Cloud workload prediction and generation models

Gilles Madi Wamba, Yunbo Li, Anne-Cécile Orgerie, Nicolas Beldiceanu, Jean-Marc Menaud

Cloud workload prediction and generation models

Gilles Madi-Wamba*, Yunbo Li*†, Anne-Cécile Orgerie†, Nicolas Beldiceanu* and Jean-Marc Menaud*
*IMT Atlantique, LS2N, Nantes, France – Email: {gmadiw14, nicolas.beldiceanu, jean-marc.menaud}@imt-atlantique.fr
†CNRS, IRISA, Rennes, France – Email: {yunbo.li, anne-cecile.orgerie}@irisa.fr

***Abstract*—Cloud computing allows for elasticity as users can dynamically benefit from new virtual resources when their workload increases. Such a feature requires highly reactive resource provisioning mechanisms. In this paper, we propose two new workload prediction models, based on constraint programming and neural networks, that can be used for dynamic resource provisioning in Cloud environments. We also present two workload trace generators that can help to extend an experimental dataset in order to test more widely resource optimization heuristics. Our models are validated using real traces from a small Cloud provider. Both approaches are shown to be complimentary as neural networks give better prediction results, while constraint programming is more suitable for trace generation.**

I. Introduction and related work

In the past ten years, the use of cloud computing technologies has significantly increased. This increase comes with a major problem, namely energy consumption. In 2010, the electricity used only by data centers accounted for between 1.1% and 1.5% of the world energy consumption. For the US, it even has increased to a level between 1.7% and 2.2% [13]. Up to 30% of the electricity consumption of a typical data center is dedicated to cooling the system [17]. In [5], Beloglazov *et al.* showed that the electricity consumption of a data center is linked to its resources usage. A lot of research focuses on optimizing the resources usage of a data center. One common way to optimize resource utilization is to use workload prediction [12].

To predict the workload of a data center, in [11] and [12], Ismaeel *et al.* use a special type of neural network called an extreme learning machine. An extreme learning machine is a single-hidden layer feed-forward neural network, where the weights connecting inputs to hidden nodes are randomly assigned and never updated [10]. In their work, as in many work in the literature, they assimilate the workload of a data center to the number of virtual machines running in the data center. Therefore, forecasting the data center workload amounts predicting the number of virtual machines requests. Our approach assimilates the workload of a data center to its CPU usage that we model with times-series. In our context this approach is more suitable as the specification of the machines of a data center directly involves their CPU capacities and consequently their energy consumption [5]. Moreover, it can help to implement energy-efficient techniques such as CPU over-commitment [4].

Another issue encountered in designing resource optimization heuristics for data center is the availability of real workload traces to validate the proposed algorithms. In most of the cases, for reasons such as confidentiality, the real data are not available in enough quantities. One of the few available traces in the community comes from Google and provides a one-month trace for about 12,500 machines [9]. Yet, this kind of hyper-scale Clouds are only minor contributors to the global energy consumption of data centers (under 4%), while small- and medium-sized data centers account for half of this global electricity consumption [19]. In this paper, we focus on these small and medium energy-hungry Clouds that present workload patterns different from the ones experienced in large-scale public clouds [19].

We present two complementary models, one based on neural networks and the other one based on constraint programming to address these issues: workload prediction and lack of real traces. Therefore, the contribution of this paper is twofold.

- To predict the CPU workload of a data center, we present and compare two machine learning models, which are respectively based on constraint programming and neural networks. To the best of our knowledge, constraint programming has not been used to predict a data center workload.
- To extend any real workload data set, we present a workload traces generator. The generator uses the learned constraint model to generate time series that are similar to real workload traces.

The rest of the paper is organized as follows: Section II presents the real workload traces that we are using in this paper. Section III presents the machine learning models, and Section IV gives the workload generator model. Section V concludes this work.

II. Real workload description

Finding real traces for data center's activity is very difficult. For this, we have partnered with a french SME company: EasyVirt, a company specializing in virtualized data center analysis. In part of its activities, EasyVirt deploys software probes in the infrastructure of their clients and archives data in a MySQL database. These probes collect system resources' consumption of the physical servers and virtual machines. For confidentiality reasons, the companies in which the data were collected are not cited. Similarly, we can not distribute collected raw data. But, the analysis can be distributed. For this paper, we selected a representative trace of a mid-sized data center embedding 50 physicals servers for about 1,000 VMs.

More precisely, from a read-only account on the VMwareV-Center, the solution recovers static and dynamic information. The static information collected is: the Datacen-

ter/Cluster/Server/VM architecture, static information of physical servers (server model, CPU model, CPU frequency, cores number, RAM available, etc.), static virtual machine information (vCPU (virtual CPU) number, allocated memory, reserved memory, VMware Tools status, VMDK size, etc). Following the recovery of this information, the probes dynamically monitor physical servers and virtual machines in order to get their consumption and virtual machines lifecycle. Various resources are monitored: processor, memory, network and disk. This monitoring is realized every 30 seconds, without impacting VMwareVCenter performance. Data are stored in a classical relational database (MySQL).

The selected trace represents six months of activity, for a database of about 2GB. All of the results presented below are based on these traces.

III. Prediction models

In order to design a scheduling heuristic that aims at optimizing resource utilization, we need to know in advance the eventual distribution of the resource usage in time. To do so, we need a workload prediction model based on historical traces. This section presents two different approaches to build such a model. The first approach makes use of constraint programming while the second one uses a neural network.

A. Workload classification

As our dataset is consequent (six months, 50 physical servers), it can present various behaviors. As a preliminary step, we employ a clustering technique, widely used in data analysis, to group the traces into subsets of similar traces. K-means [16] is one of the simplest unsupervised learning algorithm for data clustering.

Given a series $(x_1, x_2, x_3, ..., x_n)$, the parameter $k(k \leq n)$ is used for specifying the number of clusters to be created. The K-means algorithm is following the three steps:

1) Determine k centroid coordinates corresponding k clusters
2) Calculate the distance from each object to the k centroids
3) Assign each object to a unique cluster and ensure the sum of distance within a cluster is minimized.

The objective of K-means [14] can be expressed as:

$$\arg\min_{s} \sum_{i=1}^{k} \sum_{x_j \in S_i} ||x_j - \mu_i||^2 \qquad (1)$$

where, μ_i is the mean of object in S_i. Once this preliminary step has been done, one can start learning the characteristics of the traces in each subset.

B. Constraint programming model.

In [4], we very briefly sketched a constraint programming model to predict the workload of a data center. This Section refines this model that will now be used both for prediction and for generating new time series workload (in Section IV). Given a set of real workload traces modeled as time series, we proceed as follows:

- *Clustering step*: we first clusterize them into p clusters, as already presented in Section III-A. This is motivated by the fact that the workload of a data center may both depend on the day (e.g. week, weekend) as well as of the type of services it provides at specific time periods.
- *Offline step*: we extract from each cluster cl_i, some key properties that correspond to typical features of the time series (e.g. its highest peak, its number of peaks). Using these key features, we build a model $m(cl_i)$ for each cluster cl_i. This offline preprocessing is the *learning step*. Finally, the *prediction model* is given by $\cup_{i=0}^{p-1} m(cl_i)$, the union of all the models $m(cl_i)$ of each cluster cl_i.
- *Online step*: at any time t, the prediction model should be able to foretell in real time the evolution of the workload at time $t + \epsilon$, where ϵ is a time lapse to determine.

This rest of this section is structured in the following way: (1) First we recall some background on time series, (2) then we present the offline step (3), and finally we describe the online step.

1) Background on time series: The references [3], [2] describe a large set of time-series constraints, where a time series constraint is characterized by the following concepts:

- *Signature of a time series*: the *signature* of a time series is a sequence of comparison operators taking values in the set $\{<, =, >\}$. Each element of the signature is obtained by comparing two adjacent input values.

- *Pattern*: a *pattern* is a regular expression over the alphabet $\{<, =, >\}$. To find a pattern occurrence in a time series, its signature has to be computed first. A pattern occurrence is a maximal occurrence of a sequence of characters from the signature that matches the regular expression of the pattern. Table I gives examples of patterns that may occur in a time series. Figure 1 provides a visual example of the *peak* pattern.

pattern	regular expression
`increasing`	$<$
`increasing_sequence`	$< (< \mid =)^* < \mid <$
`increasing_terrace`	$<=^+<$
`summit`	$(< \mid (< (= \mid <)^* <))\,(> \mid (> (= \mid >)^* >))$
`plateau`	$<=^*>$
`proper_plateau`	$<=^+>$
`strictly_increasing_sequence`	$<^+$
`peak`	$< (= \mid <)^* (> \mid =)^* >$
`inflexion`	$< (< \mid =)^* > \mid > (> \mid =)^* <$
`steady`	$=$
`steady_sequence`	$=^+$
`zigzag`	$(<>)^+(< \mid <>) \mid (><)^+(> \mid ><)$

TABLE I: Examples of patterns and their corresponding regular expressions.

- *Feature*: given a pattern occurrence, a *feature* is a quantifiable property of the pattern.
- *Aggregator*: given one or more occurrence of a pattern p and a feature f of p, an *aggregator* is a function (e.g. $\min$, $\max$) applied to the different feature values of each occurrence of pattern p.

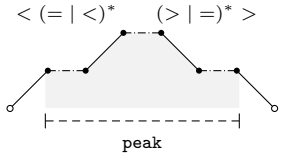

Fig. 1: Example of a peak pattern.

- *Footprint*: given a time series ts of length n and a pattern p, the *footprint* $fp_p(ts)$ of pattern p is a sequence of n values that identifies all occurrences of pattern p in the time series ts.

2) Offline step (learning step): Using constraint programming techniques, we analyze input workload traces to extract relevant features. For each cluster cl_i a model $m(cl_i)$ is built after a 3-step analysis of cl_i. For each pattern p of interest and for each input time series ts belonging to the cluster cl_i we proceed as follows:

(1) The number of occurrence of pattern p in the time series ts is computed.
(2) The footprint $fp_p(ts)$ of the pattern p is computed.
(3) Different values for the aggregation of features of pattern p are computed.

At the end, all the results for all patterns of interest are put all together. They are analyzed to extract different ranges of variation for each characteristic. Among these ranges, we select a subset of relevant ones. Those relevant ranges constitute the model $m(cl_i)$ of the cluster cl_i.

Example III.1. Consider a cluster cl containing the three following times series of length 15 each:

```
ts_1 = 5 5 4 4 6 3 7 8 9 6 3 3 1 1 1.
ts_2 = 4 3 1 1 4 2 8 5 6 2 6 5 9 8 9.
ts_3 = 3 4 5 6 6 5 6 3 4 4 2 2 7 6 9.
```

We consider the pattern $peak$, the feature $width$ of the $peak$, and the aggregations sum and max.

From steps (1) and (2) we obtain the following:

```
(nb_peak,2,[0,0,0,0,1,0,2,2,2,2,2,2,0,0,0]).
(nb_peak,5,[0,0,0,0,1,0,2,0,3,0,4,0,5,0,0]).
(nb_peak,4,[0,1,1,1,1,0,2,0,3,3,0,0,4,0,0]).
```

Each line of the form $(nb\_pat, val, fp_{pat}(ts_i))$. gives the number val of occurrence of the pattern pat in the time series ts_i as well as the footprint $fp_{pat}(ts_i)$ of the time series ts_i for the pattern pat.

From step (3) we obtain the following set of facts of the form $(aggr\_feat\_pat, val, fp_{pat}(ts_i))$, where each fact gives the value val of the aggregation $aggr$ of different values of the feature $feat$ for each occurrence of the pattern pat in the time series ts_i.

```
(max_width_peak,6,[0,0,0,0,1,0,2,2,2,2,2,2,0,0,0]).
(max_width_peak,1,[0,0,0,0,1,0,2,0,3,0,4,0,5,0,0]).
(max_width_peak,4,[0,1,1,1,1,0,2,0,3,3,0,0,4,0,0]).
(sum_width_peak,7,[0,0,0,0,1,0,2,2,2,2,2,2,0,0,0]).
(sum_width_peak,5,[0,0,0,0,1,0,2,0,3,0,4,0,5,0,0]).
(sum_width_peak,8,[0,1,1,1,1,0,2,0,3,3,0,0,4,0,0]).
```

We next put these results together and we obtain the following facts where each fact is of the form $range(r, pat, prop, min, max)$, where $prop$ is a feature of the pattern pat. min (resp. max) is the smallest (resp. largest) value that the feature $prop$ takes when evaluated wrt each time series of the cluster cl. Finally r is the range obtained by subtracting min from max.

```
range(3, peak,nb_peak,2,5).
range(5, peak,max_width_peak,1,6).
range(3, peak,sum_width_peak,5,8).
```

The range gives information on the amplitude of variation of a feature. Thus we can restrict the features of interest for the model of the cluster by setting a maximum allowed range.

Example III.2. If we set $max\_range = 4$ then the model $m(cl)$ of the cluster cl is:

```
m(cl) = {
range(3, peak,nb_peak,2,5),
range(3, peak,sum_width_peak,5,8).}
```

A time series ts is thus compatible with the model $m(cl)$ of the cluster cl if and only if :

- $2 \leq nb\_peak(ts) \leq 5$
- $5 \leq sum\_width\_peak(ts) \leq 8$

3) Online step, prediction step: This section presents how we use the model built in the offline step to predicts the next values of a time series at run time.

We first introduce the notion of prefix time series:

Prefix time series Given an index $t < n$ (where n is the length of a time series) the *prefix time series* induced by t ($pref(t)$) is the time series $x_0 x_1 \dots x_t$ which represents the workload from time 0 to time t. At time $t < n$, each value x_i ($i \in [0, t]$) is already known.

The prediction is done in three steps. Given a prefix time series ($pref(t)$), we first determine to which clusters it may belong. The second step gives an interval for the potential values of the prefix time series ($pref(t)$) at time $t+\epsilon$. Finally, we refine the interval to make the prediction more precise.

(1) First we check the compatibility of the time series ts with each cluster model $m(cl)$. The time series ts is *compatible* with cluster cl_i if the value for each characteristic of ts (e.g. number of occurrence of each pattern, footprint of each pattern, aggregation values) falls in the corresponding range of the model $m(cl_i)$ of cluster cl_i.
(2) For each compatible cluster cl, we compute the interval of potential values at time $(t + \epsilon)$, and note this interval by $I_{cl}(t + \epsilon)$. We make the union over the different compatible clusters and denote it by $I(t + \epsilon)$.
(3) We reduce the size of the interval $I(t + \epsilon)$. To do this, we consider the center time series of each compatible cluster cl_i and compute the footprint of the patterns $strictly\_increasing\_sequence$ and $strictly\_decreasing\_sequence$ to identify locations where these patterns occur. We use this information to reduce the interval $I(t + \epsilon)$.

4) Evaluation of the constraint programming based model : For the evaluation of this model, we started from a data-set of 500 real world workload traces times series. We partitioned the dataset into 5 clusters (using the method presented at Section III-A) of respective cardinalities $142, 79, 142, 106$, and 31. The model was constructed by learning from 70% of the time series of each cluster, and the prediction online step were done with the 30% remaining time series, that we call the test time series.

Those traces are of length 1008 each, meaning that each hour is represented by 42 time steps. To evaluate the cluster compatibility part of the prediction and the quality of the predicted interval, we reduced the resolution of the traces to one value per hour, that is each hour is represented by one timestep.

Each series is therefor of length 24 and is associated to a cluster. For each prefix of length k (with $k \in [3, 23]$) of each test time series we perform the following benchmarks.

(1) The first benchmark presented in Figure 2 evaluates the percentage of cases were there is at least one cluster that is compatible with the prefix time series.

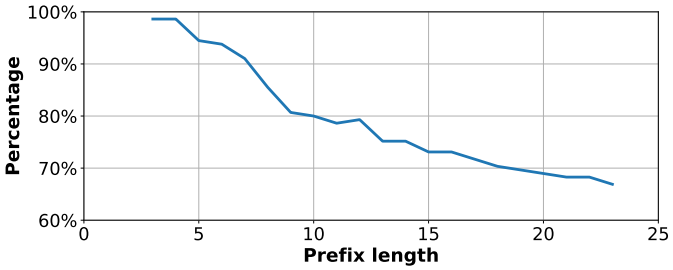

Fig. 2: Percentage of cases were there is at least one cluster that is compatible with the prefix time series.

(2) The second benchmark presented in Figure 3 evaluates the percentage of cases were the actual $(k+\epsilon)^{th}$ value of a test time series belongs to our predicted interval.

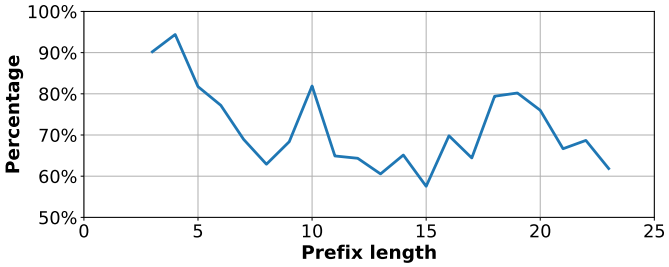

Fig. 3: Percentage of cases were the actual $(k+\epsilon)^{th}$ value of a test time series belongs to the interval $I(t+1)$ predicted.

From the curves depicted in Figures 2 and 3, we observe that the quality of the prediction depends on the cluster compatibility part of the prediction as both curves have a similar evolution.

We measure the quality of our prediction using a conventional tool in data analysis, called RMSE that we now recall.

Definition The *Root Mean Squared Error* is a standard measure used to evaluate the estimation of an unknown value. Given a sample of k predictions $pred(1), pred(2) \ldots, pred(k)$, and the a set of k observed values $val(1), val(2) \ldots, val(k)$, the RMSE of the prediction is estimated by :

$$RMSE = \sqrt{\frac{1}{k}\sum_{i=1}^{k}\big(pred(k) - val(k)\big)^2}$$

To compute the RMSE of our prediction using the constraint programming based model, we considered the center of the predicted interval $I(t+1)$ to be the predicted value. Figure 4 presents the RMSE of the prediction for each prefix length from 3 to 23.

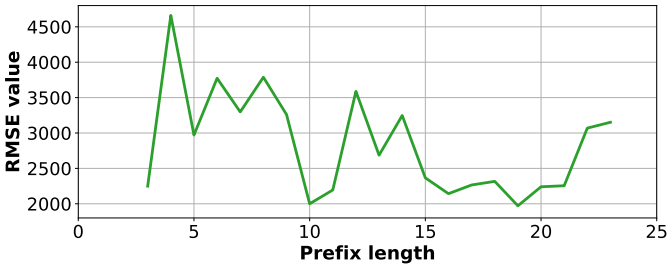

Fig. 4: Evaluating the RMSE of the prediction.

The average value of the RMSE for all prefix lengths is about 2800. This is satisfactory since the values of the time series used for these benchmarks range from 3000 to 29000. However the quality of the prediction relies on the quality of the classification of time series into clusters. Meaning that, the results may go worse if new time series that we fit into the system are not correctly classified. Since we are building a real time system, we can not neglect this aspect as new time series are also classified in real time and there is no new evaluation of the classification.

To overcome these issues, we elaborated a new model based on neural networks, the model is presented in Section III-C and does not need a clustering phase.

C. Neural networks model

Before describing our neural networks model, we recall some background on artificial neural networks.

1) Background on artificial neural network: An artificial neural network is a structured and interconnected group of nodes that reads an input in and computes an output. Figure 5 shows a simple neural network.

The Neural network of Figure 5 is structured as follows :

- One input layer having five nodes.
- One single hidden layer having three nodes.
- One output layer having a single node.

A neural network may have more than one hidden layer, and the number of nodes of each layer may vary. Each arrow of the neural network carries a *weight* w, and each node is associated with a value b called *bias*. Further, a neural network is associated with a function called the *activate function*. Figure 6 shows parameters associated with a neural network.

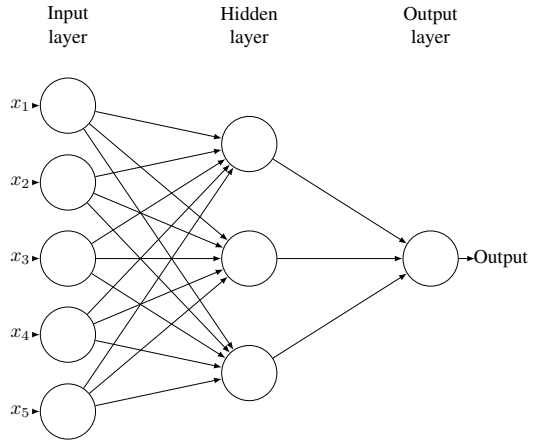

Fig. 5: A simple neural network.

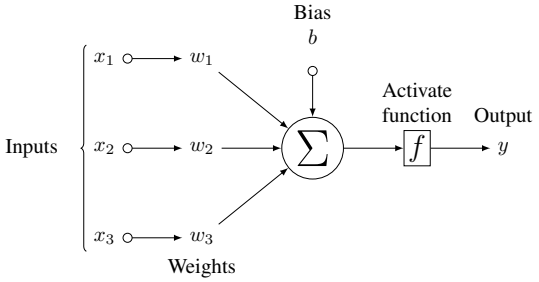

Fig. 6: Parameters of a neural network.

Using its weights, biases and activate function, a neural network computes the output y of a given input $x = (x_1, x_2 \dots x_k)$ with the formula $y = f(\sum_{i=1}^{k} w_i x_i + b_j)$, where w_i is the weight of the edge from input i to the node j of the hidden layer, and b_j is the bias of node j of the hidden layer. This formula assumes that the neural network has one single hidden layer, but can be adapted to more than one layer.

In general, we denote by W the k by m matrix of weights, by B the m by 1 matrix of biases, and by X the input vector of size k. This simplifies the expression to :

$y = f(WX + B)$ (see [15]).

The commonly used activate function is the *sigmoid function* $\sigma = \frac{1}{e^{-z}}$ where $z = WX + B$. In all what follows, we set the activate function f of our network to be the sigmoid function σ. The learning problem of our network consists in using the training data to find values for the matrices W and B in a way that minimizes the cost function C. To solve this problem, we use the gradient descent algorithm. Next section presents the gradient descent algorithm.

The gradient descent algorithm: We recall that in a neural network, the learning problem is to find weights W and biases B that minimizes a cost function of W and B. In our case the cost function is the mean squared error function $c(W, B)$.

To find W and B that minimizes the cost function, we use the gradient descent algorithm [7]. In all what follows, we consider c to be a function of w and b.

The gradient descent is a step by step algorithm. At the beginning, w and b are initialized to some values. The initial values may be chosen randomly. Lets denote w_0 and b_0 the values of w and b at step 0. The next step is to update values of w_0 and b_0 to w_1 and b_1 such that $c(w_1, b_1) \leq c(w_0, b_0)$. The change from $c(w_0, b_0)$ to $c(w_1, b_1)$ is Δc and is given by the formula $\Delta c = \frac{\Delta c}{\delta w}\Delta w + \frac{\Delta c}{\delta b}\Delta c$.

Let $\Delta v = (\Delta w, \Delta c)$ and $\nabla c = (\frac{\Delta c}{\delta w}, \frac{\Delta c}{\delta b})$ be the gradient of c, we have $\Delta c = \nabla c.\Delta v$.

We need to choose Δv such that Δc is negative i.e such that c decreases.

Let $\Delta v = -\eta \nabla c$ where η is a positive number, then $\Delta c = -\eta \|\nabla c\|^2$. Since $\|\nabla c\|^2$ and η are positive, then Δc is negative. The value η determines how high (or how small) is Δc, and is called the *learn rate*.

Next section presents how we use such a neural network to learn a model from a set of input time series.

2) Learning time series with a neural network : This section presents the design of our neural network as well as how it is trained to predict future values of time series.

The core idea behind the training of a neural network that predicts future values of time series is to learn a mapping from a prefix time series to its next value.

Given a set of real data called training inputs of the form $(X = pref(t), a(X))$ where $pref(t)$ is a time series of length t and $a(X)$ is the value of that time series at the next time step $t + 1$, the trained network should be able to predict a value $y(X)$ that is close to $a(X)$. Formally, this is done by solving a minimization problem. Let $C(W, B) = \frac{1}{2n}\sum_X \|y(X) - a(X)\|^2$ be a cost function where :

- n is the number of training input.
- W is the weight matrix.
- B the bias matrix.
- $y(X) = f(WX + B)$ is the output of the network with activate function f given the input X.

The function C is the mean squared error, also known as the quadratic error function. From its definition, we see that C becomes small when $y(X)$ tends to $a(X)$. So the objective is to find values of the matrices W and B that minimizes C.

Since the input to our neural network is a prefix time series, we designed our network to be compatible with the maximum length a prefix can take in our application. As we are dealing with time series of length 24 that represents the daily workload of a data center, we thus design our network with 24 input neurons. When we want to feed the network with a prefix time series of length $t < 24$, we append -1 to the prefix to obtain a time series of length 24. Also since we want our network to predict a single value, we design it to have a single output neuron.

To compute the gradient ∇c we used the backpropagation algorithm [1], a commonly used algorithm in machine learning.

The next section presents the evaluation of our neural network based prediction model.

3) Evaluation of the neural network based model: For the evaluation of this model, we used the same data as in the case of the constraint programming model (Section III-B4). To train the network, we extract prefixes of length 2 to 23 from each time series of the learning set, and feed the network with couple comprising those prefixes together with the next value of the time series. The number of training examples is therefor increased, since for each time series of length 24, from the learning set we extract 22 prefixes of length from 2 to 23. The neural network has 24 neurons in the input layer, one neuron in the output layer and one single hidden layer. The number of neurons in the hidden layer as well as the learning rate η were experimentally set to 65 and 0.2 respectively.

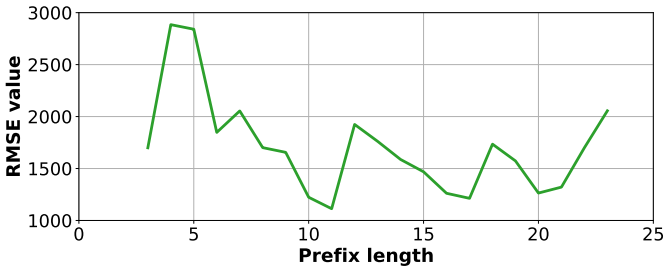

Fig. 7: Evaluating the RMSE of the prediction with the neural network model.

Figure 7 presents the RMSE for each prefix length from 3 to 23. With this model, the quality of the prediction is twice better than the results in the case of the constraint programming model were the best case RMSE value is above 2000. While the neural network model performs better than the constraint programming model, Figures 4 and 7, shows that both prediction models follow the same patterns. The explanation is that the neural network learns the patterns from the learning time series. The peaks in the RMSE prediction curve correspond to time points where there are many possible candidate patterns. The valleys in the RMSE prediction curve correspond to time points where the current pattern of the time series has been identified. When the pattern is identified, the prediction is more accurate than when it is not yet identified. This observation remains true for the peaks and valleys observed in the RMSE prediction curve of the constraint programming model in Figure 4, that is why both curves follow the same patterns.

We recall that the purpose of this model is to predict at real time the workload of the data center. Real time means that the prediction should be fast enough to be exploited by the resource optimization heuristic. The time is therefore a very important parameter to take into consideration when dealing with such model. We also want to show that our model scales correctly. To evaluate the scalability of our model, we increased the resolution of the workload traces, from 24 values per hour to 41 values per hour. These benchmarks were performed on a computer running Mac OS 10.10.5 Yosemite, with 16GB of memory and an Intel core i7 processor at 2.93 GHz.

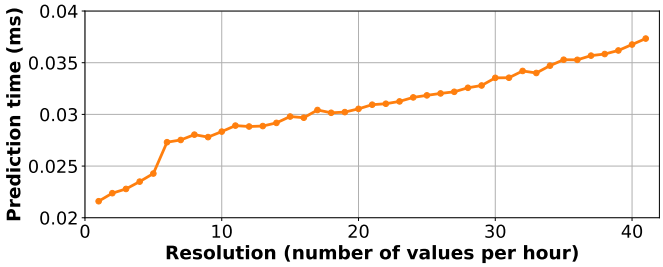

Fig. 8: Prediction time using the neural network.

From Figure 8, we can see that, with our model, the prediction time is less than $1ms$ even for time series of 41 values per hour, that is time series of length 984. This speed is due to the linear complexity complexity of forward propagation in a neural network. The number of multiplications needed to compute the output of the activation function of each neuron is linear with the number of neuron. The complexity is thus $O(k * 65) = O(k)$ where k is the number of neuron of the input layer, in our case, k is the length of the time series. These result contrasts withe the time needed for the learning of that same neural network.

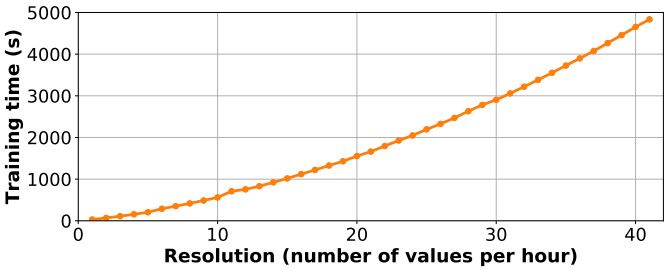

Fig. 9: Time needed to train a neural network

Figure 9 shows that the time needed to train the network goes above one hour for time series of resolution greater than 35 value per hour, i.e. series of length greater than 875. The complexity of the back propagation algorithm of our neural network is linear in the number of training epoch, the number of training example and the number of neurons. The fact that the training time may go above an hour is not an issue, since the training of the neural network is an offline process, where the time is not constrained.

These results led to Section IV on the use of the constraint programming model to generate new workload traces.

IV. Workload Generator

Sections III-B and III-C presented and evaluated two prediction models. On the one hand, the results established that the model with neural network performs better than the constraint programming model, and is also well adapted for real time prediction as it requires no data classification.

On the other hand, the advantage of the constraint programming model is that it learns a set of constraints that characterizes the time series of each cluster. This very same model can

then be used to generate time series that are compatible with a given cluster. Real data on server workload are in general not available in large quantities. Such a generator is very useful as it can generate as many data as needed. The generated data have the same patterns as the real workload data from the learning sets. The two models are thus complementary, as each one has its strengths. This section describes how new workload traces are generated in two steps from real workload data set. The first step builds a *constraint satisfaction problem* out of the available real workload traces, while the second step solves the problem to generate new traces. Before detailing the two steps, we recall some background on the notion of *constraint satisfaction problem*.

A. Background on constraint satisfaction problems

Constraint satisfaction problem A constraint satisfaction problem (CSP) [18] is a triple $P = (X, D, C)$ where :

- X is an tuple of $n > 0$ variables, $X = \langle x_0, x_1, \ldots, x_n \rangle$.
- D is a tuple of $n > 0$ domains, $D = \langle D_0, D_1, \ldots, D_n \rangle$ such that $x_i \in D_i$.
- C is a tuple of $k > 0$ constraints $C = \langle C_0, C_1, ..., C_k \rangle$.

We now define the notion of constraint.

Constrain Given a CSP $P = (X, D, C)$, a constraint $C_i \in C$ is a pair $\langle R_i, S_i \rangle$ where S_i is a subset of X and R_i is a relation over S_i. The relation R_i specifies the tuples of values forbidden among the Cartesian product of the domains of the variables in S_i.

Solution of a CSP Given a CSP $P = (X, D, C)$, a solution S of P is a tuple $\langle v_0, v_1, ..., v_n \rangle$ such that $v_i \in D_i$ and for every constraint $C_i = \langle R_i, S_i \rangle \in C$, the relation R_i holds.

We now present how to construct a CSP to generate new workload traces.

B. Building the constraint satisfaction problem, and generating new traces

Lets $P = (X, D, C)$ be the CSP that we need to construct in order to generate new workload traces. We define X, D and C as follows:

- [The set of variables X] We set the number n of variables of X to be the length of the real workload time series from the learning set. In our case $n = 24$. i.e $X = \langle x_0, x_1, \ldots, x_{23} \rangle$.
- [The set of domains D] Let m be the smallest value taken by the real workload time series from the learning set, at any time point; and let M be the largest value taken by the real workload time series from the learning set, at any time point. m and M are such that, for any time series ts of any cluster, and for any index $t \leq 23$, the t^{th} value of ts is included in $[m, M]$. $D = \langle D_0, D_1, \ldots, D_{23} \rangle$ where $D_0 = D_1 = \cdots = D_{23} = [m, M]$.
- [The set of constraints C] In Section III-B we presented the construction of a constraint programming model that captures all the features of the workload traces. We recall that a model $m(cl)$ of a cluster cl is of the form:

```
m(cl_i) = {
range(3, peak,nb_peak,2,5),
range(3, peak,sum_width_peak,5,8).}
```

To facilitate the reading, this model example considers only two characteristics of a single pattern. From the model we have the following information:

  - All the time series of cluster cl have at least 2 and at most 5 peaks, thus the amplitude of the range of variation of the number of peaks is 3.
  - When summing the widths of all the peaks occurring in each time series of cluster cl, we have a value between 5 and 8.

  We translate this information into constraints, our problem thus comprises two constraints on the whole set of variables:

  - $C_0 = \langle R_0, X \rangle$ with $nb\_peak(R_0, \langle x_0, x_1, \ldots, x_{23} \rangle)$ and $R_0 \in [2, 5]$,
  - $C_1 = \langle R_1, X \rangle$ with $sum\_width\_peak(R_1, \langle x_0, x_1, \ldots, x_{23} \rangle)$ and $R_1 \in [5, 8]$.

Using a constraint programming solver, we find the solutions $S_i = \langle v_{i,0}, v_{i,0}, ..., v_{i,23} \rangle$ of the CSP C. Each solution S_i is a time series of length 24 that has the same patterns learned from the real workload traces. The next section presents the evaluation of the quality of the time series generated by this model.

C. Evaluation of the CSP-based workload generator

To evaluate the quality of the time series generated, we ensure that it respects all the properties of the time series from the learning set. Those properties consist of all the features learned from the real workload traces. To solve the CSP model and generate solutions, we use a CSP Solver. For these benchmarks, we used Sicstus Prolog Solver [8], on a computer running Mac OS 10.10.5 Yosemite, with 16G0 of memory and an Intel core i7 processor at 2.93 GHz.

By construction of our CSP model from the learned features, and by definition of a solution of a CSP, every solution i.e. every generated time series respects all features that the model learned from each cluster. We confirmed this theoretical expectation by checking weather each generated time series of length from 24 to 984 respects the properties extracted from the respective learning sets. For each length of time series, there are at least 40 learned properties, modeled as constraints.

Further, we evaluated the time needed by the CSP to generate new time series according to the length of the time series. Figure 10 shows that for any resolution from 1 to 41, the time needed to generate a time series of corresponding length, that is of length 24 up to 984 is less than one second. This good performance is explained by the paradigm of propagation constraint [6] that is used to solve the CSP model and generate solutions.

D. Neural network-based workload generator

We also used the neural network to generate time series. To do so, we start from a small prefix, i.e. a prefix of length

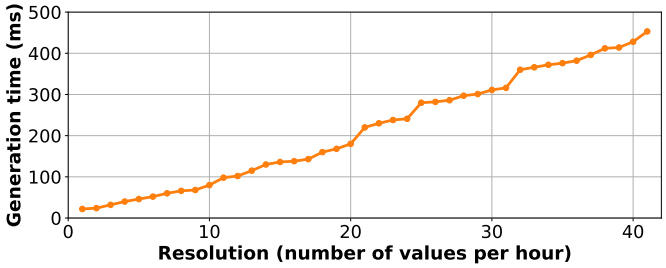

Fig. 10: Time needed to generate times series from the CSP

3, from every time series from the set of test time series; we predict the next value and repeat the process until we obtain a series of desired length. The process is described in Algorithm 1

Algorithm 1 Generate new time series with the neural network

$L_i \leftarrow Learning\ set$
$T_i \leftarrow Test\ set$
$n_i \leftarrow Neural\ network$
$S_i \leftarrow set\ of\ generated\ time\ series$
Desired length is N ▷ Each time series is of length N
$n_i.train(L_i)$ ▷ Train the neural network
for $st \in T_i$ **do**
  $p \leftarrow pref(st, 3)$ ▷ p is the prefix of length 3 of st
  while $length(p) < N$ **do**
    $y \leftarrow n.predict(p)$ ▷ Predict the next value
    $p \leftarrow p \bigoplus y$ ▷ concatenate y to p
  end while
  $S_i \leftarrow p$
end for

As expected, none of the time series generated with the neural network respects all of the learned characteristics. Another disadvantage in trying to generate a new time series with the neural network is that the neural network needs to start with a non-empty small prefix that it completed to create a full time series. On the other hand, the CSP model can rapidly generate full times series from scratch that verify all learned constraints.

V. Conclusion

In this paper, we have proposed two original workload prediction models for Cloud infrastructures. These two models, respectively based on constraint programming and neural networks, focus on predicting the CPU usage of physical servers in a Cloud data center. The predictions could then be exploited for designing energy-efficient resource allocation mechanisms like scheduling heuristics or over-commitment policies. We also provide an efficient trace generator based on constraint satisfaction problem and using a small amount of real traces. Such a generator can overcome availability issues of extensive real workload traces employed for optimization heuristics validation. While neural networks exhibit higher prediction capabilities, constraint programming techniques are more suitable for trace generation, thus making both techniques complementary. Our future work includes providing a website to access on-demand datasets produced by our generator using various real workloads that cannot be made directly publicly available.

Acknowledgments

This work has received a French state support granted to the CominLabs excellence laboratory and managed by the National Research Agency in the "Investing for the Future" program under reference Nb. ANR-10-LABX-07-01.

The authors would like to thank the EasyVirt company for providing them real workload traces from Cloud providers.

References

[1] M. K. S. Alsmadi, K. B. Omar, S. A. Noah *et al.*, "Back propagation algorithm: the best algorithm among the multi-layer perceptron algorithm," *IJCSNS International Journal of Computer Science and Network Security*, vol. 9, no. 4, pp. 378–383, 2009.

[2] E. Arafailova, N. Beldiceanu, R. Douence, M. Carlsson, P. Flener, M. A. F. Rodríguez, J. Pearson, and H. Simonis, "Global Constraint Catalog, Volume II, Time-Series Constraints," *CoRR*, vol. abs/1609.08925, 2016.

[3] N. Beldiceanu, M. Carlsson, R. Douence, and H. Simonis, "Using finite transducers for describing and synthesising structural time-series constraints," *Constraints*, pp. 1–19, 2015.

[4] N. Beldiceanu, B. D. Feris, P. Gravey, S. Hasan, C. Jard, T. Ledoux, Y. Li, D. Lime, G. Madi-Wamba, J.-M. Menaud, P. Morel, M. Morvan, M.-L. Moulinard, A.-C. Orgerie, J.-L. Pazat, O. Roux, and A. Sharaiha, "Towards energy-proportional clouds partially powered by renewable energy," *Computing*, pp. 3–22, 2016.

[5] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[6] C. Bessiere, "Constraint propagation," *Foundations of Artificial Intelligence*, vol. 2, pp. 29–83, 2006.

[7] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, 2010, pp. 177–186.

[8] M. Carlsson, J. Widen, J. Andersson, S. Andersson, K. Boortz, H. Nilsson, and T. Sjöland, *SICStus Prolog user's manual*. Swedish Institute of Computer Science Kista, Sweden, 1988, vol. 3, no. 1.

[9] Google, "Cluster data 2011," https://github.com/google/cluster-data, 2011.

[10] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[11] S. Ismaeel and A. Miri, "Using ELM techniques to predict data centre VM requests," in *IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)*, 2015, pp. 80–86.

[12] ——, "Multivariate Time Series ELM for Cloud Data Centre Workload Prediction," in *International Conference on Human-Computer Interaction. Theory, Design, Development and Practice*, 2016, pp. 565–576.

[13] J. Koomey, "Growth in Data Center Electricity Use 2005 to 2010," Analytics Press, Aug 2011.

[14] I. B. Mohamad and D. Usman, "Standardization and its effects on k-means clustering algorithm," *Research Journal of Applied Sciences, Engineering and Technology*, vol. 6, no. 17, pp. 3299–3303, 2013.

[15] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[16] V. R. Patel and R. G. Mehta, "Impact of outlier removal and normalization approach in modified k-means clustering algorithm," *IJCSI International Journal of Computer Science Issues*, vol. 8, no. 5, 2011.

[17] S. Pelley, D. Meisner, T. F. Wenisch, and J. W. VanGilder, "Understanding and abstracting total data center power," in *Workshop on Energy-Efficient Design*, 2009.

[18] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming*. Elsevier, 2006.

[19] J. Whitney and P. Delforge, "Data Center Efficiency Assessment," NRDC white paper, 2014.