CPU-based prediction with Self Organizing Map in Dynamic Cloud Data Centers

Nabila Djennane, Meziane Yacoub, Rachida Aoudjit, Samia Bouzefrane

CPU-based prediction with Self Organizing Map in Dynamic Cloud Data Centers

Nabila Djennane[1], Meziane Yacoub[2], Rachida Aoudjit[1*], and Samia Bouzefrane[2*]

[1]Lari Lab, UMMTO
[1]djennanenabila@gmail.com
[1*]aoudjit_rachida@yahoo.com
[2]Cedric Lab, CNAM
[2]meziane.yacoub@cnam.fr
[2*]samia.bouzefrane@cnam.fr

***Abstract*—The major objective of resource management systems in the cloud environments is to assist providers in making consistent and cost-effective decisions related to the dynamic resource allocation. However, because of the demand changes of the applications and the exponential evolution of the cloud, the resource management systems are constantly called into question with regard to their ability to guarantee an effective resource provisioning. To tackle these challenges, the future demand prediction is a practical solution that has been adopted in the literature. The prediction has widely relied on the CPU utilization since it is considered as a leading cause of the Quality of Service (QoS) dropping. The successful application of artificial intelligence techniques in forecasting problems motivated us to use the Kohonen Self Organizing Maps (SOM) that tries to capture the gathered empirical CPU load time series in regular behaviors to perform an accurate forecast. The proposed solution is a two-step approach that first classifies the collected data and then predicts the future CPU load. The experimental results show that our proposed system outperforms other models reported in the literature. In addition, we proved that SOM known for its strength in classification is also effective for prediction.**

***Index Terms*—Cloud Computing, Resources management, Load prediction, Times series, Clustering, Self organizing Map.**

I. INTRODUCTION

Increasingly used by companies in all industries, cloud computing is considered as a revolutionary technology for the 21st century. Cloud computing is a general term used to describe the on-demand delivery of different resources and services over the Internet. It refers to the data storage and computation through the Internet on remote powerful servers rather than using local computers. Thanks to this type of services, companies do not need to invest in their own equipment. Basically, cloud computing refers to three service categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).

Cloud providers, such as Amazon Web Services (AWS), offer virtual server storage, but also Application Programming Interface (APIs) that allow the users to transfer their workloads to remote virtual machines. An IaaS can provide servers, networks, storage space within data centers. Once users have allocated storage capacity, they can start, stop or configure the VMs and storage as they wish. The infrastructure provided as virtual machines can be small, medium, large or very large to suit different needs. Therefore, the IaaS is scalable and flexible, and can be adapted to the workload. As a consequence, the IaaS becomes as one of the most dominant services offered by cloud providers nowadays.

Although cloud computing promises a revolution in the Information and Technology (IT) world, companies still face economic challenges that require an optimal resource management, due to the growing demand for resources and the dynamic nature of the cloud usage.

The cloud resource management is a complex process. To be effective, it must allocate an appropriate amount of resources based on the current demand for applications to meet the service level agreement (SLA) and minimize the waste of resources. Indeed, under-provisioning results in a violation of the SLAs and a poor quality of service (QoS). On the other hand, over-provising wastes energy and resources and even increases costs in terms of cooling and maintenance.

Various techniques such as load balancing and consolidation have been implemented to address the resource management problem. However, the use of these techniques without taking into account the ever-changing resource requirements of hosts can lead to disastrous problems. Therefore, forecasting the future behavior of the application based on specific aspects is the only practical and effective solution for good resource provisioning [1] [2] [3] [4]. Indeed, accurately predicting the future use of host resources allows to make intelligent management decisions.

CPU is one of the main causes of resource scarcity on the cloud hosts, since it is the most dynamic, reactive and ever fluctuating resource. It is intuitive that the CPU management is inefficient, which can be disastrous for applications if they are based on reactive methods like optimisation methods.

Hence, to optimize the use of CPU resource, monitoring its condition and behaviour and correcting the causes of failure with proactive methods are necessary. It is therefore essential to apply forecasting models to manage cloud resources. However, it is very difficult to accurately predict host utilization in

a timely manner because of the very fast variation of the host utilization and the strong instability with many bursts.

The contribution of this paper attempts to meet this challenge by proposing a solution that addresses the above-mentioned issues. Indeed, our proposed methodology aims at predicting the CPU resource of the next time slot based on historical data issued from previous slots. We resort to Self Organizing Map (SOM) algorithm known for its high capacity in clustering, which we use to its full potential. The time period for which forecasting provides critical information about future load is very important. Therefore, we introduce the sliding window technique which has proven its success in predicting long time series. Thus, to predict CPU load of the next time slot we variate the observation windows. For prediction, we also use SOM through a simple but efficient method. We illustrate the feasibility of our approach by using the PlanetLab dataset generated by the COmon project [5].

The remainder of this paper is structured as follows. In section II, we discuss the literature review. In section III, we motivate our choice and present the used algorithm. Section IV describes our proposed methodology by detailing each step. Section V highlights the conducted experiments and the performed results before concluding in Section VI.

II. RELATED WORK

To ensure a good QoS and to avoid resources wasting, we need to manage resources. However, the big issue is how to predict the future use of the host in real time as the workload varies with time. Predicting future use of hosts is essential for placing virtual machines on appropriate hosts or migrating virtual machines in advance from overloaded or underloaded hosts.

Numerous studies have been carried out on the prediction in cloud computing according to various objectives of the research: Prediction of the servers' load [6], [7], [8], [9], [10], prediction of the VMs' load [11], [12], prediction of the VM use [13], [14], prediction of the host use [15].

In the context of virtual machine migrations, various workload prediction techniques are used [16], [17], [18], [19], [20].

The CPU load is one of the most studied indicator to forecast the data-center activity, as it is one of the leading causes of resource shortages on cloud hosts. Artificial Intelligence (AI) and Machine learning (ML) algorithms such as neural networks and linear regression have been widely applied in recent years to predict the CPU load.

Authors, in [21], used several prediction models to predict the execution times of tasks, based on the CPU load forecast. [22] used a recurrent neural network formed with the so-called back propagation algorithm to predict the CPU utilisation of a host on the Google Cluster trace data set.

Authors, in [23], proposed the implementation of local regression, median absolute deviation, inter-quartile range and robust local regression methods to determine when servers may be over-utilized based on CPU usage.

To improve the performance of the VM placement algorithm proposed in [23], Mason et al. [24] proposed the evolutionary neural network approach to predict the host CPU utilization based on both one-step and multi-steps ahead.

Unlike the approach described by Song et al. [25] who used the short term memory (LSTM) model in a recurrent neural network (RNN), Qazi and Aizenberg in [26] proposed a prediction mechanism using MLMVN, which is a feed forward neural network using complex-valued neurons. The CPU usage of individual host machines for the 29 days was utilized for evaluation.

In order to propose a virtual machine consolidation approach, the Kernel Density Estimation technique is used in [27] to estimate resource usage and predict future host's states. Using real web server request traces, Calheiros et al. [28] proposed an auto-regressive integrated moving average (ARIMA) model that assesses the accuracy of future workload forecasting.

In [29], an LA-based ensemble prediction algorithm based on Learning Automata (LA) theory is proposed for CPU load prediction based on several VMs gathered from the dataset of the CoMon project. The proposed approach combines the prediction values of multiple prediction models. Each model is given a weight according to its performance. Then, an autonomic approach is used to determine how much accurate each prediction model is according to their predictions.

In [30], the authors presented a workload prediction model using neural network and self adaptive differential evolution algorithm to learn the best suitable mutation strategy along with optimal crossover rate.

Sadeka et al., in [31], proposed empirical prediction models for adaptive resource provisioning in cloud computing using neuronal network and linear regression. Their prediction method applied some strategies including sliding window. They also provided evaluation metrics for validating the technique such as MAPE and RMSE.

Salam Ismaeel et al., in [32], proposed an ELM technique to predict data center VM requests based on historical data, that they combine with the clustering K-Means method. The authors presented these techniques in a small framework. They used data from real google traces.

In order to minimize energy cost and consumption in cloud data centers and to reduce the number of active physical servers on data centers, Fahimeh Farahnakian et al. in [33], proposed a dynamic virtual machine consolidation based on k-nearest neighbor regression algorithm that predicts in each host the CPU usage. To achieve SLA violation and energy cost, they adopted four algorithms of [23] that they combined with their proposed method k-nearest neighbor algorithm.

In [34], an adaptive short-term prediction strategy is proposed to adaptively select a precise short-term prediction method. The proposed approach works in three stages: preprocessing of abnormal data and replacement of outliers, adaptive selection to select a better prediction algorithm based on a dynamic threshold and finally an error adjustment by proposing an error adjustment factor for improve accuracy. The conducted experiments demonstrate the effectiveness of

the proposed approach in terms of the prediction accuracy, however it suffers from a high time cost.

To address the challenge of allocating several requests to a VM, authors in [35] proposed A Regressive Ensemble Approach for Predicting (REAP) CPU usage of a scientific application. REAP predicts intelligently the CPU utilization by integrating Genetic Algorithm-based feature selection and CPU usage prediction techniques. The experimental results show that the proposed approach enhances the accuracy by 2% and reduces the execution time by 16.2% comparatively to the existing machine learning regression models.

The authors, in [36], proposed a deep learning approach that they designed with a deep belief network (DBN) composed of multiple layered restricted boltzmann machines and a regression layer. They used the DBN to extract the high level features from data and regression layer for prediction. A MAPE error is calculated to evaluate the proposed method.

Salam Ismaeel et al, in [37], proposed a survey of the most techniques and algorithms used in proactive dynamic VM consolidation. This study focuses on energy consumption.

The authors of [38] combined linear and non linear prediction models using ARIMA and SVR. ARIMA model is used to predict time series of linear attributes while SVR model is used to extract non linear attributes from the dataset they used. Both models used MAPE and RMSE to validate their methods.

The authors of [39] consider another important parameter for prediction that is the SLA. Since this parameter is critical for both the customer and the cloud provider, they use scaled conjugate gradient neural network to make predictions possible on SLA violations.

Cloud data centers contain hundreds of thousands of servers, which host millions of VMs of different sizes, types and applications. Hence, since server resources are strongly influenced by the VMs they host, it makes more sense to focus on VMs resource management rather than server management as in [6] [7] [8].

The choice of a good data set that is representative of a real cloud computing environment is important. Authors of [27] [26] [32] [25] [21] have worked on the data set of Google [40] that gathers the collected CPU and memory traces of web servers, while in [5] collected the percentage of CPU usage by several VMs hosted in different servers located in different regions of the world.

Our research work focuses on the data set of [5] like in [23] [24] [33] [37]. The drawback of these works is that they did not exploit all the data. Some of them selected randomly some VMs while others selected randomly time intervals. This distorts the results in case of generalization.

In addition this data set is of time series type, which is an important point to take into account. The produced time series present a particular difficulty in predicting their future values. It is well known that the usual prediction methods already used in prediction problems have poor performance when used without first clustering the time series [41].

The fact that, the works cited above didn't resolve well the problem, motivates us to propose and focus in this paper on three important issues: data clustering, time series analysis and predicting the future CPU load.

III. THE SELF ORGANIZING MAP

Developed by Teuvo Kohonen, self-organized map (SOM) is a new method to represent neural networks and to perform automatic classification tasks. Unlike artificial neural networks, SOM is an unsupervised method. It is composed of simple elements (neurons) assembled into neural networks, whose the functioning is strongly influenced by the connection of the elements to each other. SOM is often used to analyze observed data whose the structure requires investigation.

A. PRINCIPLE OF SOM

SOM is a method that projects the space of observed data (of large dimensions) into a space of small dimensions such as 1, 2 or 3D, called MAP. The map is made up of a set of neurons connected to each other according to the notion of neighbourhood. Each neuron has fixed coordinates in the map space and coordinates adaptable on the space of the data called referential vectors, responsible for an area of this space.

This projection must respect the topology of the data. It is based on vector quantification methods that partition all available observations into similar groups using learning algorithms. These groups are characterized by their neighbourhood structures, which can be materialized using a discrete space called a "topological map". This space forms a small lattice on which neighbourhood structures are taken into consideration by the model. The choice of a topology depends on the nature of the problem.

The implementation of a topological map requires a distance δ on the topological space to define the notion of neighbourhood on the map. It is a discrete distance defined by the length of the shortest path connecting two neurons on the map. Thus, the order neighbourhood of a neuron can be defined as the set of neurons located at a distance d from the neuron.

The distances that link the neurons to each other allow to vary the relative influence of the different neurons which can be quantified by a K function. We often use a family of K^T functions set to T, sometimes called activation functions to better control the size of the neighbourhood as in the following:

$$K^T = K(\frac{\delta}{T}) \tag{1}$$

T is a parameter used to control the influence between neurons.

B. ALGORITHM OF SOM

Once the map topology and activation functions have been chosen, the reference vectors are obtained by learning from the data. The algorithm used to learn the map is described as in the following.

Step 0 : Initialization of the reference vectors W.

Step 1: At each iteration: As input of the map, an example of a selected learning $X(n)$ is presented. The algorithm compares the example to all the reference vectors so that the winning neuron i is the one whose reference vector $W_i(n)$ is the closest to the input $X(n)$:

$$i^* = argmin(d(W_i(n), X(n))) \tag{2}$$

Where i denotes in turn each neuron in the network and W_i its referent vector.

We evaluate the neighbourhood of the winning neuron by the equation below:

$$K_{i^*}^T(i) = K^T(\delta(i, i^*)) \tag{3}$$

Then, all neurons of the Map are updated for any i belonging to the neighbourhood of i^*. The Map is more adapted by the fact that the neurons are neighbours of i^*:

$$W_i(n+1) = W_i(n) + \alpha(t)K_{i^*}^T(i)[X(n) - W_i(n)] \tag{4}$$

where α is a learning parameter called learning step which decreases as a function of n, and is less than 1. It's also an important element because it allows to find a compromise between the convergence speed and the quality of the map unfolding.

Finally the stopping criterion of the algorithm is defined in several ways: number of iterations, difference between weight vectors, quantification error and so on.

Phases 1, 2 and 3 are called respectively as in the following:

Competition: After selecting an example from the database, we look for the neuron that most closely resembles it. This neuron called "winning neuron" is often rated BMU (Best Matching Unit).

Cooperation: In this step, the neighbourhood of the winning neuron is determined. It refers to the region of the map that is the most active and closest, in terms of the distance used, to the observation. The size of the neighbourhood of the winning neuron is controlled by the learning radius.

Adaptation: The winning neuron is modified to look more like the example and then information about its neighbourhood is disseminated. This modification is done using the activation function which controls the influence of the winning neuron on its neighbourhood.

C. AUTOMATIC CLASSIFICATION WITH KOHONEN MAPS

Once a map is learned, we have the typical profiles of each neuron called codebooks. Two close neurons in the map are close standard profiles and a set of contiguous codebooks represent a particular profile in the data.

Codebooks are a data set on which a hierarchical classification can be applied to group similar neurons. The classification of a new element can be done as follows:

- Present the new element at the card entry.
- Identify the winning neuron, the one whose codebook is close to the new element.
- Identify the cluster of the winning neuron.

IV. OUR PROPOSED APPROACH

In this section, we will highlight our methodology and describe our proposed model while depicting each component (see Fig. 1).

A. Methodology description

In the cloud computing, a data center is mainly made up of a set of virtualized servers. Services are offered to customers by allocating several heterogeneous virtual machines. The real loads of these virtual machines are often dynamic. Thus, the static allocation of resources is doomed either to waste, if it is based on a worst-case scenario estimate, or to performance degradation, if it is based on the average load. Thanks to the cloud computing model, resources can be allocated on demand and the sizing is adapted to load variation. However, resource variations, especially the CPU load of VMs are frequent and unpredictable, resulting in server overload, under-load or even inactivity. This results in degraded performance, wasted resources or violations of service level agreements (SLAs).

It appeared necessary to develop proactive methods for the management of virtual machines before it leads to under-consumption of resources on physical servers or over-consumption.

In order to make an acceptable solution that is able to predict future resources requirement in the cloud, we explored a number of existing prediction methods as proposed in the literature.

The first step of our proposed solution is to choose a representative data set from a real environment. Since the CPU resource is the most dynamic one on the servers, we choose the Planetlab data of CoMon project [5]. These data are gathered from a real workload of monitoring infrastructure. The CPU utilization data is obtained from more than one thousand VMs from servers located within more than 500 locations around the world. Data is collected every 5 minutes. So, our research problem relates to time-series analysis such as each VM is a time series data that consists of a sequence of N pairs (CPU_i, t_i), where CPU_i is the percentage of CPU utilisation at time t_i.

The Self Organizing Map mainly adopted in our proposed system is one of the most common techniques used in clustering problem. However, in our work, we propose to use it not only for this purpose but also for prediction.

Moreover, to determine the best prediction on window, we incorporate the concept of sliding window, which forecasts future CPU resources using previous data in different observation windows sizes. Our proposed models are carried out using statistical metrics as discussed in the following sections.

B. Our Proposed System Components

1) **Clustering:** A general rule in the descriptive statistics consists in firstly observing and visualizing the data

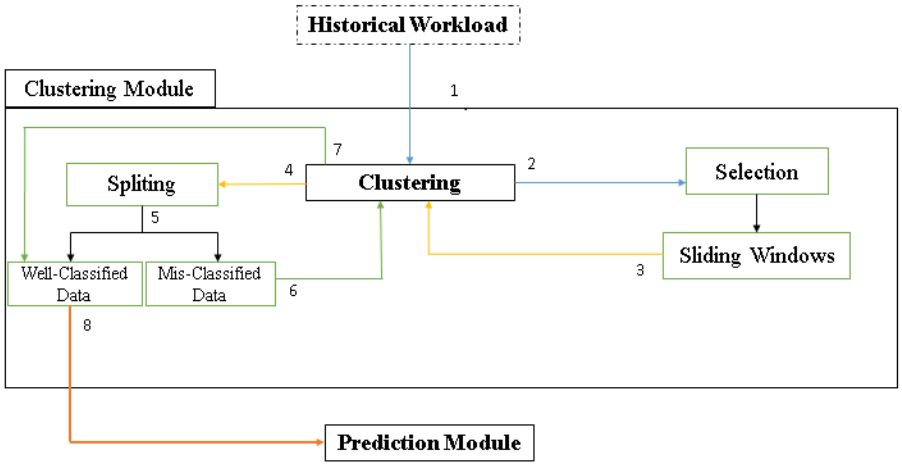

Figure 1: Proposed System Model

before making any calculations. This is due, in one hand, to the number of VMs that is very large, and on the other hand to the repetitive variation of VMs that can occur, since a clear hypotheses may be rarely available. Hence, the first step is to gather all VMs of the chosen dataset in order to make a first clustering with the SOM algorithm. The goal is to determine and to separate similar types and variations of VMs into clusters such that the VMs within a cluster become more similar that the VMs in other clusters according to Algorithm 1.

Algorithm 1: SOM-1

Input: a set of VMs $X = \{VM_1, VM_2, .., VM_N\}$
Output: a set of neurons $Y = \{y_1, y_2, .., y_M\}$
begin;
Initialize $Y = \{y_1, y_2, .., y_M\}$ randomly
repeat
  Select $VM \in$X$randomly$
  find y^* such that $d(VM, y^*) = min\{d(VM, y) | y \in Y\}$
  Update the weight of the winner y^* and its neighborhoods
  $y = y + \gamma(VM - y)$
  reduce learning rate γ
until *Convergence*;

2) **Selection** Once all the data are projected on Map and each neuron summarizes the VMs belonging to it with the same behavior, and in order to continue the next steps, we have chosen to select a subset of VMs from each neuron in order to obtain a sample of data of reduced size while keeping the heterogeneity of the whole set. In this way, our simulations for the prediction that is the purpose of our work will not be saturated. The data selection was made according to Algorithm 2.

Algorithm 2: Selection

Input: a set of neurons $Y = \{y_1, y_2, .., y_M\}$
begin;
Initialize $i = 1$
repeat
  foreach $y_i \in Y$ **do**
    $S[] \leftarrow allVMs \in y_i$
    $SOrd[] \leftarrow Ord(S, desc)$
    ; // Order S in Descending Order according to their radius error
    $SSelected[] \leftarrow$ take one out of 3 from $SOrd[]$
  end
until $i = M$;

3) **Sliding Window**
Each time we are interested in prediction problems using time series, the sliding window is requested, especially in the case of long series. Indeed, the prediction can only be made on a finite number of points. In our case, we have CPU loads as used by the VMs gathered every five minutes during 24 hours.
Our data are therefore as follows :

$$CPU_{VM_1}^{t_0}, CPU_{VM_1}^{t_1}, ..., CPU_{VM_1}^{t_k}$$

$$CPU_{VM_2}^{t_0}, CPU_{VM_2}^{t_1}, ..., CPU_{VM_2}^{t_k}$$

$$CPU_{VM_N}^{t_0}, CPU_{VM_N}^{t_1}, ..., CPU_{VM_N}^{t_k}$$

N:Number of VMs
k: number of time slot
$t_0, t_1, .., t_k$ Time slot where t_k - t_{k-1} = 5 minutes

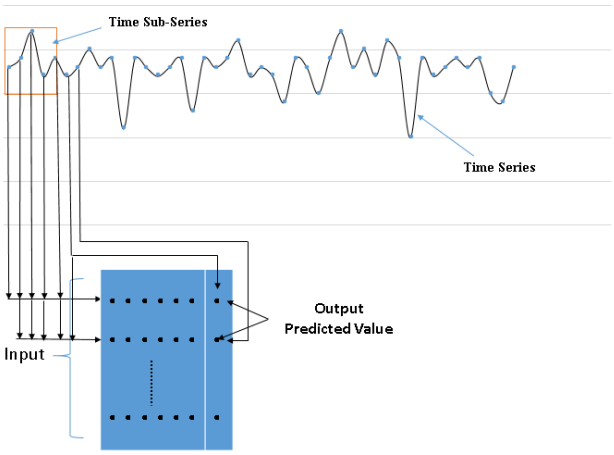

Figure 2: Scheme of slide windows

Sliding Window is a technique used to split a long time series of infinite or long observations into short series that will be used as an observation window to predict future values.
Because there is no rule to determine the appropriate size of the observation window for better prediction, we have decided to vary the window size in order to decide which one will be the right one as follows:
In Fig.2, the size of the observation window is only used as an illustrative example. In our work, we considered two scenarios. In the first one, we tried to predict the 7th value by using the previous 6 values, and in the second scenario, our aim was to predict the 13th using the previous 12 values.
For each time series, i. e. for each VM, the values were predicted from 7 to 288 in the first scenario and from 13 to 288 in the second scenario, knowing that we cannot predict the values from 0 to 6 and from 0 to 12 because of scarce values. Therefore, we use the following:

- **First Case:**
  The 6 values of each VM:
  From 0 to 6 to predict the 7th value
  From 1 to 7 to predict the 8th value
  From 2 to 8 to predict the 9th value
  .
  .
  .
  From 282 to 287 to predict the 288th value
- **Second Case:**
  The 12 values of each VM:
  From 0 to 12 to predict the 13th value
  From 1 to 13 to predict the 14th value
  From 2 to 14 to predict the 15th value
  .
  .
  .
  From 276 to 287 to predict the 288th value

We note:
$XVM_{i,j}$ the CPU measurement of VM_i at time j (i=1,n) ($j = 1$, 288)

The problem consists then in using:

- **Case 01 : Observation Window = 6**

$$\begin{array}{cccc} X_{VM_{1,1}} & X_{VM_{1,2}} & X_{VM_{1,3}} & X_{VM_{1,4}} \\ X_{VM_{1,5}} & X_{VM_{1,6}} & \textbf{to predict} & X_{VM_{1,7}} \\ X_{VM_{1,2}} & X_{VM_{1,3}} & X_{VM_{1,4}} & X_{VM_{1,5}} \\ X_{VM_{1,6}} & X_{VM_{1,7}} & \textbf{to predict} & X_{VM_{1,8}} \\ \dots & & & \\ X_{VM_{1,282}} & X_{VM_{1,283}} & X_{VM_{1,284}} & X_{VM_{1,285}} \\ X_{VM_{1,286}} & X_{VM_{1,287}} & \textbf{to predict} & X_{VM_{1,288}} \\ \dots & & & \\ X_{VM_{n,282}} & X_{VM_{n,283}} & X_{VM_{n,284}} & X_{VM_{n,285}} \\ X_{VM_{n,286}} & X_{VM_{n,287}} & \textbf{to predict} & X_{VM_{n,288}} \end{array}$$

- **Case 02 : Observation Window = 12**

$$\begin{array}{cccc} X_{VM_{1,1}} & X_{VM_{1,2}} & X_{VM_{1,3}} & X_{VM_{1,4}} \\ X_{VM_{1,5}} & X_{VM_{1,6}} & X_{VM_{1,7}} & X_{VM_{1,8}} \\ X_{VM_{1,9}} & X_{VM_{1,10}} & X_{VM_{1,11}} & X_{VM_{1,12}} \\ & & \textbf{to predict} & X_{VM_{1,13}} \\ X_{VM_{1,2}} & X_{VM_{1,3}} & X_{VM_{1,4}} & X_{VM_{1,5}} \\ X_{VM_{1,6}} & X_{VM_{1,7}} & X_{VM_{1,8}} & X_{VM_{1,9}} \\ X_{VM_{1,10}} & X_{VM_{1,11}} & X_{VM_{1,12}} & X_{VM_{1,13}} \\ & & \textbf{to predict} & X_{VM_{1,14}} \\ \dots & & & \\ X_{VM_{1,276}} & X_{VM_{1,277}} & X_{VM_{1,278}} & X_{VM_{1,279}} \\ X_{VM_{1,280}} & X_{VM_{1,281}} & X_{VM_{1,282}} & X_{VM_{1,283}} \\ X_{VM_{1,284}} & X_{VM_{1,285}} & X_{VM_{1,286}} & X_{VM_{1,287}} \\ & & \textbf{to predict} & X_{VM_{1,288}} \\ \dots & & & \\ X_{VM_{n,276}} & X_{VM_{n,277}} & X_{VM_{n,278}} & X_{VM_{n,279}} \\ X_{VM_{n,280}} & X_{VM_{n,281}} & X_{VM_{n,282}} & X_{VM_{n,283}} \\ X_{VM_{n,284}} & X_{VM_{n,285}} & X_{VM_{n,286}} & X_{VM_{n,287}} \\ & & \textbf{to predict} & X_{VM_{n,288}} \end{array}$$

Therefore, we create in each case two matrices : input matrix M_{Input} and output matrix M_{Output} written as follows:

- **Case 01 : Observation Window = 6**

$$M_{Input} = \begin{bmatrix} X_{VM_{1,1}} & X_{VM_{1,2}} & \dots & X_{VM_{1,6}} \\ X_{VM_{1,2}} & X_{VM_{1,3}} & \dots & X_{VM_{1,7}} \\ \dots & & & \\ X_{VM_{n,282}} & X_{VM_{n,283}} & \dots & X_{VM_{n,287}} \end{bmatrix} \quad (1)$$

$$M_{Output} = \left[X_{VM_{1,7}} X_{VM_{1,8}} \dots X_{VM_{n,288}}\right] \quad (2)$$

- **Case 02 : Observation Window = 12**

$$M_{Input} = \begin{bmatrix} X_{VM_{1,1}} & X_{VM_{1,2}} & \dots & X_{VM_{1,12}} \\ X_{VM_{1,2}} & X_{VM_{1,3}} & \dots & X_{VM_{1,13}} \\ \dots & & & \\ X_{VM_{n,276}} & X_{VM_{n,277}} & \dots & X_{VM_{n,287}} \end{bmatrix} \quad (3)$$

$$M_{Output} = \left[X_{VM_{1,13}} X_{VM_{1,14}} \dots X_{VM_{n,288}}\right] \quad (4)$$

The data of the input matrix in each case will be again learned using SOM to gather similar observations within the same neurons according to algorithm 3.

Algorithm 3: SOM-2

Input: M_{Input}
Output: a set of neurons $Y = \{y_1, y_2, .., y_M\}$
begin;
Initialize $Y = \{y_1, y_2, .., y_M\}$ randomly
repeat
  Steps 6 to 10 of SOM-1
until *Convergence*;

At the end of this step, we obtain two Maps from each case.

4) **Splitting**

To evaluate the quality of the obtained Maps, a strong similarity within the same neuron must be respected. Therefore, we compute the standard deviation of each neuron of Maps. Then, we define a threshold that we have set at a fix value.
The two obtained Maps in the previous step will be splited into two Sub-Maps according to the defined threshold as in Algorithm 3.

Algorithm 4: Spliting

Input: Map,Threshold;
Output: MisClassifiedData[], WellClassifiedData;
foreach *Neuron* $\in Map$ **do**
  if *STD(Y)*$\leq Threshold$ **then**
    WellClassifiedData[] $\leftarrow Data \in Neuron$
  else
    MisClassifiedData[]$\leftarrow Data \in Neuron$
  end
end

The MisClassified Data will be once again presented to SOM, in order to reclassify and better distribute them on the neurons and by the way reduce the standard deviation (See algorithm 5).

Algorithm 5: SOM-3

Input: MisClassified Data
Output: a set of neurons $Y = \{y_1, y_2, .., y_M\}$
begin;
Initialize $Y = \{y_1, y_2, .., y_M\}$ randomly
repeat
  Steps 6 to 10 of SOM-1
until *Convergence*;

5) **SOM-Prediction**

As we introduced earlier, we applied our SOM-based algorithm for both clustering and prediction. We will explain here how to make the prediction possible using SOM.

We consider the final obtained SOM Map by the previous step. Each map consists of a set of neurons and each neuron gathers a set of similar input observation windows.
The goal of SOM-Predictor is to make prediction in each one of the neurons.
Let Y_i denote the set of M neurons corresponding to each Map $Y_i = \{(y_1, y_2, ..., y_M)\}$, and

$$X_n = \{(x_1^{t_0}, x_1^{t_1}, .., x_1^{t_k}), (x_2^{t_0}, x_2^{t_1}, .., x_2^{t_k}), ..., (x_n^{t_0}, x_n^{t_1}, .., x_n^{t_k})\}$$

(where k=number of time slot) the subset of M_{input} data mapped to neuron i. For each element of X_n corresponds an observed output from M_{output}. We note the set of outputs by $O_n = \{o_1, o_2, .., o_n\}$.
The CPU load at time t_{k+1} for all observation window data belonging to the same neuron is then computed as follows:

$$Pred(y_i) = \frac{\sum_{j=1}^{n} o_j}{n} \quad (5)$$

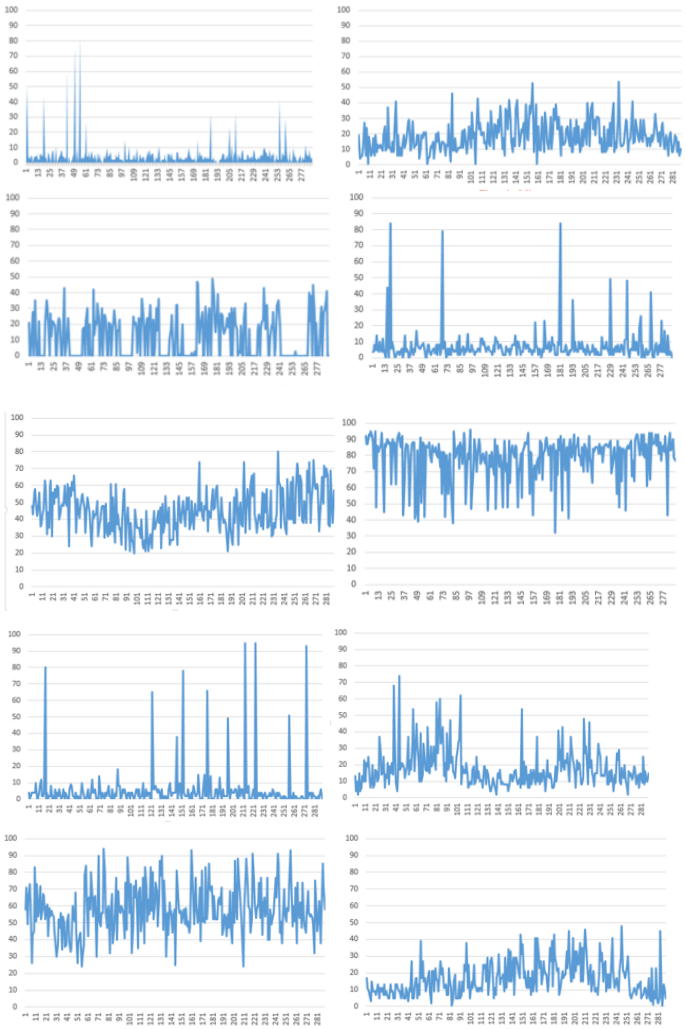

Figure 3: CPU Load of different VMs during 24 hours with 5minutes interval showing the diversity of VMs types and their variations

V. EXPERIMENTS RESULTS

In this section, the efficiency of the proposed prediction system will be evaluated using a real world CPU load trace [5] described in Table 1. Our experiments are conducted under Matlab environment using SOM toolbox [42].

First, we plot some VMs of the data set (Fig.3) to show the diversity of types and variability during 24 hours (one measurement every five minutes).

Day	Number of VMs
2011/03/03	1052
2011/03/06	889
2011/03/09	1061
2011/03/22	1516
2011/03/25	1078
2011/04/03	1463
2011/04/09	1358
2011/04/11	1233
2011/04/12	1054
2011/04/20	1033
Total	11737

Table I: Data Detail

As shown in Table 1, we have 11737 VMs and each VM contains 288 CPU traces. An Input matrix of 11737*288 is presented to SOM to be classified. Fig. 4 shows the SOM Map obtained after projection with the data by PCA tool, while Fig. 5 is the SOM Map obtained after selecting some VMs to be used in the remaining steps. The two Maps obtained (Fig. 4 and Fig.5) are not very different. This shows that the selected VMs summarize well the global data and therefore our selection method is efficient and the new obtained data set is homogeneous. The selection step is very important during the simulations. Indeed we have 11737*288 data. This number will increase considerably during sliding windows step that not only blocks our simulations but also creates redundancies of observation windows.

Fig. 6 and Fig.7 correspond to the SOM Maps obtained after classification of the observations windows for the two different cases 6 and 12 respectively. These figures clearly show that the SOM Map is spread out over the data set, which proves that the Maps are well learned in such a way that they reach all points, even the remote ones.

In statistics, the standard deviation can be used to calculate the homogeneity of several populations on the same variable. It measures the dispersion of the values of a data set around the mean. If the data values are all similar and homogeneous, then the standard deviation will be small and close to zero. If the data values are highly variable, then the standard deviation will be large and greater than zero.

The standard deviation is computed for each neuron and displayed in Fig.8 and Fig.9 by the U-Matrix. As shown by these figures, the standard deviation varies between 0 and 13.5 when the slide window is equal to 6, and between 0 and 14.6 when the slide window is equal to 12. After several simulations, we define a **threshold=2** which will be used to split the obtained SOM Maps into two parts: the well classified data and mis-classified data. We keep the neurons whose the standard deviation is less than the threshold while the dataset belonging to those whose the standard deviation is higher than the threshold will be learned again by the SOM algorithm in order to better reclassify them and to obtain two other maps in each case, that are better learned.

In order to illustrate the effectiveness of this work, the proposed SOM-predictor is compared with: Support Vector Machine (SVM) and MultiLayer Perceptron (MLP) [43] models. To make this possible, we use the same selected data set and the same size for observation windows and prediction windows.

To conduct our evaluation and comparison, we have selected two methods as described below. Our choice was based on similar works of the literature:

- Root-Mean-Square Error (RMSE): is a frequently used measure of the difference between the values predicted by a model or an estimator and the values observed defined as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \left(\hat{Y}_i - Y_i\right)^2}{N}} \tag{6}$$

- Mean absolute percentage Error (MAPE): is a measure that indicates about the mean of the dispersion between predicted and observed values, with the linear model (absolute difference). Therefore, it's a convenient indicator for comparison.
  The formula is:

$$MAPE = \frac{1}{N} \sum_{i=1}^{N} \frac{\left|\hat{Y}_i - Y_i\right|}{Y_i} \tag{7}$$

Table 2 and Table 3 show significant values and clearly highlight the chosen prediction method compared to SVM and MLP. RMSE and MAPE errors obtained with SOM are the smallest in the two cases where observation windows are equal to 6 and 12. However, for SOM, the size of observation window is clearly important such that the results of the errors show that the choice of a smaller observation window allows us to better predict.

In addition to the numerical values, we plot in figures Fig.10 Fig.11 some VMs such that the graphs of each figure show the observed values and the predicted ones by the three methods SOM, MLP and SVM for each VM. The figures show clearly that the lines of SOM follow the same variations of the observed ones. In the data center environment, the goal is not to predict the exact future CPU values but to predict the variation that can each VM take in time. The peaks of load that may occur are critical and have an important role in the consolidation of VMs, especially in their migration. Hence, through our experimentation, we have shown that SOM combined with the proposed methodology is quite robust to predict better, especially in critical cases unlike MLP and SVM which predict quite well but are weak in such cases.

In addition to our experiments, we present some VMs of the non-selected dataset to the SOM Map as shown in figure Fig.12 (with observation window=6) to predict the future CPU load with the three predictor methods. The results are shown in graphs of Fig.13 and Fig.14 and prove that SOM predicts better than MLP and SOM.

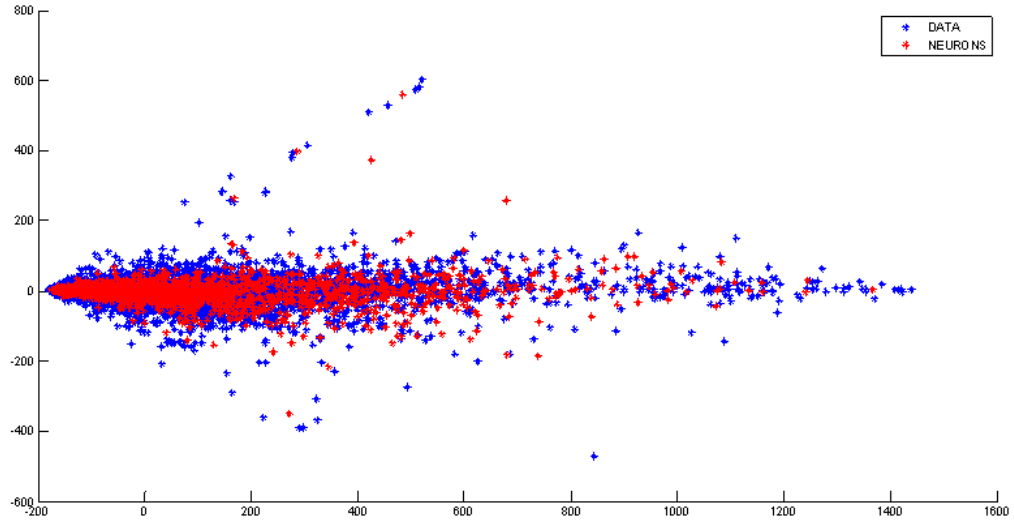

Figure 4: PCA projection of data and neurons after Classification of all VMs

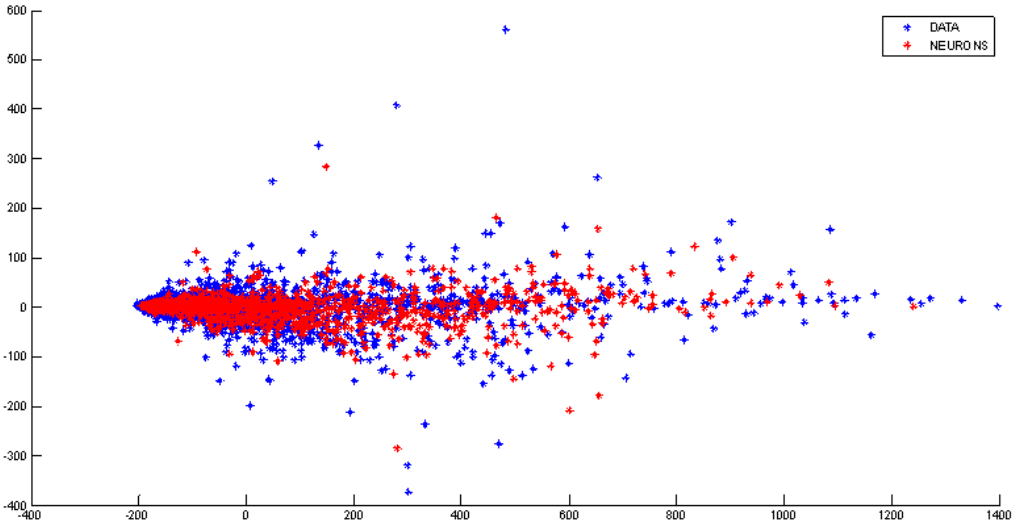

Figure 5: PCA projection of data and neurons after Classification of selected VMs

VI. CONCLUSION AND FUTURE WORKS

Coupling classification and prediction in our study represents the original aspect of the contribution. Moreover, developing a methodology based on observation and data analysis allowed us to see the data in a different way than with conventional methods. Our work used a subset of representative data set from the totality of the data unlike other existing works that focused only on limited data such as one day or a certain interval of time.

Our prediction method is quite simple but very efficient as shown by the obtained results and graphs compared to the two prediction methods SVM and MLP which are known for their robustness in regression prediction.

As a perspective, we plan to further test the proposed method by considering not only the CPU workload, but also different types of workloads (memory, storage, etc.) and investigate the correlation between them. We intend also to implement our method on a real platform for real-time decision making.

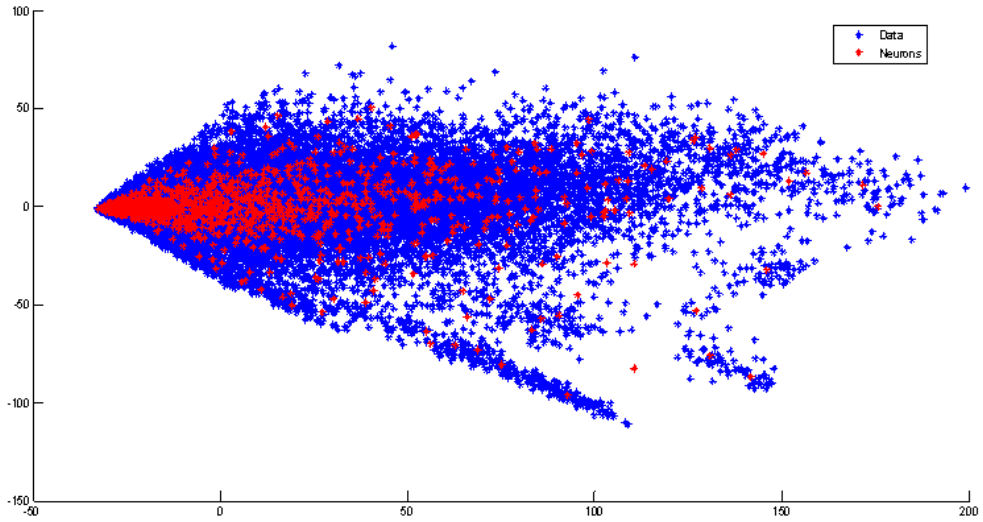

Figure 6: PCA projection of data and neurons after sliding window, Case: slide window=6

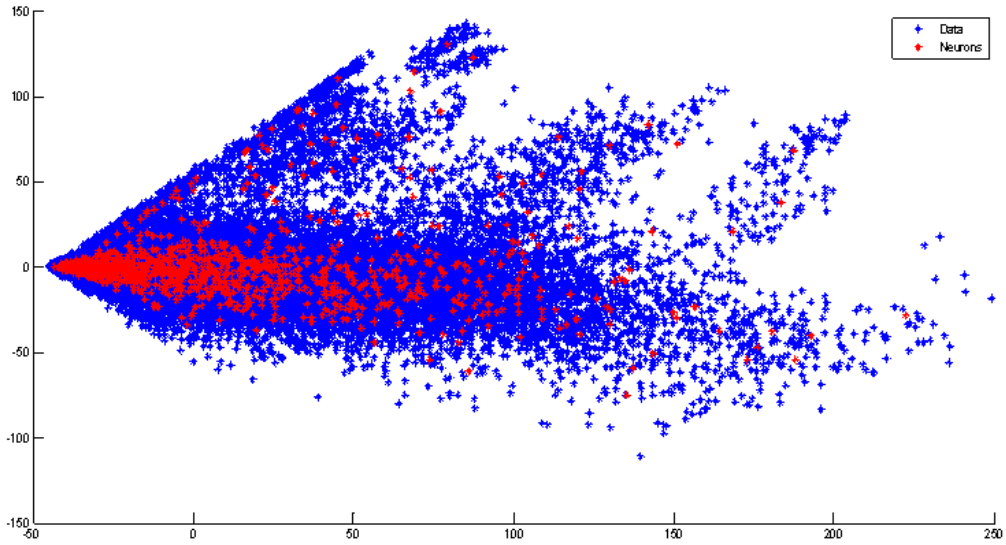

Figure 7: PCA projection of data and neurons after sliding window, Case: slide window=12

REFERENCES

[1] M. Amiri, L. Mohammad-Khanli, "Sur-vey on prediction models of applications for resources provisioning in cloud", Journal of Network and Com-puer Applications 82 (2017) 93–113

[2] M. Amiri, M., Mohammad-Khanli, L., Mi-randola, R., A sequential pattern mining model for application workload prediction in cloud environment, Journal of Network and Computer Applications (2018), doi: 10.1016/j.jnca.2017.12.015.

[3] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Toward energyefficient cloud computing: Prediction, consolida-tion, and over-commitment", IEEE Netw., vol. 9, no. 2, pp. 56–61, Mar./Apr. 2015.

[4] Boyun Liu, Jingjing Guo, Chunlin Li, Youlong Luo "Workload fore-casting based elastic resource management in edge cloud", Computers Industrial Engineering, Volume 139, January 2020

[5] https://github.com/beloglazov/planetlab-workload-traces

[6] H. Toumi, Z. Brahmi, Z. Benarfa, and M. M. Gammoudi, "Server load prediction using stream mining," in Proceedings of 2017 International Conference on Information Networking (ICOIN), pp. 653–661, IEEE, Da Nang, Vietnam, January 2017.

[7] S. Di, D. Kondo, and W. Cirne, "Host load prediction in a Google compute cloud with a Bayesian model," in Proceed-ings of 2012 International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12), pp. 1–11, IEEE, Salt Lake City, UT, USA, November 2012.

[8] N. K. Gondhi and P. Kailu, "Prediction based energy efficient virtual machine consolidation in cloud computing," in Proceedings of 2015 Second International Conference on Advances in Computing and Com-munication Engineering, pp. 437–441, IEEE, Dehradun, India, May 2015.

[9] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Ko-thari, "Server workload analysis for power minimization using consolidation," in

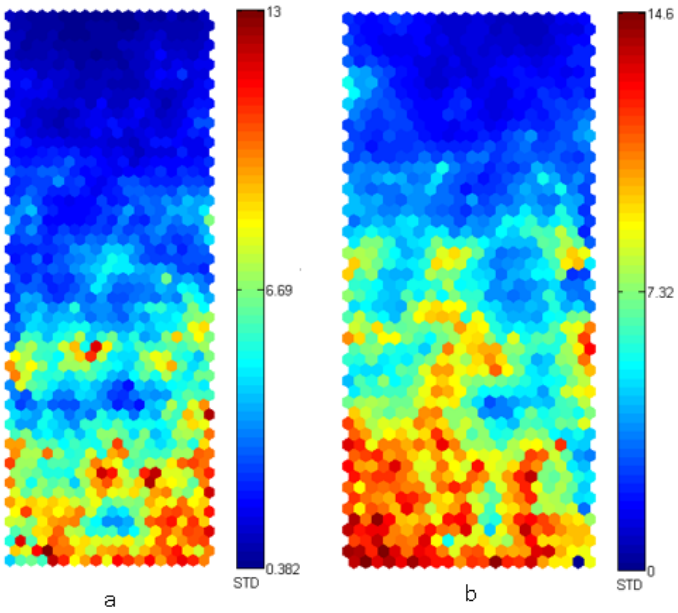

Figure 8: UMatrix showing Standard deviation of each cluster, a: Case Slide Window=6, b: Case Slide Window=12

	Slide Window=6	Slide Window=12
SOM	0,0125	0,0637
SVM	0,0813	0,0791
MLP	0,08	0,07825

Table II: RMSE ERRORs

	Slide Window=6	Slide Window=12
SOM	0,2063	0,41985
SVM	0,7329	1,17
MLP	0,7577	1,19

Table III: MAPE ERRORs

Proceedings of the 2009 Conference on USENIX Annual Technical Conference, p. 28, USENIX Associ-ation, San Diego, CA, USA, June 2009.

[10] B. Song, Y. Yu, Y. Zhou, Z. Wang, and S. Du, "Host load prediction with long short-term memory in cloud computing," Journal of Supercomputing, vol. 74, no. 12, pp. 6554–6568, 2018.

[11] J.-J. Jheng, F.-H. Tseng, H.-C. Chao, and L.-D. Chou, "A novel VM workload prediction using grey forecasting model in cloud data center," in Proceedings of International Conference on Information Networking 2014 (ICOIN2014), pp. 40–45, IEEE, Phuket, +ailand, February 2014.

[12] A. Beloglazov and R. Buyya, "Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data cen-ters under quality of service constraints," IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 7, pp. 1366–1379, 2013.

[13] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "An energy-efficient VM prediction and migration framework for overcommitted clouds," IEEE Transactions on Cloud Com-puting, vol. 6, no. 4, pp. 955–966, 2018.

[14] D. Minarolli, A. Mazrekaj, and B. Freisleben, "Tackling uncertainty in long-term predictions for host overload and underload detection in cloud computing," Journal of Cloud Computing, vol. 6, no. 1, p. 4, 2017.

[15] K. Mason, M. Duggan, E. Barrett, J. Duggan, and E. Howley, "Predicting host CPU utilization in the cloud using evolu-tionary neural networks," Future Generation Computer Sys-tems, vol. 86, pp. 162–173, 2018.

[16] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Workload analysis and demand prediction of enterprise data center applications," in Proceedings of the 10th IEEE Interna-tional SymposiumonWorkload Characterization (IISWC'07), pp. 171–180, September 2007.

[17] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscal-ing in the cloud using predictive models for workload fore-casting," in Proceedings of the IEEE International Conference on Cloud Computing (CLOUD '11), pp. 500–507, Washington, DC, USA, July 2011.

[18] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: a multiple time series approach," in Proceedings of the IEEE Network Opera-tions and Management Symposium (NOMS '12), pp. 1287–1294, Maui, Hawaii, USA, April 2012. Mathematical Problems in Engineering 11

[19] Y.Wu, K. Hwang, Y. Yuan, and W. Zheng, "Adaptive workload prediction of grid performance in confidence win-dows," IEEE Transactions on Parallel and Distributed Sys-tems, vol. 21, no. 7,pp. 925–938, 2010.

[20] Xing Chen, Haijiang Wang, Yun Ma, Xianghan Zheng, Longkun Guo: " Self-adaptive resource allocation for cloud-based software services based on iterative QoS prediction model", Future Generation Computer Systems, Volume 105, April 2020, Pages 287-296

[21] P. A. Dinda, D. R. O'hallaron, Host load prediction using linear models, Cluster Computing 3 (4) (2000) 265–280.

[22] M. Duggan, K. Mason, J. Duggan, E. Howley, E. Barrett, Predicting host cpu utilization in cloud computing using recurrent neural networks, in: The 8th International Workshop on Cloud Applications and Security 2017), 2017.

[23] A. Beloglazov, R. Buyya, Optimal online deter-ministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual ma-chines in cloud data centers, Concurrency and Computation: Practice and Experience 24 (13) (2012) 1397–1420.

[24] K. Mason, M. Duggan, E. Barrett, J. Duggan, E. How-ley, Predicting host CPU utilization in the cloud using evolu-tionary neural networks, Future Generation Computer Sys-tems (2018).

[25] B. Song, Y. Yu, Y. Zhou, Z. Wang, and S. Du, "Host load prediction with long short-term memory in cloud computing," The Journal of Supercomputing, pp. 1–15, 2017

[26] Kashifuddin Qazi ; Igor Aizenberg, " Cloud Datacenter Workload Prediction Using Complex-Valued Neural Networks", IEEE Second International Conference on Data Stream Mining & Processing August 21-25, 2018, Lviv, Ukraine

[27] Tarek Mahdhi, Haithem Mezni, A Prediction-based VM Consolidation Approach in IaaS Cloud Data Centers, The Journal of Systems & Software (2018), doi:https://doi.org/10.1016/j.jss.2018.09.083

[28] Calheiros RN, Masoumi E, Ranjan R, Buyya R (2015) Workload prediction using ARIMA model and its impact on cloud applications QOS. IEEE Trans Cloud Comput 3(4):449–458

[29] A.A. Rahmanian, M. Ghobaei-Arani, S. Tofighy, A learning automata-based ensemble resource usage prediction algo-rithm for cloud computing environment, Future Generation Com-puter Systems 79 (2018) 54–71, https://doi.org/10.1016/j.future.2017.09.049

[30] Jitendra Kumar , Ashutosh Kumar Singh, "Workload prediction in cloud using artificial neural network and adaptive differential evolution", Future Generation Computer Systems 81 (2018) 41–52

[31] Sadeka Islam; Keung, Jacky; Lee, Kevin; Liu, Anna. Empirical prediction models for adaptive resource provisioning in the cloud. In: Future Generation Computer Systems, Vol. 28, No. 1, 01.2012, p. 155-162. Research output: Journal Publications and Reviews Publication in refereed journal

[32] Salam Ismaeel; Ali Miri, Using ELM Techniques to Predict Data Centre VM Requests. In: 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing 3-5 Nov. 2015, New York, NY, USA

[33] Fahimeh Farahnakian;Pasi Liljeberg; Juha Plosila, Energy-Efficient Virtual Machines Consolidation in Cloud Data Centers Using Reinforcement Learning, 2014, 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

[34] Jing Chen ; Yinglong Wang , "An Adaptive Short-Term Prediction Algorithm for Resource Demands in Cloud Computing", IEEE Access, Volume 8, March 2020, Page(s): 53915 - 53930

[35] Gurleen Kaur, Anju Bala Inderveer Chana, "An intelligent regressive ensemble approach for predicting resource usage in cloud computing", Journal of Parallel and Distributed Computing, Volume 123, January 2019, Pages 1-12.

[36] Feng Qui; Bin Zhang, Jun Guo; A deep leanrning Approach for VM Workload Prediction in the Cloud, 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)
[37] Sadeka Islam; Jacky Keung; Kevin Lee; Anna Liu; Empirical prediction models for adaptive resource provisioning in the cloud; Future Generation Computer Systems Volume 28, Issue 1, January 2012, Pages 155-162
[38] Hishamf A. Kholidy "An intelligent swarm based prediction approach for predicting cloud computing user resource need" Computer Communications (2020), doi: https://doi.org/10.1016/j.comcom.2019.12.028
[39] Prabhat Kumar Upadhyay and al, "Scaled Conjugate Gradient Backpropagation based SLA Violation Prediction in Cloud Computing", 2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE) December 11–12, 2019, Amity University Dubai, UAE
[40] https://github.com/google/cluster-data
[41] Barnston, A., (1992). "Correspondence among the Correlation [root mean square error] and Heidke Verification Measures; Refinement of the Heidke Score." Notes and Correspondence, Climate Analysis Center.
[42] Juha Vesanto; Joha Himberg; Esa Alhoniemi and Juha Parhankangas, "SOM toolbox for Matlab" http://www.cis.hut.fi/projects/somtoolbox/package/papers/techrep.pdf
[43] R. Collobert and S. Bengio (2004). Links between Perceptrons, MLPs and SVMs. Proc. Int'l Conf. on Machine Learning (ICML)

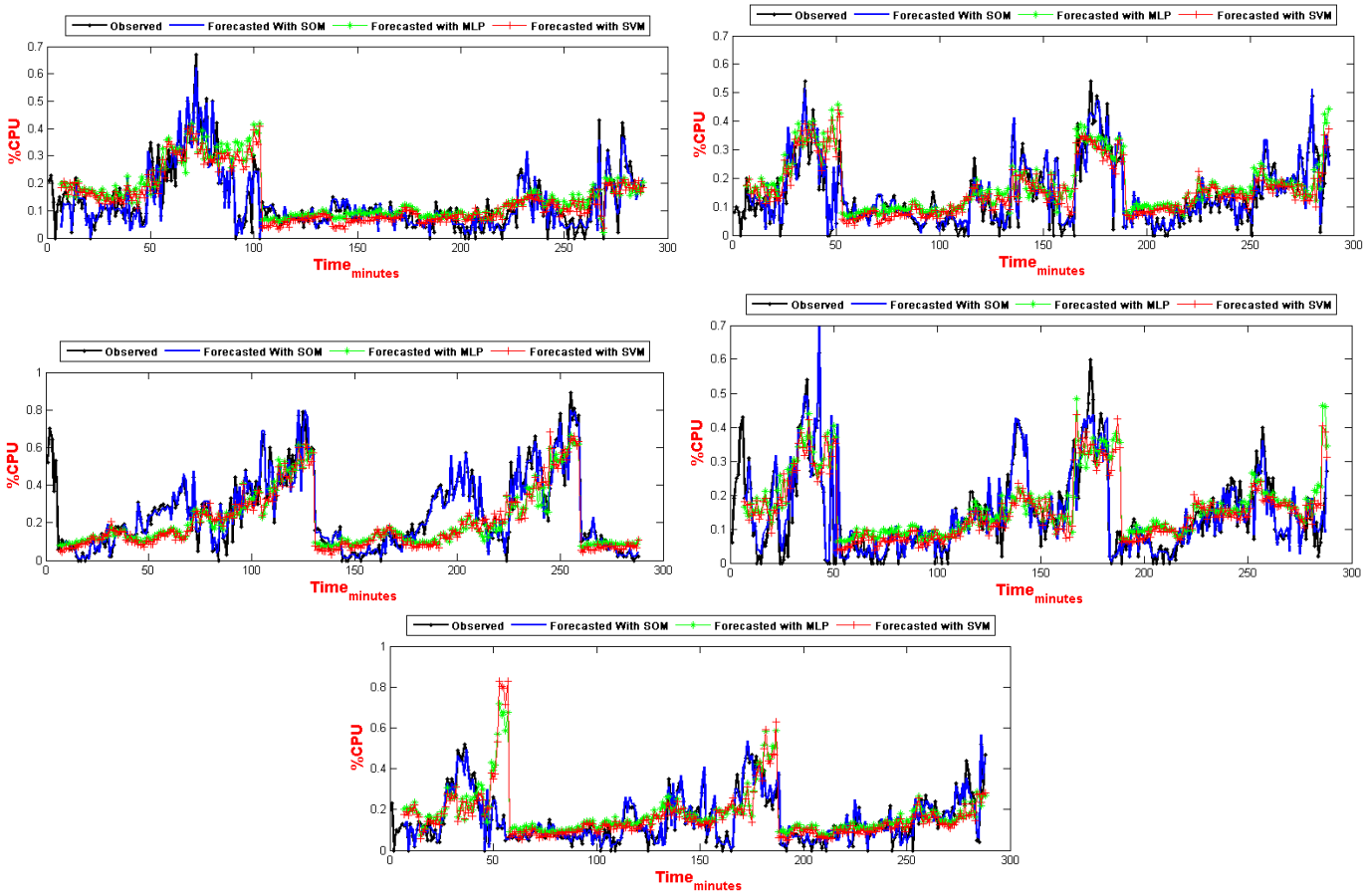

Figure 9: Observed and Predicted data for some VMs with the SOM, MLP and SVM methods, Case Slide Window=6

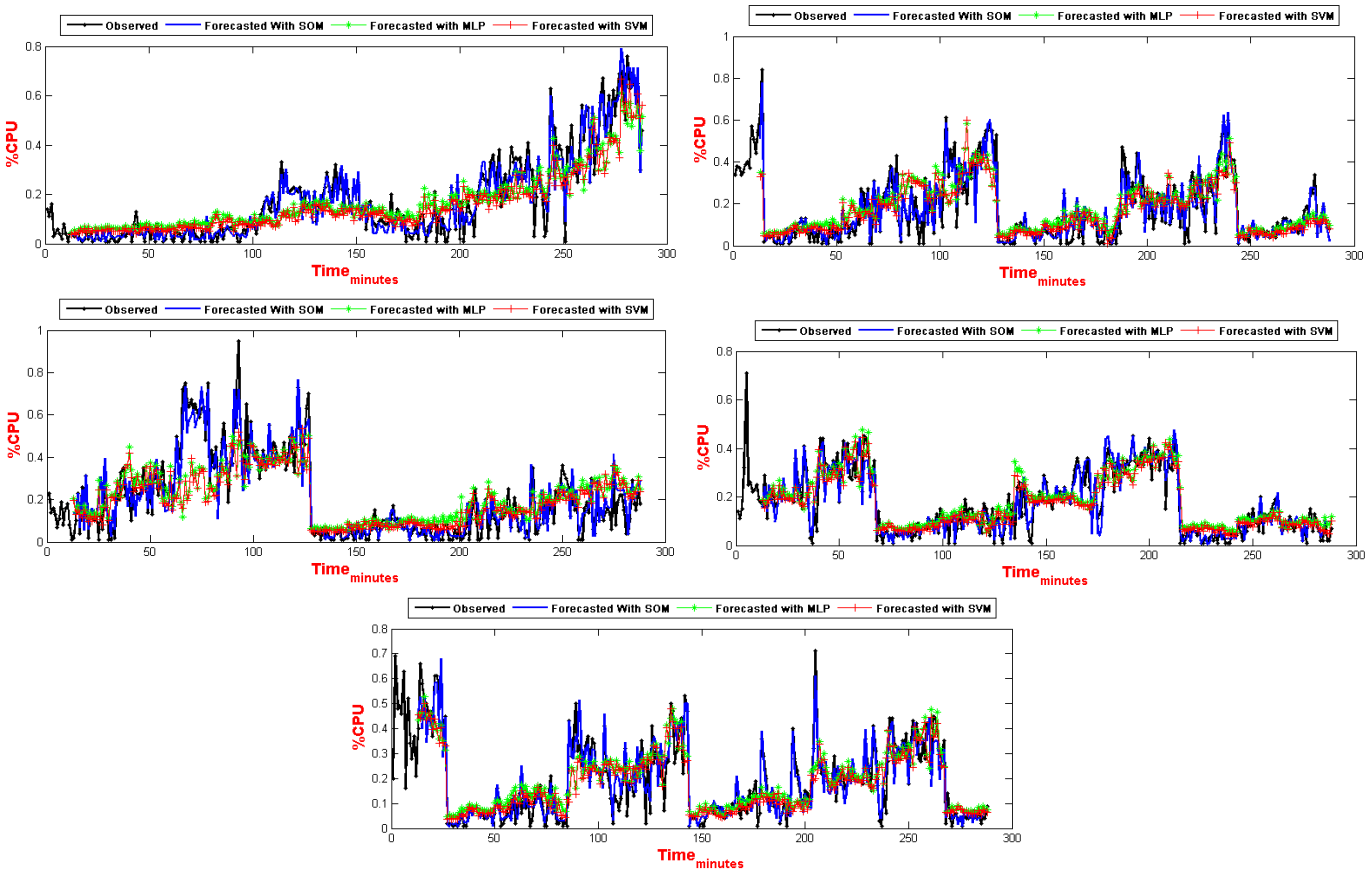

Figure 10: Observed and Predicted data for some VMs with the SOM, MLP and SVM methods, Case Slide Window=12

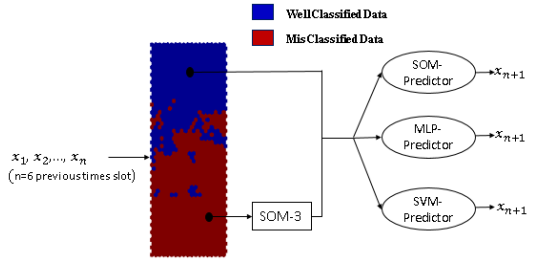

Figure 11: Prediction protocol for coming data

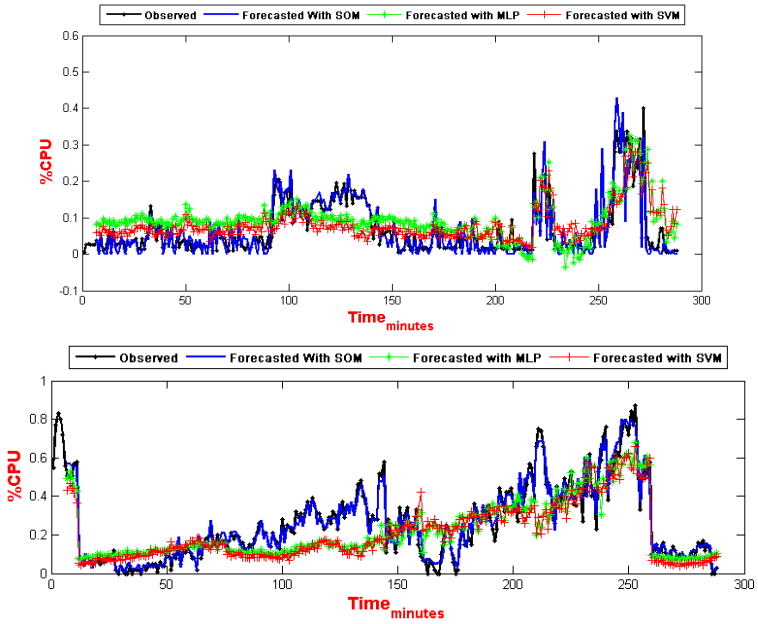

Figure 12: Observed and Predicted data for some VMs of not selected dataset using Slide Window=6

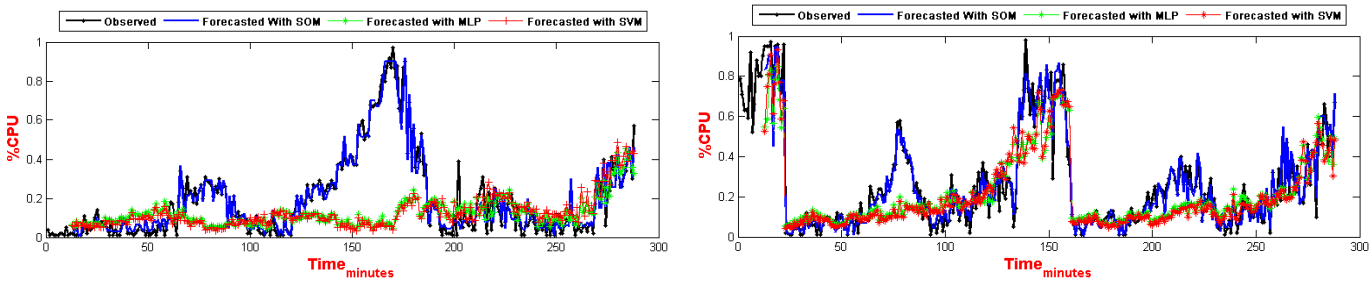

Figure 13: Observed and Predicted data for some VMs of not selected dataset using Slide Window=12