Policy-based Cloud management through resource usage prediction

CATALIN LEORDEANU and SILVIU GRIGORE and OCTAVIAN MORARU and VALENTIN CRISTEA, University Politehnica of Bucharest

Cloud computing services are becoming increasingly more widespread, mainly because they offer a convenient way of using remote computational resources at any time. Constantly satisfying client needs is a difficult task due to the limited nature of the physical resources. Careful handling of computing capabilities is critical. Cloud systems offer resource elasticity, which is essential for respecting Service Level Agreements (SLAs) or other types of contracts. This paper proposes a novel solution which offers an efficient resource management mechanism for Clouds. The solution is based on monitoring hosts belonging to the Cloud in order to obtain load data. A policy-based system uses the monitoring information to make decisions about deployment of new virtual machines and migration of already running machines from overloaded hosts. The policy-based solution is enhanced by prediction algorithms to optimize the resource usage and to make sure that the available hosts are capable of handling the increased load before it happens. This leads to more efficient resource usage and can help fulfill the SLA requirements even under heavy loads.

Categories and Subject Descriptors: D.4.7 [**Operating systems**]: Organization and design—*Distributed systems*

Additional Key Words and Phrases: Cloud computing, Resource management, Policy management, SLA, Prediction

ACM Reference Format:

Cătălin Leordeanu, Silviu Grigore, Octavian Moraru, Valentin Cristea. 2014. Cloud Management through Resource Usage Prediction *ACM Trans. Appl. Percept.* 2, 3, Article 1 (May 2010), 11 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Cloud computing is an emerging paradigm that provides remote access to computational resources and storage to end-users. In the last few years, Cloud systems have become increasingly popular. The expansion of Cloud technologies is caused both by small to medium businesses that prefer renting computing capabilities over buying them, and by end-users accessing services located in Clouds. The current trend for companies is to shift their service systems into the Cloud, unburdening themselves from the cost of purchasing and maintaining equipment. Also, third party service providers prefer offering their services using Cloud systems, accessible over the Internet, through desktop or mobile apps.

Computing resources offered by a Cloud can range from application software or services to virtual machines, servers, data storage or even entire networks. The usual approach in Cloud systems is to divide the provision of services into three layers of abstraction: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). In the IaaS model, Clouds provide direct access to physical resources, usually through virtual machines. [Fox et al. 2009]

Cloud systems can be divided into two categories: public and private. In public Clouds, resources are made available in a pay-per-use manner. The consumer and the Cloud service provider must agree upon the terms on which the quality and reliability of the services must be assured. After a negotiation process, a contract called Service Level Agreement (SLA) [Dillon et al. 2010] is usually signed. The SLA contains different quality of service attributes such as average/maximum application response time or hourly cost that must be enforced in order to fulfill the contract. Private Clouds usually refer to organization data-centers, where resources are not made available to the public, but are destined rather to internal usage.

Resource management is a very important and complex process for both business oriented and technical fields. In order to achieve responsiveness, a Cloud system must be provided with an efficient resource management mechanism. If we refer to the IaaS level of abstraction, virtual machines must be deployed on the right hosts, at the right moment of time.

A fundamental approach in Cloud systems is reusability. Cloud consumers rent computing capabilities that must always be delivered. Not all rented resources are used at all time. Unused resources must not go to waste, hence an adequate resource manager must figure out what are the unused resources, and reuse them in order to fulfill every SLA at any given time. OpenNebula[Sotomayor et al. 2008] is an open source cloud computing framework that aims to easily build and manage private cloud infrastructures. It offers multiple layered APIs allowing the user or developer to choose the degree of complexity of the cloud functions he/she wishes to use.

This paper presents an efficient, adaptable and easily extensible mechanism for managing virtual machines in an OpenNebula environment. The framework uses a policy-based system to make decisions regarding deployment of new virtual machines or migration of already running VMs. The policies refer to different parameters of the hosts that describe the amount of resource utilization, enhanced using prediction algorithms.This way, uniform utilization of the resources in the Cloud is guaranteed and frequent overloading of the hosts is prevented.

The rest of this paper is organized as follows. Section 2 describes other research related to the subject of this paper. In Section 3 we present the cloud monitoring and policy enforcement mechanisms and also contains details about the resource usage prediction which enhances the policy-based Cloud management solution. In Section 4 we describe our testbed and show various experiment which validate the proposed solutions. Finally, Section 5 draws conclusions and proposes directions for research beyond the contents of this paper.

2. RELATED WORK

Efficient management of comutational resources is a subject which has been the focus of many research projects in the past[Buyya et al. 2002] [Schwanengel et al. 2013].

Many projects tackle the problem of dynamically overlaying virtual resources on top of physical resources by using virtualization technologies, and do so with different resource models. The most widespread Cloud infrastructure is the Amazon Elastic Compute Cloud (EC2)[Ou et al. 2012] is a central part of Amazons cloud computing platform. EC2 allows users to rent virtual computers on which to run their own computer applications. EC2 allows scalable deployment of applications by providing a web service through which a user can boot an Amazon Machine Image to create a virtual machine, containing any software desired. An user can create, launch, and terminate server instances as needed, paying by the hour for active servers, hence the term elastic.

The purpose of Cloud resource managers is to obtain the maximum of performance with existing resources. The resource scheduler needs to maintain an optimum balance and make sure that there are no overloaded resources or that systems are kept idle, which would lead to a waste of energy. To maintain this balance, the resource manager needs to have access to real time monitoring data, as well as an estimation of future requirements, in order to take the best decisions. Other projects use prediction algorithms to further optimize resource allocation. The Network Weather Service[Wolski et al. 1999] is such a solution which monitors and predicts the performance of computational resources and computer networks. The predictions are based on collected monitoring data. It is a modular system containing a name server, sensors, predictors and persistent memory, all communicating with eachother through TCP sockets.

Another approach related to this paper is the Network Bandwidth Predictor [Eswaradass et al. 2006]. It estimates the bandwidth of a path between two nodes in a network by sending a small packet and measuring the round trip time. This data is then collected and used to train a neural network-based prediction module. The neural network is a simple backpropagation model and is able to accurately predict the available network bandwidth for certain network paths.

3. CLOUD RESOURCE MANAGEMENT ARCHITECTURE

The purpose of the proposed solution is to deliver an efficient, adaptable and easily extensible framework for managing virtual machines in an OpenNebula environment. The framework uses a policy-based system to make decisions regarding deployment of new virtual machines or migration of already running VMs. The policies refer to different parameters of the hosts that describe their usage in the last period of time. This way, uniform utilization of the resources in the Cloud is guaranteed and frequent overloading of the hosts is prevented.

The proposed architecture is shown in Figure 1. The framework itself consists of three interlinked modules. The Cloud is an inherently dynamic environment, so careful and continuous monitoring of host load parameters is a must. The *Monitoring module* covers this aspect. Secondly, a policy-based system must be provided to help cloud administrators enforce different host loads requirements. The system periodically checks if any general or host-specific policies become active. It also makes decisions in two types of situations: when a request for deployment is made, it chooses the right physical machine that should host the new VM. If one or more policies are triggered (i.e. a host is overloaded), it picks a virtual machine running on that host and an available destination host for it. This job is done by the *Policy module*. Finally, the *Resource Management module* interacts with the OpenNebula daemon, implementing the decisions made by the Policy module.

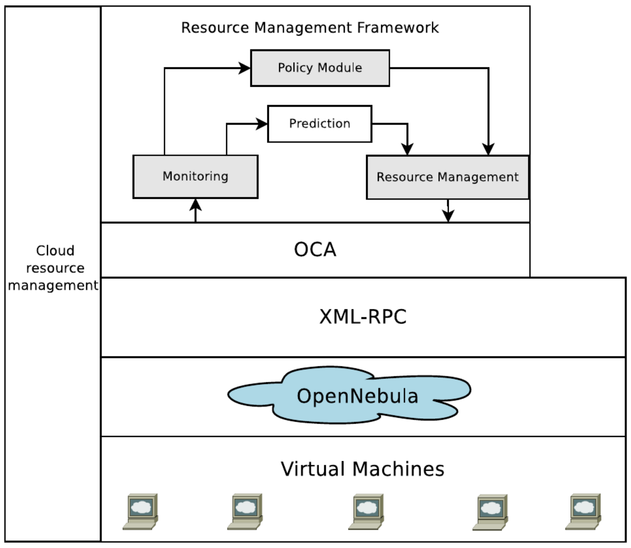

Fig. 1. Resource management architecture

In addition, the data extracted from the Monitoring module is fed to a Prediction module, that guesses future loads of the machines. The purpose of this component is to make the system react to possible host overloads immediately, or even before they actually happen. The framework is built on top of the Java OpenNebula Cloud API (OCA), which is used to access the needed OpenNebula core functions. OCA is merely a convenient wrapper for the XML-RPC methods exposed by OpenNebula.

3.1 Performance monitoring

The Monitoring module gathers information about hosts and virtual machines. The monitored parameters are cpu/memory/disk usage and network traffic. We obtain this information through the Java OCA API, by polling the OpenNebula daemon, at a certain time interval. By configuring OpenNebula to get the information from the hypervisors at a similar interval, we can obtain relevant data useful to the Policy module and to the prediction algorithms. OpenNebula deploys a number of scripts to the physical machines, used to obtain load data, which are remotely executed at each monitoring interval. In our testbed, we installed the KVM virtual machine monitor[Kivity et al. 2007] on all hosts.

At this point, the OpenNebula core contains the desired monitoring information. We can extract this data programmatically by interrogating the ON daemon via the OpenNebula Cloud API. Extracted data must be stored in order to maintain a complete history of the loads. This is necessary in the interest of having a well functioning Policy module. Moreover, the Prediction module relies on an extensive history of host and virtual machine loads. Monitored data is stored in a PostgreSQL[Momjian 2001] database. We created tables for hosts, virtual machines and templates which contain mainly static information, inserted only once per monitoring session. Additionally, Host_Monitor and VM_Monitor tables contain the actual monitoring information, inserted each time the OpenNebula daemon is polled.

3.2 Policy-based resource management

Cloud administrators can supply policies in order to specify the systems behavior in certain situations. Policies can be applied in two cases: when a virtual machine deployment request is issued, the suitable host on which the VM will run is chosen in respect of the defined policies; these are the deployment policies. For example, when a host becomes increasingly overloaded, one or more guest VMs are migrated to other hosts. The overload threshold is defined by the migration policies. The Policy module periodically checks if any general or host-specific migration policies are triggered.

3.2.1 *Policy structure.* A policy is composed of three parts: the target, the policy type and the condition. The target specifies the name of the host that the policy applies to. The name must be the unique OpenNebula identifiable host alias, that can be obtained using the *onehost list* command. If we wish to apply the policy to all the hosts in the Cloud, we can specify the all keyword as a target.

The policy type can be one of the two keywords *deploy* and *migrate*. If we specify both a general (i.e. using the all keyword) and a host-specific policy (i.e. using the hosts name) for the same type of policy, only the host-specific one is considered for that host. A condition must specify a parameter, an operator and a value. Parameters may be any of the monitored parameters described in the previous section: used cpu, used memory, used disk space or network traffic. The keywords are: cpu, mem, disk and net. Operators may be any type of comparison operators and work as expected. The value acts like a threshold for the corresponding parameter and must be given as a percentage for the first three parameter types and as an absolute value for the network traffic parameter (bits per second). Conditions may be chained using the OR operator (||), the AND operator (&&) and round parenthesis.

It is necessary to have separate types of policies for deployment and migration actions. If we were to consider only deployment policies, we could not avoid unpredicted host overloads. For example, if guest virtual machines would become cpu-intensive, there would be no way of unburdening the host and responsiveness would suffer. On the other hand, if only migration policies were to be considered, new virtual machines could be deployed on already overloaded hosts. The VM would then be migrated because of a migration policy would activate, but an unnecessary transfer would have been made. In addition, there would be no guarantee that the new host would be suitable for the VM.

As mentioned before, conditions can be chained into more elaborate policies. For example, a complex migration policy can take the following form: $(cpu\ 80\&\&mem\ 70)\|net >= 10000000$. Here, a virtual machine should be migrated if both its host cpu and memory become highly loaded or if the host generates constant high network traffic (more than 10 Megabits per second).

The policies are specified in a configuration file and can be reloaded at any time when the Policy module is running. A complete configuration file is described below:

```
1 all   deploy mem < 60
2 all   migrate cpu >= 80
3 on2   deploy cpu < 80 && mem < 60
4 on3   migrate cpu >= 90
5 on3   deploy mem < 90
```

The first line specifies a deployment policy for all hosts in the Cloud. New virtual machines should be deployed on hosts having less than 60% of their available memory used. All hosts must comply to this policy, except on2 and on3. on2 defines an even stricter deployment policy, requiring also at least 20% of free cpu to accept other virtual machines. On the other hand, on3 relaxes the global policy. This could happen in the case of hosts having more physical memory than the others. The global migration policy (defined on line 2) specifies that virtual machines should be migrated if the cpu load exceeds the 80% threshold. The on2 host keeps this policy, as there is no specific migration policy defined for it. on3 relaxes the global policy. This indicates that on3 may have a better processor or hyperthreading enabled.

3.2.2 *Policy matching.* The Policy module maintains a list of deployment and migration policies for each available host. When a new virtual machine request is issued, we must identify the hosts that support the deployment of that VM. In other words, we must call an evaluate() method for the hosts deployment policy. The values of the parameters that must be passed to the evaluate() method should be the sum between the hosts monitored parameters and the virtual machines statically determined parameters. The cpu, mem and disk values for a host are computed as the mean of the database entries of the last period of time. Experimentally, we decided that this period should be the last 2 minutes. In order to compute the net value, we need to find all the differences between two consecutive entries, divide each of them by the monitoring interval and then compute the mean of these values. Because the virtual machine is uninstantiated, we have no information about its cpu usage and network traffic. The mem and the disk parameters are static. Memory for a virtual machine is allocated at instantiation time, based on the description of the VM template. Disk usage is the amount of space the virtual machine image takes from the hosts total disk capability. The policy condition values are given in percentages, but the database entries store absolute values, so the cpu, mem and disk parameters must be expressed as ratios of the hosts maximum available cpu, memory and disk. The full algorithm for computing the values and checking if a host can sustain a new virtual machine deployment is shown below.

```
forall param in hostParameters :
        data := obtain param values from DB
        mean := 0
        if param == "net" then
                for i := 1 to data.size step 1 do
                        mean := mean + (data[i] - data[i-1])
```

```
7                   mean := mean / monitorInterval
8               mean := mean / (data.size − 1)
9           else if param == "disk" then
10              mean = data[0] ∗ 100 / hostMaxDisk
11          else
12              for i := 0 to data.size step 1 do
13                  mean := mean + data[i]
14              mean := mean / data.size
15              if param == "cpu" then
16                  mean := mean ∗ 100 / hostMaxCPU
17              else
18                  mean := mean ∗ 100 / hostMaxMemory
19          means[param] := mean
20      vmMemory := obtain from db (vm template)
21      vmDisk := obtain from db
22      vmMemory := vmMemory ∗ 100 / hostMaxMemory
23      vmDisk := vmDisk ∗ 100 / hostMaxDisk
24      means["mem"] := mean["mem"] + vmMemory
25      means["disk"] := mean["disk"] + vmDisk
26      return deployPolicy.evaluate(means)
```

At each step the Policy module checks if migration policies for each host match. Computing the host parameter means is done in the same way. The only difference from the supportsVM() algorithm is that there is no new virtual machine involved. After line 19, a migratePolicy.evaluate(means) call can be issued and the resulting boolean can be returned. If a policy matches for a host, we need to migrate a guest virtual machine in order to unload the system.

3.2.3 *Policy enforcement.* When a new virtual machine needs to be deployed, we check which hosts comply to their deployment policies. If more than one host is available, the host with fewer running guest virtual machines is preferred. After deciding which host is suitable for running the new VM, the Resource Management module comes into action. Using the Java OCA, it creates a new virtual machine and it deploys it on the designated host.

Once we decide that a migration policy was activated, we must decide which guest virtual machine should be migrated from the host and which is the destination host for the migrated VM. First, we create a list of virtual machine candidates that are suitable for migration. To do this, we subtract the VM loads from the total host loads and check if the migration policy still applies. If the resulting loads are beneath the thresholds imposed by the policy, the VM is a valid candidate. Then, we must choose one virtual machine from the resulting list. Virtual machines that were never migrated are preferred. If all VMs were previously migrated, the preferred VM is the one migrated first. Also, a virtual machine migrated less than an hour ago, will not be migrated. This is required so that virtual machines will not be continuously transferred from host to host in a short period of time. If none of the VMs were previously migrated, the VM having the highest number of load parameters greater than the others is preferred.

If the VM candidates list is empty (i.e. no virtual machine, if migrated, will reduce the hosts load to a point where the migration policy will not activate anymore), then the same sorting described in the previous paragraph is applied to the whole list of guest virtual machines. This means that we do not resolve the overloading problem in one iteration, but we rather wait for the migration process to finish, and then choose another virtual machine to migrate, sometime in the near future.

4. RESOURCE USAGE PREDICTION

The monitoring data collected using the modules we described in the previous sections are stored as time series. For a series of values for a parameter $x_1, x_2, x_3, \ldots, x_N$ we can use prediction algorithms[Castro et al. 2011] to estimate the values of $x_N + 1, x_N + 2, \ldots, x_N + h$, where h is the prediction horizon.

For the implementation we chose the Burg algorithm[Burg 1968]. The algorithm has the goal of minimizing the sum of the sqare of the errors between the measured data and the forward linear prediction E_p, as well as the sum of the error between the measured data and the previous linear prediction, which is named H_p. Those two sums can be expressed as:

$$E_p = \sum_{n=p}^{N} \left(x_n - \left(-\sum_{i=1}^{p} a_i x_{n-i} \right) \right)^2$$

$$H_p = \sum_{n=0}^{N-p} \left(x_n - \left(-\sum_{i=1}^{p} a_i x_{n+i} \right) \right)^2$$

The sum of E_{p+1} and H_{p+1} needs to be as small as possible, therefore:

$$\frac{\partial (E_{p+1} + H_{p+1})}{\partial k} = 0$$

From this, the rest of the model and algorithm can be found in [Burg 1968].

5. EXPERIMENTAL RESULTS

We installed an OpenNebula Cloud infrastructure on four physical machines. One of them (on1) acts like the head node and the other three (on2, on3 and on4) are available hosts that can support virtual machine deployment.

In order to test the resource management system we need an automated mechanism to generate host loads. Loads should consist of CPU, memory and disk usage as well as network traffic. WikiBench is a web hosting benchmark that can be used to stress-test systems[3]. It uses the MediaWiki web application to expose a Wikipedia database. Any host that has MediaWiki and a Wikipedia database dump installed acts like a Wikipedia server. We also need a Wikipedia request trace (i.e. a history of web requested Wikipedia resources). Given all these elements we can use the Wikibench application to replay the workload onto the Wikipedia installation.

In Figure 2 we show how the system reacts to a virtual machine deployment request. The policy configuration file for this scenario is the following:

```
all deploy mem < 55
on4 deploy mem < 90
```

When the simulation started, one WikiBench virtual machine was running on the on3 host. Because 1GB of RAM from the hosts total 4GB was reserved for the VM, the hosts total memory usage was somewhere around 30%. On the on2 host, two virtual machines were running: one WikiBench and one ttylinux. This is why on2s memory usage was a little higher. The on4 host was severely loaded because it was running multiple WikiBench machines. When about 10 minutes passed from the simulation

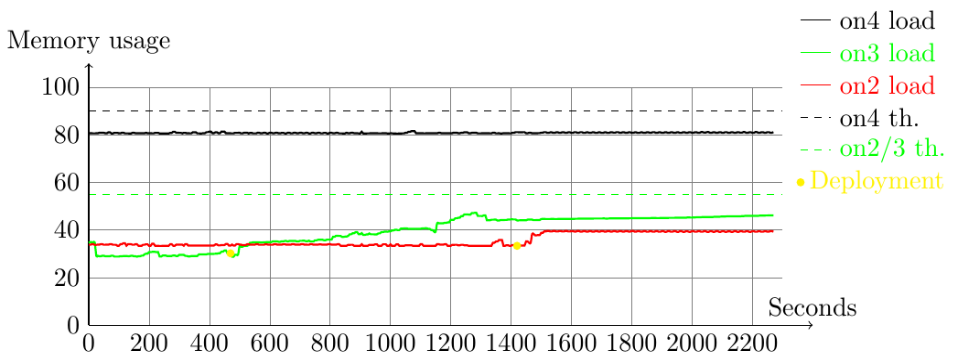

Fig. 2. Deployment of new VMs

start point, we issued a WikiBench virtual machine deployment request (first yellow bullet). In this situation, the system could clearly not pick on4 to be the destination of the new VM, as hosting another WikiBench would send its memory usage well beyond the threshold. The system picked on3 because at that point it hosted fewer virtual machines than on2. After deployment, on3s memory usage slowly grows as the new virtual machine becomes increasingly utilized. When a second WikiBench request was issued (second yellow bullet), on3s memory usage was somewhere around 45%. on3 is not a valid host candidate anymore, because another virtual machine (+25%) would make its memory load cross the policy imposed threshold. The new VM is correctly deployed to on2.

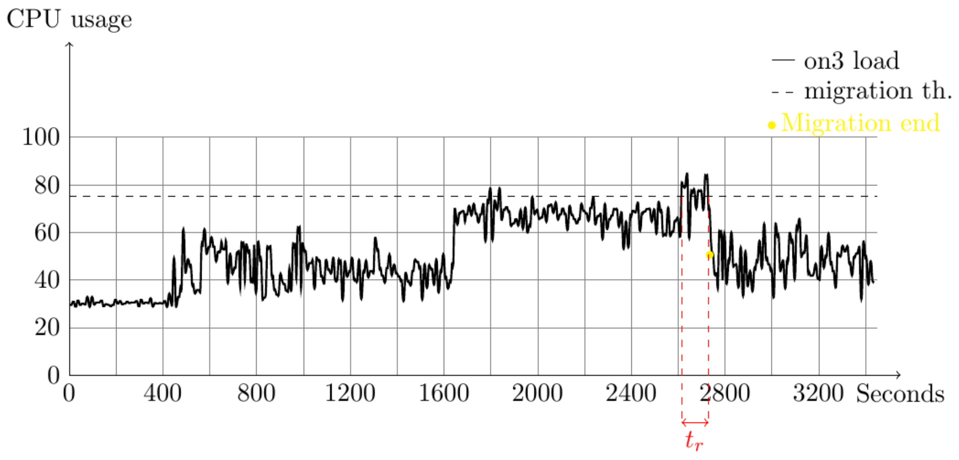

Fig. 3. Migration of a VM when reaching a threshold

In Figure 3 we show that the system reacts correctly when a migration policy becomes active. The only policy used in this scenario was on3 migrate cpu >= 75.The simulation starts with three running virtual machines on the on3 host: two ttylinux instantiated images and one WikiBench machine. We can see that because the WikiBench simulation was running, the hostss cpu load resembles real-life usage. When the cpu usage crosses for the first time the 75% threshold (around 1800 seconds), no action is taken because the mean cpu usage of the last 2 minutes is considered when checking the policy. This shows that the system is not fooled by load spikes. When the cpu usage starts to constantly exceed the policy imposed limit, the system reacts and migrates a virtual machine. The end of the migration (i.e. the moment when the migrated virtual machine is available on another host) is marked with a yellow bullet. We define the *response time (tr)* as the time passed from the moment when a parameter load starts to constantly exceed the threshold to the moment when the load is brought back under the threshold. In this case, because migration happens in just a few seconds, *tr* is approximately 2 minutes.

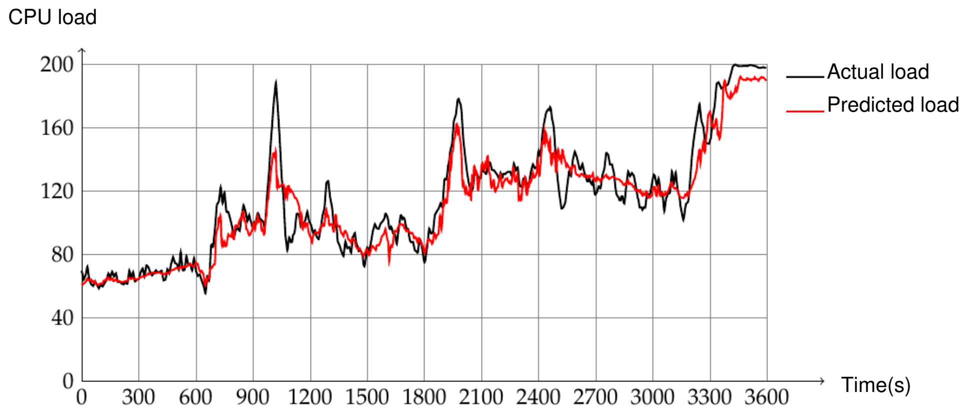

Fig. 4. Prediction of CPU load for 5 minutes in the future

In Figure 4 we used the values predicted by the Burg algorithm for the CPU load of a host for a moment in time 5 minutes in the future from the measured values. The input values are obtained through the monitoring of a host on which we launched 4 virtual machines. Each laod value is collected using intervals of 5 seconds. For each monitoring data we received, we used the Burg prediction algorithm to determine the CPU load value 5 minutes in the future, based on the data collected for the last 10 minutes. The data obtained in this experiment is close to the actual measured information, with an average error of 10.23%.

6. CONCLUSIONS AND FUTURE WORK

Efficient resource management is critical in Cloud systems. Our solution offers an efficient way to minimize resource usage. The pool of physical capabilities is limited, therefore we take advantage of reserved resources that are not used at their full capacity.

The resource management mechanism described in this paper can be integrated into an OpenNebula infrastructure. The OpenNebula scheduler offers a minimalistic policy instrument that is used to choose the appropriate host on which a new virtual machine should run. However, it lacks the possibility of auto migrating VMs off overloaded hosts. We solve this problem. The implementation is based on monitoring load data on the hosts in the Cloud and on the virtual machines deployed on them. Current host states help us decide where to send new virtual machine requests or when and what virtual machines to migrate from overloaded hosts. Both actions are taken using a policy-based system. Cloud administrators can supply deployment and migration policies in order to fulfill custom host and virtual machine load requirements.

We tested the solution with the help of the WikiBench application which provides real load for the virtual machines using Wikipedia logs. The proposed scenarios show that the framework reacts as expected when choosing a proper host for satisfying deployment requests and when making migration decisions. The results show that the use of deployment and migration policies improves resource distribution and usage in the Cloud.

This solution can be extended in two ways. Different types of policies can easily be added to reflect application response times. This parameter can be measured in the same way as the currently monitored host parameters, but its semantics is application-dependent. When policies become active, decisions can be made regarding not only migration, but deployment of new client virtual machines that could help decrease response times. Secondly, these policies can be used to enforce actual contracts in the case of a commercial-like Cloud. The framework can be perceived as a low-level tool that can help the administration of SLAs.

Acknowledgements

The work has been funded by the "*Sectoral Operational Programme Human Resources Development 2007-2013 of the Ministry of European Funds* " through the Financial Agreement POSDRU/159/1.5/S/134398.

This research presented is also supported by the following projects: "*SideSTEP - Scheduling Methods for Dynamic Distributed Systems: a self-* approach*", (PN-II-CT-RO-FR-2012-1-0084);" *CyberWater*" grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012.

REFERENCES

John Parker Burg. 1968. A new analysis technique for time series data. *NATO Advanced Study Institute on Signal Processing with Emphasis on Underwater Acoustics* 1 (1968).

Rajkumar Buyya, David Abramson, Jonathan Giddy, and Heinz Stockinger. 2002. Economic models for resource management and scheduling in grid computing. *Concurrency and computation: practice and experience* 14, 13-15 (2002), 1507–1542.

Juan R Castro, Oscar Castillo, Patricia Melin, Olivia Mendoza, and Antonio Rodríguez-Díaz. 2011. An interval type-2 fuzzy neural network for chaotic time series prediction with cross-validation and Akaike test. In *Soft Computing for Intelligent Control and Mobile Robotics*. Springer, 269–285.

Tharam Dillon, Chen Wu, and Elizabeth Chang. 2010. Cloud computing: issues and challenges. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. Ieee, 27–33.

Alaknantha Eswaradass, Xian-He Sun, and Ming Wu. 2006. Network bandwidth predictor (nbp): A system for online network performance forecasting. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, Vol. 1. IEEE, 4–pp.

Armando Fox, Rean Griffith, A Joseph, R Katz, A Konwinski, G Lee, D Patterson, A Rabkin, and I Stoica. 2009. Above the clouds: A Berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS* 28 (2009), 13.

Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. 2007. kvm: the Linux virtual machine monitor. In *Proceedings of the Linux Symposium*, Vol. 1. 225–230.

Bruce Momjian. 2001. *PostgreSQL: introduction and concepts*. Vol. 192. Addison-Wesley New York.

Zhonghong Ou, Hao Zhuang, Jukka K Nurminen, Antti Ylä-Jääski, and Pan Hui. 2012. Exploiting hardware heterogeneity within the same instance type of Amazon EC2. In *4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*.

Anna Schwanengel, Gerald Kaefer, and Claudia Linnhoff-Popien. 2013. Proactive Automated Dependable Resource Management in Cloud Environments. In *ADVCOMP 2013, The Seventh International Conference on Advanced Engineering Computing and Applications in Sciences*. 67–72.

Borja Sotomayor, Rubén Santiago Montero, Ignacio Martın Llorente, and Ian Foster. 2008. Capacity leasing in cloud systems using the opennebula engine. In *Workshop on Cloud Computing and its Applications*, Vol. 3.

Rich Wolski, Neil T Spring, and Jim Hayes. 1999. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems* 15, 5 (1999), 757–768.