VIRTUAL RESOURCE MANAGEMENT FOR DATA INTENSIVE APPLICATIONS IN CLOUD INFRASTRUCTURES

Elena Apostol[1], Valentin Cristea[2]

Cloud computing describes the model of a scalable resource provisioning technology that redirects the possibility of hardware and software leasing to the Internet, through the use of an equitable pay-per-use strategy. In this paper we present a new provisioning mechanism in Clouds used for large-scale data-intensive computing. Our project addresses the key requirements in managing resources at the infrastructure level. The proposed resource manager allocates virtual resources in a flexible way, taking into account virtual machines' capabilities, the application requirements and other defined policies and it provides on-demand elasticity. We used a QoS constraint base algorithm in order to maximize the performance, according to different defined policies. New resources can be dynamically allocated on-demand or policy-based. This is one of the novel contributions of this paper, as the majority of the cloud resource managers provides just static allocation.

In order to demonstrate the performance of the presented resource managing mechanism we provide and interpret several experimental results.

Keywords: dynamic allocation, policy, virtualization, management, Cloud infrastructure, data intensive

1. Introduction

Cloud Computing is emerging as a mainstream technology that provides virtualized physical resources and infrastructure. It is largely adopted among IT enterprises, such as Google, IBM or Amazon and it is used by millions of clients all over the world.

The fundamental concept of Cloud Computing is virtualization. Users are able to request virtual machines, with specific capabilities and then use them as needed without interruption and having total control over them.

In this paper we describe a virtual resource manager deployed in Infrastructure as a Service (IaaS) clouds that provides policy based services such as resource intensive applications. For such applications the amount of managed data exceeds the order of petabytes, so allocating accordinly with defined policies is very important.

An important component of the resource manager is the scheduler. At the infrastructure level, a scheduler is in charge of the mapping between pending virtual machines and physical nodes. Scheduling at virtual machine level is a relatively new approach and there are few implementations in this area. The process of scheduling in Cloud Systems raises

[1]Assistant professor, University "Politehnica" of Bucharest, Computer Science Department, e-mail: elena.apostol@cs.pub.ro

[2]Professor, University "Politehnica" of Bucharest, Computer Science Department

several issues. The first issue is regarding the type of scheduling algorithm that must be used in order to satisfy the user specifications. A second issue regards the performance criteria that must be met, such as load balancing or scalability. Furthermore, the preemption methods used in Grids do not apply in the same way in Clouds Systems, because usually a virtual machine resource once granted must not be interrupted for the requested duration. This becomes more of an issue when a large number of users request concurrently large amount of resources .

This paper proposes a solution to the problems discussed above regarding virtual resource management in Cloud Systems. We developed a resource manager for a cloud environment that filters users' requests accordingly to defined policies, schedules the requested virtual machines and deploys them in the Cloud.

The rest of this paper is organized as follows. In Section 2 we present related work based on the virtual resources management paradigm. In the third section the architecture for the proposed resource management mechanism is described. We continue, in Section 4, with presenting the algorithm used for scheduling the virtual resources. Section 5 presents the test scenarios and the analysis of experimental results.This paper concludes with a summary.

2. **Related Work**

There are several cloud software platforms that are doing the resource management at the infrastructure level. Examples of such of platforms include OpenNebula [2], Eucalyptus [3] and Nimbus [4]. In this sections we will present several resource provisioning mechanisms used by those systems. Most of the cloud software platforms use basic scheduling mechanisms or they do not do any real form of scheduling.

For the process of scheduling, OpenNebula uses advance reservation, preemption scheduling and VM placement constraints for balance of workload. The Match-making Scheduler (MMS) is the OpenNebula build-in scheduler module. The MMS first filters out the hosts that do not meet the virtual machine requirements. Based on the information gathered by the monitor drivers, a rank is assigned for each resource in the resulting list. The resources with a higher rank are used first to allocate virtual machines. A more complicated rank expression can be used by choosing one of the rank policies available.

Eucalyptus [3] uses a basic scheduling mechanism, that supports three types of policies: Greedy or First fit, Round robin and Power Save policy. When using the Greedy algorithm the scheduler will map the virtual machines to the first found available node. On the other hand, the Round Robin policy implies that VMs are deployed on nodes in a round robin manner.

Based on a series of conducted experiments, we concluded that our approach has a better average utilization rate than any of the above schedulers.

A solution with a better performance for the VMs scheduling algorithm is described in paper [5]. The paper presents an optimized genetic algorithm for scheduling VMs. The algorithm uses the shortest genes and introduces the idea of Dividend Policy in Economics. We can not say nothing about the way their solution behaves when the scheduler receives a large number of requests, but on a small scale we compared their results with the ones obtained by us and we observed that our utilization rate is similar to the one obtained by them.

Another idea described in our paper is policy-based allocation. Most of the cloud platforms use simple resource allocation policies. Haizea [1] is the first resource manager that offers dynamic policies to support advanced reservation. The new type of resource allocation in Haizea that uses policies for allocation is called Deadline Sensitive (DS) leas allocation and is based on swapping and preemption mechanisms. An improved algorithm is proposed in [6]. It requires rescheduling of fewer requests compared with the original Haizea algorithm. As a limitation, this solution consider that a VM request demands all of the host machine resources.

The withdraw is that the presented algorithms don't offer the possibility to specify other types of policies. Also they don't offer a more flexible resource management by relocating already running virtual machines through live migration.

3. The Proposed Architecture

In this section we do a briefly description for each component of the proposed architecture.

Figure 1 depicts the high level architecture for our resource provisioning solution. [8] [7]

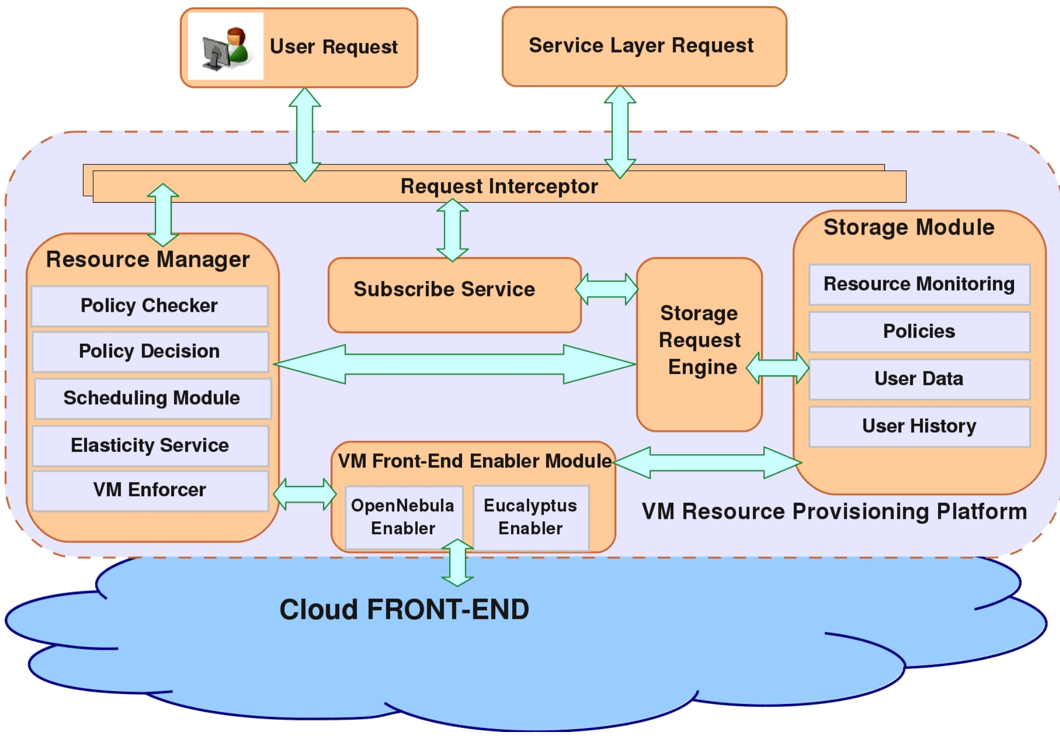

FIGURE 1. Resource Manager Architecture

There are two main modules of the architecture:

- The *Resource Manager* contains the main logic of the architecture. It receives requests from the users or from high level services, and creates a reservation for the requested resources. Furthermore, it continually checks the status of the available resources and provides elasticity in a dynamic manner to improve the overall efficiency.
- The *Storage Module* has a dual role. It stores the available policies as well as monitoring data for the user requests and resource status.

In the rest of this section we describe in detail each of the modules, their structure and functionality.

Its entry point of the proposed provisioning model is the *Request Interceptor* which takes the customer request and processes it. This component is responsible for the validation of the customer request. This involves checking the fact that the customer has authenticated and that the request refers to existing resources, where the customer has access rights.

The customer can be a user or a high level service and it must have an account on this platform. Otherwise the customer must first create an account by using the subscription mechanism provided by the *Subscribe Service*. At this stage a customer can also choose from a set of access and usage restrictions policies.

The *Request Interceptor* acts also as a gateway for the customer requests. It will send the VMs requests to the *Resource Manager* and the subscription messages to the *Subscriber Service*. The messages exchanged between those three components use a XML format. A VM request message provides the capability of customizing the virtual machines with parameters such as hardware demands, start time and end time, dependencies with other virtual machines and charging.

The *Decision Module* is the core of our architecture. It is responsible for deciding if the customer has the right to request the corresponding virtual resources and if the request can be fulfilled. It consists of five modules: *Policy Checker*, *Policy Decision*, *Scheduling Module*, *Elasticity Service* and *VM Enforcer*.

In the request processing chain, the next stage is the *Policy Checker*. It receives the available policies from the *Storage Request Engine* which acts as an intermediary in the communication with the *Storage Module*. This component requests the available policies and determines the constraints that will be used for each resources request in order to meet the user's expectations.

The list of constrains is received by the *Policy Decision* module. It analyzes the policies and compacts them in an aggregated format. Based on the policy interpretation, it generates an identifier denoting which type of resource allocation mechanism to use. This identifier is based down to the *Scheduling Module*. Thus depending on the command received from the *Policy Decision* it can use best effort based and QoS constraint based scheduling. A best-effort based scheduling algorithms must minimize the execution time, ignoring other factors, such as the cost of accessing resources. The second type of resource allocation algorithm attempts to maximize the performance, according to different QoS constraints. The *Scheduling Module* will also receive the list of virtual resources to be allocated. In the case the *Policy Decision* module decides that the request is permitted, it completes the corresponding scheduling algorithm and the associated actions are send to be executed by the VM Enforcer module. Otherwise no actions are executed and a rejection notification is send to the customer. Within this module we designed and implemented a new efficient scheduling algorithm. We will describe this algorithm in the next section.

Another novelty of our solution is the *Elasticity Service*, which provides automatic elasticity by adjusting the resource usage. This module receives information regarding current resource usage and enforces the policies for each active list of virtual resources. It can request the preemption or live migration of the one or several virtual machine currently running. The requests are directly passed to the *Policy Decision* module and then to the *Scheduling Module*.

The *VM Enforcer* module communicates with the *VM Front-End Enabler* module via a XMLRPC interface. It calls the appropriate PRC procedure for creating, deploying and migrating the VMs. It is also responsible with updating all the information that the *Scheduling Module* requires in order to apply the scheduling algorithm, such as the list of running virtual machines, the list of the virtual machines scheduled in the future, the list of available hosts (that have the ready state). Since the *VM Front-End Enabler* module is the only component communicating directly with the Cloud front-end, it can be easily adapted for use with OpenNebula, Eucalyptus or any other type of Cloud, simply by adding another Enabler to this module.

The *Storage Module* stores the available policies and monitors events and resource from the cloud system. It consists out of four separate components: the *Resource Monitoring* module, the *Policies Repository*, the *User Data* module and the *User History* module.

The *Resource Monitoring* module stores the information regarding VMs and the physical machines that are running in the Cloud System. The stored information is used to determine the current VMs resource usage.

The *Policies Repository* contains two categories of policies: general policies and per user policies. The first set of rules define how to manage and control access to resources. The purpose of the second set of policies is to insure enough available resource to satisfy the requests of the subscribed customers, in terms of the SLAs.

4. The Improved Scheduling Algorithm Description

Scheduling is one of the key issues in the workflow management, especially in the cloud systems. It is the process that maps and manages the execution of inter-dependent tasks on the distributed resources.

For our implementation we used an improved genetic algorithm. The description of the scheduling algorithm in pseudo-code and the flow of the scheduling process is presented in the next lines 4.1.

Algorithm 4.1. *:*

```
Input: hosts = list of available computing hosts
Input: runningVMs = list of currently running virtual machines
Output: scheduledVMs = list of scheduled VMs (not running)
 1: while true do
```

```
2:     checkForVMExpiration()
3:     getListOfPendingVMs()
4:     getListOfUsers()
5:     removeVMsAccordingToPolicy(pendingVMs, users)
6:     if $pendingVMs.size() < 1$ then
7:        waitForWork()
8:        continue
9:     end if
10:    calculateVMRanks(pendingVMs)
11:    scheduleSolution = doGenetic(pendingVMs, hosts, policies)
12:    for each valid mapping of the solution do
13:       populate the scheduled list
14:    end for
15:    deployVMs(scheduledVMs)
16:    waitForWork()
17: end while
```

The scheduler implements a timer that raises a Request event at predefined intervals. The flow of the scheduling algorithm consists of the following steps:

(1) **Step 1.** When the event is triggered, the scheduler checks if the currently running virtual machines must be stopped (their running interval expired).

(2) **Step 2.** Interacts with the Cloud Front-End in order to receive the list of virtual machines which are in the pending state. The pending state indicates that the virtual machines are waiting to be deployed.

(3) **Step 3.** For every virtual machine the scheduler requests from the Subscriber Server the corresponding user information.

(4) **Step 4.** It checks if every virtual machine matches the associated set of policies. If a virtual machine does not fulfill a policy it is removed and an appropriate response is returned to the user who made the request.

(5) **Step 5.** In this step the scheduler calculates a rank value for each pending machine. The rank value indicates the priority of the corresponding virtual machine. It is calculated according to four rules:
   (a) If the user choose a better subscription, his virtual machine will have a greater rank value than any other virtual machine of an user who has a more restrictive subscription.
   (b) In case of virtual machine dependencies the parent virtual machine has a greater rank value than the child's virtual machine rank value.
$$R_{parent} > R_{child}$$
   (c) A child virtual machine has a higher rank value than an independent virtual machine.
   (d) In case of subscription equality the FCFS policy is adopted.

After the rank value is computed, the list or pending virtual machines is sorted descending and then grouped according to a hybrid heuristic.

(6) **Step 6.** The handler for the genetic algorithm is called. It uses as input the pending list of virtual machines, the list of available hosts and the list of user policies. The solution of the algorithm represents the best individual. The individual has a list of mappings (vm, host, start_time). If the mapping is valid then the virtual machine can be deployed on the associated host.

(7) **Step 7.** The scheduled list represents the virtual machines which need to be deployed at their start time. In this step the scheduled list is being populated according with the solution provided by the genetic algorithm.

(8) **Step 8.** The virtual machines are deployed if their start time is equal with the current time.

In the following subsections we will illustrate in more details the chromosome encoding, the fitness function and the genetic operators used by our algorithm.

4.1. Chromosome Encoding

Each chromosome represents a solution to the scheduling problem. The Policy Decision Module checks periodically if new requests are available. If they are available it creates two lists: a list of pending virtual machines and a list of available hosts. In our case the chromosome can be seen as a sequence of mappings. The mapping contains a specific virtual machine, a host and a time value (vm_i, h_j, t), representing that the virtual machine vm_i will be deployed on host h_j when the time value t is met. The significance of the indexes is the following: i is the position of the virtual machine in the list calculated by the Policy Decision Module; j is the position of the host in the list. The virtual machines list is sorted on the rank value of each virtual machine. The rank value is based on the capabilities of the virtual machine (cpu, memory and disk) and on the policy of the user who requests the virtual machine.

4.2. Fitness Function

The objective function (or fitness function) measures the quality of an individual within the population according to some criteria. In our case, for the scheduling problem, the idea is to map virtual machine on hosts and to ensure maximum resource utilization, a well balanced load across all hosts or a combination of these.

The fitness function for an individual depends on all the mappings. In this case being the sum of the evaluation function of each mapping.

$$F_{individual_i} = \sum_{i=0}^{nM} f_M(M_i)$$

In the above formula nM is the total number of mappings the individual has which is equal with the number of the pending virtual machines. f_M is the evaluation function for a mapping and the M_i variable is the current mapping.

The evaluation function of a mapping depends on the resources the host has available in time. For example if three virtual machines are scheduled to run today, tomorrow and the day after tomorrow the host's available resources must be reduced in these periods of time according to each virtual machine. To implement this we used a gap time vector for simulating the allocation in time of the virtual machines.

A time partition reflects the available resources the host has in a specific moment of time. Therefore we define the evaluation function of a mapping as being the sum of the evaluation functions of each time partition.

$$f_M = \sum_{i=0}^{nTP} f_{TP}(TP_i)$$

In the above formula nTP is the total number of time partitions the mapping has, f_{TP} is the evaluation function for a time partition and TP_i variable is the current time partition.

The evaluation function of a time partition depends on the available resources the host has in the corresponding moment of time. We have considered that a virtual machine needs cpus, memory and hard disk space. Also each type of resource has a weight value associated with it. A virtual machine cannot run if there is no disk space or memory. Also, although the processor can be shared it's not recommended to allow more virtual machine share the same processor or core. The role of the weight value is to make an equilibrium between these three resources.

$$f_{TP} = w_{cpu} * available\_cpu + w_{mem} * available\_memory + w_{disk} * available\_disk$$

In the above formula, w_{cpu}, w_{mem} and w_{disk} are the weights associated with the three type of resources.

4.3. Population Initialization

The initial population will be generated according with the two lists received from the Policy Decision Module, a list with the pending virtual machines and a list with available hosts. The list of virtual machines is sorted on the rank of each virtual machine. The rank is based on the capabilities of the virtual machine and on the priority of the user who made the request. The population will be generated using random numbers representing indexes in the list of hosts. The initialization can be corrected to make some adjustments regarding the virtual machine specifications.

4.4. Genetic Operators

We needed to define the genetic operators in order to build the new generation and to select the pairs of individual for recombination. Also mutation can occur with a specific probability. In the following paragraphs the operators are presented.

Mutation. Every new chromosome has a certain probability for mutation. In our case the mutation operator acts as follows. It picks the new individuals created with the crossover operator and then with a probability of 5% decides whether to modify their genes or not. The process of altering the genes consists of selecting a random mapping of the individual and to randomly change the host where the virtual machine has to be scheduled.

Crossover. Each gene of the child will be randomly selected from the two parents. An example of crossover is presented in Figure 2.

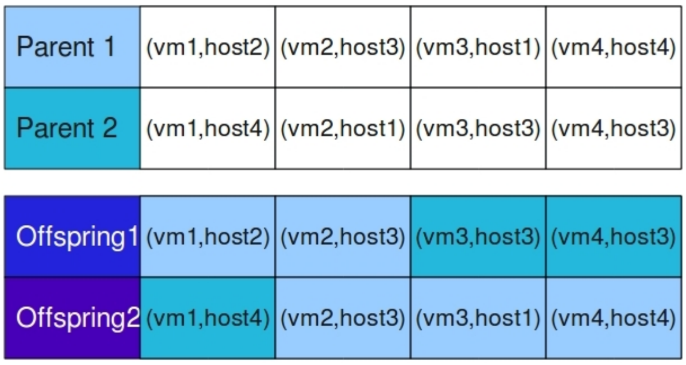

FIGURE 2. Random Crossover Operator

Selection. In order to generate the solution faster, we used an adaptation of the *tour selection*. We selected the best *n* individuals according with their fitness. One of them will be the first parent. The second parent is selected randomly from all the population.

4.5. Policy Definition

The proposed solution uses a set of defined policies to restrict access to the provisioning of VMs and to organize internal functionalities, such as live migration. The main purpose is to insure enough available resource to satisfy the requests of the users, in terms of the SLAs.

Policies are represented in an XML-based format and are stored in the Policies Repository. Policies are received from the Storage Module in a aggregated format. An example of an aggregated policy is shown in Figure 3 :

```
<policyInputTemplate >
        <policyIdentifiers>
             <identifier id="usermane" />
        </policyIdentifiers>
        <targetService name="VirtualMachineManagement">
            <targetServiceOperation name="operation_for_which_the_policy_is_appplied">
                <operationParameter name="parameter_name1">parameter_value1</operationParameter>
                .
                .
                .
                <operationParameter name="parameter_nameN">parameter_valueN</operationParameter>
            </targetServiceOperation>
        </targetService>
</policyInputTemplate>
```

FIGURE 3. Aggregated Policy Format

The structure of the aggregated policy format is the following:

- The *policyInputTemplate* tag incorporates all the information related to the policy.
- The *policyIdentifiers* tag contains the username of the user for which the policy is applied. This is the same user that made the current resources request.

- The *targetService* tag identifies the service for which that policy is valid.
- The *targetServiceOperation* tag identifies the operation associated with that policy. The main operation that we considered in our implementation of policies is the deployment of virtual resources.
- The *operationParameter* tag represents a parameter of the policy. An example of parameter is the amount charged from the user for that specific service operation.

The Policy repository consists of two sets of policies. The first set represents the general policies. They are static policies that specify basic rules that are defined by the cloud administrator. An example for a generic policy can be a rule that implies energy saving management. Energy efficiency is important and can be achieved in cloud environments using an efficient resource allocation mechanism, reallocation virtual machines if necessary and switching off idle nodes. The second set of policies is defined according to user specifications.

5. **Experimental results**

In this section we present several test sets, along with the infrastructure configuration.

5.1. **Infrastructure and Hardware Configuration**

The testbed contains the Cloud nodes and the Front-End. Our application will reside on the Front-End, along with the Apache server running the web application with the user interface that allows users to subscribe and make VM requests. The front-end has RedHat Scientific Linux as OS, a MySQL database, the Apache server and the OpenNebula middleware. The hardware configuration for the front-end consists of: 1 GB of memory, Intel Core 2 Duo E8200 processor with 2.66 GHz frequency, and a hard-disk capacity of 500 GB.

There are ten Cloud nodes, each with Xen Hypervisor installed and running. Their hardware configuration is: 1GB of memory, Intel Core 2 Duo processor and a 250GB hard-disk. The Cloud nodes and the Front-End are connected through bridges.

5.2. **Data Interpretation**

Using this scenario, we analyzed the performance of our Resource Manager and we compared our scheduling algorithm with the one used by Haizea.

In Figure 4 is presented the dependency between the number of virtual machine requests and the number of generations required to find the scheduling solution.

For the same number of virtual machine requests, we analyzed the evolution time for our algorithm and we concluded that our time is much better than the average time of a traditional genetic algorithm. Figure 5 depicts the corresponding results.

We also conducted tests to discover how much resources our scheduler wasted when allocating VMs in comparison with Haizea. The results are depicted in Figure 6. As we can observe, the utilization rate is almost always better than the one used by Haizea.

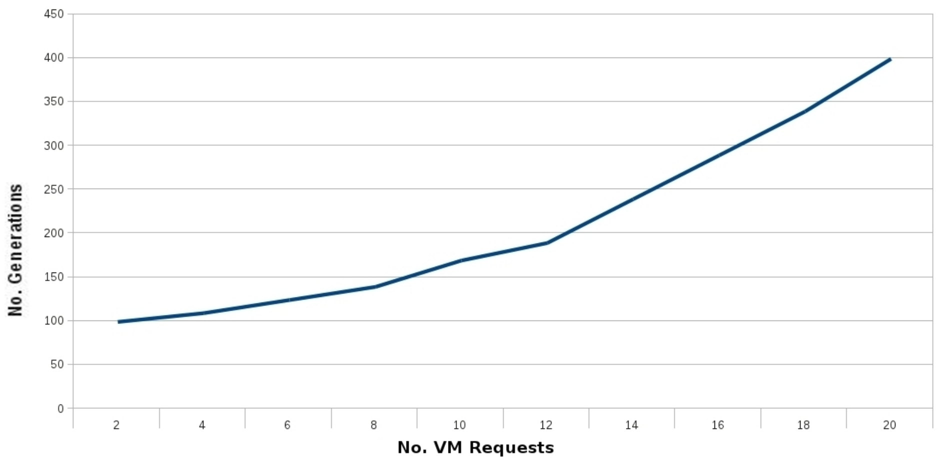

FIGURE 4. Dependency between the number of VMs requests and the number of generations

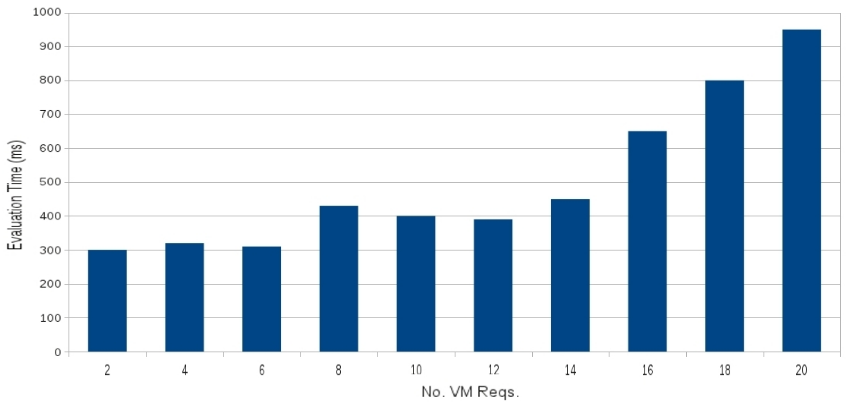

FIGURE 5. Evolution time for the proposed scheduling algorithm

6. Conclusions

The recent appearance of Cloud Computing comes with new scheduling methods, adapted to the user requirements and to the Cloud infrastructure.

In this paper we proposed a new approach for a dynamic resource allocation manager. The solution proposed is efficient and has a good utilization rate.

The entire architecture is independent from the Front-End. In order to integrate the resource manager with other Front-Ends the only module that needs to be changed is the Policy Enforcer Module because it is dependent with the management platform.

A genetic algorithm was implemented in order to find the best scheduling solution. Also, the algorithm besides the virtual machine capabilities is considering well defined

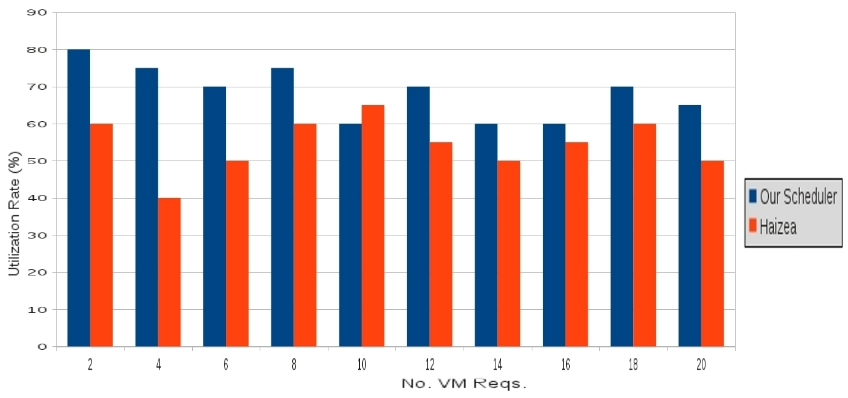

FIGURE 6. Comparison between our and Haizea Utilization Rate

policies in order to find the proper planning solution. The algorithm is optimized for performance and scalability.

The resource manager supports many virtualization platforms, such as Xen, KVM and VMware. The choice for a virtualization solution depends on the available hardware equipment.

REFERENCES

[1] *Borja Sotomayor, Ruben Santiago Montero, Ignacio Martin Llorente, and Ian Foster*, "Capacity Leasing in Cloud Systems using the OpenNebula Engine", in Cloud Computing and Applications, **vol. 2008**, 2008

[2] Open Nebula Home Page *http://www.opennebula.org/*, November 2012

[3] Eucalyptus Home Page *http://www.eucalyptus.com/*, November 2012

[4] Nimbus Home Page *http://www.nimbusproject.org/*, November 2012

[5] *Hai Zhong, Kun Tao, Xuejie Zhang,* "An Approach to Optimized Resource Scheduling Algorithm for Open-source Cloud Systems", in ChinaGrid Conference, Guangzhou, China, July 2010, pp. 124-129

[6] *Amit Nathani, Sanjay Chaudharya, Gaurav Somanib,* "Policy based resource allocation in IaaS cloud", in Future Generation Computer Systems, **vol. 28**, no. 1, January 2012, pp. 94-103

[7] *Elena Apostol, Iulia Baluta, Alexandru Gorgoi, Valentin Cristea,* "Efficient Manager for Virtualized Resource Provisioning in Cloud Systems", in IEEE International Conference on Intelligent Computer Communication and Processing, Cluj-Napoca, Romania, August 2011, pp. 511-518

[8] *Elena Apostol, Catalin Leordeanu, Valentin Cristea,* "Policy Based Resource Allocation in Cloud Systems", in 2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Barcelona, Spain, October 2011, pp. 285-289