I.J. Information Technology and Computer Science, **2018, 2, 33-44**
Published Online February 2018 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2018.02.04

Phoenix: A Framework to Support Transient Overloads on Cloud Computing Environments

Edgard H. Cardoso Bernardo[1], Wallace A. Pinheiro[2], Raquel Coelho G. Pinto[1]
[1]Instituto Militar de Engenharia (IME), Rio de Janeiro, RJ, Brazil,
[2]Centro de Desenvolvimento de Sistemas (CDS), Bras ília, DF, Brazil
E-mail: edgardbernardo@yahoo.com.br, wallace@cds.eb.mil.br, raquel@ime.eb.br

Received: 30 September 2017; Accepted: 17 November 2017; Published: 08 February 2018

Abstract—This paper aims to present a computational framework capable of withstanding the effects produced by transient overloads on physical and virtual servers hosted on cloud computing environment. The proposed framework aims at automating management of virtual machines that are hosted in this environment, combining a proactive strategy, which performs load balancing when there is not overload of physical and/or virtual machines with a reactive strategy, which is triggered in the event of overload in these machines. On both strategies, it is observed the service level agreement (SLA) established for each hosted service according to the infrastructure as a service (IaaS) model. The main contribution of this paper is the implementation of a computational framework called Phoenix, capable of handling momentary overloads, considering the CPU, memory and network resources of physical and virtual machines and guaranteeing SLAs. The results demonstrate that Phoenix framework is effective, and it has outstanding performance in handling overloads virtual machine network, which has achieved the isolation of momentary overload on the physical machine preventing the propagation of their effects on the cloud.

Index Terms—Cloud Computing, Resource Management, Load Balancing, Distributed Systems, Virtual Machine.

I. Introduction

Cloud computing can be defined as a type of parallel and distributed system, built from a collection of virtualized and interconnected computers. These are available dynamically as unified resources having their services based on services levels [1, 3].

The cloud service model has three types of service model; Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). For SaaS service model, it is available software or applications that the customer can subscribe and access from the cloud.

The PaaS offers to the customer a suitable development platform with tools to be able to support their needs.

The IaaS is bottom layer of cloud the reference model. This service model provides the resources (with computing power: CPU, memory, storage and network) and abstract into numbers of virtual machines for the cloud subscribers [18].

In cloud computing environments, there are big sets of Physical Machines (PMs) containing Virtual Machines (VMs). This combination allows multifaceted and dynamic aspects of cloud computing and requires efficient and optimized techniques for resource provisioning and load balancing. For this purpose, cloud monitoring is required to identifying overutilized and underutilized of PMs which hosting VMs [14].

Mechanisms of vertical and horizontal scaling, described by [2], should ensure an elastic behaviour to these environments [4]. However, even in a virtualized environment, provided by cloud computing, threatens, extremely harmful to the functionality of services, can emerge: instability periods, failed related to storage of data, performance reduction, among others, which can compromise seriously the credibility of some service providers.

The Resource Assignment Strategy (RAS) is based on the integration of activities performed by the cloud provider to allocate resources available in this environment to meet the needs of the hosted applications. It provides the type and amount of resources required at the time and for the time required for each user application to complete its work [16].

Transient overloads as flash crowds [5], which can demand all existent computational resources in a short period of time, still may cause an eventual interruption of service, which can lead to a break of contract between service providers and their clients. These contracts can stipulate, for example, cloud data center resources (e.g. CPU, memory, network bandwidth and storage) may be allocated based on the reduction of energy usage as on satisfaction of Quality of Service (QoS) requirements specified by users via SLAs.

Thus far, in cloud environment, there are often situations of not demand contrast with peak demands for resources this way, good Resource Allocation Strategy (RAS) is required. So, an optimal RAS should avoid resource contention, scarcity of resources, resource fragmentation and over provisioning [16].

This paper proposes a framework called Phoenix that combines set of proactive and reactive strategies in order

to support transient overload caused by flash crowds using a virtualized environment in cloud computing.

Load balancing of VMs in PMs is used as a proactive strategy, avoiding premature overload of these machines. However, once the limits established to PMs are reached too fast, the reactive strategy is triggered to find the necessary resources.

After experiments, we conclude that the proposed algorithms automatically support overloading of resources. The proposal has proven to be appropriate to the reality of cloud computing data centers, ensuring the dynamic allocation of resources in the cloud.

This paper is organized as follows: Section II presents related work, Section III presents proposed solution, Section IV discusses about the overload experiments and finally Section V provides remarks and future research directions.

II. Related Works

In cloud environment, there are often situations of peak demands and non-demand for resources. Hence to match with these uncertain demands for resources, good Resource Allocation Strategy is required. It provides an on-demand services by means of dynamic allocation of resources to provide reliable and high available services to the users.

Thus, to balance the level of supply and demanding of resources, Resource Allocation Strategy must be able to handle the following issues regarding the resources: Contention, Fragmentation, Over-provisioning and under-provisioning. [15]. Those issues have been investigated in the literature. In addition, it is necessary to take into account contracts between service providers and customers, such as SLAs. The above approaches present some ways for problem mitigation.

Zhen et. al [25] use the predictive load model to predict future uses of application resources accurately. But they used this method only in dynamic VM consolidation. If the expected future load of a server indicates an underuse of resources, the VM migration will not occur. This reduces the number of VM migration as well as the number of false hotspot events. Zhen et. al [25] also introduced the concept of "skewness" to improve the overall use of server resources. During migration, this server is used as a target, which ability can be reduced by accepting the migrant VM. Unlike the one proposed in [25], Phoenix considers a server that the server is overloaded when the server's cumulative load is greater than the parametrized upper limit. This can prevent unnecessary migrations from being initiated due to a momentary increase in resource utilization. The Phoenix uses a mechanism that accumulates historical data of the measurements of each resource value and operates a configurable delay where the value considered is an accumulated of the measured values as described by Siberschatz et al. [17]. By doing this, is avoided that resource utilization peaks are considered as overloads this mechanism is a cumulative load of the PM server or VM. In addition, the proposed architecture uses the concept of distance from the cluster balance range to find a target host when evaluating migrations.

Khanna et al. [27] monitor the resources (CPU and memory) of physical and virtual machines. They proposed the idea of fixed threshold value that would limit the maximum use of resources. If a feature exceeds a predefined threshold and there is a chance of SLA violation, then the system migrates a Virtual Machine (VM) to another Physical Machine (PM). Phoenix, in addition to considering the network resource, uses a similar principle to perform the migration, the lower cost VM being chosen to be removed from the PM. In addition, when the network overload occurs, the migration of the unaffected machines is done, starting with the lowest cost.

Anton et al. [28] proposed algorithms for efficiently mapping the energy efficiency of virtual machines to suitable cloud resources. They created different methods of VM selection, such as "minimization of migration policy," "higher potential growth policy," and "random policy of choice" to choose a migration-specific VM. The authors suggested that it is not a wise decision to keep the usage limit set because the workload is changing continuously. In their subsequent article [33], the authors proposed Inter Quartile Range (IQR) and Median Absolute Deviation (MAD) methods to dynamically find the upper limit of a server.

According to Anton et al. [27], if the current host load is greater than the upper bound, then it is considered overloaded. The concept of adaptive boundary works much better than the static threshold in the dynamic cloud environment. They also proposed methods such as Local Regression (LR) and Robust Local Regression (LRR) to predict the future load, but in those methods, hosts are considered overloaded only when the intended use is greater than or equal to 100%. Phoenix considers a overloaded PM when current and future load is greater than parameterized upper limit. This can prevent unnecessary migrations from being initiated due to a momentary increase in resource utilization. We use an exponential moving average based prediction technique [5] which is a cumulative charge of the PM server or VM.

Wood et al. [7] introduced a system called Sandpiper to automate the task of detecting overloads and determining a new mapping of physical resources to virtual resources and initiating the necessary migrations in a virtualized data center. To ensure that a small transient peak does not trigger unnecessary migrations, an overhead is only marked if the thresholds or SLAs are exceeded for a sustained time. The migration occurs only when at least k of the most recent observations, as well as the next predicted value exceeds a threshold. The limit that is considered in this article is static. The authors use the automatic regression method to calculate the next predicted value. After a hotspot detected, the VM whose maximum volume-to-size (VSR) ratio is migrated. When the system load is high, it is not possible to migrate to VM with the highest VSR. In this case, the VM swap occurs to reduce the load on the cache. According to Zhen et al. [27] this strategy will not work effectively during

the system's peak load, since switching VMs will increase the migration load unnecessarily. Phoenix uses a similar method to identify overhead, but being in contrast with Wood et al. [7], Phoenix migrates to lower-cost VM in the case of CPU or RAM overhead. In the case of network overload, the VM that is overloaded remains in the source PM while the other VMs, in descending order of volume, are migrated to PMs that are capable of receiving them.

The VOLTAIC (Volume Optimization Layer to Assign Cloud) system, as described by Carvalho and Duarte [6], functions as an autonomous resource manager for the cloud and aims to increase the quality of service provided to customers and avoid wasting computational resources [6]. This system uses controllers based on fuzzy logic to detect the saturation of PMs and proposes algorithms to reciprocate VM automatically, considering VM usage profiles and availability of resources in each PM. VOLTAIC is compatible with most platforms that support the libvirt library, such as: Xen [8], VMWare [9] and KVM [11].

Norman et al. [30] developed a management algorithm for dynamic allocation of virtual machines to physical servers. The algorithm is based on measuring historical data, predicting future demand and remapping VMs to PMs, and it is subsequently referred to as Measure-Forecast-Remap (MFR). Time series forecasting techniques and bin packaging heuristics are combined to minimize the number of PMs required to support a workload. In this algorithm, the prediction method is used to find the resource demand of individual VMs. Based on predicted values, VMs are organized in descending order and the first-fit heuristic is used to migrate the VMs. Conversely, Phoenix is a computational architecture capable of to support transient overloads, such as flash crowds, in cloud computing environments. Therefore, Phoenix uses an algorithm whose automatically manage PMs and VMs, being able to monitoring resources (CPU, RAM, Network) of PMs and VMs automatically, manage VM hosting and handling in PMs, detect and load transient overloads taking into consideration the SLAs and finally, it is also able to perform load balancing between PMs. To detect overloads, the Phoenix uses a detector that has a configurable trigger that establishes the limit of use for each PM feature. To measure the resources of the PMs and VMs it is used a mechanism that accumulates historical data of the measurements of each resource value and operates a configurable delay where the value considered is an accumulated of the measured values, as described by Siberschatz et al. [17]. By doing this, is avoided that resource utilization peaks are considered as overloads. Based on this value, considered decisions are made that result in the movement of the VM, either to treat a PM overload, the load balancing or even the identification of the VM overload. As for movements of the VMs, these are based on the increasing order of the momentary use of the resources. For this purpose, the volume of use of the resources is calculated each VM. Based on the obtained values, a VM list in ascending order of volume is established. From this list, VMs are moved as PMs. The PMs receives VM according to their capacity and it situation in relation to average capacity of t cluster's PMs.

Kochut and Beaty [26] proposed an analytical model of VM migration that provides estimates of the expected gain in response time due to a migration decision. The model is based on the M/M/1 queuing model and considers the characteristics of a virtualized environment, such as migration costs and overhead, due to the additional consumption of resources. This VM is selected for migration that minimizes system response time. Although it does not use the same algorithm, Phoenix uses the predictive model based on threshold of hotspot and or load balancing using cluster equilibrium range.

Arzuaga et. al. [31] presented a new metric that captures the load of the physical servers according to the loads of the resident VMs. The load unbalance is measured by using this metric. The proposed load-balancing algorithm follows a greedy approach. The VM that will produce the greatest imbalance metric improvement is selected for migration. In addition to load balancing, the VM migration performed by Phoenix will also make the system more efficient by making the systems more resilient to transient overloads since all post-balancing PMs will be within the cluster's equilibrium range. In this way, all PMs will have similar conditions to withstand transient overloads.

Andreolini et al. [32] proposed a new management algorithm to decide on VM overloads in a cloud environment. Instead of the traditional threshold method, the authors used the load profile evaluated through a cumulative sum-based stochastic model. This method eliminates unnecessary VM migrations due to the momentaneous increase in load. The traditional best-fit bin packing algorithm is used for reallocation of the selected VM. Phoenix also eliminates unnecessary VM migrations by using a mechanism that accumulates historical data of the measurements of each resource value and operates a configurable delay where the value considered is an accumulated of the measured values as described by Siberschatz et al. [17]. By doing this, is avoided that resource utilization peaks are considered as overloads. In addition, Phoenix also performs load balancing and PM isolation caused by overloading network beyond the SLA threshold, possibly because it is Flash Crowd or even a DDoS.

III. Proposed Solution

Phoenix is a framework to deal with transient overload, such as flash crowds, in cloud computing environments. This framework automatically manages PMs and VMs and it is applicable to the model of Infrastructure as a Service (IaaS) [3].

Phoenix framework executes the following activities: monitors resources of PMs and VMs automatically, manages movement and hosting of VMs, detects and supports transient overloads taking into account SLAs, and executes the load balance among PMs. Phoenix framework, showed in Fig.1, is composed of a series of

components that support three main modules: ***Monitor***, ***Analyzer*** and ***Configurator* (*Migrator*)**. Phoenix Admin Interface helps the administration of these modules, providing an interface that shows all parameters monitored by the framework. The components of Phoenix framework are detailed as follows:

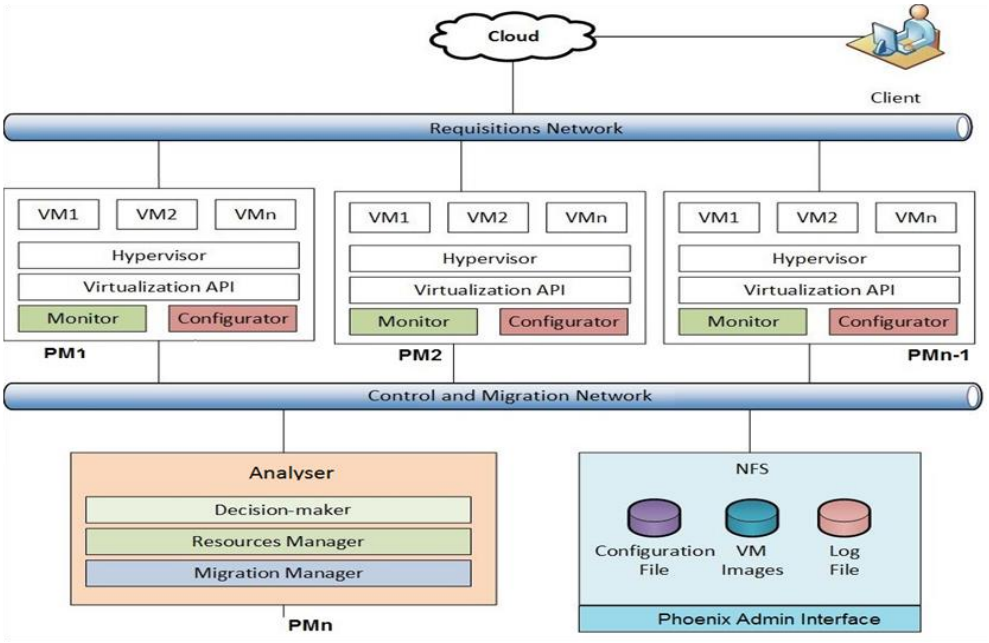

Fig.1. Phoenix Framework

PM_i – represents each PM in the framework. PMs host services provided by the cloud. PM_1 up to PM_{n-1} compose the cluster of machines that hosts IaaS services and has its resources managed by the framework. While PM hosts the *Analyzer* module, the other machines host the *Monitor* and *Configurator* modules.

VM_i – represents each VM in the framework. VMs execute clients' applications.

Requisitions Network – corresponds to the network infrastructure where users do requisitions to services hosted in VMs.

Control and Migration Network - corresponds to the network infrastructure where PMs exchange messages about their status and where VMs are moved (migrated). Thus, this network is dedicated to the management of the framework.

***Monitor* Module –** is responsible for gathering resources information (use of network bandwidth, CPU processing and memory usage) from PMs and VMs, registering also the percentage of using (on average) of these resources.

***Analyzer* Module** – analyzes relevant events generated by the *Monitor* module and evaluates the necessity of migrate VMs. This migration can be caused by: overload of PM resources, overload of VM resources, or unbalanced load considering three factors, network bandwidth, CPU processing and memory usage. Once a relevant event happens, the *Analyzer* module sends a request to the *Configurator* module.

Hypervisor – is responsible for creating, moving and destroying VMs, receiving requests directly from the *Configurator* module.

Virtualization API – is provided by *libvirt* library. This API is related to resources virtualization and it supports different hypervisors, including KVM that was chosen as virtualization platform to implement the proposed framework.

***Configurator* Module** – receives moving requests from the *Analyzer* module and uses the Virtualization API to migrate VMs. It informs the *Monitor* module when a VM migration is concluded. The *Configurator* also allows the use of different strategies of migration, defined by the *Analyzer* module.

Log Files – store information about resources used by machines related to relevant events in different moments of the day, as well as messages exchanged by the framework modules. Each *Monitor* has a log file.

Configuration File – stores information about the limits proposed to each VM and PM, based on SLA established between the service provider and its clients.

As the framework adopts a Black-box approach, the monitoring is made externally to VMs and PMs. Another aspect to be highlighted is that the framework can use different hypervisors.

A. Proposed Solution Operation

Phoenix operation was inspired on OODA loop [11]. There is a strong similarity between OODA loop phases and the process executed by the framework, such as: observation of (machines) status, orientation (based on events), planning and decision about actions to be taken and, finally, execution of the necessary actions. After this, the loop is restarted. The dynamic behaviour of Phoenix is schematically represented in Fig.2.

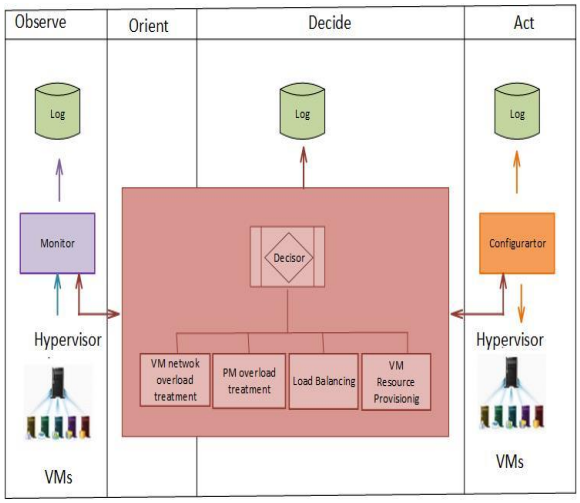

Fig.2. Functioning of Phoenix

The events observed by *Monitor* module are described as follows:

Overload of VM Network – it indicates that the percentage of network bandwidth usage is higher than the predicted in SLA. The message used by *Monitor* to indicates this event is: **OVERNETVM**.

Overload of PM – it indicates that the percentage of use of CPU, memory or network bandwidth is higher than an established limit (typically it is set to 90% of PM capacity, allowing some time of reaction, before the machine full capacity be reached). The message used by *Monitor* to indicates this event is: **OVERLOADPM**.

Relevant Event – it denotes some significant variation related to the PM status. In this case, it may be necessary to execute a load balance, depending on the comparison among PM load, the average load of all PMs and some configurable variation (that can be set in Configuration File). One or more VMs should be moved to other(s) PM(s), in order to allow the load balance. The message used by *Monitor* to indicates this event is: **RELEVANT_EVENT**.

Reachability of VM SLA – it represents that an SLA related to the usage of CPU, memory or network bandwidth was reached. Probably, a new SLA should be established between the service provider and the client, in order to allow the client VM to support the load. The message used by the *Monitor* to indicate this event is: **OVERLOADVM**.

One additional message created by *Monitor* is **MONITOR_INFO** used to indicate beginning of the monitoring operations or restart after a migration.

The *monitor* only sends messages in the situations indicated by these events, in order to minimize the number of messages generated.

Algorithm 1 demonstrates the main actions executed by *Monitor* module based on monitored events. The details about the functions used in the Algorithm 1 are described after the algorithm.

```
Input: start_Monitor/stop_Monitor
Output:info_overnetvm(vm_info)/
  info_overloadvm(vm_info)/
  info_overloadpm(pm_info)/
  info_relevant_event(pm_info)

Function Monitor {
WHILE stop_Monitor = 'No'
    get_pm_info()
    pm_CPU=calculate_accu(pm_info, 'CPU')
    pm_MEM=calculate_accu(pm_info, 'MEM')
    pm_NET=calculate_accu(pm_info, 'NET')
    list_vm=get_vm_info()
    WHILE list_pm NOT EQUAL " "
        pm_CPU=calculate_accu(pm_info, 'CPU')
        pm_MEM=calculate_accu(pm_info, 'MEM')
        pm_NET=calculate_accu(pm_info, 'NET')
        WHILE list_vm NOT EQUAL " "
            vm_CPU=calculate_accu(vm_info,'CPU')

    vm_MEM=calculate_accu(vm_info,'MEM')
            vm_NET=calculate_accu(vm_info,'NET')
            IF vm_net > limit_vm_NET
            SEND TO
            Analyzer(info_overnetvm(vm_info))
            ELSE IF vm_CPU > limit_vm_CPU OR
            vm_mem > limit_vm_MEM
            SEND TO Analyzer
            (info_overloadvm(vm_info))
        END WHILE
        IF (pm_CPU > limit_CPU OR pm_MEM >
        limit_MEM OR pm_NET > limit_NET)
            SEND TO Analyzer
            (info_overloadpm(pm_info))
        ELSE
            IF ((pm_CPU > earlier_pm_CPU +
                relevant_variation OR
                pm_CPU < earlier_pm_CPU –
                relevant_variation)
                OR
                (pm_MEM > earlier_pm_MEM +
                relevant_variation OR
                pm_MEM < earlier_pm_MEM –
                relevant_variation)
                OR
                (pm_NET > earlier_pm_NET+
                relevant_variation OR
                pm_NET < earlier_pm_NET –
                relevant_variation))
                    ealier_pm_CPU = pm_CPU
                    earlier_pm_MEM = pm_MEM
                    earier_pm_NET = pm_NET
                    SEND TO Analyzer
                    (info_relevant_event(pm_info))
            END IF
        END IF
    END WHILE
END WHILE
}
```

The *get_pm_info()* function obtains information about the resources (CPU, memory and network) of PMs.

The *get_vm_info()* function obtains a list of all VMs and information about their resources (CPU, memory and network). The information of each VM is stored in a variable named *vm_info*.

To avoid transient values of CPU, memory and network bandwidth, we proposed the use of the *calculate_accu* function. It calculates the accumulated value of each resource considering a reduction factor according to Silberschatz et al. [17]. It is expressed by the equation (1):

$$N_{i+1} = \mu * N_{instant\text{instantaneous}} + (1 - \mu) * N_i \quad (1)$$

Where:

- N_{i+1} - is a prediction for the next instantaneous value to be captured.
- μ - is a constant with a value expressed between 0 and 1 ($0 < \mu < 1$), which expresses how much the snapshot value will be taken into consideration for calculating the average. It serves as a reducer that will allow a greater or lesser lag in detection of an overload.
- $N_{instant\text{instantaneous}}$ - is the captured value of use of the resource in percentage in iteration *i*.
- N_i - is the accumulated value of use of the previous iteration feature.

The *info_overnetvm* function informs if the network of a VM is overload.

The *info_overloadvm* function returns if the limit given to CPU, memory or network bandwidth of a VM is reached.

The *info_overloadpm* function informs if the limit given to CPU, memory or network bandwidth of a PM is reached.

The *info_relevant_event* function returns if a defined variation of CPU, memory or network of a PM was reached.

The *Analyzer* module begins when it receives messages from the *Monitor* modules. It is composed of the following sub-modules: *Resource Manager*, *Decision Maker* and *Migration Manager*.

The *Resource Manager* sub-module is responsible for receiving all messages from the *Monitor* modules. After this, it calculates the volume and cost of migrating each VM, the load of each PM and the average load of the cluster composed of all PMs. All this information and the original messages sent by monitors are then repassed to the *Decision Maker* sub-module. If it decides that migrations are necessary, it informs this decision to the *Migration Manager* sub-module.

Therefore, The *Analyzer*, through the *Decision Maker* sub-module, evaluates the information related to VMs and PMs. A set of rules is used to infer actions to be taken by the framework. If it is confirmed that the limits of overload related to a machine (VM or PM) were reached or if it is necessary to balance the cluster, the framework will call the suitable action to deal with the situation. The actions are triggered obeying a priority, depending on the event impact. Our analysis indicates that the following ordering of actions should be obeyed:

(1) Treat Overload of VM (when it receives data from the *info_overnetvm* function),

(2) Treat Overload of PM (when it receives data from the *info_overloadpm* function),

(3) Treat Relevant Event (when it receives data from the *info_relevant_event* function), and

(4) Treat Reachability of VM SLA (when it receives data from the *info_overloadvm* function).

The next subsections will detail each action taken by *Analyzer* module.

A.1. Action to Treat Overload of VM

This action is quite important because most current hypervisors, such as Xen [8] and KVM [10] do not limit the network bandwidth usage. Therefore, it can jeopardize a PM that support a VM suffering, for instance, a flash crowd event. In this case, all others VMs hosted in this PM can also be affected.

We propose a strategy to deal with a VM that consumes a network bandwidth higher than the established in SLA. This VM should be kept in its original PM while others VMs should be migrated. Thus, if it is necessary, this VM is isolated in one PM ensuring its SLA. The others VMs can continue working in others PMs, having also their SLAs protected.

In this process, it is important to follow an ordering of VMs to be migrated. The VMs are sorted in order of increasing migration cost. The concept used in this work was defined in [7]. The idea is to release the resources as fast as possible, migrating the lower cost machines first.

Regarding the candidate PMs to receive VMs, we propose to prioritize PMs that have the lower load values taking as reference the cluster average load. This procedure aims to keep the cluster as balanced as possible (and, consequently, minimizing the number of migrations).

The cluster average load is expressed as the average of real values obtained of CPU, memory and network bandwidth usage. We propose that the administrator of the architecture should define a suitable weight to each one of these attributes, prioritizing what is more important depending on the architecture resources. The equation (2) that defines the Cluster Average Load (CAL) is:

$$\text{CAL} = \left(\frac{(\text{CPU} * \text{WeightCPU}) + (\text{MEM} * \text{WeightMEM}) + (\text{NET} * \text{WeightNET})}{(\text{WeightCPU} + \text{WeightMEM} + \text{WeightNET})} \right) \quad (2)$$

Fig.3 presents a simulation of Phoenix behavior when a VM consumes all network bandwidth of a PM. It also shows the network usage (percentage) of three PMs (PM1,

PM2 and PM3) in three different times (T1, T2 and T3). VMs are represented in PMs by different colors and named as: VM1, VM2, VM3, VM4 and VM5. In T1, all three PMs and their VMs are operating normally. In T2, VM1 and, consequently, PM1 are overloaded by a flash crowd event. In T3, VM1 is isolated in PM1, avoiding the propagation of the problem to other VMs and to the cluster.

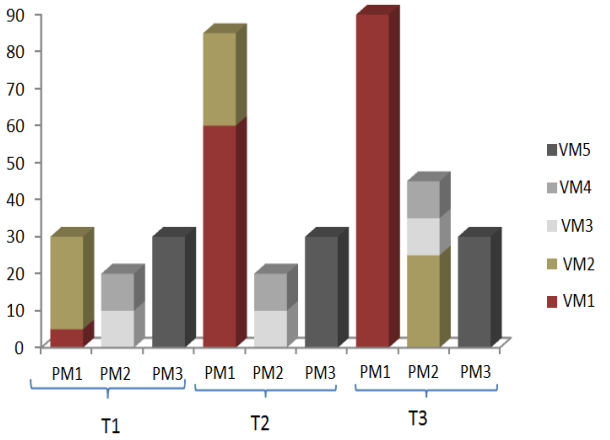

Fig.3. Behaviour of Phoenix when a network bandwidth of a VM increases beyond its specification in SLA

A.2. Action to Treat Overload of PM

During this action, the strategies to choose which VMs will be migrated as well PMs that will receive these machines are similar to the strategies used to deal with overload of VM network. However, in this case, there are no VMs overloaded.

The algorithm will move VMs to others PMs that have the resources to receive them, prioritizing VMs of lower cost [7]. The specific PMs that will receive VMs will be selected according to their load values taking as reference the cluster average load. The number of VMs to be received by each PM also considers these values. Therefore, a same PM can receive more than one VM in order to keep the cluster in balance.

A.3. Action to Treat Relevant Event

This action checks if the cluster is unbalanced, according to the parameters defined by the administrator (variation of a PM load in relation to the cluster average load is higher than a pre-defined value). If this is the case, it is chosen the PM that has the higher load and a VM (or VMs) of this machine to be migrated in order to bring this PM to the balance. Following, it is chosen the PM that has the lower load to receive the machines. After the migration, the situation is continuously reevaluated to check if more migrations are necessary.

A.4. Action to Treat Reachability of VM SLA

A message is sent to the administrator when any parameter related to VM SLA is reached. This message will contain the parameters and the corresponding values. The clients should be advised of the problem by suggesting changes in the SLA.

B. Message Exchange View

To demonstrate the message flow among the different elements of Phoenix framework, Fig.4 illustrates some scenarios divided into five frames. The first frame indicates the messages exchanged among *Monitors* and the *Analyzer* (MONITOR_INFO). This message starts the activities. The second frame shows when one *Monitor* informs the occurrence of an event to the *Analyzer*. Depending on the event, the *Analyzer* may decide that it is necessary to migrate a VM to keep the cluster in balance. The frame three presents a scenario that generates a message determining a migration (MIGRATE). The frame four depicts a migration between two PMs. In frame five, PM_{n-1} indicates the end of the migration process through the message (MIGRATION_FINISHED). Then, the PM that receives the VM informs its status with a message (MONITOR_INFO).

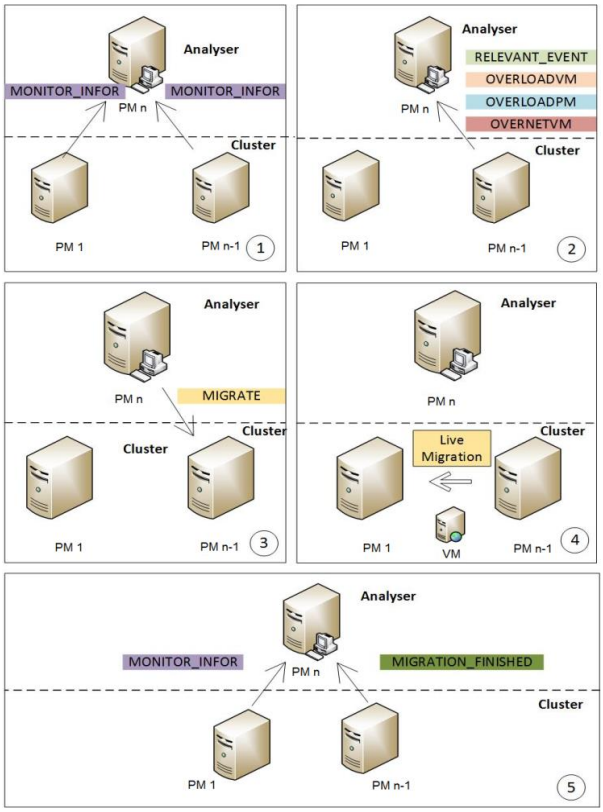

Fig.4. Message Flow among Different Elements of Phoenix Framework

IV. Experiments

A series of experiments were done in order to analyze the behavior of Phoenix framework. Fig.5 details the environment used in the experiments: four computers (LabC202, LabC203, LabC204, LabC205) containing twelve VMs; one computer (LabC201) containing the *Analyzer* module and an NFS (*Network File System*) server; and one computer (LabC206) implementing a load

generator to simulate the network requisition from clients.

The *iperf*[1] [12] program was installed in LabC206 to simulate network requisitions from clients, and the *stress*[2] program was installed in four computers (LabC202, LabC203, LabC204, LabC205) to simulate CPU and memory requisitions.

Besides, the environment implemented two gigabit Ethernet networks: one used for clients requisitions (simulation of load made by the load generator), named Requisitions Network, and other used by migration and control, named Migration and Control Network. The computers that hosted the VMs had two networks cards each one. The load generator was connected to Requisitions Network. Finally, the Control Network connected the *Analyzer* module and the NFS server to the Migration module**.** Table 1 describes hardware, software features and configurations used on these experiments.

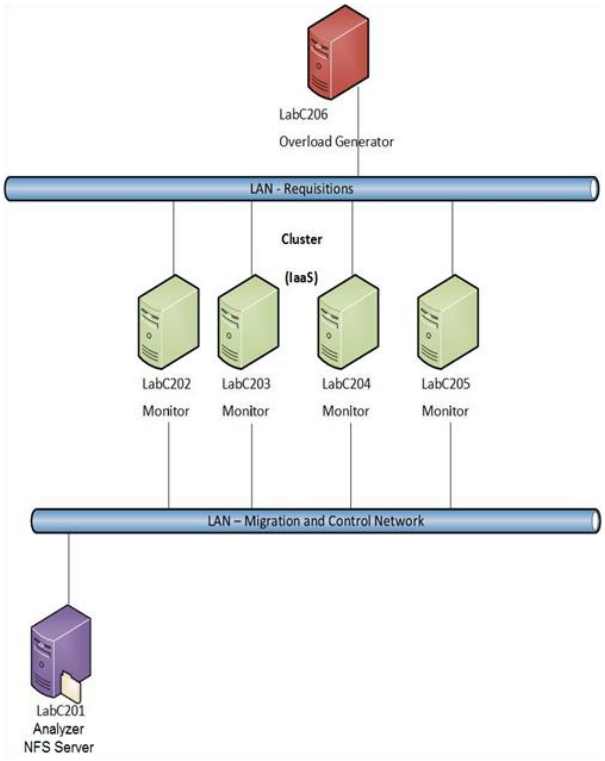

Fig.5. Environment of Experiments

The limits established to PMs (LabC202, LabC203, LabC204, LabC205) were 80% to CPU, network and memory. Beyond these limits, any PM is considered overloaded. VMs (VM1 to VM12) were set to use 1 CPU core, 1024 MB RAM and a 250 Mbps network card.

The next subsections will detail the experiments:

A. First Experiment: Load Balance

The first experiment aims to analyze the load balance effectiveness of the Phoenix framework. It demonstrates the proactive behavior that avoids the overhead of PMs in situations of regular cluster operation (without major transient overloads). The proposed algorithm evaluates if a migration will improve the balance of the cluster. Migration is performed only if this improvement is verified.

Table 1. Hardware and Software Configurations

Features of PMs and VMs	Software	Configuration
***Analyzer* (LabC201):** Intel®Core i5 E7500, 3MB Cache L2, up to 2.93 GHz, 1066 MHz FSB, 4 cores, 4 GB RAM, 1Gbps network card.	- Ubuntu 13.10 - Python 2.7 - SSH - NFS Server	IP=192.168.7.1
Load Generator (LabC206): Intel® Core2Duo, 3Mb Cache L2, up to 3.0 GHz, 1066 MHz FSB, 4GB Ram de 4GB, 1Gbps network card.	- Ubuntu 13.10 - Iperf (client)	IP=192.168.7.6
PM (LabC202 a LabC205): Intel®Core i5 E7500, 3MB Cache L2, up to 2.93 GHz, 1066 MHz FSB, 4 cores, 4 GB RAM, enabled Virtualization Technology, two 1Gbps network card.	- Ubuntu 13.10 - Python 2.7 - Libvirt 1.1.1 - Psutil 0.6.1-2 - Qemu-kvm.5.4 - NFS client - SSH - Stress 1.0.1 - Iperf (server)	IP=200.20.188.68 to 200.20.188.71 CPU cores=4 Memory size=4096 Network capacity = 1Gbps IP=192.168.7.2 to 192.168.7.5 CPU limit=80% Memory limit=80% Network limit=80%
VM (VM1 to VM12): VCPU - 1 core 1GB RAM 250 Mbps network card.	- Ubuntu 13.10 - Stress 1.0.1 - Iperf (server)	NAME=”VM1 to “VM12” IP=192.168.7.101 to 192.168.7.112 CPU SLA = 1 core Memory SLA = 1024 MB Network SLA = 250 Mbps

To analyze different balance ranges, the load balance algorithm was tested with four ranges related to the cluster average load: 3%, 5%, 8% and 10%.

Initially, in this experiment, twelve VMs were allocated in only one PM in order to cause an unbalanced cluster. This experiment was repeated five times to each range.

To present the experimental results, it was used Phoenix Admin Interface. the information provided by this interface considering the initial scenario of this experiment (all VMs in the same PM). At the top of Fig.6, PMs graphs are shown. Below each PM, graphs of its VMs are presented. The *x*-axis presents the sample collection time points (in seconds). For each graph, the *y*-axis can indicate percentage levels of: network bandwidth (green line), CPU processing (blue line) and memory usage (magenta line). Besides, in PM graphs, the *y*-axis can also indicate percentage levels of: PM overloads limit (cyan line), PM load (yellow line) and CAL (1) (red line).

[1] https://iperf.fr/
[2] https://launchpad.net/ubuntu/trusty/armhf/stress/1.0.1-1ubuntu1

After detecting the unbalanced cluster, the algorithm moved VMs trying to balance the cluster. Table 2 presents the results obtained to the different ranges of balance. It can be observed that lower ranges demanded more time and more migrations to generate a balanced cluster.

In our experiments, the range of 8% presented the best distribution considering lower time, number of migrations and standard deviation of VMs per PMs (standard deviation zero indicates exactly three VMs in each PM).

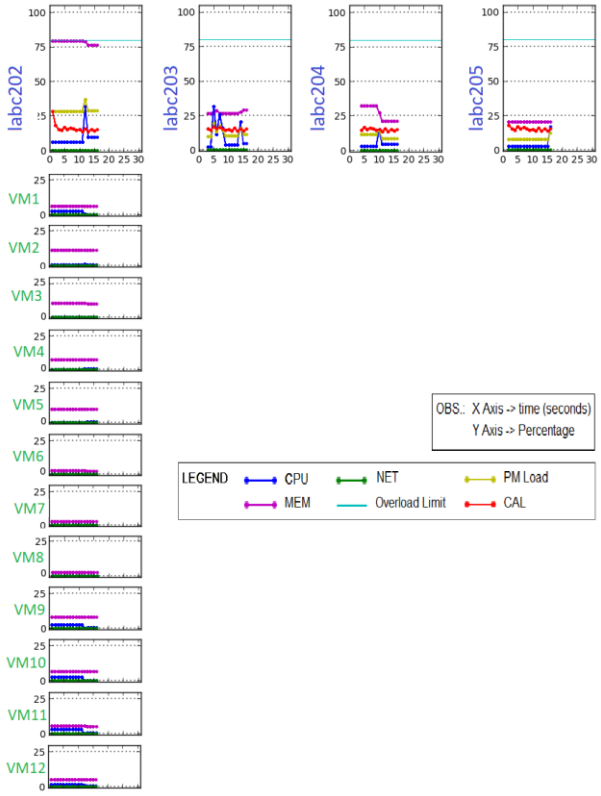

Fig.6. Initial Distribution of VMs to the Cluster

Table 2. Experimental Results considering different Ranges of Balance

Range	Average Number of Migrations	Average Time	Standard Deviation of VMs per PMs
3%	13	3 min and 5 sec	1,2
5%	8	2 min and 38 sec	0,8
8%	8	1 min and 19 sec	0
10%	7	1 min and 10 sec	1,4

Fig.7 shows the final cluster status for the range of 8%. It is possible to observe that all PMs presented their PM load near to CAL (1). This fact demonstrates that the cluster was balanced (lines in yellow compared to lines in red).

B. Second Experiment: Overload of a VM Network

This experiment started from the balanced clusters generated by the first experiment considering the range of 8%. Then, an overload of a VM network was created simulating a flash crowd event in one VM. The algorithm considered different actions to deal with this situation, such as:

1. When a VM is overloaded, this VM may not be held in its PM, migrating firstly VMs of higher cost (strategy inspired in VOLTAIC);

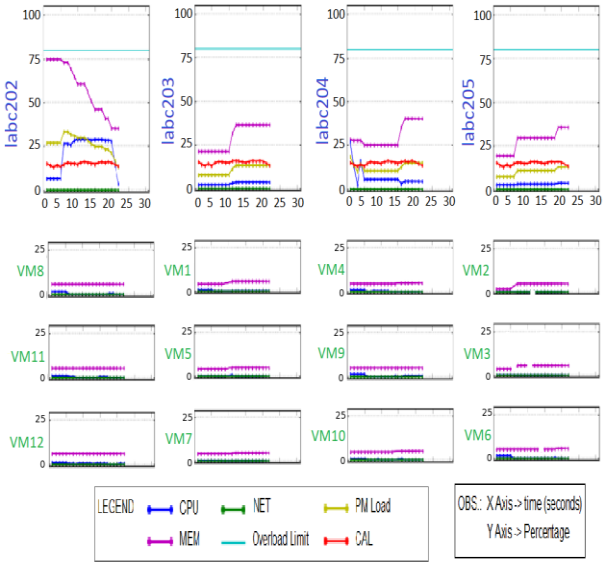

Fig.7. Final Cluster Status for the range of 8%

2. When a VM is overloaded, this VM may not be held in its PM, migrating firstly VMs of lower cost (strategy inspired in Sandpiper); and
3. When a VM is overloaded, this VM is held in its PM, migrating firstly VMs of lower cost (strategy proposed by Phoenix framework);

The first strategy did not isolate the overloaded VM in the original PM. It was observed that, initially, as the overload VM had a relative lower cost related to other VMs, it was migrated. After a period of time, the overloaded VM became the higher cost VM and it was migrated repeatedly among PMs avoiding the correct operation of the cluster, causing damages to all PMs and VMs. In this process, SLAs of several VMs could not be supported. In VOLTAIC strategy [6], it is proposed to allocate, as the final destination a PM that can support all anomalous VMs. But, doing this, SLAs of these VMs allocated in the same PM will eventually be not supported.

The second strategy also did not isolate the overloaded VM in the original PM. It caused unnecessary migrations as observed in the first strategy, jeopardizing the operations of other VMs and PMs. After a period of time, when the overload VM became the highest cost VM and occupied all resources of the host PM, this strategy isolated this VM in one PM.

The third strategy was quite effective, because it isolated the overload VM in a PM since the beginning, avoiding the migration of this VM through the cluster and the negative effects of this migration (overload of different PMs). At the same time, it migrated the others VMs to PMs, allowing their regular operation. Doing this, all VMs have their SLAs respected once they were

allocated to PMs with sufficient resources.

The *iperf* generated a traffic of about 300Mbps, while the VMs network limit was 250Mbps. This caused a network overload at VM8. Fig.8 shows the increase of the network bandwidth usage in VM8 during this experiment. It should be highlighted that labc203 presented a slightly lower PM load than the other PMs.

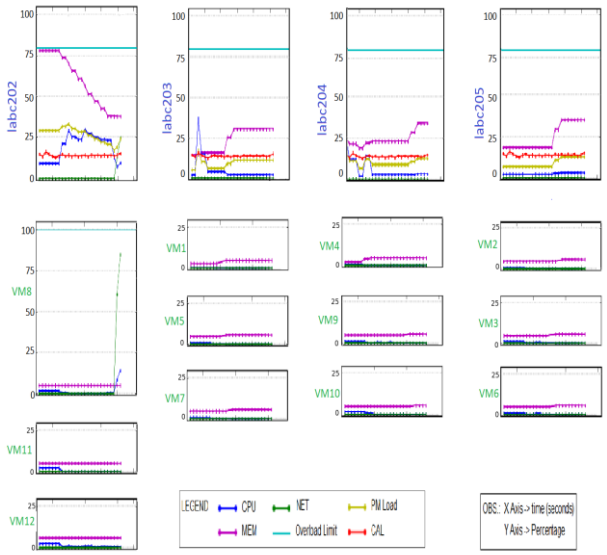

Fig.8. VM Network Overload - Initial State

Once the overload was detected, Phoenix framework started to migrate VMs that were not overloaded. Fig.9 shows the status of the cluster after VM9 was migrated. It is possible to see that the VM8 network limit was reached (100%). Therefore, the labc202 load (showed in yellow) is higher than PMs load in other machines.

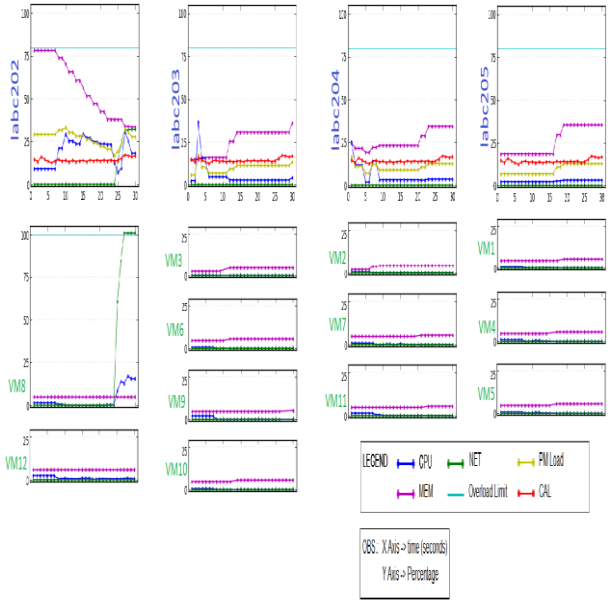

Fig.9. VM Network Overload - Intermediate State

Finally, Fig.10 shows the final status of the cluster after the last migration (VM12). Thus, VM8 was isolated in labc202. It is possible to notice that the two VMs migrated were sent to labc203. It is worth to remember that this machine presented initially a lower load than the others. Because of that, the cluster was balanced after the migration, considering a PM load variation of 8% related to the CAL (1).

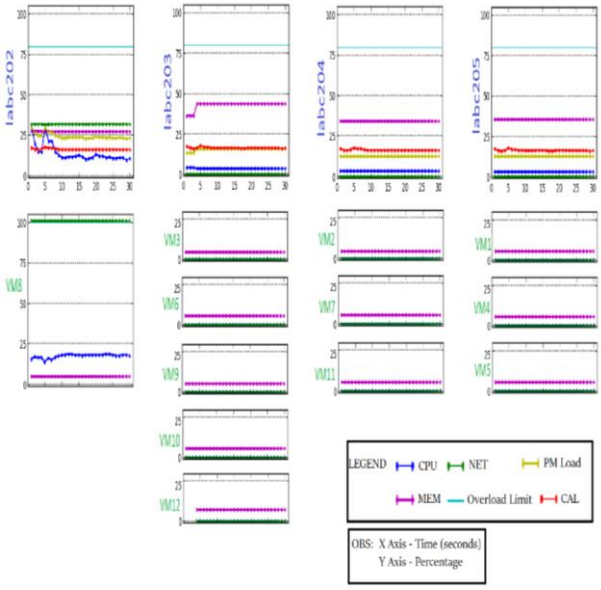

Fig.10. VM Network Overload - Final State

V. Conclusion

This paper proposed the combination of proactive and reactive strategies through a framework called Phoenix. Those strategies minimize the effect of momentary overloads in a cloud environment. Phoenix is divided into three main modules. The first one is the *Monitor* module that interacts with *libvirt* and gets the statistics of each physical and virtual machine monitored and stores this information. The second is the *Analyzer* module that uses the profiles of PMs and VMs obtained through *Monitor* to establish the strategy to be adopted for each situation. The third module is the *Configurator* which implements the strategies defined by the *Analyzer* in virtualized environment using the *libvirt*. The *Analyzer* module performs resource reallocation algorithms. The algorithms analyze PMs and VMs profiles and establish patterns of behavior. Based on observed behaviors, PMs that best suit to the demands of the VMs are chosen. In addition, *Monitor* detects when PMs are in situations of saturation and fire engines that try to ease the burden of PMs to avoid the degradation or even isolate VMs with network overload to avoid breakage of any SLA.

The experimental results showed the heuristics effectiveness and load balancing support for PMs and VMs overload with automatic firing profile survey algorithms, analysis and VMs migrations. Algorithms select physical and virtual machines in a situation of saturation as candidates of better conditions relocations. The results show that the application of heuristics and migration selection algorithms allow the containment of

PMs network feature. Therefore, in the experiments carried out, the proposal proved to be adequated to the reality of cloud computing data centers, ensuring the quality of service. In addition to the proposal for dynamic allocation of resources in the cloud, this architecture brings as a contribution to the implementation of a strategy for containing overloads in virtualized environments. Based on the results, we can conclude that Phoenix is an effective proposal for dynamic reallocation of virtual elements on physical servers.

There are some features that should be highlighted in Phoenix and its competitor's architectures. These features help to minimize the effects produced by the transient overloads on physical and virtual servers hosted on cloud computing environments. It is noteworthy that Phoenix groups a set of features that do not exist simultaneously in other approaches. Highlighting the strategy that isolates a VM, when its network bandwidth usage increases beyond its specification in SLAs. If all network bandwidth of a PM is requested by one VM, the others VMs are migrated.

As future work, we intend to use Multi-Criteria Analysis to better qualify/prioritize CPU, memory and network resources usage. We also intend to study the use of predictive algorithms to analyze and identify profiles of PMs and VMs with larger trends overload in order to minimize the impact of overload on the virtualized environment. Besides, it is possible to perform an optimization in the load-balancing algorithm consolidating VMs in a minimum number of PMs. This would enable the optimization of PMs in order to save energy (green computing).

REFERENCES

[1] Rajkumar Buyya, et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation computer systems, v. 25, n. 6, p. 599-616, 2009.

[2] Rewehel E, M. Mostafa, M.-S. M. A Survey on Load Techniques in Cloud Computing. Internacional Journal of Engineering Research & Technolgy (IJERT), v. 3, n. 2, p. 178–183, 2014.

[3] Mell, Peter, Grance, Timothy. The NIST definition of cloud computing, recommendations of the national institute of standards and National Institute of Standards and Technology, p. 800-145, 2011.

[4] Weber, Andreas et al. Towards a Resource Elasticity Benchmark for Cloud Environments. In: 2nd International Workshop on Hot Topics in Cloud Service Scalability (HotTopiCS 2014). ACM, 2014.

[5] Pan C, M. Atajanov. Flash Crowds Alleviation via Dynamic Adaptive Network. In: Proceeding of Internet Conference 2004. p. 21-28, 2004.

[6] Carvalho, Hugo E. T. Duarte, Otto Carlos. "VOLTAIC: volume optimization layer to assign cloud resources." Proceedings of the 3rd International Conference on Information and Communication Systems. ACM, 2012.

[7] Wood, Timot. et al. Sandpiper: Black-box and gray-box resource management for virtual machines. Computer Networks, v. 53, n. 17, p. 2923–2938, dez. 2009.

[8] Barham, Paul et al. Xen and the art of virtualization. ACM SIGOPS Operating Systems Review, v. 37, n. 5, p. 164-177, 2003.

[9] Eduard Bugnion, et al. Bringing Virtualization to the x86 Architecture with the Original VMWARE workstation. ACM Transactions on Computer Systems (TOCS), v. 30, n. 4, p. 12, 2012.

[10] Boyd, John R. The essence of winning and losing. Unpublished lecture notes, 1996:< http://defence and the national interest. d-n-i-net> Acess in 2017, February 4.

[11] A. Kivity, et al. KVM: the linux virtual machine monitor. Proceedings of the 2007 Linux Symposium, p. 225–230, 2007.

[12] Iperf-fr. IPERF.:The TCP/UDP Bandwidth Measurement Tool. < https://iperf.fr//>.

[13] Zhao Y. Z. Y, Huang W H. Adaptive Distributed Load Balancing Algorithm Based on Live Migration of Virtual Machines in Cloud. 2009 Fifth International Joint Conference on INC, IMS and IDC, 2009.

[14] Amanpreet Kaur, Bikrampal Kaur, Dheerendra Singh,"Optimization Techniques for Resource Provisioning and Load Balancing in Cloud Environment: A Review", International Journal of Information Engineering and Electronic Business(IJIEEB), Vol.9, No.1, pp.28-35, 2017. DOI: 10.5815/ijieeb.2017.01.04

[15] Pooja S. Kshirsagar, Anita M. Pujar,"Resource Allocation Strategy with Lease Policy and Dynamic Load Balancing", International Journal of Modern Education and Computer Science(IJMECS), Vol.9, No.2, pp.27-33, 2017.DOI: 10.5815/ijmecs.2017.02.03

[16] V. Vinothina, R. Sridaran, and P. Ganapathi, "A survey on Resource Allocation Strategies in Cloud Computing," International Journal of Advanced Computer Science and Applications, vol. 3, no. 6, pp. 97–104, 2012.

[17] A. Silberschatz, P. Galvin, G. Gagne, Operating System Concepts Essentials, 2nd, Wiley Publishin g, 2013.

[18] M. Mohamaddiah, A. Abdullah, S.Subramaniam, & M. Hussin. "A Survey on Resource Allocation and Monitoring in Cloud Computing," International Journal of Machine Learning & Computing, vol. 4, no. 1, pp. 31–38, 2014.

[19] Zoha Usmani, Shailendra Singh, A Survey of Virtual Machine Placement Techniques in a Cloud Data Center, In Procedia Computer Science, Vol 78, 2016, Pages 491-498, ISSN 1877-0509.

[20] Sunilkumar S. Manvi, Gopal Krishna Shyam, Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey, In Journal of Network and Computer Applications, Volume 41, 2014, Pages 424-440, ISSN 1084-8045.

[21] Dillon, Tharam; WU, Chen; CHANG, Elizabeth. Cloud computing: issues and challenges. In: Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on. IEEE, 2010. p. 27-33.

[22] Parikh S, M. A survey on cloud computing resource allocation techniques. Nirma University International Conference on Engineering (NUiCONE); Ahmedabad. 2013. p. 1-5.

[23] J. M. Galloway, K. L. Smith, and S. S. Vrbsky. Power Aware Load Balancing for Cloud Computing. In Proceedings of the World Congress on Engineering and Computer Science, volume 1, pages 19-21, 2011.

[24] Sunilkumar S. Manvi, Gopal Krishna Shyam, Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey, In Journal of Network and Computer Applications, Volume 41, 2014, Pages 424-440, ISSN 1084-8045.

[25] Xiao Zhen, Song Weijia, Chen Qi "Dynamic Resource Allocation Using Virtual Machines for Cloud Computing

Environment," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1107-1117, June 2013.doi: 10.1109/TPDS.2012.283.
[26] Kochut Andrzej, Beaty Kirk. On Strategies for Dynamic Resource Management in Virtualized Server Environments. In: 15th IEEE International Symposium on Modeling, Analysis, And Simulation of Computer And Telecommunication Systems, October; 2007. p. 193–200.
[27] Khanna Gunjan, Beaty Kirk, Kar Gautam, Kochut Andrzej. Application performance management in virtualized server environments. In: Proc of network operations and management symposium, 10th IEEE/IFIP; 2006. p. 373–81.
[28] Beloglazov Anton, Abawajy Jemal, Buyya Rajkumar. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener Comput Syst 2011:755–68. Elsevier.
[29] Srikantaiah Shekhar, Kansal Aman, Zhao Feng. Energy aware consolidation for cloud computing. Clust Comput 2009; 12:1–5.
[30] Bobroff Norman, Kochut Andrzej, Beaty Kirk. Dynamic placement of virtual machines for managing SLA violations. In: 10th IFIP/IEEE International Symposium on Integrated Network Management, IM'07. IEEE; 2007. p. 119–27.
[31] Arzuaga Emmanuel, Kaeli David R. Quantifying Load Imbalance on Virtualized Enterprise Servers. In: WOSP/SIPEW'10. ACM; 2010. p. 235–42. January.
[32] Andreolini Mauro, Casolari Sara, Colajanni Michele, Messori Michele. Dynamic Load management of Virtual Machines in a Cloud Architectures. In: Cloudcomp 2009, LNICST 34; 2010. p. 201–14.
[33] Beloglazov Anton, Buyya Rajkumar. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud Data Centers. Concurr Comput: Pract Exper 2012:1397–420. Wiley InterScience.

Authors' Profiles

Edgard H. C. Bernardo received his MSc in systems and computing at the Military Institute of Engineering (IME). He works at the Military Institute of Engineering (IME). His research interest is Cloud Computing focusing Virtualized infrastructures, Algorithms and Systems Development.

Wallace A. Pinheiro received his D.Sc. in Computer Science from Federal University of Rio de Janeiro in 2010. He works at the Centro de Desenvolvimento de Sistemas (CDS), the Center of Technology and Science of the Brazilian Army, in Bras ília, Brazil. His current research interest includes: information retrieval, cloud computing, data quality and data mining.

Raquel C. G. Pinto received his PhD in systems engineering and computing at the Federal University of Rio de Janeiro (UFRJ). She is an associate professor at the Section of Mathematics and Systems Engineering, Military Institute of Engineering (IME).

How to cite this paper: Edgard H. Cardoso Bernardo, Wallace A. Pinheiro, Raquel Coelho G. Pinto, "Phoenix: A Framework to Support Transient Overloads on Cloud Computing Environments", International Journal of Information Technology and Computer Science(IJITCS), Vol.10, No.2, pp.33-44, 2018. DOI: 10.5815/ijitcs.2018.02.04

