A machine learning approach to predict Virtual Machine Workload

Ivan Zarro
ivan.zarro@tecnico.ulisboa.pt
Instituto Superior Técnico
January 2021

ABSTRACT

More companies have been moving to the cloud recently, due to the several advantages that cloud provides such as elasticity. Elasticity allows companies to acquire or release computational resources based on their needs, without human intervention, with the help of the auto-scaler mechanism. The most common one is the reactive auto-scaler which is very limited and sometimes ineffective since SLA are violated and resources are underutilized. The alternative is the predictive auto-scaler that forecasts the workload of VMs in order to be prepared before the demand occurs. Microsoft Azure and Google Cloud don't provide this tool and only recently Amazon AWS start offering it. The proposed solution aims to verify if the combination of three different machine learning algorithm (ARIMA, LSTM and Random Forest) can forecast the VM workload, in terms of CPU metric, better than each of the individually algorithms. The results have proved that the combination of the algorithms, in three different strategies, provided better results than each of the individual algorithms.

Author Keywords

Virtual Machine, Workload Prediction, Machine Learning, CPU

INTRODUCTION

Since the cloud since offers elasticity, companies are deploying their applications in there, which allows them to dynamically acquire and release computational resources depending on the needs.

To acquire or release computational resources dynamically, it is necessary to do it without human intervention, otherwise it would imply one of the following two things:

- **Over provisioning**: which would incur extra costs and an under-utilization of the available resources [23].
- **Under provisioning** : which would have an impact on the performance and violation of the Service Level Agreement (SLA) [23].

To provide elasticity and comply with the Quality of Service (QoS) requirements, major cloud platforms like Amazon AWS, Microsoft Azure and Google Cloud offer the auto-scaling mechanism. This mechanism has four options, which are: deploy VMs, shutdown VMs, add more resources to VMs or remove resources from VMs. The most popular one is the reactive auto-scaling and it's a simple mechanism. The way it works is to do a pre-determined action when a defined threshold value is crossed, for example deploy a new instance when the CPU value of the virtual machine crosses the 70%. However, sometimes is not very efficient, mainly because of its reactive nature. Many problems arise with this approach, for instance, in the latter example a new instance will be deployed after the threshold is reached but the new instance might only be utilized for one request that uses 5% of CPU. This means that there was not a need to deploy a new instance, because the other one could handle it without compromising the QoS. Other problem that could arise with the latter example is that suddenly the demand starts to rise in a relatively fast way that the system gets overloaded and, since virtual machines instantiation takes time, the performance is affected in such a bad way that users of the application might leave [3].

Both problems lead to unnecessary costs that, with today's technology, could be solved since the workload often follows patterns. If we could predict that future demand is not high enough to deploy a new instance or that is high enough to have virtual machines ready to respond to the incoming request, then we could maximize the use of the resources and comply with SLA, meaning a reduction in costs and energy. The mechanism that could do this is called predictive auto-scaler, which looks to the history of the workload to predict the future workload to make proactively decisions before the workload change.

The real challenge is forecasting the future workload with accuracy enough that is better than the common reactive model.

Several studies have been conducted in order to find the best combination of metric and machine learning algorithm that could predict the workload, however there is a lack of unanimously between the researchers, mainly because some combinations are great in a set of conditions, but not to good on other [14]. In addition, Microsoft Azure and Google Cloud don't offer any tool that could predict the workload of Virtual Machines and was only in November of 2018 that Amazon AWS presented theirs, however there is not any research about its performance. This leads to companies to implement their own predictive auto-scaler in case they want to reap the benefits of it.

Objectives

Since previous research shows that proposed solutions to predict workflow perform reasonable in a set of conditions, but unsatisfactory on other, we decided to verify if a combination of 3 different Machine Learning algorithms can outperform the prediction of each one of them alone. In order to achieve the desired goal, this research focused on:

- Find a dataset that contains CPU usage(%) of Virtual Machines.
- Develop 3 different machine learning models capable of predicting the CPU usage(%).
- Develop different strategies to implement different systems. By different strategies we mean different combination of the predictions in order to maximize the accuracy.

BACKGROUND

In this section we present the Background related to our research, namely concepts about Auto-Scaler, Virtual Machine (VM) workload prediction, Time Series and the chosen algorithms (ARIMA, LSTM and Random Forest)

Auto-scaler

In several occasions, applications being executed in cloud computing environments will have to scale their computing resources when encountered with different workload requirements. The auto-scaling problem for applications can be defined as how to, without human intervention, provision or deprovision computing resources in order to satisfy fluctuant application workload, while using the least amount of resources and avoiding SLA violations [23].
The most common are the reactive auto-scalers that just respond to the system status, for example, if the defined threshold reaches the 70% of CPU utilization, deploy a new VM [20]. Amazon Auto-Scaling service uses this type of auto-scaler [23].
However, only take action when a threshold is crossed leads to several issues, such as:

- the instantiation of new VMs is not an immediate operation and clients might notice, damaging user experience. This can lead to users leave the application, potential financial losses and not meeting the minimum required QoS [3].
- if the actual CPU utilization keeps changing from 65% to 75%, then the system keeps adding unnecessary VMs deployment, thus possible increase in costs and damaged QoS [20].
- unnecessary VM migration when a VM is constantly being migrated from one physical server to other [27], which also can damage user experience.

As we can see reactive auto-scalers are not very efficient. The other option which is predictive auto-scaler, tries to predict the workload so that the system is prepared before the demand occurs, which enable SLA's QoS targets to be met with the least resources possible [3].

Benefits of Predicting the Workload of VMs

In case public cloud providers, private cloud owners or even clients want to improve the efficiency of their physical/virtual machines, they should try to predict the future workload in order to reduce costs and maximize the use of their resources.With the prediction of Virtual Machines behaviour, several problems can be mitigated, lowering the costs and increasing overall customer satisfaction. Some of the benefits are presented below:

- **Maximize resource utilization:** by predicting workload, we can estimate the minimum necessary active Physical Machines hosting the virtual machines, thus decreasing energy costs.
- **Reduce Virtual Machine migration:** it's known that migrating a virtual machine takes time, Quality of Service can take a hit and most of the times is unnecessary. By predicting future workload, we can check if it was only a small peak load and don't do anything, or prepare the migration of the virtual machine before the high demand occurs. holidays.
- **Reduce Virtual Machines instantiation:** instantiating virtual machines is not an immediate operation and end-users might notice, leading to several problems such as users leaving the application or poor QoS [3]. If instead of waiting for peak loads to happen, we allocate resources in advance, the risks of losing clients or not deliver acceptable QoS drop.

Time Series

Time Series is a collection of data points that are placed in a sequence by the same order they were collected, over the corresponding period of time. This means that the order of the data observed is crucial, unlike other type of machine learning datasets where each data point is handled in the same way, regardless of the place in the dataset.
Accordingly to Shumway et al. [28], we can interpret Time Series as a stochastic process, since it is a set of observations x1, x2 . . . of a random variable Xt, indexed by time t, where x1 corresponds to the value of the first timestamp, x2 corresponds to the value of the second timestamp, and so on.

Autoregressive Integrated Moving Average model

Widely known as ARIMA, this model is the fusion between the ARMA model with the Integrated component. This component, I(d), allows the model to transform non stationary into stationary time series, by applying one or more simple differentiations. The order *d*, is the number of times the time series will be differentiated, which typically is no more than two. A model with order *d* equal to zero assumes that the original series is stationary. A model with order *d* equal to one might mean that the time series has a constant average trend (non-stationary), so it should result into a stationary time series after applying the differencing part. The differencing part I occurs before the ARMA part, since stationarity is essential in order to apply ARMA with greater accuracy. [31].

Stationarity

In order for ARIMA to make predictions, the observed time series should be stationary. This means that its statistical characteristics don't change over time, such as the mean, variance and covariance [31].This makes the series easier to be analyzed by the learning model. A non-stationary time series shows seasonal effects, trends, and other structures that depend on the time index, unlike a stationary one.

Stationarity tests

Since all the stationarity properties are hard to achieve simultaneously, there is a higher probability of the times series to not be stationary.

There are several ways to verify if a time series is stationary or not. Two of the most used methods are:

- **Augmented Dickey-Fuller (ADF) :** type of statistical test developed to verify the null hypothesis that a unit root is present. If the unit root exists, then the null hypothesis is accepted and the series is considered non-stationary. To determine the result, we need to look at the *p-value* of the test. If the *p-value* is lower than the value of a certain threshold, typically 0.05, then the test rejects the null hypothesis, which might mean that the time series is stationary. Additionally to the *p-value*, we should also look at the *Test Statistic*. If the *Test Statistic* is less than the critical value at 5% or even 1%, then it means that we can reject the null hypothesis with a significance level of 5% or 1%, respectively. Otherwise the null hypothesis is accepted and the time series is non-stationary [4] [15].

- **Kwiatkowski-Philips-Schmidt-Shin (KPSS) :** usually used as a complement of the Augmented Dickey-Fuller test. This test verifies the null hypothesis of the absence of the unit root, unlike the previous test. To determine the result we need to also look at the *p-value* of the test. If the *p-value* is lower than the value of a certain threshold, typically 0.05, then the test rejects the null hypothesis, which might mean that the time series is non-stationary [15] [17].

Supervised Learning

Supervised learning is becoming more common nowadays. In this type of machine learning, the input data of the training data is paired with the corresponding output value to learn the mapping function from the input to the output. The mapping function will serve to predict the output value for a given input value.

Ensemble Learning

Ensemble learning is a process that builds a predictive model by incorporating several models. The idea, which the results are well known, is that the aggregation of models tend to improve the prediction accuracy [24]. It combines the models, with different weights or not, so that the performance is better than all of them alone. Analogously, when humans want to make a difficult decision, they look for other sources of information.

Ensemble techniques, like boosting and bagging, produce a strong learner by grouping weak learners in order to solve problems, like supervised learning ones.

A weak-leaner is a type of algorithm which the prediction results are only marginally correlated with the true values, for instance it can predict slightly better than random guessing.

A strong-learner is a type of algorithm which the prediction results are well-correlated with the true values.

Random Forest

Random Forest is a type of Decision Trees ensemble learning method, hence the name. Before we present Random Forest, it is import to understand what Decision Trees are, since they are the core of the method. Decision Tree is a relatively simple predictive model that utilizes a set of binary thresholds to predict continuous values or rules to predict categorical values, depending if we are dealing with a regression or classification problem, respectively. Decision trees are relatively fast in terms of execution speed, however if it grows in complexity it might lead to a loss of accuracy in unseen data and sub optimal accuracy on training data [9]. So, in 1995, Tin Ho [9] proposed a method with the purpose of increasing the accuracy of both unseen and training data. The method was denominated random decision forests and consists on following the stochastic modeling principle and building several trees in randomly selected sub spaces of the feature space [9]. Experiments made by Ho, proved the validity of his theory, where trees complement their predictions and improve the accuracy if they are trained on distinct sub spaces.

This experiment led Breiman to combine his bagging ensemble technique with the concept developed by Ho, and developed the well known machine learning algorithm called Random Forests [2].

Random Forests take advantage of the bagging method to solve the common problem of overfitting that happens with Decision Trees, by reducing the variance without increasing the bias. As previously described, this is accomplished by training several decision trees in parallel and in different subsets of data. Once all decision trees have been established, the model will average the forecasts and come out with a final value. The performance of the model can be improved with a higher number of decision trees, however it comes with a cost, since the higher the number of decision trees, the higher the execution time. Different sets of hyperparameters should be evaluated in order to achieve the results we want [2].

Deep Learning

Deep Learning is one of the most popular and promising Machine Learning methods that has demonstrated great accomplishments in diverse fields and applications such as handwritten digits, speech recognition and stock market forecasts [19]. Deep Learning involves around the use of artificial neural networks, which are based on the actual communication and computation that occurs in the brain of animals.

Artificial Neural Networks

Artificial Neural Networks, mostly known as Neural Networks (NN), is a mathematical composition that can identify complex nonlinear relationships between input and output data. It has been proven by the literature the efficiency and usefulness of this computing system, especially when the characteristics of the problems are hard to describe using physical equations [21].

Artificial Neural Networks are composed by a single input layer, a single output layer and a defined number of hidden layers. Each one of these layers is constituted by the most basic unit of a neural network, the neuron. Similarly to the human brain, neural networks is a set of many neurons wired together with the purpose of establishing communications between them. A layer is a set of neurons grouped together where the learning process of the neural network happens. There are three types of layers, but only the input and output ones are mandatory in a neural network.

Recurrent Neural Networks

Recurrent Neural Networks (RNN) belong to the family of Artificial Neural Networks (ANN) and are specialized in processing sequential data [25]. Contrary to most NNs, RNN exhibit temporal dynamic behaviour because they are capable of using their internal memory in order to process sequential data of variable magnitude. This makes RNNs well-suited for time series prediction.

Long Short-Term Memory

Long Short-Term Memory, mostly known as LSTM, is a type or RNN architecture which have been proven by the literature to outperform other type of RNNs on plentiful temporal processing tasks [8].

Hochreiter and Schmidhuber developed LSTM in 1997 [10], with the purpose of overcoming one of the main limitations of RNNs, which occurred during the requirements of learning long-range time dependencies [26]. Nowadays, LSTM is not only capable of learning long-term dependencies, but also is widely used of solving a variety of problems, including time series forecasting.

LSTM is a variant of RNNs that has a way of carrying information across many timesteps [5]. Francois Chollet described this additional feature with an easy to understand metaphor:

"Imagine a conveyor belt running parallel to the sequence you're processing. Information from the sequence can jump onto the conveyor belt at any point, be transported to a later timestep, and jump off, intact, when you need it. This is essentially what LSTM does: it saves information for later, thus preventing older signals from gradually vanishing during processing" [5].

RELATED WORK

In this section we present the studies that most influenced this research and the main conclusions we reached regarding the metrics and algorithms.

Metrics

Several attempts have been made in order to estimate future resources to deal with the demand, however there is not a consensus in which metrics should the algorithms have as an input. All the methods presented here differ in terms of what is their input. Most methods focus on a combination of these three metrics CPU, Memory and RAM usage. CPU is a common metric in almost all the methods. In [12], Jheng *et al* tried to predict the workload computation with the average of CPU, Memory and RAM utilization of a Virtual Machine. In [29], Tseng *et al* tried to predict CPU, memory utilization and energy consumption, based on historical data of those metrics. In [27], Sato *et al* proposed predicting resource usage based on historical CPU and RAM metrics, in proportion to the number of accesses. In [20], Radhika *et al* proposed a method that gathers CPU and RAM utilization from a Virtual Machine to predict the workload. In [30], Xue *et al* present "PRACTISE" which is a neural network-based framework to predict future loads, peak load, and when those are going to happen. They extracted data from IBM data centers and retrieved 4 metrics: CPU, memory, disk and network bandwidth. A difference between these methods and the next two, that also collect CPU, is that the next two ones focus on collecting data from a group of Virtual Machines instead of just a single virtual machine. In [22], Qiu *et al* criticize other approaches that estimate demands based on historical data from a single Virtual Machine. Since the cloud is a complex network system of VMs, there is a temporal and spatial correlation between a group of VMs that those models neglect. Besides that, due to the high veracity data in data sets, there is a vast amount of information that is incomplete and can lead to inaccurate prediction if it's based on only one Virtual Machine history data. In [22], Qiu *et al* focused on CPU utilization, but their approach could also be applied to other metrics, such as memory utilization. Also, in [13], they show that prediction based on workload from individual VMs tends to provide inaccurate results, due to the fact that the workload is noisier and more random, thus less predictable. In addition, they found that VMs that are configured to cooperate on an application, have workloads that tend to vary in a relatable way. As a result, they use this to filter noisy from individual VM's data, thus improving prediction accuracy. In [11], Iqbal *et al* conducted a study where they observed that the behaviour of the same type of VMs produces different performance on the same workload. They claimed that the usual machine-learning models to predict VM's performance would become obsolete, once the performance varies, needing to be retrained in order to predict future required resources. So, they proposed an algorithm that learns from some resource configurations that are maintained, regardless of VM's performance, such as arrival rate and response time.

Machine Learning Algorithms

Once the metrics are gathered, we can exploit the characteristics of the workload by using machine learning techniques in order to find patterns and predict future workload. Similar to the metrics, there isn't an agreement in which machine learning algorithm to use, having the researches focused on different algorithms. In [12], Jheng *et al* tried to predict the

workload using Grey Forecasting model. They did an experiment where results show that grey algorithm is appropriate for tendencies that continually increase or decrease, however when the tendency becomes more complex, it is not suitable. Despite this conclusion, it's not very clear the details in which the experiment occurred nor how other algorithms would perform under the same conditions. In [29], Tseng *et al* used a Genetic Algorithm to predict the future workload. They did an experiment where the proposed Genetic Algorithm is compared with the Grey Forecasting model. Both algorithms try to predict CPU and memory utilization on two type of tendencies, one being stable raise and fall, while the other is an unstable fluctuation. The experiment consists of 30 time slots (1 slot = 1hour) and since algorithms like GA need historical data to train the model, the first 2 slots are for collecting data. However only at the 19th time slot is the GA capable of find optimal solutions. They concluded that the GA model has a superior prediction accuracy compared to the Grey model, because grey model prediction is based on historical tendency, it fails to predict at the turning point of it [29]. In [27], Sato *et al* proposed an autoregressive model for predicting the resource usage, based on the history of CPU and RAM usage, in proportion to the number of accesses. However, no experiments were done to see how the algorithm would perform neither comparison with other algorithms. In [20], Radhika *et al* used a deep learning technique termed Recurrent Neural Network with Long Short-Term Memory (RNN-LSTM) to predict CPU/RAM utilization. They claim that the algorithm can predict with accuracy in case of any variance and that, compared to timeseries techniques like ARMA and ARIMA, this method gives better results for predicting future workload. However, they didn't provide any experiment where the they compared the proposed algorithm with any other.
In [3], Calheiros *et al* chose the time-series model ARIMA. Since this model has been successfully utilized for time series predictions and that workload typically follows time-dependent patterns, this model gives good expectations. Workload patterns may vary depending on what is being processed, so they focused on trying to find patterns in web requests, because previous research observed that web workloads tend to present strong autocorrelation. However, this model may not be effective if the workload they are trying to predict is not well known nor time-dependent, thus and accordingly to them, the model needs a strong knowledge about the application workload behaviour. The workload estimation is given at a one time-interval in advance. This interval should be long enough for VMs to be instantiated if needed and without compromising the QoS. They did an experiment where they train the model with 3 weeks of data and try to predict the next one. Although they only provided one experiment, the results show that the method achieves good accuracy.
In [22], Qiu *et al* proposed a deep learning model that can learn from all VM's workload data in the cloud, claiming that this model is more powerful than others that learn workload data from a single machine, because the workload data is stochastic and non-linear. They did an experiment where 8 days of CPU data from 1000 VMs were used for training, in order to predict the next 2 following days. Other 4 prediction models were also used for comparison, including ARIMA, where the deep learning model performed 1.3% better in a short time interval and 2.5% better in several time intervals (5min, 15min, 30 min). The improvement is more significant in longer time intervals because comparative models, such as ARIMA and EWMA, predict better in shorter time intervals. In [13], Khan *et al* , once the temporal correlations are identified, they use a Hidden Markov Model to predict different co-clusters and future workload of VMs. They did an experiment based on 21 days of CPU utilization, where they trained the model with 17 days of data and predicted the level of workload of the next 4 days. In this experiment, they made a comparison with some algorithms, however they are from the last decade, therefore should not be considered. The experimental results on 1212 VMs, showed that its overall prediction accuracy ranges between 60% to 95% on a level 3 workload, 80-85% on a level 4 workload and 75% to 80% level 5 workload.
In [6], Cortez *et al*, they mentioned that ARMA/ARIMA models are not that effective in predicting complex behaviours and that have difficulties in predicting patterns that not have appeared before. In the real world, the probability of unexpected events happen is high so a model that is based on past information, will have difficulties in forecasting future results. Therefore, and since neural network models also rely on past information, they added an online updating module to give agile responses to suddenly changes in workload. They accomplished this by monitoring errors on the prediction performance, and if they happen, the neural network model is retrained. Since the computational cost of the neural network training and prediction is low, the model can be retrained online quickly at low cost. They did an experiment where they used 3 methods for comparison: ARIMA model, BaselineNN and the PRACTISE framework proposed. They used the first 14 days for training the models and the next 46 days for evaluating the prediction accuracy. All 4 metrics were predicted (CPU, memory, disk and bandwidth). The results show us that: PRACTISE model consistently achieves less than 12% of false negatives across all metrics and that is consistently accurate when compared to the other 2 models on the CPU, memory, disk and network bandwidth metrics. They also showed a more challenging case where the trends of periodical pattern change, and again, PRACTISE outperforms the other 2 models, mainly because of the online updating component.

To help cloud users find a workload predictor that is the best one for their cloud activity, in [14], Kim *et al* conducted an experiment with 21 predictors where they evaluated them in terms of accuracy for job arrival time prediction in four realistic workload patterns, that are presented in Figure 1.
The evaluation results were measured with MAPE (Mean Absolute Percentage Error), and they show that all the four workload patterns have different best predictors, meaning that it's important to have strong knowledge about the workload before choosing a prediction algorithm and that there isn't an universally best algorithm. Results are shown in Figure 2 .

From the Related Work, we concluded that the research around predictive auto-scaler is very ambiguous. There isn't

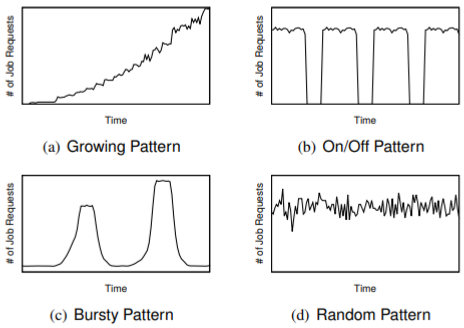

Figure 1. Cloud Workload Patterns. X-axis represents time and Y-axis represents the number of requests [14]

WL	Rank	Predictor	MAPE	WL	Rank	Predictor	MAPE
GR	**1**	**L-SVM**	**0.28**	OO	**1**	**G-SVM**	**0.22**
	2	AR	0.29		2	ARMA	0.30
	3	ARMA	0.30		3	L-SVM	0.44
	Avg.	–	0.51		Avg.	–	0.69
	Worst	Qua.Reg	2.75		Worst	Loc.Cub.Reg	1.25
BR	**1**	**ARIMA**	**0.38**	RN	**1**	**G-SVM**	**0.45**
	2	BRDES	0.41		2	Lin.Reg	0.46
	3	L-SVM	0.43		3	L-SVM	0.46
	Avg.	–	0.75		Avg.	–	0.52
	Worst	mean	3.35		Worst	Dec.Tree	0.62

Figure 2. MAPE Results of Workload Predictors Under Four Different Workload Patterns. (WL: Workload, GR: Growing, OO: On/Off, BR: Bursty, RN: Random) [14]

a clear reason on why each study chose the respective metrics and there isn't much useful information about the real impact of the metrics, besides CPU usage. Also, there isn't a consensus round which machine learning algorithm is the best. A reason could be that the workload may present several patterns, and some machine learning are better for one and worse for other. Other reason could be that some machine learning methods work better with a set of metrics and perform inaccurate with others.

CPU PREDICTION SYSTEM

In this research, we propose a system to predict future CPU usage (%), using ARIMA, LSTM and Random Forest. We chose these algorithms for several reasons. One of the reasons is because the literature not only has shown that ARIMA, LSTM and Random Forest have performed very well in time series forecasting but also they are very different. And since these algorithms are different, they will eventually perform better in a set of conditions and worse on others. For example, ARIMA can not deal with nonlinear relationships and LSTM has a difficult time dealing with linear relationships. Not only time series might have both linear and non-linear relationships, but also it is hard to identify them. Also, due to other influential factors, the final selected model might not be the best one for future use, which gives more power to using this type of approach. By combining different models, we increase the probability of identifying different patterns, which will eventually increase forecasting accuracy. We expect that the combination of these 3 algorithms will complement each other. This section addresses the architecture and implementation of the latter stated solution.

Dataset

During the implementation of this research, three datasets from Materna Data Centers in Dortmund were considered [1]. The datasets contain the performance metrics of three distinct Virtual Machines and each dataset has a timespan of around a 1 month, divided in timesteps of 5 minutes. The workloads in the traced VMs originated from highly critical business applications of known companies.

Pre-Processing

We decided to choose the CPU usage in terms of percentage as the only metric to forecast. We reached this conclusion for three reasons. First of all, the CPU metric is by far, the most used metric in the Related Work. The second reason is that there is no consensus on the best or set of best metrics. And finally, more than one feature would bring an higher level of complexity to the three machine learning algorithms, which could jeopardize the delivery and results of this research.
After the removal of the non-chosen metrics, we are left with the percentage of CPU in each timestep, ordered by time.

Training set, Validation set and Test set

In order to acquire the best possible machine learning model, we separated the dataset in three parts. The three parts are the training set, validation set and testing set. The model is fit on the training set with certain hyperparameters, and the fitted model is used to predict the responses on the data of the validation set [7]. This allowed to make appropriate changes, namely tune in the hyperparameters to then repeat the previous process. This method prevents overfitting and underfitting and opens room for improving the accuracy.
The division of the datasets was made accordingly:

- **Training set:** From 0% until 60% of dataset.
- **Validation set:** From 60% until 80% of dataset
- **Testing set:** From 80% until 100% of dataset

After the hyperparameters were chosen, the model was then trained with the training set plus validation set, in other words, it was trained with the first 80% of the data so that the final model is reached. The final model was then evaluated on the test set, which contains data that the model never seen. This provided an unbiased sense of model effectiveness [16].

Sliding Window Method

Since one of the purposes of this research is to predict the % of CPU, then it becomes clear that we are dealing with a regression problem. Although time series forecasting is not a supervised learning problem, it can be framed as one so that LSTM can forecast. The way to do it, is by applying the sliding window method. All we need to to do is use previous time step as input variable and the next time step as output value. Additionally, this approach also allows to choose between one or multi step forecast, in other words, how far in the future we want to predict.

Performance Measures

It is necessary to evaluate the performance of each of the algorithms in order to open space for improvement. It became clear in the Background section that ARIMA, Random Forest and LSTM are very different algorithms, therefore each one of them needs a different treatment in order to tune in the hyperparameters. However, the performance measures will be the same for all of them, which is the Mean Absolute Error (MAE). This metric is negatively oriented, which means that a lower value represents a more accurate prediction.
An absolute error is the difference between a forecasted value and the respective true value. To verify the overall quality of a prediction model, it was used the MAE which does the average of all absolute errors between the predicted values and the corresponding true values.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_{predicted} - y_{true}| \tag{1}$$

Architecture

The proposed solution consists on a system that has the 3 chosen algorithms running in parallel, where one of them is the primary algorithm. The primary algorithm is the algorithm which the predictions are sent to the auto-scaler. There is also a fixed strategy, which we explain in the next section, that chooses the primary algorithm at each timestep. At each timestep, the 3 algorithms will be retrained with the true value of that timestep, in order to predict the next one.

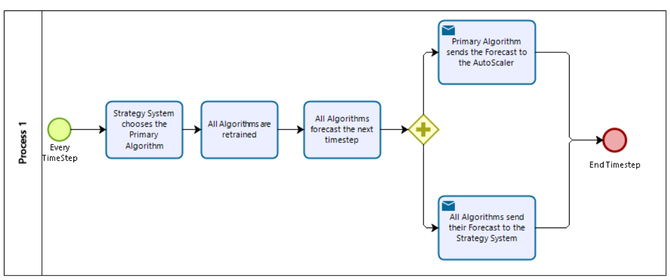

Figure 3. Proposed solution architecture using BPMN

Hardware and Software used

All code and algorithms were developed using Python 3.7.6 because it is one of the most powerful and versatile programming languages. Python was run on the Jupyter Notebook which is a very popular open-source web application amongst data scientists.

The hardware used for running the developed solution has the following specifications:

- **System Model:** ASUS X555LJ x64-based PC
- **Processor:** Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz, 2397 Mhz, 2 Core(s), 4 Logical Processor(s)
- **RAM:** 8 GB

RESULTS

This section is dedicated to present the results obtained from the application of our system in 3 timeseries. We explain how we tuned the hyperparameters in each of the algorithms, as well the proposed strategies of the system in order to choose the primary algorithm at each timestep.

Baseline

Before the start of making predictions, it was important to establish a baseline to check how a common-sense approach would perform. Occasionally, common-sense approaches turn out to be so accurate that they are hard to beat, which would make a machine-learning solution pointless [5]. Since common sense derives from humans, it carries a lot of valuable information that a machine-learning model does not contain. The common-sense approach served as way to verify the performance of the machine learning algorithms proposed. Analogously to the machine learning algorithms, the common sense was evaluated with MAE. The common sense approach utilized consists on finding what is the average value on the training set plus validation set and then use that value as the forecasted value to all the true values in the testing set.
The calculated baselines for the three different timeseries are established below.

- **TimeSeries 1:** 7.1
- **TimeSeries 2:** 5.8
- **TimeSeries 3:** 5.0

Stationary tests

Once we had the data ready to be fitted on the model, it was necessary to verify if the time series were stationary or not, due to the reasons explained in the Background. For this, the KPSS and ADF tests were used. ADF test on all three datasets proved that the time series were stationary, while the KPSS test on all three datasets gave the opposite result. We concluded that the time series are difference stationary, so they need to be differencied in order to be stationary.

Hyperparameters

Tuning in the hyperparameters consisted on repeatedly modify the hyperparameters, train the model, evaluate it on the corresponding validation set and repeat again until we were satisfied with the results.
Before tuning the hyperparameters, we decided that would be better to only forecast what is the percentage of the CPU in five minutes, in other words a one timestep forecast. We reached this conclusion due to several reasons. The first one was because the model was already divided in timesteps of five minutes. The second one it was because the models would perform worse in terms of accuracy the farther in time we forecast. The third one it was due to the fact that forecasting one timestep means that we predict what is going to be the percentage of CPU necessary in five minutes, and five minutes should be more than enough time to apply horizontal or vertical scaling, for instance it takes less than 100 seconds to deploy a new Amazon EC2 [18].

ARIMA

The algorithm was trained on the training set with different combinations of the *p, d* and *q* hyperparameters values where the values ranged from 0 to 5. After the training, the performance of the model was evaluated on the validation set using MAE, and the model that had the lowest MAE became the chosen one. Below, we show the conclusions we reached from the experimentation, namely the best combination of values of *p, d* and *q* for the three timeseries together with the corresponding graphs comparing the forecasted values against the predicted values for each timeseries

- **TimeSeries 1:** The algorithm performed better on the validation set with the hyperpameters (1, 1, 1) as *(p, d, q)* and resulted on a Mean Absolute Error of 3.98 in the test set, which is a roughly 44% increase in performance, compared to the baseline.

- **TimeSeries 2:** The algorithm performed better on the validation set with the hyperpameters (1, 1, 1) as *(p, d, q)* and resulted on a Mean Absolute Error of 2.75 in the test set, which is a roughly 52% increase in performance, compared to the baseline.

- **TimeSeries 3:** The algorithm performed better on the validation set with the hyperpameters (1, 1, 0) as *(p, d, q)* and resulted on a Mean Absolute Error of 1.96 in the test set, which is a roughly 60% increase in performance, compared to the baseline.

LSTM

It is not easy to configure neural networks since there is no good theory on how to do it. But we decided to explore different configurations and recommendations made in François's Book [5]. It consisted on trying different epochs, number of layers, number of neurons per layer and other procedures in order for the model to be prone to generalization and prevent overfitting and underfitting. Underfitting usually happens at the beginning of training, when the model still has a lot of potential for progress because the model hasn't found any relevant patterns on the data [5], and can be solved by increasing the number of training epochs, neurons and hidden layers. Overfitting usually happens when the model is starting to learn patterns that belong to the training data, however they are useless when it comes to new data [5], and can be solved by decreasing the hyperparameters values.

We were also aware of the phenomenon called information leaks [5], which prevented us from overfitting the model on the validation set by not leaking too much information from the validation set, in other words, we chose a model that could perform better on new data and not the best model on the validation set. Therefore, generalization is the goal and it is usually manifested when the training and validation loss decrease and stabilize at the same point.

Additionally to the hyperparameters, several sliding window method were tested and we concluded that the algorithm would performed better, on all models, when the input was two and the output was one, meaning that the algorithm would focus on the two previous timesteps to forecast the next one.

Below, we show the conclusions we reached from the experimentation, namely the combination of the hyperparameters values together with the corresponding graphs comparing the forecasted values against the predicted values for each timeseries.

- **TimeSeries 1:** The chosen hyperpameters of the LSTM for the first Timeseries were : Two LSTM layers with relu as activation function with 75 and 35 neurons correspondingly and 10 epochs. The last layer is a Dense layer which is used for outputting the prediction. In terms of performance, there was a 45% increase compared to the baseline, since the Mean Absolute Error in the test set is 3.95.

- **TimeSeries 2:** The chosen hyperpameters of the LSTM for the second Timeseries were : Two LSTM layers with relu as activation function with 130 and 65 neurons correspondingly and 7 epochs. The last layer is a Dense layer which is used for outputting the prediction. In terms of performance, there was a 55% increase compared to the baseline, since the Mean Absolute Error in the test set is 2.61.

- **TimeSeries 3:** The chosen hyperpameters of the LSTM for the third Timeseries were : Two LSTM layers with relu as activation function with 150 and 100 neurons correspondingly and 8 epochs. The last layer is a Dense layer which is used for outputting the prediction. In terms of performance, there was a 60% increase compared to the baseline, since the Mean Absolute Error in the test set is 1.98.

Random Forest

Similarly to the approach used in ARIMA, different values of *n-estimators*, which represents the number of trees is the forest, were tested. The model that had the lowest MAE on the validation set was the chosen one.

- **TimeSeries 1:** The algorithm performed better on the validation set with an *n-estimators* value of 105. The Mean Absolute Error on the test set equals to 4.14, which is a roughly 42% increase in performance, compared to the baseline

- **TimeSeries 2:** The algorithm performed better on the validation set with an *n-estimators* value of 96. The Mean Absolute Error on the test set equals to 3.23, which is a roughly 44% increase in performance, compared to the baseline

- **TimeSeries 3:** The algorithm performed better on the validation set with an *n-estimators* value of 101. The Mean Absolute Error on the test set equals to 1.97, which is a roughly 60% increase in performance, compared to the baseline

As we can see, the LSTM model had the lowest MAE on 2 timeseries and Random Forest on 1. These are the Mean Average Values that the strategies have to outperform in order to be to worth combining the different algorithms in a system.

Strategy for combining the predictions

We now explain the three strategies that we created in order to take advantage of all the three machine learning algorithms

and the correspondingly results. Regardless of the strategy there are some things that all have in common, namely, at every prediction there is one primary algorithm and two secondary algorithms, but the only prediction that counts is the one that comes from the primary one, which is sent to the auto-scaler. Depending on the strategy, the secondary algorithms will have a chance to trade position with the primary algorithm.
With this is mind, we propose three strategies with the goal of having as the primary algorithm, the one we believe is going to predict with the best accuracy the next timestep.

- **Strategy 1:** The first strategy is a simple one. At every timestep, we calculate the absolute error of each algorithm prediction. The one that has the lowest absolute error is selected to be the primary algorithm for the next timestep. Behind this strategy there is a belief that an algorithm will perform better in the next timestep than the others, if it was the best predictor at the current timestep.

- **Strategy 2:** The second strategy consists on first discovering which algorithm got the lowest MAE on the validation set and call him the privileged one. Then, at every timestep, we check who got the lowest absolute error. In case the privileged one got the lowest absolute error, then he is going to be the primary algorithm for the next timestep. Otherwise, if one of the remaining two got the lowest absolute error, then it is also necessary that in the next step it has the lowest absolute error. In other words, a non-privileged model needs to have the lowest absolute error twice in a row for it to become the primary predictor. Behind this strategy there is a belief that the privileged algorithm will perform better than the others, so it is easier for it to be the primary predictor.

- **Strategy 3:** The third and final strategy consists on using an weighted average to decide what is going to be the primary predictor in the next timestep. For each algorithm , the weighted average gives a 40 percent weight to the absolute error of the previous timestep forecast and a 60 percent to the absolute error of the current timestep forecast. The algorithm that has the lowest value becomes the primary predictor for the next timestep. Behind this strategy there is a belief that the algorithm that predicted with more accuracy in two consecutive timesteps, giving more weight to the current timestep, will perform better than the others at the next timestep.

Obtained Results

In this section we will present the results obtained from the application of the strategies in each timeseries. The results are both promising and positive since 9 of the 9 possible combinations of timeseries and strategies (3x3), performed better than any individual model for each time series. In the Strategy 2, LSTM was defined as the "privileged" model for the first and second timeseries and ARIMA for the third one, since these were the models that performed better on the validation set for the correspondingly timeseries.

- **Timeseries 1:** In the first timeseries, the MAE to beat was 3.95. All strategies beat this value with similar MAEs, namely 3.83 ,3.85 and 3.84 correspondingly, which gives an average of 3% increase in performance. The dominant algorithm at all three strategies was LSTM, which was expected since the MAE to beat was produced by this algorithm. However, ARIMA made a similar MAE (3.98) and got the least amount of share as being the primary algorithm, while Random Forest got the worst MAE (4.14) but made a close call as being the dominant primary algorithm. In the best strategy for this timeseries, which was the strategy 1 with a MAE of 3.83, LSTM got a 38% share as being the primary algorithm, following by a 35% by Random Forest and finally 27% of ARIMA. All in all, it was a very balanced share of the primary place.

- **Timeseries 2:** In the second timeseries, the MAE to beat was 2.61, also provided by LSTM. All the 3 strategies beat the previous value and developed a MAE of 2.52, 2.57 and 2.54, respectively, which is an average of 2.7% increase in performance. The dominant algorithm at all three strategies was LSTM, which was expected since the MAE to beat was produced by this algorithm. If we look at the best strategy in this timeseries, we can see that LSTM got 42% as being the primary algorithm, following by a 29% share of both the other two strategies. Again, it was a fairly balanced share of the primary place.

- **Timeseries 3:** The final timeseries had the lowest and most difficult MAE to beat, which was 1.96, outputted by ARIMA. All strategies beat this value with the exact same MAE of 1.94, which is a 1% increase in performance. Besides the strategy 2 where clearly a model, in this case ARIMA, is going to outperform the other two because it is "the privileged one", in the other two strategies Random Forest was able to be the primary algorithm more time than any previous model in any previous timeseries, with a 42% share in the first strategy and 46% in the third one.

Finally, in Table 1, we provide information that summarizes our investigation and gives more details about the results of the experiment.

CONCLUSIONS

The Information Age is elevating the standards in terms of computation requirements, which translates in a constant look for innovation and an efficient use of resources. This research focused on improving previous solutions related to the Auto-Scaler mechanism, namely the predictive component. Auto-scaler is a mechanism that allows applications to, without human intervention, increase or decrease computing resources depending on the workload. It can be divided in reactive auto-scaler and predictive auto-scaler, where the main difference is that the reactive one simply follows pre-determined rules while the predictive one tries to forecast future demand. The literature has already provided the flaws regarding the reactive auto-scaler and why the predictive auto-scaler might be a better idea. Several studies have been conducted in order to prove the latter statement, which gave promising results, however predictive auto-scaling seems to be in early stages since Microsoft Azure and Google Cloud don't supply this tool. The only major cloud platforms that offers this service is the Amazon AWS, but the service is less than 2 years old

	TimeSeries1	TimeSeries2	TimeSeries3
Baseline MAE	7.1	5.8	5.0
ARIMA MAE	3.98	2.75	1.96
ARIMA vs Baseline	+44%	+52%	+60%
LSTM MAE	3.95	2.61	1.98
LSTM vs Baseline	+45%	+55%	+60%
Random Forest MAE	4.14	3.23	1.97
Random Forest vs Baseline	+42%	+44%	+60%
Strategy 1 MAE	3.83	2.52	1.94
Strategy 2 MAE	3.85	2.57	1.94
Strategy 3 MAE	3.84	2.54	1.94
Best Strategy vs Best Algorithm	+3.13%	+3.57%	+1.03%

Table 1. Table captions should be placed below the table.

and there are no studies on its performance. In addition, the literature regarding predictive auto-scaler is not very clear. There is not a consensus on which metrics and algorithms to use, probably because the workload is different in every case, thus it needs different treatment. It has also been shown that algorithms perform better in a type of workload and worse on other. Since workload often follows different kind of patterns, we propose a system that combines 3 well-known algorithms with the purpose of improving the accuracy of each of them alone.
The proposed system consists on:

- Collecting CPU metric(%) from the VMs which the workload we want to predict.
- Tune in the hyperparameters using the training and validation set on each of the chosen algorithms, namely ARIMA, LSTM and Random Forest.
- Implement one of the strategies proposed. The chosen strategy will determine which algorithm will forecast the next timestep.

Since the chosen algorithms are different and might complement each other, we believe that this combination leads to a better generalization for predicting future workload. The results have shown that the best combination performed 3.13%, 3.57% and 1.03% better in accuracy on the timeseries 1, 2 and 3 respectively, compared to the best algorithm in each of the timeseries. The results have also shown that all proposed combinations of algorithms lead to an increase in accuracy, compared to the best algorithm in each of the timeseries. Although the increases in accuracy seem low, small numbers in big corporations like Amazon, Microsoft and Google make a huge difference. Since the investment in using one algorithm or our proposed system does not seem very disparate, even small companies could reap the benefits of it. From this results we can conclude that perhaps, it is a good idea to combine different kind of algorithms to forecast Virtual Machine Workload. However more research needs to be done.

Future Work

One of the major difficulties of this research was tuning in the hyperparameters of the algorithms. There is a high probability that an experienced data scientist could improve the models of the chosen algorithms, which could lead to an improvement of the proposed system.
In the future, some things can be tested. Different kind of datasets with more data should provided a different perspective and maybe even better results, because algorithms like deep learning tend to perform better with more data. A different metric or a combination of metrics might bring more complexity, however it can also provide good results. Although the combination of these algorithms and the respective strategies provided better results than the individual algorithms, this does not mean that it is the best possible combination or the best possible strategy. Therefore, different algorithms could be grouped as also different strategies could be tested in order to maximize the accuracy of the forecast.

REFERENCES

1. The grid workloads archive (http://gwa.ewi.tudelft.nl/).
2. Breiman, L. Random forests. *Machine learning 45*, 1 (2001), 5–32.
3. Calheiros, R. N., Masoumi, E., Ranjan, R., and Buyya, R. Workload prediction using arima model and its impact on cloud applications' qos. *IEEE Transactions on Cloud Computing 3*, 4 (2014), 449–458.
4. Cheung, Y.-W., and Lai, K. S. Lag order and critical values of the augmented dickey–fuller test. *Journal of Business & Economic Statistics 13*, 3 (1995), 277–280.
5. Chollet, F. *Deep Learning with Python*. Manning, Nov. 2017.
6. Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., and Bianchini, R. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), 153–167.
7. Gareth, J., Daniela, W., Trevor, H., and Robert, T. *An introduction to statistical learning: with applications in R*. Spinger, 2013.
8. Gers, F. A., Eck, D., and Schmidhuber, J. Applying lstm to time series predictable through time-window approaches. In *Neural Nets WIRN Vietri-01*. Springer, 2002, 193–200.

9. Ho, T. K. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1, IEEE (1995), 278–282.

10. Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

11. Iqbal, W., Erradi, A., Abdullah, M., and Mahmood, A. Predictive auto-scaling of multi-tier applications using performance varying cloud resources. *IEEE Transactions on Cloud Computing* (2019).

12. Jheng, J.-J., Tseng, F.-H., Chao, H.-C., and Chou, L.-D. A novel vm workload prediction using grey forecasting model in cloud data center. In *The International Conference on Information Networking 2014 (ICOIN2014)*, IEEE (2014), 40–45.

13. Khan, A., Yan, X., Tao, S., and Anerousis, N. Workload characterization and prediction in the cloud: A multiple time series approach. In *2012 IEEE Network Operations and Management Symposium*, IEEE (2012), 1287–1294.

14. Kim, I. K., Wang, W., Qi, Y., and Humphrey, M. Empirical evaluation of workload forecasting techniques for predictive cloud resource scaling. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, IEEE (2016), 1–10.

15. Kočenda, E., and Černỳ, A. *Elements of time series econometrics: An applied approach*. Charles University in Prague, Karolinum Press, 2015.

16. Kuhn, M., Johnson, K., et al. *Applied predictive modeling*, vol. 26. Springer, 2013.

17. Kwiatkowski, D., Phillips, P. C., Schmidt, P., Shin, Y., et al. Testing the null hypothesis of stationarity against the alternative of a unit root. *Journal of econometrics 54*, 1-3 (1992), 159–178.

18. Mao, M., and Humphrey, M. A performance study on the vm startup time in the cloud. In *2012 IEEE Fifth International Conference on Cloud Computing*, IEEE (2012), 423–430.

19. Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R., and Muharemagic, E. Deep learning applications and challenges in big data analytics. *Journal of Big Data 2*, 1 (2015), 1.

20. Naomi, F. A rnn-lstm based predictive autoscaling approach on private cloud.

21. Pitta, S., Venkata Praveen, T., and Prasad, M. Artificial neural network model for rainfall-runoff -a case study. *International Journal of Hybrid Information Technology 9* (03 2016), 263–272.

22. Qiu, F., Zhang, B., and Guo, J. A deep learning approach for vm workload prediction in the cloud. In *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, IEEE (2016), 319–324.

23. Qu, C., Calheiros, R. N., and Buyya, R. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys (CSUR) 51*, 4 (2018), 1–33.

24. Rokach, L. Ensemble-based classifiers. *Artificial intelligence review 33*, 1-2 (2010), 1–39.

25. Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *nature 323*, 6088 (1986), 533–536.

26. Sagheer, A., and Kotb, M. Time series forecasting of petroleum production using deep lstm recurrent networks. *Neurocomputing 323* (2019), 203–213.

27. Sato, K., Samejima, M., and Komoda, N. Dynamic optimization of virtual machine placement by resource usage prediction. In *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, IEEE (2013), 86–91.

28. Shumway, R. H., and Stoffer, D. S. *Time series analysis and its applications: with R examples*. Springer, 2017.

29. Tseng, F.-H., Wang, X., Chou, L.-D., Chao, H.-C., and Leung, V. C. Dynamic resource prediction and allocation for cloud data center using the multiobjective genetic algorithm. *IEEE Systems Journal 12*, 2 (2017), 1688–1699.

30. Xue, J., Yan, F., Birke, R., Chen, L. Y., Scherer, T., and Smirni, E. Practise: Robust prediction of data center time series. In *2015 11th International Conference on Network and Service Management (CNSM)*, IEEE (2015), 126–134.

31. Zhang, G. P. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing 50* (2003), 159–175.