosting the maximum number of VMs, each one executing the stress command [23]. As first step, we investigated on how many machines with the same virtual hardware template the user can instantiate on a single physical host, without over-provisioning CPU and RAM. The possibility to over-provision the available resources can be enabled or disabled in the configuration parameters of the platforms. We tried to reach a 100% CPU utilization on each VM, by balancing between user/system and I/O processes. The stress utility allows to specify the kind of stressing operation through some parameters, such as:

- cpu, which describes the number of forked processes executing a sqrt() command (user level);
- vm, the number of forked processes executing malloc()/free() commands (system level);
  - vm-bytes is the number of bytes, expressed in MB, which every vm worker has to allocate;
- hdd, the number of forked processes executing a write() command (I/O level);
  - vm-keep specifies to re-dirty memory, instead of freeing and reallocating it.

We used a combination of the three parameters which leads to a 0% time percentage in which CPU is idle and to a trade-off between user, system and I/O CPU utilization, by setting these parameters:

- cpu 1;
- vm 1 and vm-bytes 350M;
- hdd 500 and vm-keep.

V. Virtual Machine Deployment

A. OpenStack

OpenStack only gives the chance to create *ephemeral* virtual instances, which means that the VM image no longer exists after the user deletes it. As a consequence, it is possible to create multiple VMs from the same image. The only way to obtain persistent images is to spawn an image from a volume previously created through the block service (Cinder). When the user creates a new virtual machine from an image stored in the Glance repository for the first time, the deployment deals with these steps:

- request handling;
- virtual network configuration;
- image creation, which in turn is divided into the following sub-steps:
  1) the image is copied from the repository to a directory (called _base) of the VMs datastore;
  2) this image is then converted from its original format to raw;
  3) the image is copied again obtaining a different version for each requested flavor (we can refer to this as "flavor image");
  4) the image is resized using qemu-img resize according to the specifications of the requested flavor;
  5) the filesystem of the image is checked and resized to the specific filesystem;
  6) an instance disk is finally created through qemu-img create by specifying qcow2 as output format.

When the user needs a secondary (ephemeral) disk for the VM, the additional steps concern the creation of a disk which is formatted according to a particular filesystem and then attached to the VM. All the subsequent requests of VMs with the same image and flavor are not affected by steps from 1 to 5. Instead, when creating a VM with the same image but a different flavor, only steps 1 and 2 are skipped.
Prior to create the virtual machine, the scheduler needs to know where to allocate it. OpenStack uses nova-scheduler to select a host where to spawn the new VM. The cloud administrator can decide to deploy a specific type of scheduler by choosing among different policies, such as:

- filter (the default one), which defines a number of filters the compute nodes have to satisfy and computes a ranking score for each of them;
- chance, which chooses random one of the available hosts with enough resources;
- simple, which adopts a spread first policy, by selecting the least loaded node (it is no longer supported in OpenStack Grizzly and Havana).

B. OpenNebula

OpenNebula creates and handles two datastores, which stand for the storage areas holding the VM disks. The original VM images are registered in a datastore, which we will refer to as images datastore, whereas those related to the running VMs are stored into the so-called system datastore. OpenNebula does offer the possibility to create a persistent image: by creating a VM with a persistent disk, every modification of the instance is preserved even if it is destroyed. Obviously, only one VM at a time can be created starting from a persistent image. Non-persistent images, instead, are deleted when the VM is canceled. OpenNebula entrusts part of the VM creation to a centralized Transfer Manager (TM) which is pre-configured with various scripts, among which one of them is executed according to the chosen storage model (NFS shared in our case). This component is in charge of handling the transfer of an image between a source (the front-end) and a destination (a compute host). In particular the actions executed by the TM when creating a non-persistent/persistent image are:

- clone (non-persistent), which creates an exact copy of a disk from the images datastore to the system datastore of the target host;
- ln (persistent), which creates a symbolic link in the target system datastore that points to the source image.

When a deployment request for a non-persistent VM arrives to the TM, a clone script is executed: it copies the selected image it in the system datastore of the target host. On the other hand, if the user requests a VM with a persistent image, a ln script is executed and a VM root disk is created as symbolic link pointing to the original image.
In both cases, after the VM disk is created, the virtual guest is deployed by using virsh. The second step deals with the network configuration which basically consists in:

- adding a new port to the virtual switch;
- defining a treatment for the packets that pass through this port by establishing two flows in the virtual switch.

As argued before, the other factor that can influence the provisioning time is the scheduling algorithm. In OpenNebula lots of pre-defined scheduling policies are available, such as packing, striping and load-aware, in addition to the opportunity of specifying customized rank expressions. In our investigation we adopted the so-called packing policy, that allows to use the whole available CPU and RAM of each physical host. The platform also defines a configurable time value (scheduling interval) between two scheduling actions, fixed by default to 30 seconds. We reduced this parameter up to 1s in order to minimize the whole idle time during consecutive allocations of VMs.

VI. Experimental Campaign

A. Hardware and Software Configuration

The whole experimental test-bed has been set up on three clustered Dell PowerEdge M610 Blade Servers, each of them equipped with:

- 2 * QUAD-CORE 2.50 GHz Intel Xeon E5420 (totally eight cores up to 20 GHz);
- 64 bit CentOS 6.4 operating system;
- 8 GB of RAM;
- 4 Gigabit Network Interface Cards (NICs) (even though only two NICs have been used for our tests).

Those nodes are interconnected by means of Gigabit Switches, namely DELL PowerConnect M6220, which are themselves linked to the Storage Area Network (SAN) module by fiber optic cables. As a consequence, the underlying physical network infrastructure provides 1Gbps as theoretical throughput.
Storage for both images and instances folders is implemented by using an OpenFiler (version 2.99.1) server which exposes two different partitions through NFS protocol (version 4.0). The adoption of a shared filesystem-based datastore allows to enable the live-migration, but it can also become a bottleneck in the infrastructure and degrade VMs performance if the virtualized services perform disk-intensive workloads.
As explained before, the experimental infrastructure consists of three clustered nodes, of which one plays the role of the front-end (or controller in the OpenStack terminology), while the others act as hosts (or compute nodes). All the nodes are interconnected by means of gigabit LAN interfaces. We used a couple of OpenvSwitch (version 1.11) bridge interfaces, suitably configured to separate the exchange of data among VMs from the traffic related to the external network.
With regards to OpenStack, we used Folsom version with a controller on which Nova, Keystone, Glance, Cinder, Dashboard and Quantum packages with OpenvSwitch plug-in were installed. The compute nodes only have Nova software and the OpenvSwitch plug-in. Concerning OpenNebula software configuration, instead, we used version 4.2.

B. Experimental Parameters

Before setting up our experimental campaign, we initially tuned the parameters involved in the creation of a virtual machine, in order to evaluate their influence on the provisioning and scheduling time measures. This allowed us to neglect the variability of some parameters and to set a specific combination of values. In particular we considered:

- the server load, as concerns the number of VMs already instantiated on the host;
- over-provisioning, that indicates if the amount of RAM and CPU physical resources can be virtually exceeded;
- the virtual machine image:
  1) type (qcow2 and raw);
  2) size;
  3) persistence;
- the scheduling algorithm;
- the instance type, also referred to as "flavor" or virtual machine template, through which the user can assign:
  1) the number of vCPUs (expressed as CPU slots);
  2) the amount of RAM (expressed in MB);
  3) an optional ephemeral secondary storage;
  4) the number of virtual network interfaces.

Our first experiments showed no appreciable variation in the results due to the server load, the over-provisioning option and the VM flavor, so we decided to consider them as fixed. The picked values are shown in table I. Since we aimed at obtaining the relevance of the basic steps of the elaboration time (T_k^e) on the provisioning interval, we did not focus on the variability of the time measures due to different sizes and formats of the VM image. The dependence of the provisioning time on these parameters was well studied in [7] and [10]. We excluded the variability analysis of the copy of the image, because this case scenario is of poor interest to us: the provisioning time, indeed, is obviously affected by the copy of the image, which is in turn dependent on the network load of the switch connecting the clustered nodes and the NFS storage server. For this reason in the experimental campaign, we only consider one case that involves the copy of the image, by fixing the size and the format.

C. Experimental Scenarios

As explained in the previous sections, the two platforms are intrinsically different when spawning a new virtual machine. When a user requests a VM with a non-persistent image in OpenNebula, the deployment basically consists in copying the original image into the target system datastore and creating the VM by using virsh. On the other hand, the first time a user deploys a VM in OpenStack, a full copy of the image stored in the Glance back-end is needed; then the image is converted and resized according to the flavor, created using qemu-img and spawned. In this perspective the time it takes to have a running VM can be comparable with the deployment of a non-persistent image in OpenNebula.
Instead, if the VM is deployed starting from a persistent image in OpenNebula, a symbolic link to the original image

Table I: Fixed Parameters

Host Load	Scheduling Algorithm	OverProvisioning	VM Instance Type		
			vCPU	RAM(MB)	vNIC
No VM	Fill First Scheduler	Disabled	1	512	1

Table II: Variable Parameter

Secondary Storage
Yes/No

Table III: VM Allocation

VM Allocation			
OpenNebula		OpenStack	
Persistent	Non-Persistent	1^{st} Allocation	2^{nd} Allocation

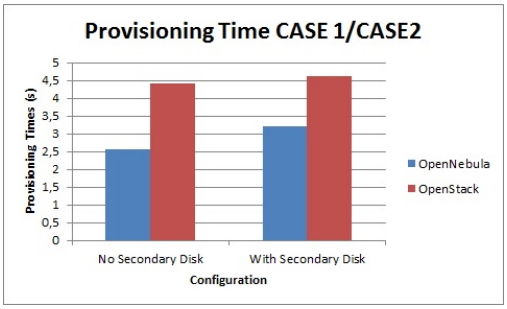
is generated (without copying it). In OpenStack, after the first creation of a VM, all the subsequent requests for the same image does not involve its copy. Here is why the provisioning time in the latter case can be compared to the one achieved with the deployment of a persistent image in OpenNebula. By having this analysis in mind, we took into account three different cases in our investigation:

1) case 1 involves the deployment of a persistent qcow2 image with a virtual disk size of 1.7 GB (effective size of 8.0 GB) in OpenNebula without a secondary disk and the allocation subsequent to the first one of the same VM type in OpenStack;
2) case 2 differs from case 1 because of adding a secondary ephemeral disk to the instance;
3) case 3 deals with the creation of a non-persistent image without a secondary disk in OpenNebula and the first allocation of such a VM in OpenStack.

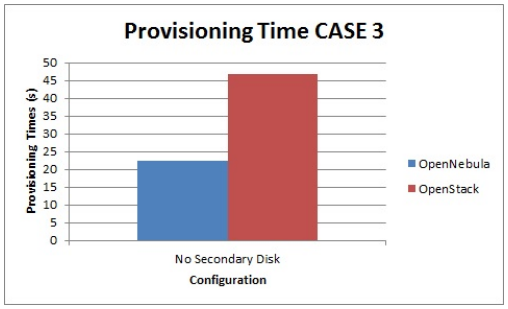
D. Results and Analysis

1) Provisioning Times: as remarked by figure 1(a), the creation of an additional ephemeral disk does not have a significant impact on the provisioning time in both the platforms. Therefore, in the analysis of the remaining case, we neglected the presence of the secondary disk. In the first considered case (the deployment of a VM with a persistent image in OpenNebula and the second creation of the same VM in OpenStack) OpenNebula outperforms OpenStack: for the latter, the mean provisioning time is about 42% longer (1,85 seconds). By default, OpenStack gives the chance of injecting metadata and an asymmetric key (for SSH password-less access) into the VM: anyway in our analysis we avoided this possibility and we used no firewall for the VMs. In figure 1(c) the variability of the obtained measures is illustrated: OpenStack has more stable results compared to OpenNebula, and this is due to a greater impact of the hosts monitoring process. The deep analysis of the logs of the platforms allowed us to obtain the principal contributions of the provisioning time. As shown in figure 1(e), we were able to identify four factors that are: the time to issue the VM request, the communication overhead in the platform, the scheduling time and the creation of the VM. In both the platforms the creation of the VM accounts for about 80% of the entire time and the presence of different modules in OpenStack also allowed to recognize a 12% additional overhead, which is related to the communication with the scheduler and the compute module. Nevertheless, the time it takes for OpenStack to spawn a new VM is longer with respect to OpenNebula and it is the reason why we investigated on the influence in percentage terms of the VM deployment tasks (figure 1(f)). OpenStack requires about 39% of the whole time for handling the VM creation and interacting with the other services, such as Glance and Quantum to manage the image and to get network-related information respectively. The management of the virtual networking (basically OpenvSwitch and tun/tap drivers) and the virtualization drivers accounts for about 30%, just as the creation of the VM image. OpenNebula, on the contrary, shows a simpler internal architecture and employs fewer commands for the management of the network and virtualization drivers. Anyway it should be observed that we were not able to estimate the additional overhead in the OpenNebula platform, because of the smaller amount of data which can be deduced from the logs. The greatest impact on the provisioning time is related to the interaction with KVM and OpenvSwitch drivers and the execution of the VM deployment script.

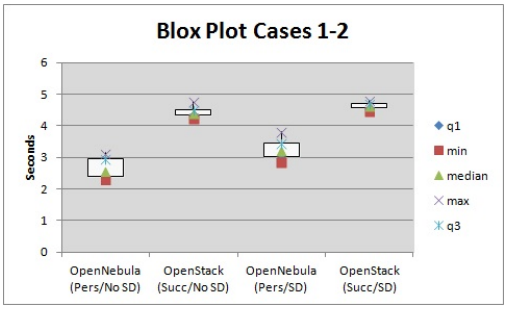
Figures 1(b), 1(d) and 1(g) show the provisioning times of the third case, which deals with the creation of a non-persistent virtual appliance in OpenNebula and the first allocation of the same VM in OpenStack. The time needed for the creation of the instance are prominently different for the two platforms: the time it takes for OpenStack almost doubles the one achieved in OpenNebula (1(h)). This time takes into account the copy of the whole image, which is executed between two directories, both exported via NFS protocol. Anyway the time required by OpenStack is double because (as stated in section V.A) the image, which is qcow2, is converted to raw and this conversion demands for about 20 seconds additional time.



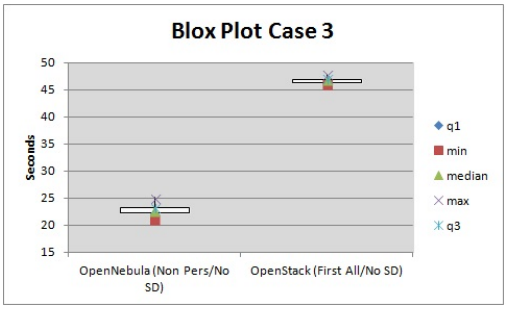
(a) Cases I-II



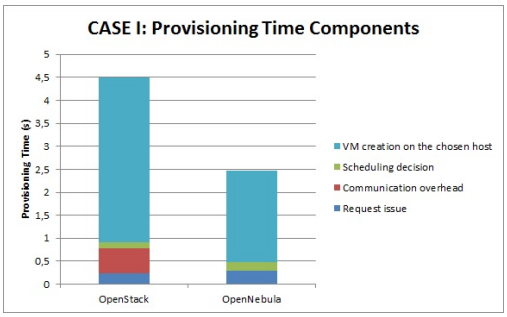
(b) Case III



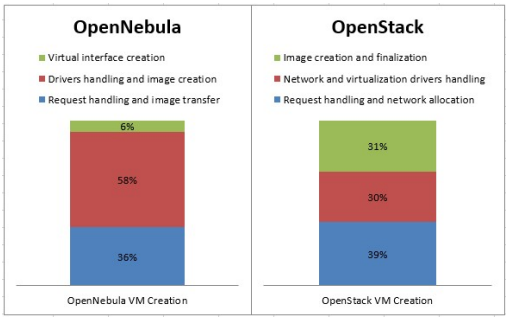
(c) Cases I-II



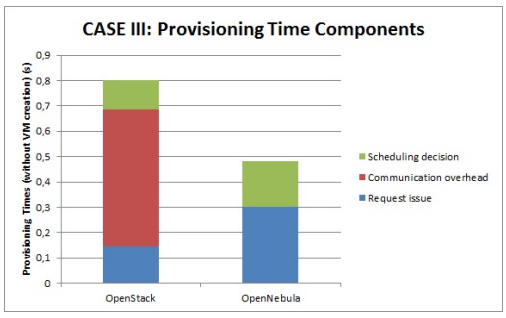
(d) Case III



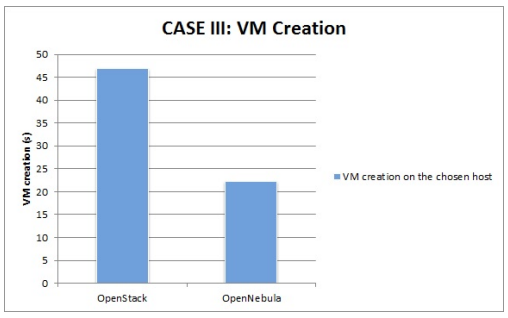
(e) Provisioning Components CASE I



(f) VM Creation CASE I



(g) Provisioning Components CASE III



(h) VM Creation CASE III

Figure 1: Provisioning Times Results

2) Scheduling Times: in figure 2 the mean values for the scheduling times in both the platforms are presented: the scheduling algorithm implemented by OpenNebula is affected by slightly longer times with respect to OpenStack. Nevertheless we expected these results by comparing the complexity of the algorithms which are implemented by the two platforms. The OpenStack scheduling algorithm consists in two stages: the first one has the goal of selecting hosts according to the filters specified in the Nova configuration file. We configured RAM and CPU filters with an "allocation_ratio" of 1.0, in order to disable over-provisioning. Afterwards, all the hosts that did not pass the filtering phase are excluded from the final

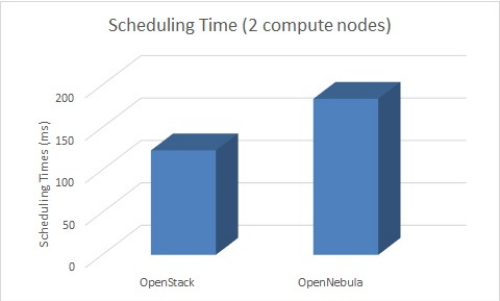
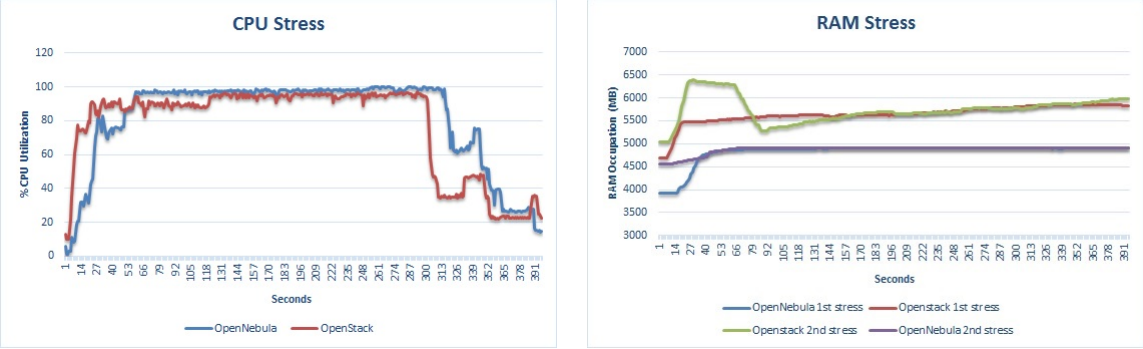


Figure 2: Scheduling Times Results



(a) CPU Stress

(b) RAM Stress

Figure 3: Stress Results

decision. In the second phase there is a weighting process of all the filtered hosts according to the specified cost functions. The default one returns the amount of free RAM of the host and has a weight, which represents the behaviour of the algorithm (fill-first 1.0 or spread-first -1.0). So for each cost function, every host obtains a score and the final ranking is calculated as follows:

$$rank_i = \sum_j w_j s_{ij} \quad \forall i \tag{3}$$

where s_{ij} is the score of node i related to cost function j and w_j is the weight of the j-th function. Besides the default cost function, we also included another one for CPU slots usage, in order to choose a target with the largest combined utilization of CPU and RAM.
OpenNebula uses a match making scheduler which implements the rank scheduling policy. The allocation decision is taken after these steps:

- VM filtering, in which the VMs that request more storage than the available amount are filtered out and assume a pending state;
- host filtering, during which the servers that do not meet VM requirements or do not have enough resources (CPU and RAM) are filtered;
- hosts ranking, that is the evaluation of the remaining hosts, using a ranking score policy obtained through the information periodically collected by the monitor driver.

Regarding the ranking factors, OpenNebula allows to configure the policy by using a predefined or a custom one. We chose the packing policy which has the objective of minimizing VMs fragmentation and the number of active nodes, by selecting the node with more VMs running on it.
Anyway it should be pointed out that the scheduling time accounts on the provisioning interval for about 2,78% in OpenStack and about 7,15% in OpenNebula.

3) Stress: the experiments aimed at evaluating the impact in terms of CPU and RAM utilization when stressing the two platforms. We deployed the maximum number of VMs and we executed the linux stress command on each of them, by applying the parameters explained in section VI A and a duration interval set to 300 seconds. In figure 3(a), it can be noticed that, during almost all the stress time, the level of CPU utilization remains quite stable for both the platforms, even if the mean percentage achieved by OpenNebula is 97,45%, while for OpenStack it is slightly inferior, by settling on 93,2%. Figure 3(b) exhibits, instead, the occupation of RAM (in MB) after the allocation of 8 virtual machines and during the stress test. The graph compares two (almost consecutive) stress tests: in the OpenNebula case, during the stress, the curves mostly overlap and they are characterized by a mean value of 4895 MB. OpenStack experiments shows a higher RAM occupation (5694 MB as mean value) and the second test exhibits a 10% increment in the pick RAM occupation value with respect to the first experiment.

VII. Conclusions and Future Work

In this work we propose a comparison of a restricted number of IaaS open-source projects, with the main goal of selecting the one that properly meets the distinctive features that are typical of virtualized mission-critical infrastructures. When dealing with the virtualization of this kind of infrastructures, characterized by a great complexity and fault tolerance needs, a platform that assures a low provisioning time is preferable. This is obviously a leading factor when a system, composed of a consistent number of nodes greatly coupled among them, has to be virtualized. For this reason, bearing in mind these requirements, we planned and implemented an experimental campaign, aiming at performing a deep analysis of the provisioning time. After a raw-level evaluation of the available IaaS open-source projects, we picked OpenNebula and OpenStack and we set up two identical test-beds with the same hardware configuration in order to achieve our objectives. To be sure to guarantee the exact reproducibility of the proposed experiments, we tried to deeply describe all the parameters taken into account to perform our campaign. Our efforts were directed at evaluating the impacts of the different key factors of the provisioning time, by pointing out their relevance in each of the platforms. In this perspective, the results can also be read in order to find out the most impacting contribution to the provisioning time and to realize if there is room to improve the creation of a VM. Our evaluation showed that, although the great success of OpenStack, OpenNebula offers slightly more suitable performances when creating a virtual machine in some cases. The next steps consist in assessing and also proposing new metrics and new benchmarking techniques which can be applied with the objective of extending the proposed experimental campaign.

References

[1] G. Carrozza, V. Manetti, A. Marotta, R. Canonico, and S. Avallone, "Exploiting sdn approach to tackle cloud computing security issues in the atc scenario," in *Dependable Computing*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, vol. 7869, pp. 54–60.

[2] G. Arnold, "Internet protocol implementation experiences in pc-nfs," in *Proceedings of the ACM Workshop on Frontiers in Computer Communications Technology*, ser. SIGCOMM '87. New York, NY, USA: ACM, 1988, pp. 8–14.

[3] P. Sempolinski and D. Thain, "A comparison and critique of eucalyptus, opennebula and nimbus," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 417–426.

[4] Eucaliptus web site. [Online]. Available: http://www.eucalyptus.com/

[5] Nimbus web site. [Online]. Available: http://www.nimbusproject.org/

[6] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann, "What are you paying for? performance benchmarking for infrastructure-as-a-service offerings," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 2011, pp. 484–491.

[7] Y. Ueda and T. Nakatani, "Performance variations of two open-source cloud platforms," in *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, 2010, pp. 1–10.

[8] Wikipedia web site. [Online]. Available: http://en.wikipedia.org/wiki/Main_Page

[9] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, "How is the weather tomorrow?: towards a benchmark for the cloud," in *Proceedings of the Second International Workshop on Testing Database Systems*, ser. DBTest '09. New York, NY, USA: ACM, 2009, pp. 9:1–9:6.

[10] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012, pp. 423–430.

[11] Rackspace web site. [Online]. Available: http://www.rackspace.com/

[12] Amazon ec2 web site. [Online]. Available: http://aws.amazon.com/ec2/

[13] Microsoft azure web site. [Online]. Available: http://www.windowsazure.com/

[14] S. Garg, S. Versteeg, and R. Buyya, "Smicloud: A framework for comparing and ranking cloud services," in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, 2011, pp. 210–218.

[15] E. H. Forman. and S. I. Gass, "The analytic hierarchy process–an exposition," *Oper. Res.*, vol. 49, no. 4, pp. 469–486, Jul. 2001.

[16] U. e Habiba and S. Asghar, "A survey on multi-criteria decision making approaches," in *Emerging Technologies, 2009. ICET 2009. International Conference on*, 2009, pp. 321–325.

[17] P. S. L. R. E. Z. V. Makhija, B. Herndon and J. Anderson, "VMmark: A scalable benchmark for virtualized systems," VMware Inc, CA, Tech. Rep., September 2006.

[18] Wmware virtualization web site. [Online]. Available: http://www.vmware.com/

[19] J. P. Casazza, M. Greenfield, and K. Shi, "Redefining server performance characterization for virtualization benchmarking," *Intel Technology Journal*, vol. 10, no. 3, pp. 243–251, Aug. 2006.

[20] A. Kivity, "kvm: the Linux virtual machine monitor," in *OLS '07: The 2007 Ottawa Linux Symposium*, Jul. 2007, pp. 225–230.

[21] Openvswitch web-site. [Online]. Available: http://openvswitch.org/

[22] The cloud service measurement index consortium. [Online]. Available: http://csmic.org/

[23] Stress utility repository for centos. [Online]. Available: http://openvswitch.org/

[24] Openstack web site. [Online]. Available: http://www.openstack.org/

[25] Opennebula web site. [Online]. Available: http://opennebula.org/about:about