Journal of Network and Computer Applications 148 (2019) 102443

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Resource needs prediction in virtualized systems: Generic proactive and self-adaptive solution

Souhila Benmakrelouf [a,*], Nadjia Kara [a], Hanine Tout [a], Rafi Rabipour [b], Claes Edstrom [b]

[a] Department of Software Engineering and IT, École de Technologie Supérieure, University of Quebec, Canada
[b] Ericsson, Canada

ARTICLE INFO

Keywords:
Prediction
Resource needs
Machine learning
Kriging
Genetic algorithm
Adjustment

ABSTRACT

Resource management of virtualized systems in cloud data centers is a critical and challenging task due to the fluctuating workloads and complex applications in such environments. Over-provisioning is a common practice to meet service level agreement requirements, but this leads to under-utilization of resources and energy waste. Thus, provisioning virtualized systems with resources according to their workload demands is essential. Existing solutions fail to provide a complete solution in this regard, as some of them lack proactivity and dynamism in estimating resources, while others are environment- or application-specific, which limits their accuracy in the case of bursty workloads. Effective resource management requires dynamic and accurate prediction. This work presents a novel prediction algorithm, which (1) is generic, and can thus be applied to any virtualized system, (2) is able to provide proactive estimation of resource requirements through machine learning techniques, and (3) is capable of real-time adaptation with padding and prediction adjustments based on prediction error probabilities in order to reduce under- and over-provisioning of resources. In several virtualized systems, and under different workload profiles, the experimental results show that our proposition is able to reduce under-estimation by an average of 86% over non-adjusted prediction, and to decrease over-estimation by an average of 67% versus threshold-based provisioning.

1. Introduction

Virtualization is one of the key technologies leveraged to provide scalability, better management flexibility, optimized resource sharing and lower costs in data centers. To capitalize on this technology, it is essential to provision virtualized systems with resources dynamically according to their workload demands. However, the complexity of virtualized systems and applications, their fluctuating resource demands over time (Shyam and Manvi, 2016), and their dynamic and heterogeneous environments all impose a real challenge in resource management, which requires optimizing resource utilization while avoiding Service Level Agreement (SLA) violations. A common practice is to over-provision resources to meet various SLA requirements established with clients. However, this increases the costs incurred in data centers in terms of energy consumption (Goudarzi and Pedram, 2016) and capital expenditure since more resources have to be available. Scalable and elastic allocations of resources is necessary and crucial for the dynamic adjustment of resource capacity to actual demand in real time, while minimizing SLA violations and delays in resource scaling.

Effective and accurate prediction of resource demands is fundamental to real-time resource needs planning and virtualized resource management in data centers. It helps to meet service-level agreement stipulations (by minimizing under-provisioning), anticipate needs in terms of middleboxes (e.g., Load Balancer, Firewall) and lay the groundwork for proactive job scheduling. Thus, consequently this will improve the resource usage, service performance and reduce costs (by minimizing over-provisioning) (Shyam and Manvi, 2016). Several studies have proposed diverse techniques to address these issues (Shyam and Manvi, 2016; K Hoong et al., 2012; Iqbal and K John, 2012; Lloyd et al., 2013; Liang et al., 2011; Hu et al., 2013; Pezzè and Toffetti, 2016), however none of them has provided a complete solution. Some of these approaches do not offer proactive and adaptive management of resources or even consider SLA requirements. Moreover, some of these solutions are environment-specific or application-specific. This limits their accuracy in the case of unexpected and large amounts of data (Shyam and Manvi, 2016), which is a major drawback in the cloud context, where we are dealing with highly dynamic and bursty workloads.

To address these limitations, we propose in this work a novel solution,

* Corresponding author.
E-mail address: souhila.benmakrelouf.1@ens.etsmtl.ca (S. Benmakrelouf).

https://doi.org/10.1016/j.jnca.2019.102443
Received 1 April 2019; Received in revised form 2 August 2019; Accepted 9 September 2019
Available online 13 September 2019

which is generic enough to be applied to any virtualized system or application. It is able to dynamically generate and adjust predictions in real time and offers proactivity in terms of estimating resource demand by anticipating future changes in the system. Our approach provides an algorithm for the dynamic, accurate and effective prediction of resource needs by developing and leveraging different methods and techniques. Black-box prediction methods derive models from the system behavior without requiring any knowledge of the system internals (Gambi, 2013). The adaptability and the efficiency of these methods make them appropriate for virtualized, dynamic and complex environments such as data centers. The proposed algorithm also employs machine learning methods and time series to remain a few steps ahead in the dynamic estimation of resource needs. Furthermore, because prediction is not always sufficiently accurate, adjustments are sometimes needed. Therefore, a dynamic adjustment technique is devised and employed in the prediction algorithm to reduce the under- and over-estimation of resources. A thorough experimentation was also conducted to study the efficiency and performance of the proposed algorithm with different systems and workloads.

European Telecommunications Standards Institute (ETSI) introduced Network Function Virtualization (NFV) concept to reduce cost and accelerate service deployment. The high-level NFV framework proposed by ETSI identified three main working domains (see Fig. B1 in appendix B): (1) Virtualized Network Function, (2) NFV Infrastructure (NFVI) and (3) NFV Management and Orchestration (MANO). The latter covers the orchestration and lifecycle management of physical and/or software resources and the lifecycle management of VNFs. The proposed resource prediction Algorithm can be integrate into MANO and more specifically as a part of Virtualized Infrastructure Manager (VIM).

The main contributions of this work are threefold:

- Novel algorithm for dynamic and multi-step ahead prediction of resource needs in virtualized systems without any prior knowledge or assumptions on their internal behaviors.
- Dynamic and adaptive adjustment of prediction based on the estimated probability of prediction errors, and padding strategy to reduce under-estimation (SLA violations) and over-estimation (resource loss) of resource needs.
- Dynamic determination of the optimal sizes of the sliding window and the predicted data that minimize under- and over-estimations through Genetic Algorithm (GA) intelligence.

The rest of the paper is organized as follows. First, we review the state of the art related to resource demand prediction in the context of virtualized system resource management. Second, we present our proposed approach and we explain the algorithm, the methods and the strategies we used. Third, we evaluate the performance of the prediction algorithm. Finally, we analyze and discuss the main results and conclude the article.

2. Related work

Focus on resource management of virtualized systems as a research area has recently been gaining steam, and several techniques have been proposed in this regard. In this section, we study existing work on the techniques used in this domain. To highlight our contributions, in Table 1, we classify the most recent approaches and compare them based on key features needed for efficient prediction of resources in virtualized systems.

2.1. Time series

Time series (Fu, 2011) is a collection of observations presented chronologically, characterized by a large data size, high dimensionality and continuous update. In his review, Fu (2011) categorized time series data into representation and indexing, similarity measure, segmentation, visualization and mining. He also considered the similarity measure and segmentation as the core tasks for various time series mining tasks. To analyze time series data, various methods and techniques have been used, namely, Support Vector Regression, auto-regression, Expectation Maximization Algorithm, hidden Markov models, and Fourier transforms. When it comes to the segmentation of time series into subsequences for preprocessing or trend analysis, an important observation has proved the effectiveness of a dynamic approach that uses a variable window size, rather than a fixed one, to flexibly identify the time points.

In the same context, the studies conducted in (Islam et al., 2012); (Tran et al., 2016) revealed that the input window size impacted the prediction model accuracy, which has been improved by the use of the sliding window strategy. Contrary to all historical data, the sliding window data allows prediction models to follow the trend of recently observed data and the underlying pattern within the neighborhood of predicted data mainly in the case of highly fluctuant and bursty workloads. Therefore, it allows achieving more accurate predictions.

2.2. Prediction

Prediction approaches can be mainly categorized into two classes. The first category is based on models deduced from the system behavior analysis. Existing approach derived from such analytical models focus mainly on auto-regression and moving averages (K Hoong et al., 2012) (Yu et al., 2011) (Iqbal and K John, 2012), multiple linear regression (Lloyd et al., 2013), Fourier transform and tendency-based methods (Liang et al., 2011) (Gandhi et al., 2011), and cumulative distribution functions (Goudarzi and Pedram, 2016). Researchers in (Lloyd et al., 2013) evaluated the relationships between resource utilization (CPU, disk and network) and performance using multiple linear regression technique to develop a model that predicts application deployment performance. Their model accounted for 84% of the variance (coefficient of determination) in predicting the performance of component deployments. However, all these models are static and non-adaptive to unexpected changes in system behavior or environment. This is due to their use of configuration-specific variables. For its part, the second category of resource prediction approaches is based on machine learning methods. Applications in this class are dynamic and adaptive, but less accurate than the model-based approaches as they may be affected by the non-reliability of measurement tools, which may lead to erroneous values.

To achieve predictions that are both dynamic and more accurate, recent research studies have proposed combining both approaches into hybrid solutions (Gambi, 2013) (Amiri and Mohammad-Khanli, 2017). Numerous studies have proposed machine learning methods for the dynamic resource usage prediction. These methods include the Kalman filter (Zhang-Jian et al., 2013) (Wanget al., 2012), Support Vector Regression (SVR) (Hu et al., 2013) (Huanget al, 2013) (Wei et al., 2013), Artificial Neural Networks (ANN) (Islam et al., 2012) (Tran et al., 2016) (Maet al., 2017), and Bayesian models (Shyam and Manvi, 2016). The authors in (Shyam and Manvi, 2016) proposed a Bayesian model to determine short- and long-term virtual resource requirements for applications based on workload patterns at several data centers, during multiple time intervals. The proposed model (Shyam and Manvi, 2016) was compared with other existing work based on linear regression and support vector regression, and the results showed a better performance for the Bayesian model in terms of mean square error. Nevertheless, as the proposed model was based on workload patterns generated from resource usage information during weekdays and weekends, it may be unable to respond to quick and unexpected changes in resource demands.

Researchers in (Hu et al., 2013) suggested a multi-step-ahead CPU load prediction method based on SVR and an integrated Smooth Kalman Filter to further reduce the prediction error (KSSVR). The results of their experiments showed that KSSVR had the best prediction accuracy, followed successively by standard SVR, Back-Propagation Neural Network (BPNN), and the Autoregressive model (AR). Nevertheless, when dealing with small and fixed size training data from heavily loaded and highly

Table 1
. Taxonomy of prediction approaches.

Study Reference	Criteria							
	Target	Granularity	Model	Adaptability	Prediction based on	Adjustment/Padding	SLA Sensitive	Energy Aware
Shyam and Manvi (2016)	Short- and Long-term Prediction of Resource Demand	Applications in Cloud	Bayesian Model	Non Adaptive	Cyclical and/or Seasonal patterns	Does Not Apply	Does Not Apply	Does Not Apply
K Hoong et al. (2012)	Short-term Prediction	Network Traffic	ARMA time series					
Iqbal and K John (2012)	One-step ahead Prediction		Double Exponential Smoothing, ARMA, Artificial Neural Network and Wavelet		Historical Data, Training Data			Performance and Energy Overhead of Predictors
Lloyd et al. (2013)	One-step ahead Prediction	Multi-tier Applications Deployment Performance in Cloud	Multiple Linear Regression		Historical Data			Impact of Applications Deployment on Resource Usage and Performance
Liang et al. (2011)	Long-term Prediction	CPU Load	Fourier Transform, Tendency-based Method to predict the variation		Seasonal Variation			Does Not Apply
Hu et al. (2013)	Multi-step ahead Prediction		Support Vector Regression (SVR) and Smooth Kalman Filter	Non Adaptive to Unexpected Changes	Training Data (size of training data)			
Pezzè and Toffetti (2016)	Self-adaptive cloud controller	Cloud Controller	Kriging models	Self-adaptive			Minimization of SLA Violation	
Our Approach	Online Multi-step ahead Prediction	Generic	Kriging Method and Time Series	Self-adaptive	Dynamic and Variable Sliding Window	Reactive Error Adjustment and Adaptive Padding	Reduction of under-provisioning	Reduction of over-provisioning

variable interactive machines, KSSVR provides decreased prediction accuracy.

Gambi et al. (Pezzè and Toffetti, 2016) proposed self-adaptive cloud controllers, which are schedulers that allocate resources to applications running in the cloud, based on Kriging models, in order to meet the quality of service requirements while optimizing execution costs. They used Kriging models to approximate the complex and a-priori unknown relationships between: (1) the non-functional system properties collected with runtime monitors (e.g., availability, and throughput), (2) the system configuration (e.g., number of virtual machines), and (3) the service environmental conditions (e.g., workload intensity, interferences). Their test results confirmed that Kriging outperforms multidimensional linear regression, multivariate adaptive regression splines and Queuing models. However, the relatively poor performance of controllers using pure Kriging models revealed that Kriging-based controllers require the availability of a larger set of training values for improved performance.

2.3. Adjustment and padding

To avoid under-estimation of resource needs, a prediction adjustment has been proposed in several studies. The adjustment was introduced as a padding to be added to the predicted data as a prediction cap (Qazi et al., 2014). This padding was prefixed (e.g., 5%) (Qazi et al., 2014) or calculated dynamically using various strategies. The latter include measuring the relationship between the padding value and the confidence interval, which is defined as the probability that real demand is less than the cap (Jiang et al., 2013), or considering the maximum of the recent burstiness of application resource consumption using fast Fourier transform and recent prediction errors through a weighted moving average (Shen et al., 2011), or using the confidence interval based on the estimated standard deviation for prediction errors (Liu et al., 2016).

2.4. Proposed approach

The time and effort needed to build analytical models (off-line modeling) limit their usefulness in dynamic and real-time applications despite their accuracy (Wei et al., 2013). Based on historical observed data, it is known that these models are not able to capture behavioral changes in applications or the systems. Furthermore, techniques based on threshold rules that assume linearity and stability in the system behavior are not realistic solutions, given the complexity and the unpredictable behavior of current systems, as well as their internal and external interactions (Pezzè and Toffetti, 2016). Furthermore, being application-specific, these solutions lack the ability to adapt to cloud dynamics, where their models are generated based on the analysis of a specific application or system for a given environment and behavior. In contrast, data-driven approaches relying on machine learning methods are able to outperform analytical models and adapt to changes by deriving models from the system behavior without requiring any knowledge of the system internals. However, existing resource prediction models in the cloud consider an excessive allocation of resources in order to avoid SLA violations during peak demand periods (Shyam and Manvi, 2016). This leads to a waste of resources and energy, and increases operating costs. Table 1 provides the taxonomy of the most recent relevant approaches in these contexts.

Our approach differs in many aspects from the studies in the literature, and provides a generic, dynamic, and self-adaptive solution for resource prediction. In this proposition, we leverage black-box techniques to provide a generic solution, which can be applied to any system, with no assumptions or knowledge of the systems' internal functionalities being required. We also provide an adaptive solution capable of accommodating changes in the system behavior through real-time data analysis. Moreover, our solution provides multi-step ahead resource demand prediction by leveraging the Kriging machine learning method and time series, and proposing a dynamic sliding window technique.

Further, this work presents a novel dynamic adaptive padding and reactive error adjustment which can mitigate resource under-estimations and over-estimations to reduce SLA violations and reduce resource loss due to typical excessive allocation of resources.

3. Approach overview

In the present work, we propose a generic, dynamic, and self-adaptive prediction of the resource needs in virtualized systems. The proposition aims to minimize under-estimation, which can lead to possible SLA violations, and reduce over-estimation, which that causes a loss of resources, without any prior knowledge of the system or any assumption regarding its behavior or load profile. To that end, we propose a novel prediction algorithm that involves three main techniques. The first mechanism leverages the Kriging method for dynamic machine learning-based prediction. The second technique considers the input of the algorithm, namely, the resource utilization data collected from the system, as a time series with a variable sliding window. This technique benefits from Genetic Algorithm (GA) to dynamically provide the optimal size of the sliding window and the optimal number of predicted data, helping to minimize number prediction errors due to under- and over-estimation. This enables our algorithm to dynamically provide the prediction based on the resource utilization data collected in real time reflecting the current system state. The third and last technique adjusts the prediction based on the estimated probability of prediction errors and a variable padding. Fig. 1 presents the main components of the proposed resource prediction algorithm.

This section gives an overview of the proposed solution, while the details are left for the following sections. Table A.1 Table A1 describes the notations and the symbols used in the rest of the paper. Algorithm 1 presents the proposed approach for resource needs prediction. It starts by initializing the size of the sliding window and the number of predicted data (n_i, m_i) to their maximums (maximums of a given sets of values defined in configuration phase), while the error adjustment coefficient and the padding values are set to zero (Line 1 to Line 4). After resource utilization data is collected, an initialization phase (Line 5) is performed. It consists of consecutive training and prediction steps based on the Kriging method, in which we gather sufficient data (named historical data) to apply our techniques for adjustment and optimization. The prediction and its adjustment are applied based on the historical resource utilization data for each pair (n_i, m_i) of sets of all possible combinations of n_i, m_i values (Line 6 to Line 10). The results obtained, which form the adjusted predicted data and their corresponding combination of (n_i, m_i), are used by Genetic Algorithm (Algorithm 4) to determine the optimal sliding window and prediction sizes (n_s, m_s) that minimize under-estimation and over-estimation (Line 11). With the optimal pair (n_s, m_s) determined, the upcoming resource consumption prediction is performed based on the Kriging method (Line 12) and the adjustment algorithm (Line 13), using the two previous error adjustment values (Algorithm 2). Once the first resource utilization value is collected, it is compared to its corresponding adjusted predicted data. If there is an under-estimation greater than 10%, the padding value is evaluated (Algorithm 3) and the prediction/adjustment processes are resumed, taking padding into account. We defined the threshold based on empirical studies. Otherwise, the resource utilization data are gathered for the next prediction step (Line 18). Our online prediction process continues to estimate the resource utilization repeatedly while the system is monitored and its relevant data are collected.

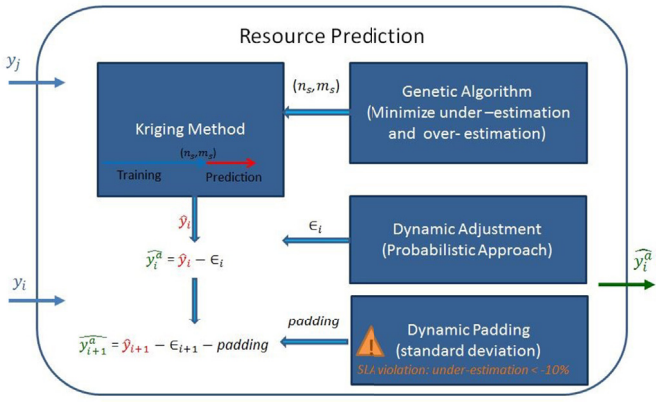

Fig. 1. Resource prediction components.

Algorithm 1 Complexity: The time complexity of the proposed resource consumption prediction algorithm depends essentially on three parts: (1) time taken by the Kriging method to train and predict the next resource demand, (2) the time complexity of adjustment and padding, and (3) the time complexity of GA to provide the optimal size of the sliding window and the number of predicted data (n_s, m_s). The time complexities of each part of our algorithm, namely, the Kriging method, adjustment, padding and GA, are evaluated in Sections 4, 5, 6, and 7, respectively. In Algorithm 1, the initialization of parameters (the sliding window size, the number of predicted data, the error adjustment value, the padding value), as well as the resource utilization data collection (Line 1-Line 4) have a time complexity *O(1)*. During the initialization phase (Line 5), several training and prediction steps using the Kriging method are performed with a time complexity of $O(k\, m_i\, n_i^3)$, where k is the number of repetitions used to collect sufficient data for adjustment and optimization. Then, the assessment of the optimal (n_s, m_s) is performed using GA (Line 11), based on the results of the prediction and adjustment of the historical data (Line 7–10). These two steps have time complexities of $O(IPL)$ and $O(P\, m_i\, n_i^3)+O(PN_I)$, respectively, where P is the size of the population of (n_i, m_i). The prediction (Line 12), adjustment of predicted resource demands (Line 13), and padding (Line 15) have time complexities of $O(m_s n_s^3)$, $O(N_I)$, and $O(n_s)$, respectively. Finally, data collection, the evaluation of padding values and provision of the estimation of resource demands have a time complexity of *O(1)*. Consequently, the time complexity of Algorithm 1 is $O(1) + O(k\, m_i\, n_i^3) + O(P\, m_i\, n_i^3)+O(PN_I)+O(IPL)+O(m_s n_s^3)+O(N_I)+O(1)+O(n_s)+O(1)$, which is equivalent to $O(P\, m_i\, n_i^3)$ due to the highest order of n_i^3.

Algorithm 1
Resource Consumption Prediction (y_i, n, m).

1: $(n_i, m_i) :=$ (max ($\{n_i\}$), max ($\{m_i\}$))
2: Initialize error adjustment coefficient $\epsilon_{i-1} := 0, \epsilon_{i-2} := 0$
3: Initialize padding: $= 0$
4: Collect observed data: y_i , $i \in [1..n_i]$
5: Initialization Phase (n_i, m_i)
6: **for** each collected data window
7: **for** each (n_i, m_i)
8: Predict historical data$(\hat{y}_i, n_i, m_i)$ = Kriging (y_i, n_i, m_i)
9: Adjust historical data $(\hat{Y}_i^a, n_i, m_i, \epsilon_{i-1}, \epsilon_{i-2})$ = **Algorithm 2** $(\hat{y}_i, I_{i-2}, I_{i-1}, \epsilon_{i-1}, \epsilon_{i-2},)$
10: **end for**
11: Genetic Algorithm (n_s, m_s) = **Algorithm 4** $(\hat{y}_i^a, \{(n_i, m_i)\})$
12: Predict next data $(\hat{y}_i, n_s, m_s)$ = **Kriging** (y_i, n_s, m_s)
13: $(\hat{Y}_i^a, n_s, m_s, \epsilon_{i-1}, \epsilon_{i-2})$ = **Algorithm 2** $(\hat{y}_i, I_{i-2}, I_{i-1}, \epsilon_{i-1}, \epsilon_{i-2})$
14: Collect observed data: y_i
15: padding:= **Algorithm 3** $(\hat{y}_i^a, y_i, n_s, m_s, threshold)$
16: return the adjusted prediction of resource utilization $\widehat{y_i^a}$
17: ***if*** (padding $= 0$)
18: Collect observed data: y_i , $i \in [i+1..i-1+m_s]$
19: ***else***
20: Go to step 7
21: ***end***
22: **end for**

4. Prediction

Kriging (Krige, 1951; Matheron, 1963) is a spatial interpolation procedure that uses statistical methods for prediction. It assumes a spatial

correlation between observed data. In other words, observed data close to each other in the input space are assumed to have similar output values (Pezzè and Toffetti, 2016). Kriging is able to model a system based on its external behavior (black-box model) and generic data. It also provides adaptability to linear, non-linear and multi-modal behaviors of the system (i.e., runtime training), with a complexity that varies with the number of samples used for the model fitting. These characteristics are exactly what make the Kriging method suitable for online adaptive and dynamic prediction, specifically in the cloud context (workload variability), which has also been proved in the literature (Pezzè and Toffetti, 2016). In this work, we adapt Kriging in order to provide dynamic and adaptive resource consumption prediction. Next, we explain how the proposed method works.

Through interpolation, the method predicts the unknown value $\hat{y}_p$ by computing weighted linear combinations of the available samples of observed data y_i in the neighborhood given in Equation (1) (van Beers and Kleijnen, 2004):

$$\hat{y_p} = \sum_{i=1}^{n} \omega_i \, y_i \tag{1}$$

where ω_i is the weight associated with the estimation, and $\sum_{i=1}^{n} \omega_i = 1$.

To quantify the weight ω_i for each observed datum (Equation (1)), the method determines the degree of similarity between the observed data, from the covariance value, according to the distance between the sample points (y_i , y_j), using the semivariogram $\gamma(h)$ given by (Gratton, 2002) (Liuet al., 2014):

$$\gamma(h) = \frac{1}{2N(h)} \sum_{i=1}^{N(h)} \left(y_i - y_j\right)^2 \tag{2}$$

where $N(h)$ is the number of all pairs of sample points (y_i , y_j) (i.e., observed data) separated by the distance h.

The empirical semivariogram allows deriving a semivariogram model (e.g., Spherical, Gaussian, Exponential) to represent the semivariance as a function of separation distance. The semivariogram model is used to define the weights ω_i and to evaluate the interpolated points (i.e., predicted data). Hence, the weights ω_i are obtained by resolving the following linear equation system (Gratton, 2002) (Liuet al., 2014):

$$\begin{bmatrix} A & 1 \\ 1^T & 0 \end{bmatrix} \begin{bmatrix} \vec{\omega} \\ \mu \end{bmatrix} = \begin{bmatrix} B \\ 1 \end{bmatrix} \tag{3}$$

where $A_{ij} = \gamma(h_{ij})$ is a value of semivariogram corresponding to distance h_{ij} between y_i and y_j, $B_{ip} = \gamma(h_{ip})$ is the value of semivariogram to be calculated according to the distance h_{ip} between y_i and y_p (point to estimate), $\vec{\omega} = [\omega_1, \ldots, \omega_n]^{\mathrm{T}}$ is the weight vector, and μ is the Lagrange multiplier.

Finally, the calculated weights are used in Equation (1) to estimate y_p.

In our proposition, we use the Kriging method to predict the next m_i values of CPU consumption ($\hat{y_p}$ in Equation (1)) using n_i observed values of CPU consumption (y_i in Equation (1)) as training data. To estimate resource needs for multi-step ahead prediction, the method determines the weights of observed CPU consumption data (training phase) by solving the linear system in Equation (3), which has a time complexity of $O(n_s^3)$, with n_s being the number of training data (V Srinivasan et al., 2008). Hence, at each prediction phase, the Kriging method has a time complexity of $O(m_s\ n_s^3)$, with m_s being the number of predicted values. A complexity of $O(m_s\ n_s^3)$ is acceptable as the size of the training data (sliding window) and the numbers of the predicted data are variable and relatively small values for use in closely tracking the system behavior. Indeed, we conducted divers experiments on the impact of training/prediction window size variation on prediction performance. The results show an increase in prediction errors when the training data size increases specifically for fluctuating workloads. This is due to real-time data-driven algorithm process and its need for continuous adaptation to changes in system resource demands. Therefore, the sizes of training and predicted data are relatively small even with GA optimization (n_i, m_i $\in$ (Gambi, 2013; Pezzè and Toffetti, 2016) in our experiments) and the time complexity of the prediction decrease consequently.

5. Adjustment

To improve the efficiency of the prediction method and reduce the under-estimation caused by significant changes in resource demand, we propose a dynamic prediction adjustment strategy based on the estimated probability of prediction errors (Algorithm 2) and a variable padding technique. Algorithm 2 describes the proposed strategy. We determine the error adjustment coefficient $\in_i$ that reflects the current tendency for under/over-estimation and we add it to the predicted data. In the event of a significant under-estimation, particularly one over the given tolerance threshold (Algorithm 3), padding is added to the adjusted predicted data in order to prevent critical under-estimation and SLA violations (Algorithm 2, line 14). We set tolerance threshold to 10% because further experiments showed that thresholds below 10% induced excessive padding and consequently an over-estimation of resource needs. . Otherwise, the padding value is null.

In our probabilistic approach, we consider the prediction error e_i as a continuous random variable, denoted by X. Its probability density function (PDF), $\rho(x)$, defines a probability distribution for X (Dunn, 2014). The probability that X will be in interval I, with $I = [x_1\ , x_2]$, is given by:

$$\Pr(x \in I) = \int_I \rho(x)dx = \int_{x_1}^{x_2} \rho(x)dx \tag{4}$$

with $\rho(x) \geq 0$ for all x and $\int \rho(x)dx = 1$.

Based on historical data, we set the I as an interval of values between the minimum and the maximum of previously observed errors $I = [e_{\min}, e_{\max}]$. Additionally, we define two probability intervals I_{Proba} and I'_{Proba} ($I_{Proba} = [0, 0.1[$ *and* $I'_{Proba} = [0.1, 1]$) and we assume that: (1) the PDF is Gaussian (most common PDF for continuous process (Dunn, 2014)); and (2) an error e_i is more likely if its probability $\Pr(x \in I)$ belongs to $[0.1, 1]$ and is less likely if its probability $\Pr(x \in I)$ belongs to $[0, 0.1[$.

Algorithm 2
Adjustment of the Prediction ($\hat{y}_i, I_{i-2}, I_{i-1}, \in_{i-1}, \in_{i-2}, $).

1: Compute $\Pr_{i-2}(x \in I_{i-2})$ and $\Pr_{i-1}(x \in I_{i-1})$
2: ***if*** $\{\Pr_{i-1}(x \in I_{i-1}), \Pr_{i-2}(x \in I_{i-2})\} \in I_{Proba}$ and sign $(\in_{i-1}) = \text{sign}(\in_{i-2})$
3: $\in_i :=$ maximum $(|\in_{i-1}|, |\in_{i-2}|) \times \text{sign}(\in_{i-1})$
4: ***else***
5: ***if*** ($\in_{i-1} > 0$ and $\in_{i-2} > 0$)
6: $\in_i :=$ minimum $(\in_{i-1}, \in_{i-2})$
7: **else**
8: $\in_i := \in_{i-1}$
9: **end**
10: ***end***
11: ***if*** $(\in_i > 0)$
12: $\widehat{y_i^a} := \hat{y}_i + \in_i$
13: ***else***
14: $\widehat{y_i^a} := \hat{y}_i$ - $\in_i$- padding
15: ***end***
16: ***return*** $(\hat{y}_i^a, n_s, m_s, \in_{i-1}, \in_{i-2})$

Each time a prediction is made, the probabilities of two previous error intervals $\Pr_{i-2}(x \in I_{i-2})$and $\Pr_{i-1}(x \in I_{i-1})$, are compared, along with the error indicators (i.e., under-estimation if $e_i < 0$;over-estimation otherwise) (Line 2). If the two probabilities belong to the same probability interval (I_{Proba}) and they have the same indicators, we assume that the system may have a stable tendency and the current prediction is adjusted by the maximum of the previous errors (Line 3). Otherwise, we assume

that there is a change in the workload and/or in the system behavior, and hence the current prediction is adjusted either (1) by the minimum of the two previous error adjustment coefficients (if they are positives, which denote two consecutive over-estimations, in order to minimize the over-estimation (Lines 5–6)), or (2) by the most recent error adjustment coefficient in order to track the change (Lines 7–8). It is relevant to mention that the number of error intervals considered in prediction adjustment was dictated by several executions of Algorithm 2 to find the best adjustment.

The time complexity of the prediction adjustment (Algorithm 2) is influenced by the evaluation of the probabilities of the two previous error intervals (Line 1) using a numerical integration of the probability density function (Eq. (4)). The integral is computed via Simpson's rule (Abramowitz and Stegun, 1972) (MathWorks, 2017a) (MathWorks, 2017b) in the interval $[e_{\min}, e_{\max}]$ with N_I equally spaced points, which is performed with a time complexity of $O(N_I)$. The calculation of the error adjustment coefficient $\in_i$ (Line 2–10) has a time complexity of $O(1)$. Also, the calculation of adjusted data (Lines 11–15) and its return (Line 16) both have a time complexity of $O(1)$. Hence, the time complexity of the prediction adjustment algorithm is $O(N_I) + O(N_I) + O(1) + O(1) + O(1)$, which is equivalent to $O(N_I)$.

6. Padding strategies

If the under-estimation exceeds a tolerance threshold (e.g., 10%), an additional adjustment, called padding, is computed (Algorithm 3) and added to the adjusted predicted data in the next prediction step in order to quickly address the gap between the observed data and the predicted data. This ultimately prevents a long-lasting under-estimation and SLA violations. In this context, we tested two padding strategies, namely, ratio-based and standard deviation-based strategies in order to measure the amplitude of variation in resource consumption and adjust the next prediction results accordingly. The first strategy is based on a ratio r between the observed data and the adjusted data, and the error between them (Equation (5)):

$$padding = r\left(\hat{y}_i^a - y_i\right) \tag{5}$$

where $r = \left\lceil y_i / \hat{y}_i^a \right\rceil$

This ratio-based padding shows large over-estimations when significant fluctuations, such as a sharp increase followed by a sharp decrease, occur in the workload. Therefore, we propose another padding strategy that considers workload variability. It is based on the standard deviation of the previous observed data. The mean of previous standard deviations of observed data is considered as a value of the padding (Equation (6)):

$$padding = mean\left(\sigma_j\left(y_{i-n_s}, y_i\right)\right) \tag{6}$$

where $j \in \{1, \ldots, l\}$, l is the number of under-estimations greater than 10% and n_s is the optimal number of training data in the interval I_i.

Algorithm 3
Padding ($\hat{y}_i^a, y_i, n_s, m_s, threshold$).

1: ***if*** $(\hat{y}_i^a < y_i)$ ***and*** $(\frac{\hat{y}_i^a - y_i}{y_i} < threshold)$
2: $\sigma_{current} := \sigma_j\ (y_{i-n_s}, y_i)$
3: ***if*** $\sigma_{current} > 2\sigma_{previous}$
4: $padding := mean\ (\sigma_j, j \in \{1, \ldots, l\text{-}1)$
5: ***else***
6: $padding = mean\ (\sigma_j, j \in \{1, \ldots, l)$
7: **end**
8: ***else***
9: $padding = 0$
10: ***end***
11: ***return*** *padding*

The time complexity of the padding in Algorithm 3 depends on the computation of standard deviation of the previous observed data (Line 2), which is $O(n_s)$, and the mean of previous standard deviations of observed data (Line 4 or Line 6), corresponding to $O(l)$. The rest of the statements in this algorithm each have a time complexity of $O(1)$. Hence, the time complexity of the padding algorithm is $O(1)+O(n_s) + O(1)+ O(l)+O(1)+O(1)$, which is equivalent to $O(n_s)$ having $n_s > l$.

7. Optimization

A time series is an ordered collection of values obtained through repeated measurements, typically over equally-spaced time intervals (Wei, 1990), and its analysis allows the extraction of a model that describes particular patterns in the collected data. Therefore, for dynamic resource consumption prediction, we consider real-time data collected from the system as time series, where at each sliding window, the prediction of the next resource demand is performed. Each set of i observed data is used in the training phase of the prediction model in order to predict the next j values. Afterwards, the sliding window is slid forward by j values at each prediction step in order to continuously keep track of, and predict, the system resource usage.

With the efficiency of the prediction model being mainly affected by the size of the sliding window, as a first step, we studied the performance of the typical fixed sliding window strategy. We tested the prediction model with respect to the mean absolute percentage error (MAPE) metric. Next, experiments were carried out with different sliding windows, by varying both training data and predicted data numbers (e.g., (13–20), (10–15), (7–10), respectively), and the ratio between them. Although the fixed sliding window strategy was able to provide efficient prediction results with MAPE <10%, a prior testing and evaluation phase was required to determine the best pair (n_s, m_s). Furthermore, the results showed critical performance degradation when abnormal system behaviors were observed. Abnormal behavior includes, for instance, a sharp workload increase or decrease. Our observations showed that an efficient and adaptive prediction, which is able to deal with typical and unpredictable system behaviors, is a function of both the size of the sliding window and the number of predicted samples. Both parameters have direct impacts on the accuracy of the prediction, particularly when there are significant fluctuations in the workload. Therefore, the main objective in this section is to find the best sliding window size (n_s) and the best number of predicted data (m_s) that minimize resource demand over-estimation and under-estimation before each prediction process.

7.1. Problem formulation

We define these goals in a multi-objective optimization problem which we formulate as follows:

- Minimize over-estimation: The mean of over-estimations is equal to the total of over-estimations divided by the number of over-estimation occurrences in historical data. An adjusted predicted datum is considered as an over-estimation if it is greater than its corresponding observed data.

$$F1 = \min\left(\frac{1}{n_{oe}} \sum_{k=1}^{l} \sum_{j=1}^{n} \gamma_i \left(\sum_{i=1}^{m} \left[\left(\hat{y}_{ijk} + \varepsilon_{ijk} \right) - y_{ijk} \right] \beta_i \right) \right) \tag{7}$$

- Minimize under-estimation: The mean of under-estimations is equal to the total of under-estimations divided by the number of under-estimation occurrences in historical data. An adjusted predicted datum is considered as an under-estimation if it is less than its corresponding observed data.

$$F_2 = \min\left(\frac{1}{n_{ue}}\sum_{k=1}^{l}\sum_{j=1}^{n}\gamma_i\left(\sum_{i=1}^{m}\left[\left(\hat{y}_{ijk} - \varepsilon_{ijk} - \alpha_i padding\right) - y_{ijk}\right]\left(1-\beta_i\right)\right)\right) \tag{8}$$

Thus, our multi-objective optimization problem is:

$$F = \min\{F_1, F_2\} \tag{9}$$

subject to.

$$n \in \mathbb{N} \tag{c1}$$

$$m \in \mathbb{N} \tag{c2}$$

$$m \leq n \tag{c3}$$

$$\alpha_i = \{0, 1\} \tag{c4}$$

$$\beta_i = \{0, 1\} \tag{c5}$$

$$\gamma_i = \{0, 1\} \tag{c6}$$

These objective functions aim to minimize resource wastage and SLA violations, respectively. The constraints c1, c2 ensure that the sizes of data and the sliding window belong to the natural numbers, while c3 confirms that the number of predicted data is less than or equal to the sliding window size. Finally, constraints c4, c5 and c6 define the decision variables for padding, over-estimation and the sliding window size, respectively, as binary variables. The solution of this problem is the best combination of sliding window size (n_s) and the predicted data number (m_s) which minimizes over-estimation and under-estimation of the resource requirements, allows dynamic prediction, and improves the performance of the latter.

7.2. Heuristic algorithm-Genetic Algorithm

Genetic Algorithms (GAs) (Tout et al., 2016) (Mitsuo and Cheng, 2000) (Yang, 2010) form an abstraction of biological evolution, which imitate the natural evolution process to solve an optimization problem. GAs are heuristic methods that aim to determine the best solution by simulating the propagation of the fittest individuals over consecutive generations. The fitness of a solution is based on a function that aims to minimize or maximize (a) particular objective(s). Through crossover, mutation and selection operators, Genetic Algorithm is able to generate diversified individuals and find the best among them. This approach might be a good alternative to an exhaustive search, as shown in Section 8.

In this paper, we use GA to determine the optimal size of the sliding window and the optimal number of predicted data (n_s, m_s). Hereafter, we explain how we adapted GA to solve our optimization problem defined in the previous section.

7.2.1. Individuals of populations

Each individual in the populations is represented by two elements. The first element is a size of the sliding windown_i, and the second one is a size of the predicted datam_i. The set of (n_i, m_i) combinations constitutes a population. For instance, the set {(5, 3), (10, 3), (10, 7), (15, 6), (15, 10), (20, 4)} represents a population of six different individuals.

7.2.2. Fitness function and selection

The fitness function in GA aims to assign a score to each individual according to its efficiency in resolving the problem. In this work, the score of a solution is computed by evaluating the objective functions F_1 and F_2. Next, the selection of individuals is carried out by evaluating their fitness. Individuals with a higher fitness are more likely to reproduce in the next generations.

7.2.3. Crossover and mutation

Next, crossover and mutation operators are applied with r_c. and r_m. rates, respectively. Typically, the crossover of two individuals can be conducted by swapping elements from both individuals, resulting in two offspring. For instance, individuals (10, 3) and (20, 7) can be crossed over each other's first element to generate the two offspring (20, 3) and (10, 7). Alternatively, mutation can be realized by randomly flipping an element in different individuals, selected based on r_m. For example, an individual (10, 7) could be mutated in its second element to yield (10, 2).

7.2.4. Algorithm 4- Genetic AlgorithmAlgorithm 4

Genetic Algorithm $(\hat{y}_i^a, \{(n_i, m_i)\})$ (Tout et al., 2016).

1: Initialize N = population size; r_c = crossover rate; r_m = mutation rate
2: Initialize population index $j = 0$
3: Generate the initial population P_j
4: **for** each individual (n_i, m_i) in P_j
5: Evaluate objective functions $F_1(\hat{y}_i^a, (n_i, m_i))$, $F_2(\hat{y}_i^a, (n_i, m_i))$
6: **end for**
7: **do**
8: Select x best (n_i, m_i) and insert them into P_{j+1}
9: Crossover $r_c \times n$ individuals to produce new offspring(n_i, m_i) and insert them into P_{j+1}
10: Mutate $r_m \times n$ individuals to produce new offspring(n_i, m_i) and insert them into P_{j+1}
11: **for** each individual (n_i, m_i) in P_{j+1}
12: Evaluate objective functions $F_1(\hat{y}_i^a, (n_i, m_i))$, $F_2(\hat{y}_i^a, (n_i, m_i))$
13: **end for**
14: j = j+1
15: **while** stopping criteria is not met
16: **return the fittest** (n_s, m_s) from P_j

All the steps of the adapted GA are described in Algorithm 4. It starts by generating the initial population. The fitness of each individual is computed by evaluating both predefined objective functions that aim to minimize under- and over-estimations. The fittest individuals in the population are selected and inserted into the next population. Crossover and mutation operators are applied to produce a new generation of solutions (n_s, m_s), and the fitness of these individuals is calculated. This process is repeated until the stopping condition is met. The latter can be defined as the time constraint, number of generations, or any other adequate criterion. Finally, the fittest pairs (n_s, m_s) from the last generation are reported. If several solutions are provided by GA, the solution that most minimizes the under-estimation is selected for the next prediction because we consider that the under-estimation is more critical than the over-estimation in terms of cost of SLA violations.

The complexity of Algorithm 4 depends on many factors, namely, the fitness function evaluation, the population size, the individuals' length (number of elements in each individual) and the number of generations (i.e., iterations). The initialization of parameters and the generation of the initial population each has a time complexity *O(1)* (Lines 1–3). The evaluation of the fitness function has a time complexity *O(N)*, where *N* is the size of the population (Lines 4–6). The tournament selection, the crossover and the mutation (Lines 7–15) have a time complexity *O(INL)*, where *I* is the number of generations and *L* is the length of an individual (length of $(n_s, m_s) = 2$). Finally, the return statement (Line 16) has a time complexity *O(1)*. Thus, the time complexity of Algorithm 4 is *O(1)*+*O(N)*+*O(INL)*+*O(1)*, which is equivalent to *O(INL)*.

8. Experimental evaluation

We evaluated the cost of the resource demand prediction in terms of SLA violation and resource wastage by computing the probability of under-estimations (e.g. $Pr^{PredictData}_{UnderEstim}$), the mean of over-estimations

Table 2
Parameters of Genetic algorithm.

Parameter	Value
Population size	6 pairs(n_i, m_i)
Initial Population	{(10,3), (10,5), (10,7), (20,5), (20,10), (20,14)}
Selection	Tournament selection function
Crossover probability	0.8
Mutation probability	0.2
Crossover operator	Single point
Mutation operator	Random modification of n_i *or* m_i values
Number of generations	201
Stopping criteria	Number of generations

(e.g. $E^{PredictData}_{OverEstim}$) and the mean time between under-estimations (*MTBUE*) for both predicted and adjusted data. Also, we considered the mean of over-estimations in the case of static provisioning of resources (threshold-based provisioning), that is, an over-provisioning of resources applied in legacy systems. It represents the maximum of allocated resources for a specific system and load profile. We use this metric to compare the resource gains between our approach (prediction and adjustment) and the static provisioning approach. Details of the calculation of suggested metrics are presented in Table A.1Table A1.

To evaluate our algorithm with different types of virtualized systems/applications, we used data from the OpenIMS telecommunication service platform (Configurations 1 and 2 presented in Table 3), data furnished by an IMS service provider (Datasets 1 and 2), and data extracted from Google cluster data available online (Reisset al., 2014) (Datasets 3 and 4). By processing data from various systems and load profiles, we aim to evaluate the ability of our algorithm to accurately predict the resource utilization within various types of systems/applications, workloads and situations (predictable vs. unpredictable workloads).

8.1. Experimental setting

8.1.1. Testbed setup

The testbed was built with 4 servers, all connected to the same local area network. For IMS Core components, we used OpenIMS Core (Fraunhofer, 2017), which is an open source implementation of IMS Core developed by Fraunhofer FOKUS IMS Bench. The virtualization of IMS Core (CSCFs and HSS) is based on Linux container technology (Containers, 2017). Further, a SIP client was created using the instantiation of SIPp version 591 (October 2010), which is a traffic generator (GAYRAUDet al., 2017).

The Virtualized P-CSCF and HSS were hosted on Server 1, whereas, the Virtualized S-CSCF and I-CSCF were hosted on Server 2. The high-level monitoring of virtualized CSCF entities and the management of resources was done by the Host Monitoring Agent (HMA) and Resource Management Agent (RMA), respectively. HMA and RMA were deployed on both Server 1 and Server 2. Server 3 hosted SIPp and the Session Monitoring Engine (SME). DNS (Domain Name System) was hosted on Server 4. Each physical machine had an Intel Core i7 3.6 GHz CPU, 24.576 GB of RAM, and a Linux Ubuntu operating system version 14.04. Fig. 2 presents the testbed.

For the parameters of the Kriging method, we essentially set its type to universal Kriging, and the variogram model to the spherical one, with a range of 0.2 and a sill of 1. These parameters were set through tests trying to find the configuration minimizing prediction errors.

Table 3
Description of load profiles (OpenIMS platform).

Load Profile	Description
Configuration1 (Conf1)	Start at 150 CPS, increment: 50 CPS/10 s until 400 CPS, 400 CPS constant during 100 s, 600 CPS constant for 300 s, 200 CPS decrement: 50 CPS/50 s until 50 CPS
Configuration2 (Conf2)	Start at 150 CPS, increment: 50 CPS/10 s until 400 CPS, 400 CPS constant during 10 s, 600 CPS constant for 300 s, 5 CPS during 60 s, 400 CPS constant for 300 s

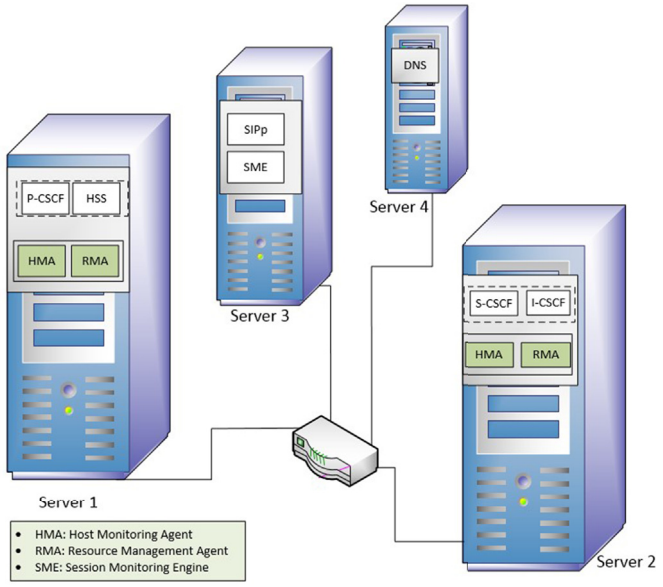

Fig. 2. Testbed.

Regarding the GA configuration, we used the values presented in Table 2. The population was initialized as {(10,3), (10,5), (10,7), (20,5), (20,10), (20,14)}, the selection was based on the tournament selection function (Mitsuo and Cheng, 2000), the single-point crossover was used, while a random modification of n_i or m_i values was performed for the mutation. The crossover and the mutation probabilities were set to 0.8 and 0.2, respectively. Finally, the stopping criterion was set to the number of iterations. We set the value of this criterion based on multiple executions of this algorithm while trying to find the optimal pair (n_s, m_s).

8.1.2. Data/load profile

We focused our tests on the CPU consumption of the virtualized S-CSCF node because the CPU load has a significant effect on the performance (Liang et al., 2011), and a performance analysis of OpenIMS showed that S-CSCF is a bottleneck in IMS (Mkwawa and Kouvatsos, 2008). Then, we performed several tests on the OpenIMS platform using different load profiles, representing for instance, a sharp decrease (Configuration1 (Conf1)), or a sharp decrease followed by a sharp increase (Configuration2 (Conf2)) of the workload. The data from the OpenIMS platform was collected every 5 s.

The data from the IMS service provider represents CPU consumption collected from multiple machines every 5 min for 16 h. In the present article, we present two examples of datasets (different numbers and amplitudes of workload fluctuations), namely, Dataset1 (Dset1) and Dataset2 (Dset2).

The Google cluster data trace (clusterdata-2011-2) represents 29 days' worth of cell information from May 2011, on a cluster of about 12,500 machines (packed into racks, and connected by a high-bandwidth cluster network). A cell is a set of machines, typically all in a single cluster, that share a common cluster-management system which allocates jobs to the machines. A job is comprised of one or more tasks (accompanied by a set of resource requirements) (Reisset al., 2014). Because of the large size of the Google cluster data, we propose to extract the CPU consumption of tasks from multiple data files for a given machine and a given job. For instance, we present Dataset3 (Dset3) and Dataset4 (Dset4) that denote the CPU consumption of tasks identified as 85 and 42, respectively, collected every 5 min.

The descriptions of the load profiles, Configuration1 and Configuration2, are presented in Table 3.

8.1.3. Assumptions

- The monitoring tools are sufficiently accurate.
- The monitoring and collection of data from the system are continuous and in real time.
- The Linux container-based virtualization and the CPU core isolation technique guarantee that each service/application cannot access others' hardware and software resources. They also ensure that collected data reflect the effective service/application consumption.

8.2. Results and analysis

In this section, we present the evaluation of the ability of our algorithm to accurately estimate resource consumption, as well as the efficiency of the proposed techniques. First, we used two machine learning methods, namely, Kriging and SVR, to predict CPU consumption, and we compared their performance. Next, we evaluated the proposed prediction with adjustment and padding by processing data from various systems and assessed the cost in terms of SLA violation and resource wastage.

8.2.1. Prediction using Kriging vs. SVR methods

Beside the Kriging method, we also tested support vector regression (SVR) to predict resource consumption. We then compared their performance in terms of prediction errors and execution time. The Kriging is a spatial interpolation-based method, while SVR is a kernel-based learning method. SVR identifies the optimal hyperplane that maximizes the margin between the vectors (support vectors) of the considered classes. This optimal hyperplane is defined as a linear decision function (finds optimal parameters w and b) (Cortes and Vapnik, 1995) (Smola and Schölkopf, 2004):

$$f(x) = wK(x) + b \tag{10}$$

where.

x is input data, w is the weight vector, and b is the bias parameter.

$K(x)$ is a kernel function (e.g., linear, radial basis function (RBF))

When the kernel function is set to Radial Basis Function (RBF), two parameters must be considered, namely, C and gamma. The parameter C, common to all SVR kernels, trades off misclassification of training samples against simplicity of the decision surface. The parameter *gamma* defines how much influence a single training sample has (cikit-learn. "Support Vec, 2019).

In our experiments, we used the CPU consumption data from OpenIMS platform tests (using different load profiles), and data from an IMS service provider and Google cluster data trace, which present several types of variations/fluctuations. Then, we performed prediction (without adjustment) with a fixed-size sliding window and a fixed number of predicted data: $(n_i, m_i) = (10, 5)$ in the case of Kriging and $(n_i, m_i) = (10, 10)$ in the case of SVR. The SVR kernel function was set to RBF since it is more appropriate for nonlinear datasets, while C and Gamma values were set to 0.1. These parameters were set following extensive testing performed to find the configuration minimizing the prediction error.

In the first part of the experiments, we used the Kriging and SVR methods to predict CPU consumption for several configurations and datasets, and then compared their performance in terms of MSE (Mean Squared Error) and execution time. In the second part of the experiments, we tested the SVR method by varying the training and prediction data size and using Configuration 1 data to evaluate their impact on prediction performance. Fig. 3 presents the results for each dataset, while Table 4 illustrates the MSE and the execution time of the considered datasets.

The results indicate that the Kriging method outperforms the SVR method in terms of prediction error (MSE), except for Dataset3 and Dataset4 (data from Google cluster). Indeed, Fig. 3 shows that the predicted CPU consumption using the Kriging method fits the observed workload better than the SVR method, with the exception of some areas where there are sharp CPU usage increases/decreases. For instance, the CPU usage prediction error was 645.51 in the case of Kriging, while it

Table 4
MSE and execution time of Kriging vs. SVR prediction.

Configuration/ Dataset	Kriging method		SVR method	
	MSE[1]	Execution time (sec)	MSE	Execution time (sec)
Configuration1	**123.40**	0.53	253.33	0.004
Configuration2	**645.51**	0.47	819.04	0.005
Dataset1	**5.78** 10^{-4}	0.55	7.7 10^{-4}	0.004
Dataset2	**0.58**	0.57	1.01	0.004
Dataset3	30.15	0.43	**15.95**	0.003
Dataset4	30.96	0.44	**15.72**	0.003

1:MSE: Mean Squared Error

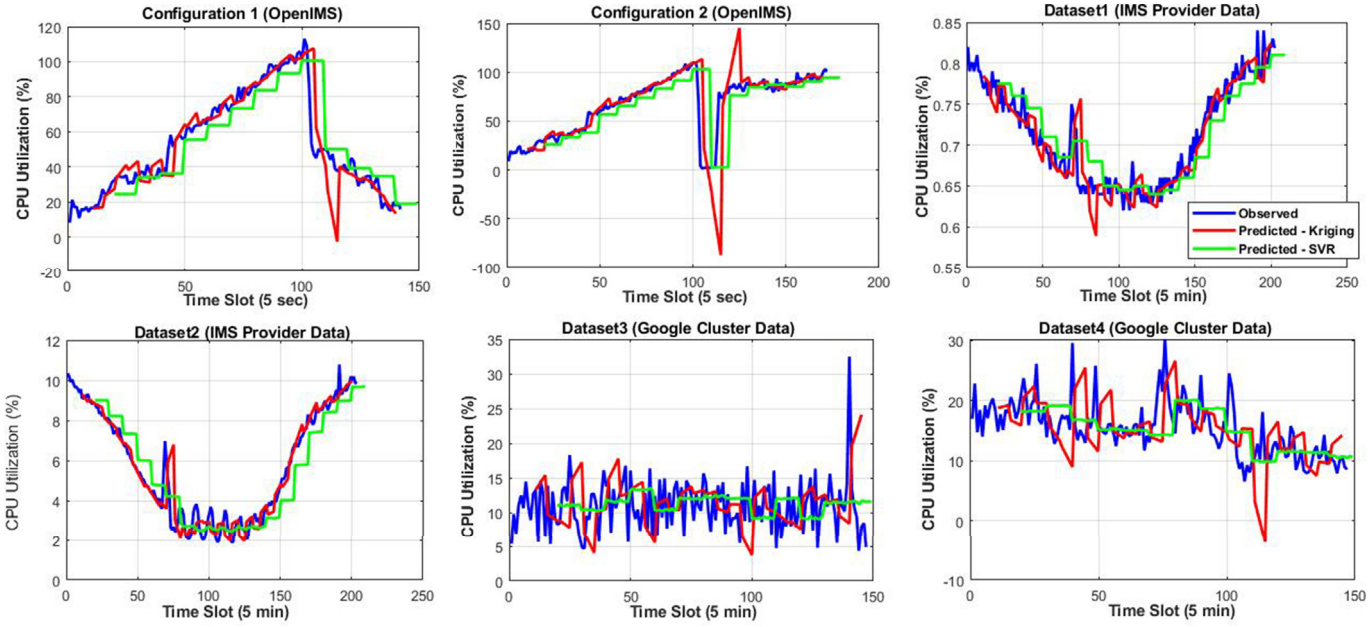

Fig. 3. Prediction of CPU consumption for OpenIMS, IMS and Google cluster data: Kriging vs. SVR.

was 819.04 in the case of SVR for Configuration2 (see Table 4).

On the other hand, we notice that the Kriging prediction error is greater than the SVR prediction error in the presence of sharper variations in short bursts, as observed in Dataset3 and Dataset4. The results also show that the SVR method is faster than the Kriging method (see Table 4, execution time). Furthermore, we observe (see Fig. 4) that the prediction error (MSE) increases when the training/prediction data size increases in the case of the SVR method (see Table 5). The latter is then sensitive to training/prediction data size, which is a major drawback, especially in the context of the dynamic sliding window.

To summarize, these experiments show that the Kriging method is able to accurately predict CPU consumption, but at higher cost in terms of execution time compared to SVR. Furthermore, the latter loses efficiency proportionally to the window size, and is more sensitive to large variations and peaks in the observed data. These results validate the choice of the Kriging method to predict CPU consumption in a dynamic context. Furthermore, we propose historical data-based adjustment, which improves the efficiency of the spatial locality-based Kriging prediction and reduces the under-estimation caused by sharp variations in resource usage.

8.2.2. Prediction and adjustment

First, we defined a set of alternative scenarios: (1) prediction with fixed-size sliding window and fixed number of predicted data: $(n_i, m_i) = (10, 5)$, and without adjustment; (2) prediction with variable number of predicted data: $(n_i, m_i) = (10, [7, 5])$, adjustment and standard deviation-based padding (strategy 3 defined in Section 8.2.3 padding strategies), and (3) prediction with the sliding window size and the number of predicted data selected dynamically by GA (Algorithm 4), adjustment and standard deviation-based padding (strategy 3). The sliding window and predicted data size values (n_i, m_i) for scenarios 1 and 2 were selected among initial GA population pairs that had provided better prediction performance in prior Kriging prediction tests. Next, we experimented with several types of variations/fluctuations of CPU usage with a small period (5 s) or a large period (5 min) of data collection (e.g., Conf1 and Dset1, respectively) to evaluate our approach to dealing with various workload situations.

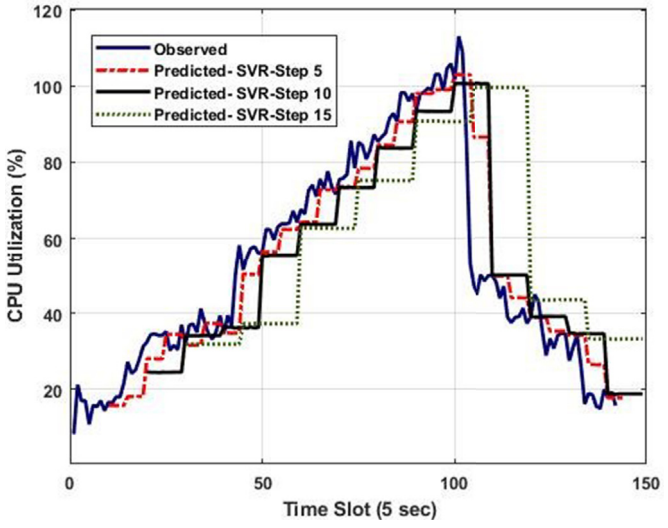

Fig. 4. SVR prediction of CPU consumption in case of Configuration 1: variation of predicted data size.

Table 5
MSE and execution time of SVR prediction-variation of training and predicted data size.

Training/prediction data size	SVR Method	
	MSE	Execution time (sec)
3	82.80	0.01
5	144.54	0.007
10	253.33	0.004
12	493.53	0.003
15	664.68	0.002

Table 6
Characteristics of prediction scenarios.

Scenario	Strategy	*Adaptive*	*Dynamic*
Scenario 1 Without Adjustment, Padding and GA	Sliding window size, Number of Predicted Data	×	×
	Prediction - Kriging	✓	✓
	Adjustment and Padding	Does Not Apply	
Scenario 2 Adjustment and Padding without GA	Sliding window size, Number of Predicted Data	✓	×
	Prediction - Kriging	✓	✓
	Adjustment and Padding	✓	✓
Scenario 3 Adjustment, Padding and GA	Sliding window size, Number of Predicted Data	✓	✓
	Prediction - Kriging	✓	✓
	Adjustment and Padding	✓	✓

Table 6 details the characteristics of our scenarios. In all the scenarios, the prediction is dynamic and adaptive to the workload profile. However, the sliding window size and the number of predicted data are adaptive and dynamic only in the third scenario, which represents our approach. Furthermore, the adjustment and the padding are dynamic and adaptive in the second and the third scenarios. By defining and testing these scenarios, we aim to compare the impact of each proposed technique on the CPU consumption prediction accuracy and the cost in terms of under-estimation and over-estimation.

Fig. 5, Fig. 6 and Fig. 7 present the CPU consumption prediction results for the defined scenarios and various systems and workload profiles. During the initialization phase, several training and prediction steps using the Kriging method are performed (without adjustment) until sufficient data for applying adjustment and optimization are collected. Therefore, we observe under-estimation cases before the 60 time slot (Figs. 5–7). Mainly, the results show that the prediction using the Kriging method can follow the CPU usage trend, and can adapt to changes for all considered configurations and datasets. However, it is less effective for large and unexpected variations (e.g., Configuration2, see Fig. 5) or many fluctuations (e.g., Dataset4, see Fig. 7) occur, which cause long and significant under-estimations and SLA violations. Therefore, we propose the adjustment of the prediction and the standard deviation-based padding (strategy 3 outperforms in almost all scenarios and datasets: see Section 8.2.3 Padding strategies) to reduce long and/or frequent under-estimations. The adjusted prediction and variable padding results show a clear improvement in terms of significant under-estimation reduction.

However, we observe some cases of large over-estimation, for instance: Dataset3 and Dataset4 (Fig. 7, plot: adjusted without GA). These over-estimations arise mainly when the variation and the magnitude of the CPU consumption in the current prediction phase are different from the previous one. It is mainly due to the adjustment and the padding values computed using the results of previous prediction steps. Further, we notice from several experiments that the prediction performance is influenced by the size of the sliding window and the number of predicted data. Therefore, we use GA to dynamically provide the size of the sliding window and the number of predicted data minimizing under-estimation and over-estimation at each prediction step. The results of the adjusted prediction with the use of GA (see Figs. 5–7 plots adjusted with GA) mostly show a decrease in over-estimation, while under-estimation decreases as well (e.g., see Fig. 7: Dataset3 and Dataset4) or is close to the results of the adjusted prediction without GA (e.g., see Fig. 5: Configuration1, Configuration2). Therefore, we conclude that the accuracy and the adaptability of our algorithm are improved significantly thanks to the prediction and the adjustment techniques combined with GA.

To evaluate and quantify the efficiency of our algorithm, we compute

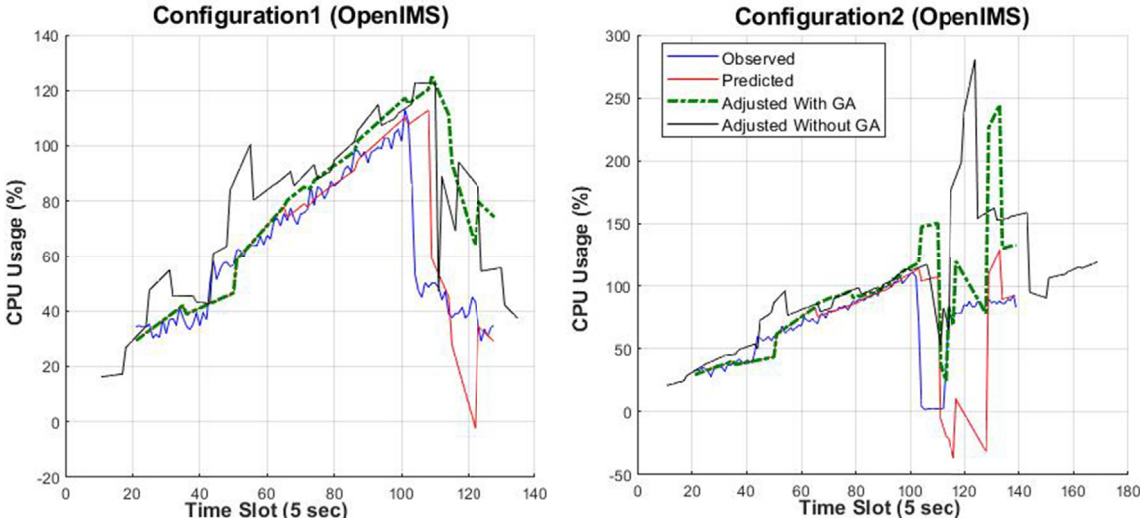

Fig. 5. Prediction and adjustment of CPU consumption in cases of Configuration 1 and 2 (OpenIMS).

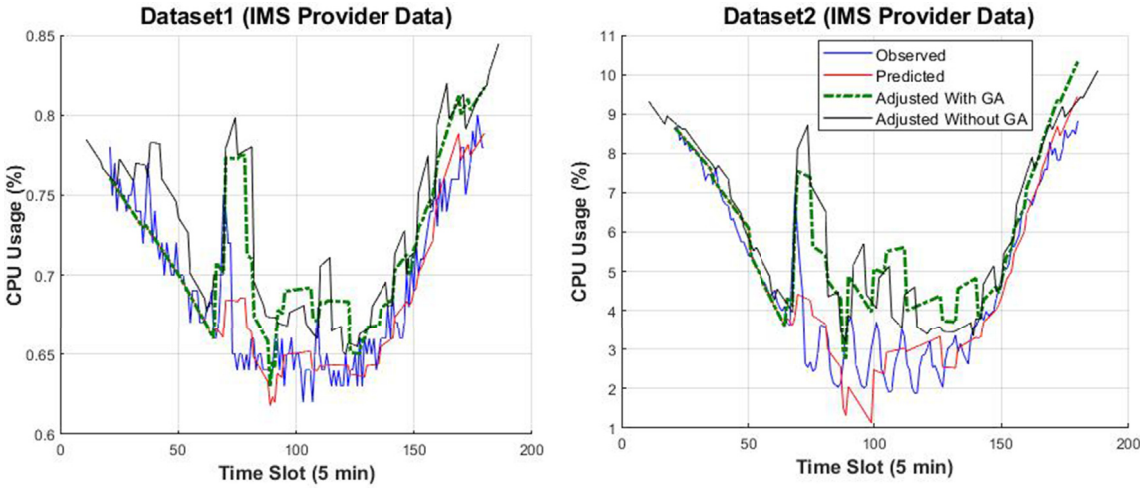

Fig. 6. Prediction and adjustment of CPU consumption in cases of Datasets 1 and 2 (IMS).

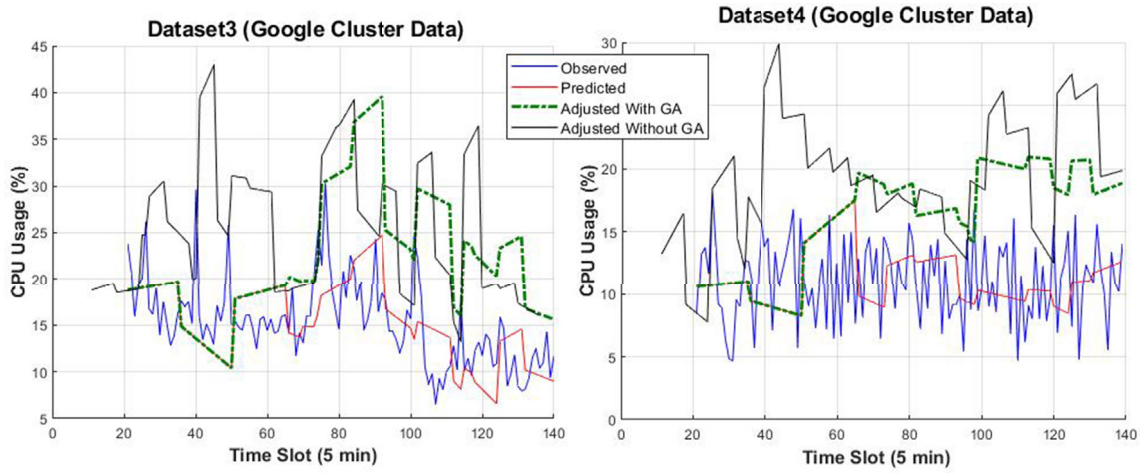

Fig. 7. Prediction and adjustment of CPU consumption in cases of Datasets 3 and 4 (Google cluster data).

the probability of under-estimation, the mean of over-estimations, the mean of over-estimations for threshold-based provisioning, and the mean time between under-estimations (*MTBUE*) for both predicted and adjusted data. Fig. 8 summarizes the evaluation metrics for the defined scenarios, configurations and datasets.

As shown in Fig. 8 c, the prediction without any adjustment is characterized by a large probability of under-estimation (between 0.44 and 0.54) (Fig. 8.c1), a mean of over-estimation under 10% for all configurations and datasets (Fig. 8.c2), and a short MTBUE: less than 17 s for all configurations and less than 21 min for all datasets (Fig. 8.c3). These

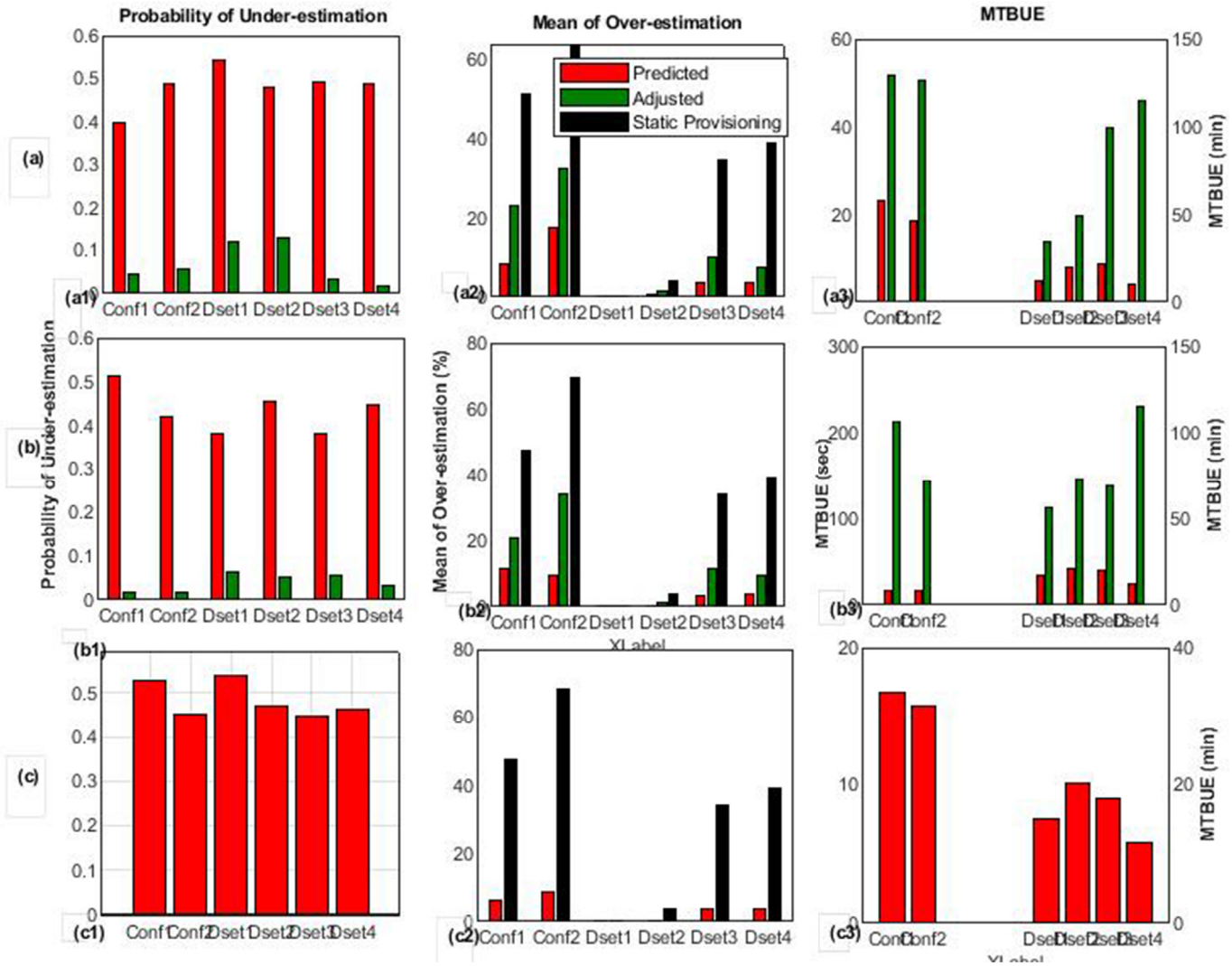

Fig. 8. Results of Evaluation Metrics in case of: (a) Prediction (dynamic sliding window), Adjustment and GA, (b) Prediction with fixed-size sliding window, Adjustment without GA, (c) Prediction with fixed-size sliding window and fixed number of predicted data.

results reveal the limitations of using the prediction without adjustment specifically in the presence of fluctuating workloads.

A comparison of the results presented in Fig. 8.a1, Fig. 8.b1 and Fig. 8.c1 shows that the proposed prediction with adjustment and padding (strategy 3) minimizes the under-estimation for all configurations and datasets remarkably. Indeed, the prediction adjustment allows reducing the under-estimation by 86% on average as compared to prediction without adjustment. For instance, the probability of under-estimation decreases from 0.39 to 0.04 for Configuration1, and from 0.48 to 0.01 for Dataset4 (see Fig. 8.a1). Moreover, our approach results in a significant increase in the mean time between under-estimations (MTBUE). For example, the MTBUE increases from 20 min to 50 min and from 10 min to 115 min in the case of Dataset2 and Dataset4, respectively (see Fig. 8.a3).

Regarding resource wastage due to over-estimation, our algorithm significantly improves the efficient consumption of CPU resources in comparison with threshold-based provisioning (static provisioning) for both prediction and adjustment, for all configurations and datasets. Actually, our approach is able to reduce over-estimation by 67% on average, as compared to the threshold-based provisioning (see Fig. 8.a2 and Fig. 8.b2). For instance, the mean of over-estimation of CPU consumption decreases from 64% to 32% and from 0.31% to 0.03% for Configuration2 and Dataset1, respectively (see Fig. 8.a2).

The main improvements provided by GA to the proposed algorithm are the dynamic selection of the sliding window size and the number of predicted data, as well as the flexibility and the adaptability to changes in the workload while minimizing SLA violations and resource wastage. As shown in Fig. 8 a and Fig. 8 b, the probability of under-estimation decreases (see Fig. 8.a1 and Fig. 8.b1), and the MTBUE increases when using GA in both Dataset3 and Dataset4 (see Fig. 8.a3 and Fig. 8.b3). Further, the scenario with GA improves the over-estimation results (see Fig. 8.a2 and Fig. 8.b2), in almost all configurations and datasets, except for Configuration1 and Dataset2. For instance, the usage of GA enables decreasing the probability of under-estimation and the mean of over-estimation from 0.05 to 0.03 (see Fig. 8.a1 and Fig. 8.b1) and from 12% to 10% (see Fig. 8.a2 and Fig. 8.b2), respectively, for Dataset3. In contrast, the scenario without GA gives better results for Configuration1, Configuration2, Dataset1 and Dataset2. However, the probability of under-estimation, as well as the mean of over-estimation, remain close for the two scenarios (adjustment without vs. with GA). This difference in performance between scenarios 2 and 3 (without GA, with GA, respectively) may be explained by the use of a small and prefixed sliding window and predicted data sizes $(10,\ [7,5])$ in scenario 2, which allows a quick adaptation to recent variations. However, scenario 2 loses in flexibility and adaptability of prediction, and is less accurate in the case of a fluctuating workload (e.g., Fig. 5, Dataset3 and Dataset 4). Therefore, the challenge is to find the optimal trade-off between the cost of SLA violations and the cost of resource wastage (minimizing under-

Table 7
Results of evaluation metrics of various padding strategies for configuration2.

Padding Strategy	*Probability of under-estimation*	*Mean of over-estimation (%)*	*Mean of over-estimation Static provisioning (%)*	*MTBUE (sec)*
ratio	**0.05**	41.88	65.3	**54.16**
std1	0.07	31.55	64.48	49.375
std2	0.058	32.28	**63.61**	50.71
std3	0.06	**31.26**	64.48	50

estimation and over-estimation), while ensuring adaptability and flexibility for the prediction algorithm.

8.2.3. Padding strategies

First, we computed the padding value dynamically based on the prediction error and the ratio r between the adjusted data and the observed data (see Equation (5)). We obtained good results in terms of under-estimation probability, but the mean of over-estimation was observed to be significantly higher when the workload tended to fluctuate.

Then, we tested standard-deviation-based (std) padding with different strategies by considering:

- strategy 1 (std1): the std of observed data in the previous prediction step.
- strategy 2 (std2): the mean of the previous std that were computed in the case of under-estimation greater than 10%.
- strategy 3 (std3): the mean of previous std that were computed in the case of under-estimation greater than 10%. If the current std value (current under-estimation >10%) is greater than twice the previous std value ($\sigma_{current} > 2\sigma_{previous}$), it is excluded from the mean std estimation.

The results of the std-based padding strategies show an improvement in terms of reducing over-estimation. However, the selected strategy, namely, strategy 3 (see Algorithm 3), outperforms in almost all scenarios and for all datasets. For instance, the mean of over-estimations of adjusted data in the case of Configuration2 with ratio-based padding is about 41.88, whereas the std-based padding reduced the over-estimation mean to 31.26 (std3). Table 7 presents the evaluation metrics in each padding strategy using Configuration2 data.

9. Conclusion and future work

This work presents a generic, dynamic and multi-step ahead prediction of resource demand in virtualized systems. Based on a time series and machine learning method, our proposed algorithm is able to provide real-time prediction of resource needs without any prior knowledge or assumptions on the system or its internal behavior. When unexpected workload fluctuations occur, the proposed algorithm is capable of adapting to these changes within a relatively short delay (e.g. on average, 40 s in Configuration 2). Our algorithm also includes a dynamic adjustment based on the estimation of prediction error probability, and padding strategies to minimize SLA violations and reduce over-estimation. Furthermore, the proposed algorithm is able to dynamically generate the size of the sliding window and the number of predicted data, bringing flexibility to the prediction and improving its performance. Thorough experiments were conducted using various virtualized systems and different workload profiles. The results show that our algorithm is able to reduce the under-estimation average by 86% as compared to prediction without adjustment. Further, our algorithm decreases the over-estimation average by 67% against threshold-based provisioning. For future work, additional experiments and analyses using datasets from different types of systems and applications would be valuable to reinforce the general characteristics of our algorithm. Additionally, further investigations and evaluations should be conducted in order to improve the adjustment delay, the trade-off between under-estimation and over-estimation, as well as the trade-off between historical data size and computational efficiency. Moreover, tests and prediction performance comparison using various machines learning (e.g. hybrid approach using Kriging and SVR), as well as investigations of abnormal behavior will be conducted.

Acknowledgment

This work has been supported in part by Natural Science and Engineering Research Council of Canada (NSERC), in part by Ericsson Canada and in part by Rogers Communication Canada.

Appendix A

Table A.1
Terms, definitions and symbols/acronyms

Symbol/Acronym	Definition
y_i	Observed data
$\widehat{y}_i$	Predicted data
$\widehat{y}_i^a$	Adjusted data
e_i	Error of the ith prediction: $e_i = \widehat{y}_i - y_i$
X	Continuous random variable that represents the observed error e_i
I	Interval of errors $I = [e_{\min}, e_{\max}]$for each prediction step
PDF	Probability Density Function (e.g., Normal, non-parametric)
$\Pr(x \in I)$	Probability that X is in the interval I.
I_{proba}	Interval of probability (e.g., $I_{Proba} = [0, 0.1[$)
$\in_i$	Error adjustment coefficient (e.g., min or max of errors) in the interval I_i
l	Number of sliding windows
n_{oe}	Number of over-estimations
n_{ue}	Number of under-estimations
α_i	Indicates whether or not a padding is added
β_i	Indicates whether or not there is an over-estimation
γ_i	Indicates whether or not the size of the sliding window is applicable
m_i	Number of predicted data in the interval I_i
n_i	Number of observed data within a sliding window used for training data in the interval I_i
n_s	Optimal number of training data in the interval I_i

(*continued on next column*)

Table A.1 (*continued*)

Symbol/Acronym	Definition
m_s	Optimal number of predicted data in the interval I_i
r	rounded ratio between the observed and the adjusted data: $r = \lceil y_i / \widehat{y_i^a} \rceil$
σ_j	Standard deviation: $\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \overline{y})^2}$ of the jthunder-estimation if it is less than −10%
$\overline{y}$	$\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$
$Pr_{UnderEstim}^{PredictData}$	Probability of under-estimation for predicted data: $\frac{number\ of\ underestimations\ in\ predicted\ data}{number\ of\ predicted\ data}$
$Pr_{UnderEstim}^{AdjustData}$	Probability of under-estimation for adjusted data: $\frac{number\ of\ underestimations\ in\ adjusted\ data}{number\ of\ adjusted\ data}$
$E_{OverEstim}^{PredictData}$	Mean of over-estimation for predicted data: $\frac{1}{n_{oe}}\sum_{i=1}^{n}(\hat{y}_i - y_i)$
$E_{OverEstim}^{AdjustData}$	Mean of over-estimation for adjusted data: $\frac{1}{n_{oe}}\sum_{i=1}^{n}(\hat{y}_i^a - y_i)$
$E_{OverEstim}^{Thres}$	Mean of over-estimation for static provisioning (threshold-based provisioning) $E_{OverEstim}^{Thres} = \frac{1}{n_{oe}}\sum_{i=1}^{n}(Thres - y_i)$
Thres	Over-provisioning of resources in legacy networks. It represents the maximum allocated resources for a specific system (e.g., OpenIMS) and load profile.
$uptime_i$	The ith uptime moment where there is no under-estimation.
$downtime_i$	The ith downtime moment where there is under-estimation after the ith uptime moment.
MTBUE	Mean Time Between Under-Estimations: $\frac{\sum_{i=1}^{n}(uptime_i - downtime_i)}{number\ of\ under-estimations}$
CPS	Call Per Second
MSE	Mean Squared Error $= \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$

Appendix B

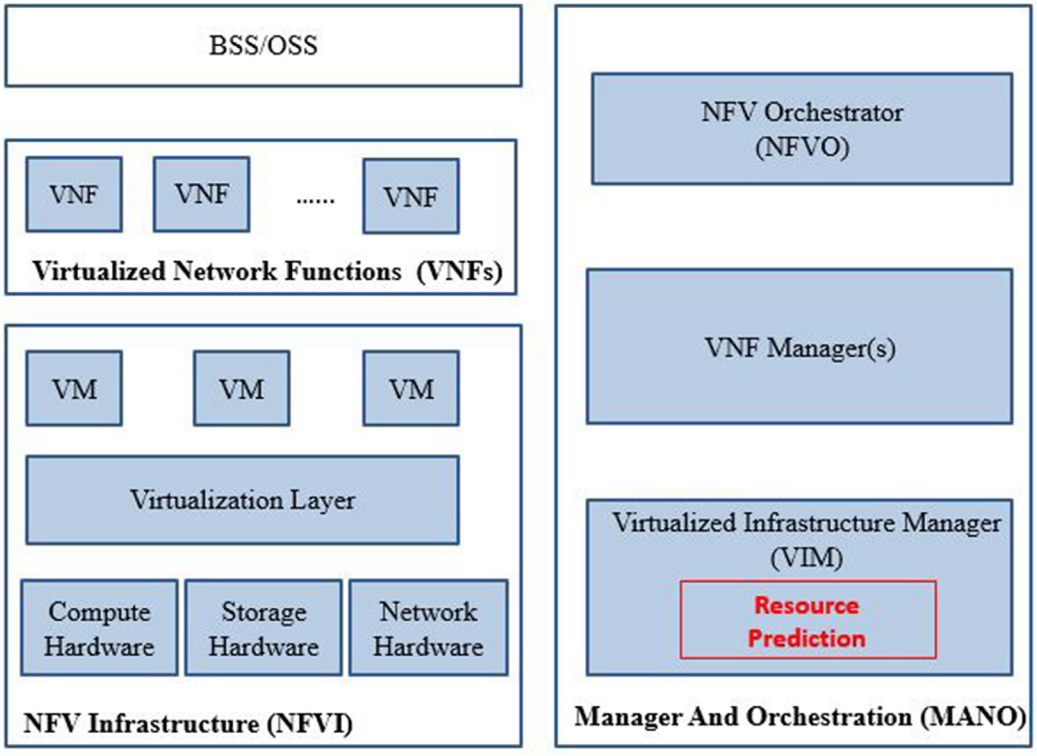

Fig. B.1. High-level NFV framework (ETSI GS NFV 002 V1.1.1, 1010).

References

Abramowitz, M., Stegun, I.A., 1972. Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables, vol. 55. National Bureau of Standards applied Mathematics Series. Online. http://people.math.sfu.ca/~cbm/aands/abramowitz_and_stegun.pdf. (Accessed 28 April 2017).

Amiri, M., Mohammad-Khanli, L., 2017. Survey on prediction models of applications for resources provisioning in cloud. J. Netw. Comput. Appl. 82, 93–113.

Cikit-learn. "Support vector machines". Online https://scikit-learn.org/stable/modules/svm.html#svm-regression, accessed: 2019-03-15.

Containers, Linux. Online. https://linuxcontainers.org/. (Accessed 6 June 2017).

Cortes, C., Vapnik, V., 1995. Support-vector networks. Mach. Learn. 20 (3), 273–297.

Dunn, P.F., 2014. Measurement and Data Analysis for Engineering and Science. CRC Press, Taylor & Francis, p. 616.

ETSI GS NFV 002 V1.1.1. "Network functions virtualisation (NFV);Architectural framework" Online https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf, accessed: 2019-07-31.

Fraunhofer, F.O.K.U.S.. Open source IMS core by cnd. Online. http://www.openimscore.org/. (Accessed 6 June 2017).

Fu, T., 2011. A review on time series data mining. Eng. Appl. Artif. Intell. 24 (1), 164–181.

Gambi, A., 2013. "Kriging-based Self-Adaptive Controllers for the Cloud". PhD thesis. University of Lugano. Online. http://doc.rero.ch/record/32769/files/2012INFO008.pdf.

Gandhi, A., Chen, Y., Gmach, D., Arlitt, M., Marwah, M., 2011. Minimizing data center sla violations and power consumption via hybrid resource provisioning. In: Green Computing Conference and Workshops (IGCC), 2011 International. IEEE, pp. 1–8.

R. GAYRAUD et al. "SIPp". Online http://sipp.sourceforge.net/, accessed: 2017-06-06.

Goudarzi, H., Pedram, M., 2016. Hierarchical SLA-driven resource management for peak power-aware and energy-efficient operation of a cloud datacenter. IEEE Transactions on Cloud Computing 4 (2), 222–236.

Gratton, Y., 2002. Le krigeage: La méthode optimale d'interpolation spatiale. Les articles de l'Institut d'Analyse Géographique. Online. https://cours.etsmtl.ca/sys866/Cours/documents/krigeage_juillet2002.pdf2002.

Hu, R., Jiang, J., Liu, G., Wang, L., 2013. CPU load prediction using support vector regression and kalman smoother for cloud. In: In Distributed Computing Systems Workshops (ICDCSW), 2013 IEEE 33rd International Conference on. IEEE, pp. 88–92.

Huang, C.J., et al., 2013. An adaptive resource management scheme in cloud computing. Eng. Appl. Artif. Intell. 26 (1), 382–389.

Iqbal, M.F., K John, L., 2012. Power and performance analysis of network traffic prediction techniques. In: Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium. IEEE, pp. 112–113.

Islam, S., Keung, J., Lee, K., Liu, A., 2012. Empirical prediction models for adaptive resource provisioning in the cloud. Future Gener. Comput. Syst. 28, 155–162.

Jiang, J., Lu, J., Zhang, Long, G., 2013. Optimal cloud resource auto-scaling for web applications. In: Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on. IEEE, pp. 58–65.

K Hoong, P., Tan, I.K., Keong, C.Y., 2012. Bittorrent network traffic forecasting with ARMA. Int. J. Comput. Netw. Commun. 4 (4), 143–156.

Krige, D.G., 1951. A statistical approach to some basic mine valuation problems on the witwatersrand". J. S. Afr. Inst. Min. Metall 52 (6), 119–139.

Liang, J., Cao, J., Wang, J., Xu, Y., 2011. Long-term CPU load prediction. In: Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference. IEEE, pp. 23–26.

Liu, J., Shen, H., Chen, L., 2016. CORP: "Cooperative opportunistic resource provisioning for short-lived jobs in cloud systems". In: Cluster Computing (CLUSTER), 2016 IEEE International Conference on. IEEE., pp. 90–99

Liu, G., et al., 2014. An indicator kriging method for distributed estimation in wireless sensor networks. Int. J. Commun. Syst. 27 (1), 68–80.

Lloyd, W., Pallickara, S., David, O., Lyon, J., Arabi, M., Rojas, K., 2013. Performance implications of multi-tier application deployments on Infrastructure-as-a-Service clouds: towards performance modeling. Future Gener. Comput. Syst. 29 (5), 1254–1264.

Ma, K., et al., 2017. "Spendthrift: Machine Learning Based Resource and Frequency Scaling for Ambient Energy Harvesting Nonvolatile Processors" Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific. IEEE, pp. 678–683.

Matheron, G., 1963. Principles of geostatistics". Econ. Geol. 58 (8), 1246–1266.

MathWorks, 2017. Numerically evaluate integral, adaptive Simpson quadrature. Online. https://www.mathworks.com/help/matlab/ref/quad.html. (Accessed 28 April 2017).

MathWorks, 2017. Numerical integration. Online. https://www.mathworks.com/help/matlab/ref/integral.html#btdd9x5. (Accessed 28 April 2017).

Mitsuo, G., Cheng, R., 2000. Genetic Algorithms and Engineering Optimization", vol. 495. J. Wiley and Sons, New York, N.Y.

Mkwawa, I.M., Kouvatsos, D.D., 2008. Performance modelling and evaluation of handover mechanism in IP multimedia subsystems. In: Systems and Networks Communications, 2008. ICSNC'08. 3rd International Conference on. IEEE, pp. 223–228.

Pezzè, A. Gambi M., Toffetti, G., 2016. Kriging-based self-adaptive cloud controllers. IEEE Transactions on Services Computing 9 (3), 368–381.

Qazi, K., Li, Y., Sohn, A., 2014. Workload prediction of virtual machines for harnessing data center resources. In: Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on. IEEE, pp. 522–529.

Reiss, C., et al., 2014. Google cluster-usage traces: format + schema. Online. https://drive.google.com/file/d/0B5g07T_gRDg9Z0lsSTEtTWtpOW8/view.

Shen, Z., Subbiah, S., Gu, X., Wilkes, J., 2011. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In: Proceedings of the 2nd ACM Symposium on Cloud Computing. ACM, p. 5.

Shyam, G.K., Manvi, S.S., 2016. Virtual resource prediction in cloud environment: a Bayesian approach. J. Netw. Comput. Appl. 65, 144–154.

Smola, A.J., Schölkopf, B., 2004. A tutorial on support vector regression. Stat. Comput. 14 (3), 199–222.

Tout, H., Talhi, C., Kara, N., Mourad, A., 2016. Selective mobile cloud offloading to augment multi-persona performance and viability. in IEEE Transactions on Cloud Computing 99, 1–14.

Tran, D., Tran, N., Nguyen, B.M., Le, H., 2016. PD-GABP - a novel prediction model applying for elastic applications in distributed environment. In: Information and Computer Science (NICS), 2016 3rd National Foundation for Science and Technology Development Conference on. IEEE, pp. 240–245.

V Srinivasan, B., Duraiswami, R., Murtugudde, R., 2008. Efficient kriging for real-time spatio-temporal interpolation Linear kriging. In: 20th Conference on Probablility and Statistics in Atmospheric Sciences, pp. 1–8.

van Beers, W.C.M., Kleijnen, J.P.C., 2004. Kriging interpolation in simulation: a survey. In: Proceedings of the 2004 Winter Simulation Conference, vol. 1. IEEE, p. 121.

Wang, W., et al., 2012. Application-level cpu consumption estimation: towards performance isolation of multi-tenancy web applications. In: 2012 IEEE 5th Iinternational Conference on Cloud Computing. IEEE, pp. 439–446.

W. W. S. Wei. "Time Series Analysis: Univariate and Multivariate Methods". Redwood City, Calif.; Don Mills, Ont.: Addison-Wesley. vol. 478 p. 1990.

Wei, Z., Tao, T., ZhuoShu, D., Zio, E., 2013. A dynamic particle filter-support vector regression method for reliability prediction. Reliab. Eng. Syst. Saf. 119, 109–116.

Yang, X., 2010. Engineering Optimization: an Introduction with Metaheuristic Applications, vol. 347. Wiley, Hoboken, N.J.

Yu, Y., Song, M., Ren, Z., Song, J., 2011. Network traffic analysis and prediction based on APM. In: Pervasive Computing and Applications (ICPCA). IEEE, pp. 275–280.

Zhang-Jian, D., Lee, C., Hwang, R., 2013. An energy-saving algorithm for cloud resource management using a Kalman filter. Int. J. Commun. Syst. 27 (12), 4078–4091.

Souhila Benmakrelouf holds a Master's degree in Software Engineering from the School of Superior Technology (ETS), University of Quebec (Canada) and Bachelor degree in Computer engineering from university of Houari Boumadienne, Algiers (Algeria). Currently, she is PhD student in ETS. Her main research interests include Cloud computing, resource management, machine learning techniques.

Nadjia, Kara holds a Ph.D. in Electrical and Computer Engineering from Ecole Polytechnique of Montreal (Canada), a Master's degree in Electrical and Computer Engineering from Ecole Polytechnique of Algiers (Algeria). She has several years of experience in research and development. She worked in industry for more than ten years. From 2005 to 2015, she held adjunct professor positions at INRS-EMT (Canada), University of Sherbrooke (Canada), and Concordia University (Canada). Since 2009, she is a full professor at the department of software engineering

Hanine Tout received the Ph.D. degree in software engineering from École de Technologie Supérieure (ÉTS), University of Quebec, Montreal, Canada and the MSc degree in computer science from the Lebanese American University, Beirut, Lebanon. She is currently a Postdoctoral Fellow at Ericsson, Montreal, Canada. Her research interests include 5G, IoT, machine learning, mobile cloud computing, mobile virtualization, optimization, Web services, security and formal verification. She is serving as TPC member for IMCET'16, NTMS 2016 and SSCC-2018 and a reviewer in IEEE Communications Letters, Computers & Security journal, IEEE Transactions on Cloud Computing and several international conferences. She is a student member of the IEEE.

Rafi Rabipour has been engaged in research and development at Bell-Northern Research, Nortel Networks, and Ericsson. His work at Bell-Northern Research and Nortel Networks was mainly focused on the development of digital signal processing algorithms aimed at improving the performance of wireless and VoIP products. At Ericsson he participated in research in the domain of Cloud Computing, on topics such as non-linear performance characteristics of virtualized applications, specific facets of the Internet-of-Things, as well as approaches to resource management. He holds a Master's degree in Electrical Engineering from McGill University in Montreal.

Claes, Edstrom is Senior Specialist Cloud Computing and is based in Montreal. He is responsible for initiating and executing exploratory projects in the areas of NFV and Cloud technologies. His research interests include application transformation, resource management and automation in cloud computing environments.Before joining Ericsson Canada in 2007, Edstrom spent +15 years working for the company in Sweden and Denmark in various technology roles. He has been working with system studies for WCDMA, tender support for initial 3G/UMTS offerings roll out and system upgrades for 2G/3G networks in the Nordic region and development of AMPS/D-AMPS networks in North America.