



Dynamic and elastic monitoring of VMs in cloud environment

Younes Khair¹ · Abdeslem Dennai¹ · Youssef Elmir¹

Accepted: 19 May 2022 / Published online: 17 June 2022

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract

Virtualization of computer systems has made it possible to deliver infrastructure as a service, deploying multiple virtual machines (VMs) on a single physical server to share available resources such as CPU, memory and cache, I/O. However, virtual machines (VMs) in the cloud can have varying workloads over time, which can lead to performance degradation and wasted idle resources. In view of the importance of this challenge, many research works have focused on their solutions to control and monitor these virtual machines in a dynamic, permanent and elastic manner. In this context, this article introduces the VMs-Monitor component, a software solution that acts as an extension of the OpenNebula open source cloud platform. Such a component must make it possible to monitor the consumption of VM-CPU and VM-Memory resources of running virtual machines and to plan instantiation actions (allocation, suspension, resumption) according to the different metrics and by retrieving information about the resources available on the hosts. The model evaluation showed performance gains as the results indicated that the host provides the CPU and memory capacity to the VMs based on the continuous allocation and suspension of the VMs, which has been balanced during the experiment.

Keywords Cloud IaaS · Virtualization · Virtual machine monitoring · OpenNebula

✉ Younes Khair
ynss.khair@univ-bechar.dz

Abdeslem Dennai
dennai.abdeslam@univ-bechar.dz

Youssef Elmir
elmir.youssef@univ-bechar.dz

¹ SGRE Laboratory, Department of Computer Science, University of Tahri Mohammed, Bechar, Algeria

1 Introduction

Faced with the cost of local managing of the computing infrastructures, many companies have decided to have it managed by external providers. [1] The latter, known as IaaS (Infrastructure as a Service), makes resources available to companies in the form of virtual machines (VM—Virtual Machine), where most computing infrastructures consist of virtual systems. which facilitates maintenance, migration, scaling and different levels of abstraction for those who use them [2]. Thus, companies only use a limited number of virtual machines capable of satisfying their needs. This contributes to the efficient use of resources and reduced IT infrastructure costs for cloud customers.

Cloud computing facilities are often supported by virtualization technologies that were originally defined by Popek and Goldberg [3] as efficient, isolated duplicates of a real machine, which in turn enable the cloud to acquire or release multiple virtual machine virtual machines on the same physical machine and in such a way that the loss of any component of the system does not cause the system to fail [4]. As this ability to customize is to achieve the efficient resource usage of the available physical resources, it leads to maintaining the required performance levels for the service provider. Many research works have focused on this problem indicating that the main reason for the inefficient resource usage is due to the varying workloads in the running applications on virtual machines; the latter's survival in an overloaded or underused state leads to insufficient resources being obtained [5]. This in turn affects the quality of service (QoS) and results in financial losses for the service provider's operator. Considering this background, monitoring virtual machines appears as a key point when booting on the host, including resource consumption at the platform or infrastructure layer level. As a result, the monitoring mechanism is indispensable to improve the quality of virtualization in the cloud environment [6, 7].

This article presents an adaptive monitoring model called VMs-Monitor to monitor and control virtual machines instances in terms of consuming virtualized resources, including CPU and memory consumption. VMs-Monitor acts as an extension of the OpenNebula open source cloud platform. The VMs-Monitor regularly and dynamically monitors the virtual machines running and plans actions for instantiation (allocate, suspend, resume) according to the metrics of the resources available on the OpenNebula platform; each instance of a running virtual machine requires an average calculation of the use of appropriate resources (CPU, memory, maximum/minimum threshold). As a result, when the VM instance exceeds the maximum resource usage, a new instance must be created. In return, if the minimum threshold is reached, the VM instance will be suspended. In other words, the objective is to maintain the use of resources of any active VM instance between these two thresholds. In addition, the status recovery of the Host is necessary to meet the requirements of the instantiation of VMs. The process defines a saturation threshold based on CPU usage status and memory for each Host.

The remainder of this article will first discuss related studies and summarize the tendency of cloud computing technologies and virtualization in Sect. 2. In Sect. 3, the proposed component is presented, revealing how the mentioned contributions are

developed. Section 4 shows the evaluation methods and a discussion of the results. Finally, the final section presents the conclusions and future work.

2 Background and related work

2.1 Anatomy of cloud computing

2.1.1 Virtualization

Virtualization technology is used to allow the presence of multiple virtual machines on the same physical machine [1, 8, 9]. Also is considered one of the most important components of cloud computing. Due to its allocated capacity, which involves optimizing the investment of available physical resources.

The concept of virtualization can be modified according to any component of the computer system [10, 11]. However, virtualization makes it possible to make a logical separation between, on the one hand, the resources providing the service and, on the other hand, the service itself. Otherwise, it is represented by a layer of abstraction and isolation between physical resources and applications in an independent and self-sufficient way [12].

Performance, efficient use of shared resources, performance, scalability, reliability/availability, flexibility, etc., are just a few of the advantages of virtualization [13]. Virtualization is the essential element of cloud Computing. It offers abstraction, isolation, encapsulation, and consolidation of applications and servers [14].

In hardware resource virtualization, the component that provides abstraction and isolation is called Virtual Machine Monitor (VMM) or Hypervisor. Figure 1 shows the basic architecture of a node in a cloud environment [16]. VMM is a software layer between hardware and operating systems that divides the hardware layer from the application (software) layer into an abstraction layer, allowing multiple VM instances to be run on a single physical machine. Systems running above this layer access the hardware layer as if they were running natively on hardware [3].

A virtual machine is required to set up a basic cloud service. When a significant number of virtual machines are produced using virtualization technology, managing

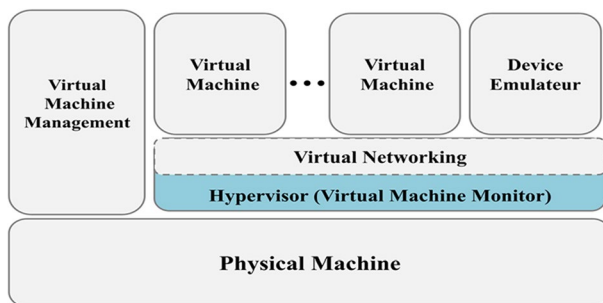


Fig. 1 Basic architecture of a node [15]

them using the native instructions becomes extremely difficult, necessitating the use of a virtual machine management platform [17]. The virtual management platform typically includes a virtual machine for allocating, suspend, delete, and performing a migration in real time [18]. A variety of popular open-source virtualization management platforms can be used. In addition, with a more user-friendly network interface, it is easier to monitor a large number of virtual state machines, and it is easier to manage account permissions, thus providing a user-friendly virtual machine creation process, which has many advantages [19]. Figure 2 shows the layers of Virtual Machine Manager. Through these layers, physical and virtual resources are grouped into virtual resources accessible through virtual machines. Therefore, these layers: virtual infrastructure manager (VIM) and cloud infrastructure manager (CIM) and hypervisors, constitute the group of basic tools for the deployment of services cloud IaaS from physical [20].

2.1.2 OpenNebula

Today, different platforms support the management of data centers which provide “infrastructure as a service.” Therefore, require technical expertise to create a virtual machine, as well as understanding of the CPU and memory requirements for running an application and which operating system to install [21].

OpenNebula is a cloud infrastructure management platform that is completely open source [4]. It is designed to deploy and manage complex and heterogeneous cloud infrastructure efficiently. This platform manages the virtual resources of public and hybrid clouds. It presents a layered architecture that allows the orchestration and management of storage, network, and virtualization technologies to allow the dynamic deployment and placement of services in a virtualized VM hardware environment [20].

The OpenNebula architecture, illustrated in Fig. 3, consists of several components, specialized in different aspects of virtual infrastructure management [4].

The OpenNebula solution adopts strictly layered and modular architecture, which can be integrated into different environments and different virtualization technologies

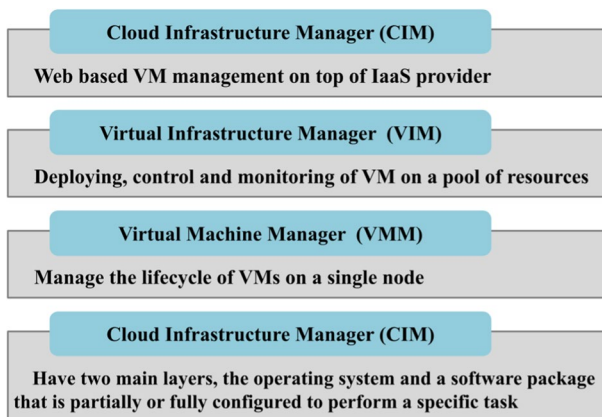


Fig. 2 VM management layers of a cloud infrastructure platform

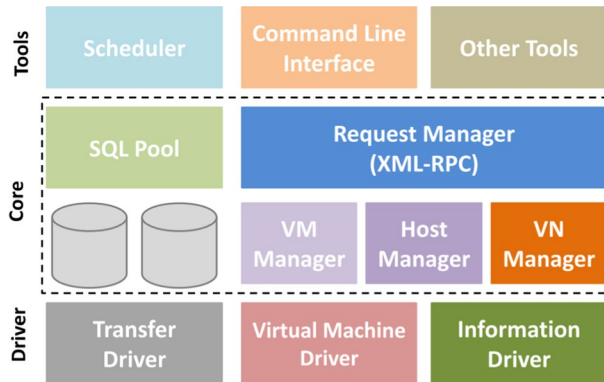


Fig. 3 OpenNebula-architecture [4]

(such as KVM, Xen, VMware, and VirtualBox) [22]. It organizes the IaaS in a cluster centered around the Front-end node and multiple back-end nodes, Front-end node and back-end nodes communicate via SSH. It also provides at least one physical network link the frontal node and other runtime nodes. OpenNebula is divided into the following layers: tools, cores, and drivers, as shown in Fig. 3. The tool layer contains architecture management tools. It consists of the CLI (command line interface) command-line interface. These components communicate via a set of APIs based on the XML-RPC interface [22].

2.1.3 Monitoring in the cloud computing

Resource management is one of the most difficult considerations of the Infrastructure-as-a-Service architecture. Indeed, the cloud environment is continuously expanding, making infrastructure management more difficult [1].

In general, cloud monitoring solutions are necessary functions for the automated scaling of services on both the provider and client sides [23]. On the one hand, it is a key technique for the monitoring, analyzing and management of physical and virtual computer resources, on the other hand, continue to assess and measure the performance, reliability and energy consumption of the infrastructure or applications [24].

Monitoring is obviously instrumental for all activities covered by the role of the cloud, as monitoring functions are facilitated and integrated in an adaptive way into the cloud architecture. More generally, cloud computing involves many activities for which monitoring is a necessary task allows the users and providers to measure workload, monitor the performance of their physical and virtual machines and hence monitor the quality of service offered by the provider. This is then an important element for the monitoring of Service Level Agreements (delay, availability), which are contractually guaranteed upon by the user and provider [25].

The cloud monitoring system depicted in Fig. 4 is composed of three parts: 'Data,' 'Processing,' and 'Reporting.' These layers are considered to be the location of the monitoring system agents, who interact with the cloud stack at different levels [5].

The data layer in this part, the monitoring agent is responsible for collecting information and performance indicators of physical and virtual machines [23]. This layer provides a set of expandable adapters to integrate heterogeneous data coming from different sources. These different adapters are distributed across the cloud infrastructure and all layers of the cloud stack.

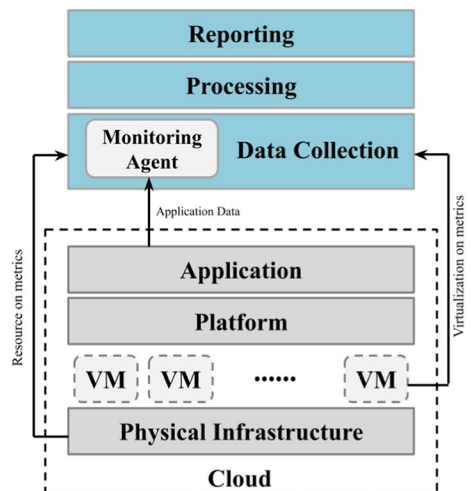
The processing layer describes queries on data contained by the data layer [26]. This one processes, maintains, and reports on all of the data collected by the data layer.

The reporting layer is concise to help understand the functioning and performances of services Cloud, including both the infrastructure report and the application report; the first focuses on the use and performance of physical and virtual resources in the cloud. The second is for the usage of resources by each application to ameliorate the exploitation and use of resources.

2.2 Related works

The emergence of cloud computing based on virtualization technologies offers huge opportunities to host virtual resources at low cost without the need to own an infrastructure [2]. Virtualization technologies allow users to acquire, configure, and be billed as they are used. However, cloud data centers mostly comprise of heterogeneous commodity servers hosting multiple virtual machines (VMs) with varying potential specifications and fluctuating resource usages, which can lead to unbalanced resource utilization within the servers, and to performance degradation and Service Level Agreement (SLA) violations [5]. To achieve effective resource management, these challenges need to be addressed and solved by including various interrelated operations such as task assignment, VM migration, resource prediction, VM allocation and by using management strategies. Load balancing, which turned out to be an NP-hard problem [24, 27].

Fig. 4 Components of the cloud monitoring system



From multiple perspectives, this work identifies challenges and analyzes algorithms for elastic resource management in a cloud environment, specifically focusing on dynamic and elastic monitoring of virtual machines in a cloud environment.

In the literature review [5, 8, 14, 24, 25, 28–30], many research efforts have been introduced to monitor cloud environments. These researches concentrate on monitoring the physical and virtual resources of cloud infrastructures. The majority of monitoring solutions focus on low-level metrics, such as CPU consumption, memory consumption, or network bandwidth [31].

Monitoring the system resources consumed by virtual machines on a physical server is a key element in ensuring the proper operation of an IaaS platform. Typical metrics are CPU or memory occupancy, disk and network access, or latency. There are many tools to collect these metrics [32].

The problem that now arises and more precisely the cloud environments have specific needs in terms of dynamism and elasticity for efficient monitoring with regard to Quality of Service (QoS), which remains a challenge [30].

Monitoring VMs instances become an opportunity for researchers to find solutions to this problem using an conventional monitoring system located within a virtual machine [29]. This monitoring is restricted to those resources that the virtual machine can see only. Chen, Chien-Chih, and al. in their work [5] implemented dynamic resource allocation and energy saving algorithms. And others, the author creates an infrastructure architecture based on cloud software OpenStack to implement these two algorithms with live virtual machine migration. They utilize the power distribution unit (PDU) to monitor power consumption and monitor system status. The performance of the two proposed algorithms is verified by the PDU power consumption records [33]. Saswade, Nikhil, Vinayak Bharadi, and Yogesh Zanzane [28] proposed implementing cloud infrastructure to monitor the status of virtual machines in real time as well as their energy consumption (CPU and memory use) in a cloud environment. In [28] the proposed solution is based on a monitoring software that is implemented as a multi-platform library written in the python programming language. This program's main purpose is to capture and retrieve CPU and memory information about currently running processes. It saves and integrates this monitoring data into a database and then displays it on this cloud infrastructure as empirical support for the user.

Wailly, Aurélien, and Pascal Legouge [14], in their study, suggested a method for monitoring the security of a virtual machine in a virtual computing environment. The method includes receiving automatic instructions related to an interruption in the operating system after an event in the virtual machine. The hypervisor uses the hardware resources of the host system to execute the instructions and send the data flow to the operating system, including the execution result. The hypervisor receives the instructions of at least one machine from the operating system associated with an interrupt in the operating system [34]. On the other hand, the sent data stream (the data stream is sent as a result of the execution) is created in a second stream and analyzed by a security agent working for an entity separate from the virtual machine.

The main objective of cloud automatization is to monitor the use of resources in the cloud environment properly [31]. Yang, Chao-Tung, and Tsung-Yueh Wan [25] introduce a mechanism that has been developed to alert the user when the system utilizes cloud resources and virtual machines (instances). This mechanism also helps

companies to keep a close watch on the security and performance of their essential apps hosted on AWS and Azure. As a result, the user must configure his working environment, which is hosted on the cloud and where the user's desired behaviors are automatically monitored. A graph depicting the instance data points produced by the cloud service client is shown to the user via the user interface.

To summarize the various works cited above, a comparative study is conducted to show the existing gap in the state of the art on resource management. Table 1 shows this compilation.

Based on this architecture [24], the Component communicates with OpenNebula, using (OCA) OpenNebula Cloud API that encapsulates access to OpenNebula's XML-RPC API and implicitly associated features [21]. To perform the communication, methods are chosen to access and define the OpenNebula XML-RPC address. This integration enables OpenNebula to execute queries for dynamic and elastic virtual machine monitoring [5]. The commitment to provide high-quality service to customers leads to the need to face the compromise of energy performance [6]. The authors discuss in [35] an energy-efficient cloud computing architecture framework based on energy-efficient resource allocation and scheduling algorithms. These methods focus on dynamic virtual machine consolidation, in which only one host is chosen to hibernate, based on the lowest CPU utilization or the least power divided by the number of virtual machines it hosts. According to the results, the energy-efficient cloud computing paradigm offers enormous potential for improving energy efficiency under a variety of dynamic workloads while also providing considerable cost savings.

To reach a consistently active management target, resource allocation must be adjusted to the external load on the cloud. Considering that the cloud's load varies over time. This means that performance management's task is to manage the process of allocating resources and dynamically adjusting them [36]. In this work [8], Wuhib and al. provide a prototype implementation for load balancing, service differentiation, and energy efficiency under variable load. On the one hand, they demonstrate a cloud management architecture based on OpenStack that allows for dynamic resource allocation for interactive and computationally intensive applications. On the other hand, this work involves the construction of scalable modular components as an extension of the OpenStack cloud platform, providing for the integration of additional management goals.

Several technical gaps have been identified in the present works, particularly in terms of dynamic and elastic virtual machine monitoring. Ensure elasticity by managing the allocation and release of resources, deployment of VMs based on current resource utilization, and maintain effective consolidation by managing the full life-cycle of a virtual machine. On the other hand, the process of automation for which the user will only have to configure his working environment deployed on the cloud and guarantee elasticity by managing the allocation and release of resources, the deployment of VMs according to the current use of resources. These challenges must be addressed and handled in order to fulfill effective resource management by understanding how to best exploit the cloud environment.

Table 1 Synthesis of related work

Authors	Level	Observed metric	Methods	Cloud environment
Chen et al. [5]	IaaS	CPU, and memory	Dynamic resource allocation and energy saving	Executes over Open-stack
Saswade et al. [28]	IaaS PaaS	CPU, and memory	Monitor VMs status in real-time	Microsoft Azure and Amazon AWS
Wailly et al. [14]	IaaS	Data stream	Monitoring the security of a virtual machine	Executes over Open-stack
Yang et al. [25]	IaaS, PaaS	Data and resources	Monitor the security and performance	AWS and Microsoft Azure
Moniruzzaman et al. [24]	IaaS	Execution time as metric	Load balancing	Executes over OpenNebula
Beloglazov et al. [35]	IaaS	CPU	Dynamic consolidation	CloudSim toolkit
Wuhib et al. [8]	IaaS	CPU	Load balancing	Executes over Open-stack
Nazir, Babar [8]	IaaS	Power consumption	Placement and migration	CloudSim toolkit

* Infrastructure as a Service (IaaS), * Platform as a Service (PaaS)

3 Proposed architecture

This section introduces the architecture of the proposed VMs-Monitor based on the OpenNebula cloud platform. The OpenNebula platform offers numerous Application Programming Interfaces (APIs) Fig. 5 with a variety of functionality for flexibly managing cloud resources [37]. The OpenNebula is a wrapper that incorporates and aggregates existing functionalities from other software components or infrastructure platforms. It was created to help with software structure integration and compatibility [38], which are open-source client application programming interfaces and system interfaces for communicating with various cloud services [39]. The XML-RPC API, the Ruby OpenNebula Cloud API, and the OpenNebula Cloud Java API can be used to implement this integration. OpenNebula also has an interface driver that permits it to interact with other infrastructure components such as authorized drivers, virtualization, monitoring, and storage.

The XML-RPC interface is mainly used to automate tasks related to the management of VMs, that can include several operations, such as checking the availability of virtual machines and instantiating/suspension/resumption virtual machines. Among others, extract information via low-level interfaces with the OpenNebula core.

Based on this architecture, the VMs-Monitor communicates with OpenNebula, using (OCA) OpenNebula Cloud API Fig. 6, that encapsulates access to OpenNebula’s XML-RPC API and implicitly associated features [22].

This integration allows VMs-Monitor to execute queries for dynamic and elastic monitoring of virtual machines. The component benefits a total view on the instantiation of the virtual machine. This means that it is deployed to control the virtual machine instances during its execution by retrieving information relating to available resources. Consider the following scenario :

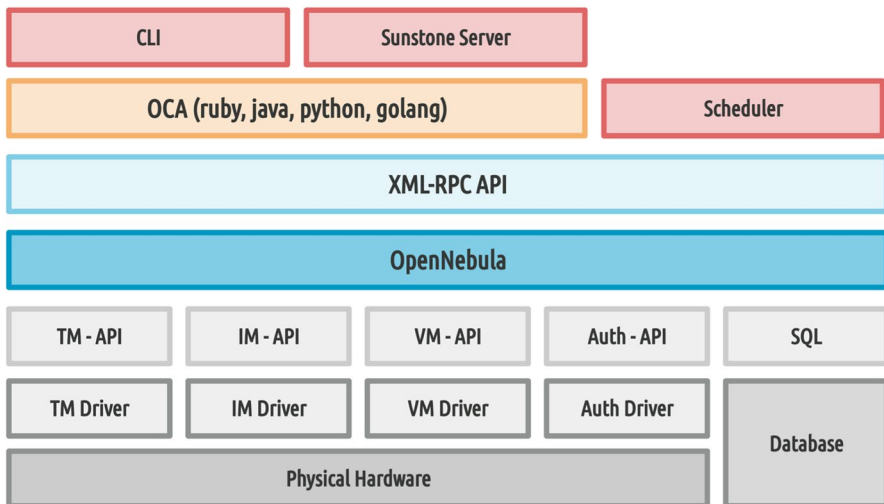
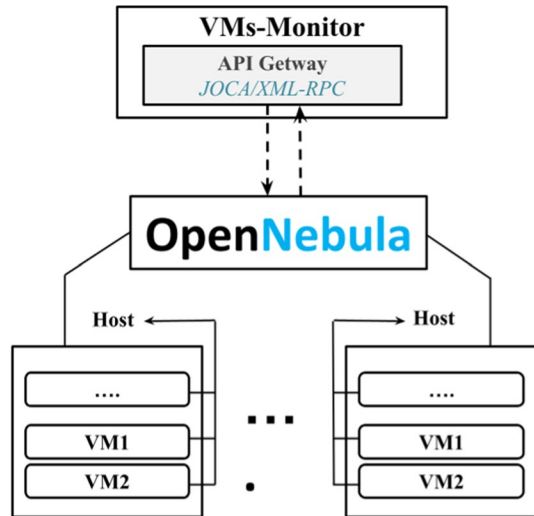


Fig. 5 OpenNebula interfaces [4]

Fig. 6 VMs-Monitor architecture



- : Host capacity: it represents the trivial constraint to accept the instantiation operations of new VMs while ensuring the availability of resources (memory, CPU, etc.).
- : CPU Usage: the rate at which each VM instance actively consumes CPU on the Host.
- : Memory Usage: the rate at which the virtual machine instance is currently consuming memory.

3.1 System model

The VMs-Monitor component regularly and dynamically monitors the running virtual machines and plans actions for instantiation (allocate, suspend, resume) according on the metrics of the resources available on the OpenNebula platform; its instantiation actions are only triggered when the component is authenticated on the OpenNebula platform. OpenNebula features an internal user/password authentication and access checklist authorization mechanism by default, as seen in Fig. 7. OpenNebula Core manages authentication and authorization, which can also be delegated to external modules. Any OpenNebula interface communicates with the Core using XML-RPC [4], which contains the user's session.

The instantiation of virtual machines is triggered by the VMs-Monitor only after retrieving measurements and information on the status of available resources. An appropriate virtual machine template must be created to launch the virtual machine instance. Figure 8 illustrates the template configuration's type of instance, which specifies the most appropriate parameters for instantiating a virtual machine. Each instance type has a specific performance profile (instance name, amount in terms of memory and CPU, storage, and the image on which this instance will be executed, etc.).

During the test, the component displays output information when a new virtual machine is successfully created or an existing virtual machine is suspended.

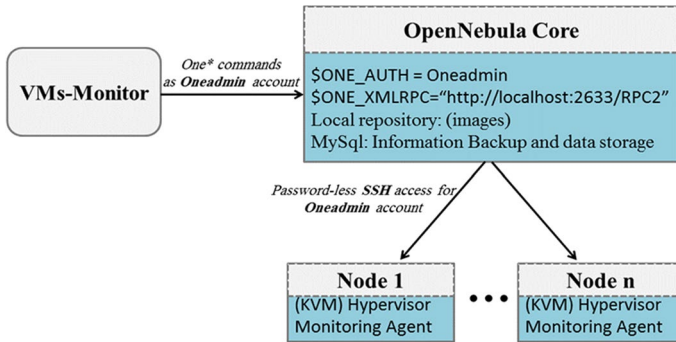


Fig. 7 Interaction with OpenNebula

To illustrate the process of instantiating virtual machines, we provide a flow-chart detailing this, as shown in Fig. 9. The process of our solution consists of two approaches: the state of the virtual machine instance and the state of the host.

To begin, each instance of a running virtual machine requires an average calculation of the appropriate resource usage by using a tuple (CPU, memory, maximum/minimum threshold) for this operation. And by each iteration, the VMs-Monitor monitors the status of each generated instance.

As a result, when the virtual machine instance exceeds the maximum resource usage, a new instance must be created. In return, if the minimum threshold is reached, the virtual machine instance will be suspended. In other words, the objective is to maintain any active virtual machine instance’s resource utilization between these two thresholds. In addition, in order to fulfill the requirements of virtual machine instantiation, host state recovery is necessary. The procedure fulfills this by defining a saturation threshold for each host based on the current status of CPU and memory consumption.

3.2 Status monitoring

The main purpose of the VMs-Monitor, as shown in process flowchart (Fig. 9), is to monitor virtual machines in a dynamic, permanent and elastic manner. Through the VMs-Monitor we have developed two algorithmic solutions: “StatVM” and “StatHost”. Furthermore, with the “StatVM” algorithm, a set of operations are performed on the virtual machine iteratively such as instantiation/ suspension/resumption. Then, with the

```

String VMTemplate =
  NAME=VM-Inst-1 #specify the name
  CPU=1 #How many CPUs required
  DISK="tIMAGE=1" #source image
  MEMORY=512 #RAM in MB
  HYPERVISOR=kvm
  NIC=[NETWORK="Private Cloud"] #connect to virtual network
  
```

Fig. 8 Template configuration model

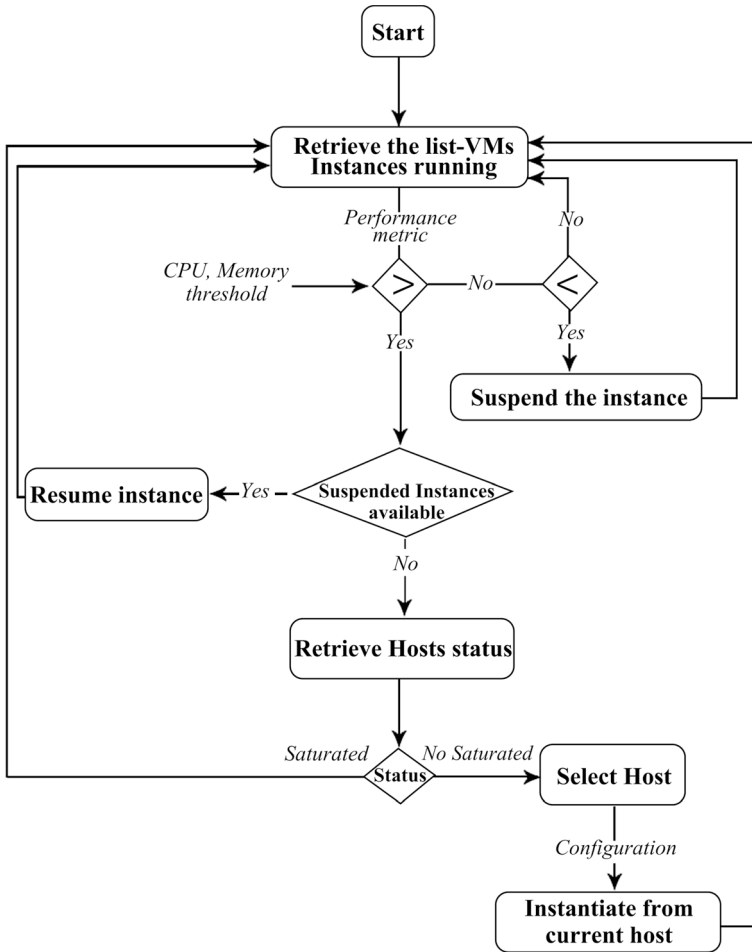


Fig. 9 Instantiation process flowchart

“StatHost” algorithm, the host status is retrieved based on the available resources and the ability to receive a new instance of the virtual machine. Detailed pseudo-code of proposed “StatVM” and “StatHost” is given in Algorithms 1 and 2. For a better perception of the proposed algorithms, the different notations used are summarized in Table 2.

3.2.1 Vm status

The algorithm is based primarily on a score associated with each active VM instance. For this, we take into consideration the utilization rate of the memory and the CPU, as shown by Algorithm 1.

Table 2 Notations and definitions

Notations	Definitions
<i>List_VMs_id</i>	VMs list identify by ID
UpperTh	Upper Threshold
LowerTh	Lower Threshold
VM_id	VM identify by ID
<i>VM_status</i>	Virtual machine status
<i>H_status</i>	The state of the host
VCpuU	VM CPU Utilization
VMemU	VM Memory Utilization
<i>H_status</i>	The state of the host
HCpuU	Host CPU Utilization
HMemU	Host Memory Utilization

Algorithm 1: StatVM Algorithm

```

Input : List_VMs_id ;
UpperTh ← 0.9 : Upper threshold;
LowerTh ← 0.5 : Lower threshold;
Score(Id_VM instance running) :
/* definition of score */;
Output : VM_status;
Begin;
for each VM_id in List_VMs_id do
    Score ← CalculateScore[VM_id];
    if Score[VM_id] > UpperTh then
        if suspended Instances available then
            resume instance;
        else
            if
                Host has enough ressources for VMs
            then
                Create new instance VM
            end
        end
    end
    else
        if Score[VM_id] < LowerTh then
            Suspend VM_id
        end
    end
end
Return H_status;
End

```

At each iteration, in the first step, the algorithm tries to collect information from each active VM instance to calculate the score. The score is formulated using the two parameters (VCpu, VMemory) expressed as a percentage defined as follows:

$$\text{Score}_{\text{VM}} = \text{VCpu}U * \text{VMem}U \quad (1)$$

Next, the VCpuU and VMemU variables, respectively, represent the CPU and memory consumption of each active VM instance. The total consumption score depends on the consumption of one of the variables; if one of the two variables is extensively used, the total score will increase. Therefore, the score of each active VM instance calculated will be compared to the predetermined threshold. If the score exceeds the upper threshold, a new VM instance will be created, if and only if the Host has sufficient resources for the new instance, or resume a suspended VM instance if it available. However, if the score is below the lower threshold, the VM instance will be suspended.

3.2.2 Host status

To accept the allocation of a new VMs instance, the Host's status must be retrieved (determine if the Host is Saturated or not). The status of the Host is mainly based on available resources, namely memory and CPU [40]. Therefore, the host is considered saturated due to one of two possibilities:

- The use of CPU exceeds the previously determined threshold.
- The use of Memory exceeds the previously determined threshold.

Algorithm 2: StatHost Algorithm

```

Input : H_list;
UpperTh ← 0.9 : Upperthreshold;
Output : H_status;
Begin;
for all Host in H_list do
  HCpuU ← get Host Cpu used;
  HMemU ← get Host Memory used;
  if
    (HCpuU > UpperTh) or (HMemU > UpperTh)
  then
    | exclude the Host from receiving new VMs;
    | H_status ← Saturated
  else
    | select Host;
    | H_status ← No saturated;
  end
end
Return H_status;
End

```

The pseudo-code for the StatHost is shown in Algorithm 2. The Saturated host is further analyzed based on the overload threshold UpperTh. Next, to reveal host saturation, each host has two possible scenarios: CPU overload or memory overload. Therefore,

it is sufficient to ensure that the maximum use of one of the metrics (CPU, memory) is not exceeded at a maximum UpperTh threshold (the threshold is set at 90%). This signifies that if the memory usage exceeds the upper threshold; even if the CPU usage is below the upper threshold, the Host is considered saturated and does not accept new instances. Furthermore, the Host is unable to utilize all its memory but abuses the CPU excessively.

4 Experimental environment and results

To prove the efficiency and accuracy of VMs-Monitor component proposed in this work, an evaluation was made by conducting a series of experiments to monitor the consumption of resources by virtual machines. The experimental environment and the experimental results are defined in this section. In experimental environment, we describe our experimental environment including hardware specification and software specification. And there we show in experimental results the performance of VMs-Monitor when subjected to different test scenarios.

4.1 Experimental environment

The experimental environment consists of two parts. In the first part, we show the hardware constraints of the system requirements (CPU, memory, etc.). Later, in the second part of this section, we show the software constraints related to the different operations of the proposed architecture.

4.1.1 Hardware constraints

The experimental environment consists of a physical machine Equipped with an Intel (R) Xeon® W-2102 CPU @ 2.90 GHz, 2904 MHz, 8Cores, a 16 GB Memory, and a 500 GB hard drive. Two virtual machines were provided: one machine used as a front-end machine for the cloud administration system OpenNebula, and the other is the node to instantiate virtual machines. Table 3 summarizes the resources used to implement the experimental environment.

4.1.2 Software constraints

Our architecture was implemented with OpenNebula V 5.4 and the Kernel-based Virtual Machine KVM as the Hypervisor [22]. The VMs-Monitor component is developed in Java to make it easier to interact with OpenNebula cloud systems through its Java-OCA API (Java OpenNebula Cloud API), which uses the XML-RPC protocol for communication and data recovery.

Tables 4 and 5, respectively, show the detailed specifications of the software used, including the Java programming language ,the OpenNebula platform. And the pre-configured resources on each virtual machine created.

Table 3 Specifications of the machine used for the experiments

Criteria	Specification
CPU	Intel(R) Xeon® W-2102 CPU @ 2.90 GHz, 2904 MHz, 8Cores
RAM	16 GB
Hard disk	500 GB
Hypervisor	KVM
Host OS	CentOS 7
Guest OS	Ubuntu 18.04 LTS server

Table 4 Software specification

Software	JAVA	OpenNebula	Java-oca
Version	netbeans 8.2	5.4	Java-oca-5.4

As One-admin, the VMs-Monitor will connect to OpenNebula's default front-end via the interface URL $ONEXMLRPC = \epsilon http : // localhost : 2633 / RPC2e$, which is familiar with the XML-RPC API returned by the OpenNebula core (were created by the one-host tool as hosts). The OpenNebula Front-end connects to the Hypervisor Hosts using SSH keys without a password, utilizing the public key of user-one-admin. This is essential to establish communication between the OpenNebula Core and front-end drivers, as well as virtualization hypervisors "KVM" [4].

The experiment focused on the consumption of CPU and memory and the availability of resources necessary for the permanent monitoring of virtual machine instances. In fact, these two measures are the most relevant to our study and constitute the limited resources in a cloud environment. The functionality was tested by injecting memory and CPU workload into virtual machines to observe its behavior. Therefore, it is assumed that the load to be injected will increase the use of hardware resources such as (CPU, memory) of each VM instance to ensure reproducibility of the experiments. Suppose that the overuse of material resources is below or above the static thresholds (Upper/Lower). In this case, VMs-Monitor can automatically trigger monitoring by instantiating a new VM instance or even several VM instances.

Virtual machine monitoring is performed by the VMs-Monitor and only after authenticating as a client interacting with OpenNebula Core [20]. Figure 10 illustrates this situation by highlighting the identification between the course of the experiment and the monitoring procedure. Once interconnected, the VMs-Monitor retrieves all resource state information from the host and active instances, by running the statHost and statVM algorithms discussed in the previous section. Then, and at each iteration, if it is necessary to create a new instance, the component creates a configuration template which specifies the performance profile (CPU, memory, etc.).

Table 5 Based resources to virtual machines created

Criteria	Specification
CPU	CPU 4 cores with VT-x/ AMD-V,KVM
RAM	Based Memory 6096 Mo
Storage	10 GB

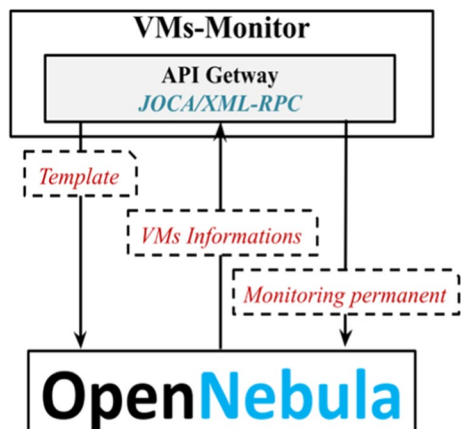
4.2 Experimental results and discussion

In this section, we present the main results of this paper. For all experimental tests, we used the same configurations template, and we'll unroll the algorithm that supports, executes, and monitors the VM instantiation process by setting the upper and lower thresholds to 90% and 10%, respectively. Figure 12a and b show a graphical representation of the behavior of the virtual machine. On this graph, we take each virtual machine's CPU Score and memory as a point during the test period. There are four behaviors:

- : Idle: If over a given period, the value of CPU and memory consumed is entirely less than 5%.
- : Oversized: If the value of CPU and memory consumed over a given period is entirely less than 80%.
- : Undersized: If the value of CPU and memory consumed over a given period is entirely greater than 5%.
- : Busy: If over a given period, the value of the CPU and memory consumed is entirely greater than 80%.

The basic configuration begins with a single active virtual machine. Therefore, we can see through Fig. 11 that the CPU and memory consumption rate during the experiment varies according to the time and the resource stress. Furthermore, according to the

Fig. 10 Experiment flow diagram



consumption curves deduced from the Sunstone interface [4], where the abscissa axis expresses the total execution time of the VM, Fig. 12 illustrates the times implicated in VM allocation actions. Therefore, the consumption rate exceeds the allocated value for most execution periods when the virtual machine is running, while the memory consumption rate is at a maximum of 95%. The allocation process started at 13:00H, on average; VMs-Monitor performs regular monitoring with an interval of 120 seconds (takes 50 seconds on average to instantiate virtual machines). Additionally, when the average CPU and memory utilization exceeded the 90% threshold for the first time (the CPU consumption rate exceeded the 98% and 92% thresholds for the memory consumption rate), in response to the solution imposed by the algorithm (described in Sect. 3.2).

Figure 13 shows a snapshot of a portion of the dashboard that displays the instantaneous CPU and memory usage monitoring values. However, VMs-Monitor starts allocating 1-4 additional VM instances during the test. From Section 13a and 13b of the figure, VM/CPU or VM/MEMORY displays the latest memory consumption reported by the monitoring probes (CPU per hour and memory per hour). As expected, the VM instances were suspended, respectively, when the minimum CPU and memory consumption rates reached the lower threshold during the test.

The utilization of host resources is depicted in the following figures. In this experiment, we observed that host consumption changes with the increase and decreases in material resource consumption by active VM instances. According to Fig. 14, the Host status varies greatly depending on the allocation and suspension of VM instances. This indicates that the Host provides CPU and memory capacity for VM instances based on the continuous allocation and suspension process of VM instances, which appears to be balanced during the experiment. This prevents the Host from reaching the maximum threshold.

Figure 15 shows the CPU and memory consumption of the hosts when they are at rest and when they are in use. It can be seen that the CPU consumption is very low when the Host is idle and increases when the Host is in use, but it does not exceed the allocated resources. In addition, memory consumption also remains low when the Host is idle, increasing when the Host is in use, which indicates that the host was not saturated at any point during the process. Therefore, there is no performance degradation and waste of idle resources. In this regard, VMs-Monitor achieved a better result. Adjusting instantiation actions (set to 80% and 10% threshold) should help monitor and control machine instances in an elastic manner.

5 Conclusion

As part of this manuscript, general principles are proposed for the implementation of an adaptive monitoring model called VMs-Monitor component respecting two issues: performance degradation and waste of idle resources. This Monitor component acts as an extension of the OpenNebula open-source cloud platform. It is designed to monitor and control virtual machine instances in terms of consumption of virtualized resources. Aware of regularly and dynamically monitoring running virtual machines and planning

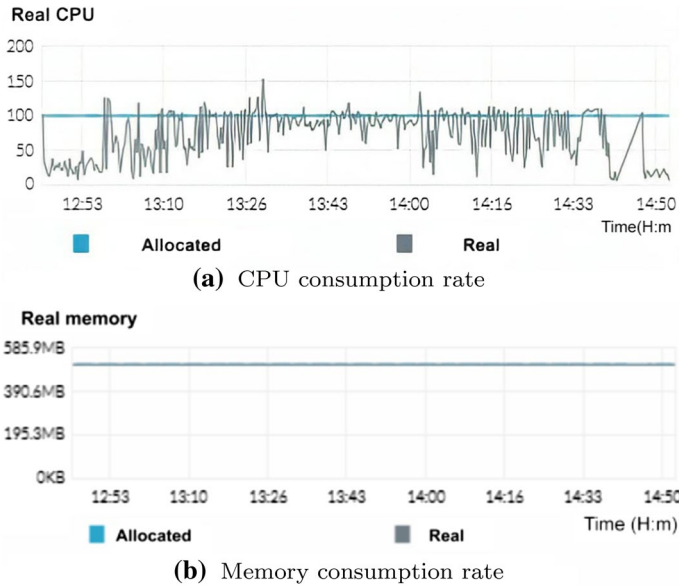


Fig. 11 the CPU and memory consumption rate for the virtual machine

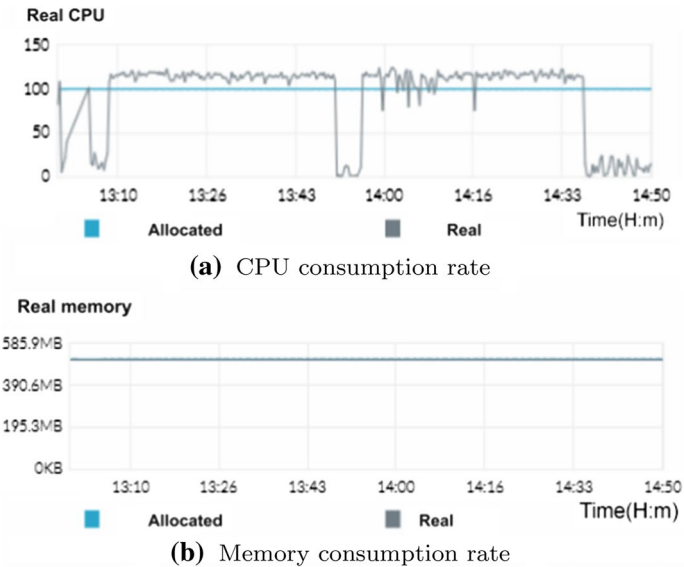
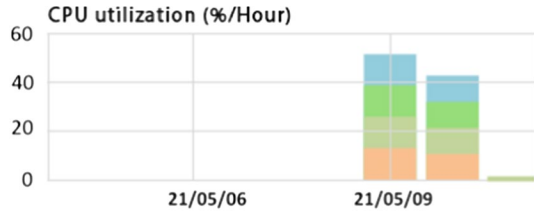
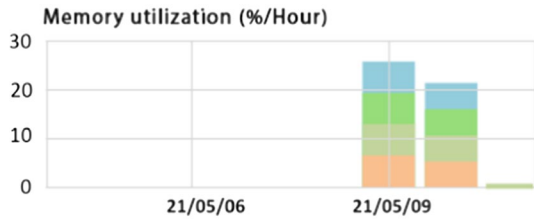


Fig. 12 The first Instance-VM allocated in response to the metrics imposed

Fig. 13 instantaneous CPU usage and Memory monitoring values



(a) instantaneous CPU usage monitoring values

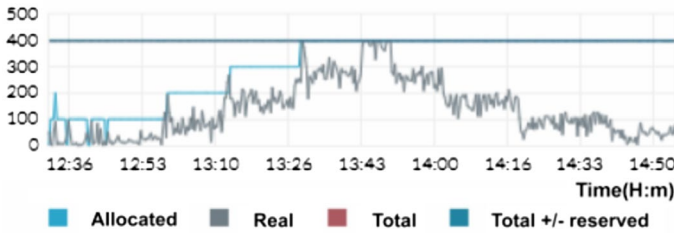


(b) instantaneous Memory usage monitoring values

instantiation actions (allocate, suspend, resume) based on available resource metrics on the OpenNebula platform.

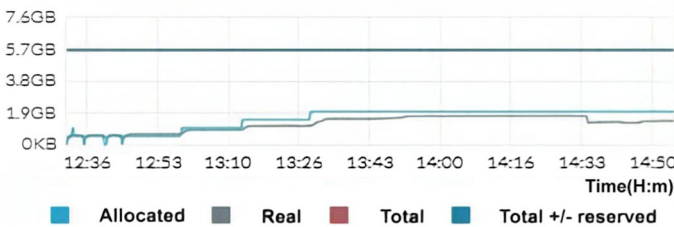
In the current implementation of the VMs-Monitor, algorithmic solutions are initially implemented to enhance the performance metrics available on the OpenNebula platform and to retrieve various information related to virtual machines and hosts.

CPU



(a) the CPU consumption rate for the Hosts

MEMORY



(b) the Memory consumption rate for the Hosts

Fig. 14 The CPU and memory consumption rate for the Hosts

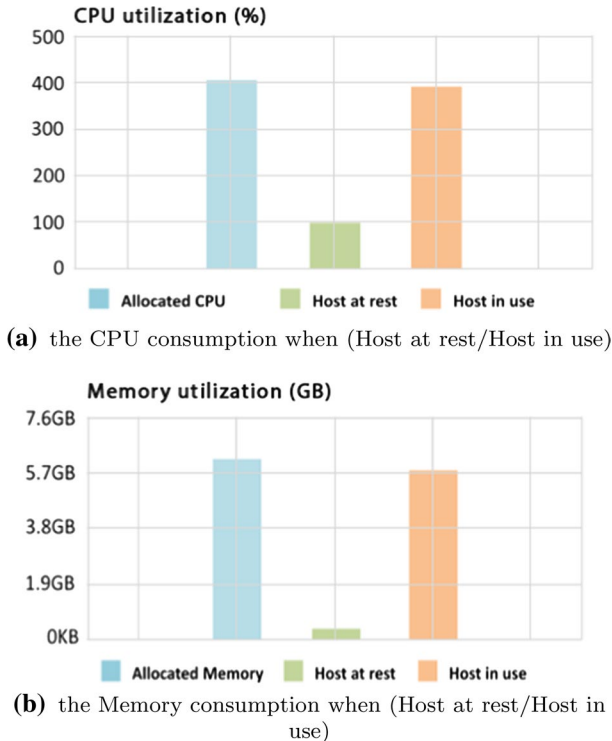


Fig. 15 The CPU and memory consumption when (Host at rest/Host in use)

Furthermore, the interaction between the monitor component and the OpenNebula cloud environment is discussed. Throughout the realization of this work, various potential axes are identified to extend the proposed model. One of the short-term perspectives is to apply the general set of monitor component design principles for other load paths (disk, NIC) and more nodes and different heterogeneous hosts.

On the other hand, another possible improvement of the algorithm would be the inclusion of a load prediction module That can be coupled with a learning module; it could predict resource utilization rates by analyzing a historic consumption.

References

1. Aceto G et al (2013) Cloud monitoring: a survey. *Comput Netw* 57(9):2093–2115
2. Khair Y, Dennai A, Elmir Y (2020) A survey on cloud-based intelligent transportation system. *International Conference in Artificial Intelligence in Renewable Energetic Systems*. Springer, Cham
3. Popek GJ, Goldberg RP (1974) Formal requirements for virtualizable third generation architectures. *Commun ACM* 17(7):412–421
4. OpenNebula OpenNebula home page. <http://www.OpenNebula.org/>, Accessed May 2021

5. Chen C-C (2015) Implementation of a cloud energy saving system with virtual machine dynamic resource allocation method based on openstack. Seventh international symposium on parallel architectures, algorithms and programming (PAAP). IEEE, 2015: 190–196
6. Usmani Z, Singh S (2016) A survey of virtual machine placement techniques in a cloud data center. *Procedia Comput Sci* 78:491–498
7. Qie X, Jin S, Yue W (2019) An energy-efficient strategy for virtual machine allocation over cloud data centers. *J Netw Syst Manag* 27(4):860–882
8. Wuhib F, Stadler R, Lindgren H (2012) Dynamic resource allocation with management objectives-Implementation for an OpenStack cloud. In: 8th International Conference on Network and Service Management (cnsm) and 2012 Workshop on Systems Virtualization Management (svm). IEEE, 309–315
9. Yang C-T et al (2017) Virtual machine management system based on the power saving algorithm in cloud. *J Netw Comput Appl* 80:165–180
10. Choi JY (2019) Virtual machine placement algorithm for energy saving and reliability of servers in cloud data centers. *J Netw Syst Manag* 27(1):149–165
11. Smith JW, Sommerville I (2013) Understanding tradeoffs between power usage and performance in a virtualized environment. In: 2013 IEEE Sixth International Conference on Cloud Computing. IEEE, 725-731
12. Kumar R, Charu S (2015) An importance of using virtualization technology in cloud computing. *Glob J Comput Technol* 1(2):56–60
13. Mosa A, Paton NW (2016) Optimizing virtual machine placement for energy and SLA in clouds using utility functions. *J Cloud Comput* 5(1):1–17
14. Wailly A, Legouge P (2020) "Method for monitoring the security of a virtual machine in a cloud computing architecture." U.S. Patent No. 10,540,499. 21 Jan. 2020. Google Patents
15. Paul AK, Sahoo B (2017) Dynamic virtual machine placement in cloud computing. In *Resource Management and Efficiency in Cloud Computing Environments*, IGI Global, pp 136–167
16. Mendel R, Garfinkel T (2005) Virtual machine monitors: current technology and future trends. *Computer* 38(5):39–47
17. Zhang X, Dong Y (2008) Optimizing xen vmm based on intelvirtualization technology. In: 2008 International Conference on Internet Computing in Science and Engineering. IEEE. pp 367–374
18. Yang C-T, Liu J-C, Huang K-L et al (2014) A method for managing green power of a virtual machine cluster in cloud. *Future Gener Comput Syst* 37:26–36
19. Yang C-T, Wan T-Y (2020) Implementation of an energy saving cloud infrastructure with virtual machine power usage monitoring and live migration on OpenStack. *Computing* 102(6):1547–1566
20. Yadav S (2013) Comparative study on open source software for cloud computing platform: eucalyptus, openstack and OpenNebula. *Int J Eng Sci* 3(10):51–54
21. Khair Y, Dennai A, Elmir Y (2021) An experimental performance evaluation of OpenNebula and eucalyptus cloud platform solutions. *International Conference on Artificial Intelligence in Renewable Energetic Systems*. Springer, Cham
22. Calle-Romero P et al (2019) Virtual Desktop Infrastructure (VDI) Deployment Using OpenNebula as a Private Cloud. *International Conference on Applied Technologies*. Springer, Cham
23. Mohamaddiah MH et al (2014) A survey on resource allocation and monitoring in cloud computing. *Int J Mach Learn Comput* 4(1):31–38
24. Moniruzzaman ABM, Nafi KW, Hossain SA (2014) An experimental study of load balancing of OpenNebula open-source cloud computing platform. 2014 International Conference on Informatics, Electronics Vision (ICIEV). IEEE, pp 1–6
25. Yang C-T, Wan T-Y (2020) Implementation of an energy saving cloud infrastructure with virtual machine power usage monitoring and live migration on OpenStack. *Computing* 102(6):1547–1566
26. Jin H et al (2013) A VMM-based intrusion prevention system in cloud computing environment. *J Supercomput* 66(3):1133–1151
27. Bacanin N et al. (2022) Modified firefly algorithm for workflow scheduling in cloud-edge environment. *Neural Computing and Applications*, pp 1–26
28. Saswade N, Bharadi V, Zanzane Y (2016) Virtual machine monitoring in cloud computing. *Procedia Comput Sci* 79:135–142
29. Sun Y et al. (2010) An architecture model of management and monitoring on cloud services resources. 2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE). Vol 3, IEEE
30. Shahidinejad A, Ghobaei-Arani M, Masdari M (2021) Resource provisioning using workload clustering in cloud computing environment: a hybrid approach. *Clust Comput* 24(1):319–342

31. Shi J et al. (2014) Design of a comprehensive virtual machine monitoring system. 2014 IEEE 3rd International Conference on Cloud Computing and Intelligence Systems. IEEE
32. Frédéric D, Menaud J-M (2015) Synthesizing realistic CloudWorkload traces for studying dynamic resource system management. 2015 International Conference on Cloud Computing and Big Data, Huangshan, China
33. Nazir B (2018) QoS-aware VM placement and migration for hybrid cloud infrastructure. *J Supercomput* 74(9):4623–4646
34. Saurabh KG, Buyya R (2012) Green cloud computing and environmental sustainability. *Harnessing Green IT: Principles and Practices*, pp 315–340
35. Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Gener Comput Syst* 28(5):755–768
36. Fard MV, Sahafi A, Rahmani AM et al (2020) Resource allocation mechanisms in cloud computing: a systematic literature review. *IET Softw* 14(6):638–653
37. Calle-Romero PE, Lema-Sarmiento PA, Gallegos-Segovia PL et al. (2020) Virtual Desktop Infrastructure (VDI) deployment using OpenNebula as a private cloud. In: *Applied Technologies-1st International Conference, ICAT 2019, Proceedings*. Springer, pp 440–450
38. Maenhaut P-J et al (2020) Resource management in a containerized cloud: status and challenges. *J Netw Syst Manag* 28(2):197–246
39. Masdari M, Nabavi SS, Ahmadi V (2016) An overview of virtual machine placement schemes in cloud computing. *J Netw Comput Appl* 66:106–127
40. Mour S et al. (2014) Load management model for cloud computing. *The 9th International Conference for Internet Technology and Secured Transactions (ICITST-2014)*. IEEE

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.