An online learning model based on episode mining for workload prediction in cloud

Maryam Amiri [a,*,1], Leyli Mohammad-Khanli [a,2], Raffaela Mirandola [b,2]

[a] *Faculty of Electrical and Computer Engineering, University of Tabriz, 29 Bahman Blvd, Tabriz, Iran*
[b] *Dipartimento di Elettronica, Informazione e Bioingegneria Politecnico di Milano, Via Golgi 42, 20133 Milan, Italy*

ABSTRACT

The resource provisioning is one of the challenging problems in the cloud environment. The resources should be allocated dynamically according to the demand changes of the applications. Over-provisioning increases energy wasting and costs. On the other hand, under-provisioning causes Service Level Agree-ments (SLA) violation and Quality of Service (QoS) dropping. Therefore the allocated resources should be close to the current demand of applications as much as possible. Thus, the prediction of the future workload of applications is an essential step before the resource provisioning. In our previous work, we proposed a Prediction mOdel based on SequentIal paTtern mINinG (POSITING), which considers the correlation between different resources and extracts behavioural patterns of applications independently of the fixed pattern length explicitly. Although POSITING provides reliable results, it is not able to adapt according to the workload variations. The application behaviour might change and drift due to the dynamic nature of cloud. For this purpose, we investigate the capabilities of online learning for POSITING. This paper proposes a Prediction mOdel based on epIsode miNing with the capabiliTy of onlIne learNinG (RELENTING) based on POSITING. Thus, in addition to the accuracy, adaptability, one of the most important characteristics of the application prediction models, is fulfilled. The performance of the proposed model is evaluated based on both real and synthetic workloads. The experimental results show that the proposed model adapts to the behavioural changes of the application and learns the new behavioural patterns rapidly in comparison to the other state-of-the-art methods such as moving average, linear regression, neural networks and hybrid prediction approaches.

1. Introduction

Elasticity is one of the prominent features of cloud computing [1,2] to deploy applications such as health care services [3] on it. Elasticity is the degree of the system adaptability to the workload changes by provisioning and deprovisioning the resources automatically in a way that the allocated resources match the current demand [4]. Under-provisioning causes Service Level Agreements (SLA) violation, Quality of Service (QoS) dropping and the customer dissatisfaction. This may lead to the loss of customers and a decrease in revenue. On the other hand, Over-provisioning wastes energy and resources and it even increases costs like network, cooling and maintenance [5]. Thus, the resources allocated to the application should be close to its demand in a way that SLA is satisfied and resources wasting is minimized [5]. The future demand prediction is the only practical and effective solution for the

* Corresponding author.
E-mail addresses: maryam.amiri@tabrizu.ac.ir (M. Amiri), l-khanli@tabrizu.ac.ir (L. Mohammad-Khanli), raffaela.mirandola@polimi.it (R. Mirandola).
[1] PhD Student.
[2] Associate Professor.

Published Journal Article available at: https://doi.org/10.1016/j.future.2018.04.044

fast resources provisioning and the rapid elasticity implementation [5,6]. If the sudden increase of the future demand is predicted, the resource manager scales up the infrastructure and prepares Virtual Machines (VMs) according to the predicted future demand before the surge of demand occurs. In the same way, according to the demand reduction, the allocated resources are released. Indeed, allocated resources match the demand and the rapid elasticity is accomplished. The most important characteristics of the application prediction models are as follows [5]:

- Accuracy: The prediction models are evaluated by the accuracy of the predicted results. The accuracy of the models can be measured in different ways [5]. Generally, the models whose outputs are closer to the actual values are more reliable [7].
- Adaptability (online learning): The cloud environment is dynamic and is changing continuously. Therefore, the prediction model should be able to adapt to the changes. For this purpose, the model should learn the behavioural changes of the application.
- Proactive: The prediction should be proactive. It means that before the workload burstiness occurs, the model should be able to predict the future demand sooner in a way that the resource manager has enough time to provide the appropriate resources.
- Historic Data: The different types of resources are allocated to cloud services [8]. So, the application behaviour is affected by the different resources. An effective prediction model should investigate all the resources. It should also consider the correlation between them.

The newest and the most common proposed predictors are based on machine learning techniques [5] such as Neural Network (NN) [6,9–11], Moving Average (MA), regression based methods [12–14] and Hybrid Prediction Approaches (HPAs) [6,9,15]. The machine learning techniques usually model the application behaviour as a time series. Most of the methods are based on a sliding window [5]. These models cannot extract all the useful patterns whose length is less/more than the fixed length. Choosing the length of the pattern (the length of the sliding window) for different regions of workloads is one of the most important challenges in these methods [5]. Moreover, when they are applied to bursty cloud workloads, they have limited accuracy [16].

The common prediction models such as NN, Support Vector Machine (SVM) and regression based methods are the discriminative learning approaches. The discriminative methods need much training data. The behavioural changes of the application workload might start after training the prediction model. To adapt to the workload changes, the models should be retrained using new data. Gathering the new data and retraining the models might be very time consuming. Furthermore, the delay of the learning phase might lead to drastic loss in cloud. Due to the cloud nature, retraining and continuous updates of the prediction models should not be time consuming and require the heavy computations [5]. Therefore, these predictors could not satisfy the essential characteristics of the prediction models of cloud applications.

In our previous work [17], we proposed a prediction model based on episode mining, called POSITING, which investigates the correlation between different resources and extracts the behavioural patterns independently of the fixed pattern length explicitly. As the red dashed box in Fig. 1 shows, POSITING considers the application behaviour in the past, extracts the behavioural patterns and stores them in the off-line pattern base. Based on the extracted patterns and the recent behaviour of the application, POSITING predicts the future demand of different resources. POSITING alleviates the shortcomings of the prior predictors with its ability to extract the patterns of different length explicitly. However, the application behaviour can change and drift due to the dynamic nature of cloud. So, since the off-line learned patterns are extracted from the past behaviour of the application, they might not be useful to predict the current behaviour. It means that POSITING is not able to adapt to the workload changes. On the other hand, the computation resources consumption of the prediction model should be reasonable [5]. So, storing the entire data and then processing them are impractical. For this purpose, in this paper, we investigate the capability of online learning for POSITING. This paper develops POSITING and proposes a pREdiction modeL based on Episode miNing with the capabiliTy of onlIne learNinG (RELENTING).

To measure the strength of off-line patterns to predict the current behaviour of the application, we define a consistency criterion for them. According to the blue dashed box in Fig. 1, RELENTING compares the predicted results of POSITING with the observed behaviour and updates the consistency criterion of the off-line patterns. If there is no reliable off-line pattern for prediction, based on the recent and the current observed behaviour of the application, the online pattern mining engine extracts the new behavioural patterns and stores them in the online pattern base. Both of the online and the off-line pattern bases are used to predict the behaviour of the application. Thus, in addition to leaning new behavioural patterns, the reliable off-line patterns are recognized and used for prediction. So, the contributions of this paper are as follows:

- In this paper, for the first time, we define a new criterion, called *Consistency*, for the patterns extracted in the off-line mode. The consistency of each pattern is updated based on its ability to predict the current behaviour of the application. Thus, in addition to the patterns learned in the online-mode, the reliable off-line patterns are identified and used for prediction (Section 4.1).
- For the first time, this paper presents a comprehensive prediction model based on the pattern mining with the capability of online learning. While the model predicts the status of all the allocated resources, it also learns the new behaviour of the application rapidly without gathering the new data and retraining the model. So, time and space complexities of RELENTING are reasonable in a way that its deployment is affordable (Sections 4.2 and 4.3).

The rest of the paper is organized as follows: Section 2 reviews related works on the prediction of the workload in cloud. An overview of POSITING is presented in Section 3. Section 4 introduces RELENTING in detail. We present the experimental results in Section 5. Finally, the paper is concluded with our future work in Section 6.

2. Related work

In general, according to our previous work in [5], the prediction models proposed for cloud applications are classified into three main groups: control theory, queuing theory and machine learning techniques. Furthermore, in [17], we propose a new type of the predictor based on Sequential Pattern Mining (SPM). In the following subsections, each group is discussed briefly.

2.1. Control theory

In control models, the goal is to control resources shared between cloud applications [5]. In [18] a Single Input Single Output (SISO) model maps the CPU share of the application to the inverse of its response time. In [19], the resources usage of all VMs hosted on a server is mapped to their performance by using a Multi Input Multi Output (MIMO) model. The interference among VMs

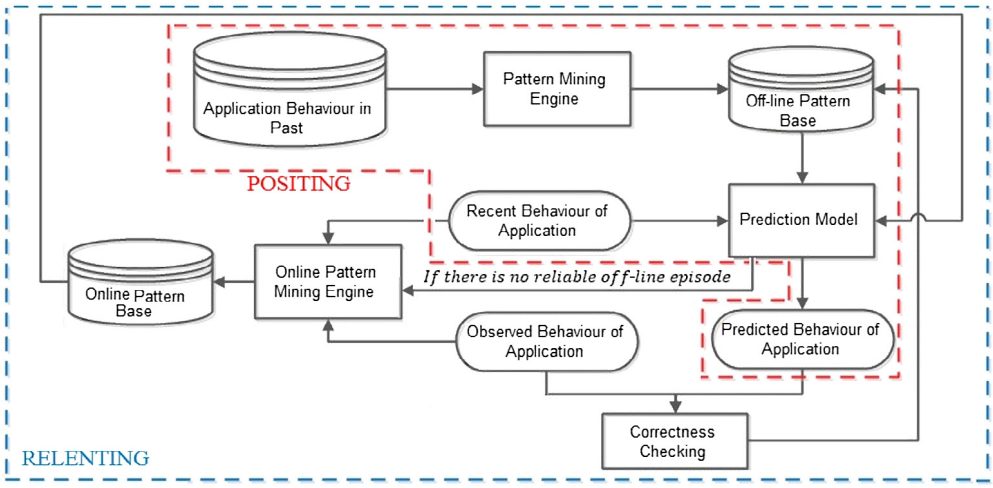


Fig. 1. The scheme of POSITING and RELENTING.

hosted on a server is handled by using the dynamic adjustment of resources allocated to applications. Wu et al. in [20] present a feed back control algorithm whose goal is to maximize the profit rate. The cost and the benefit are calculated for different combinations of reconfiguration actions and VMs. A combination with the maximum profit and the minimum cost is selected.

Some controllers assume a restrictive constraint: the linear controllers assume that the application behaviour is linear [21]. Thus, there is potential of instability. Although the non-linear controllers model the application behaviour accurately, their mathematical computations are complex [5]. Some controllers such as fuzzy controllers [22,23] are based on the rule based approaches. The ability of the controllers depends on the defined rules. Furthermore, the rule based approaches do not have the learning capability.

2.2. Queuing network

The Queuing Network (QN) models the relationship between the workload and the performance criteria [20]. In QN, each server allocated to the application is a queuing system [24]. These models have parameters such as the requests arrival rate and the average resources requirements of requests that should be specified [24]. The parameters can be estimated by solving some equations resulted from the system evaluation.

In [25], the response time of the transactional workload and the throughput of batch jobs are modelled by using the open QN and the closed QN respectively. In [26], the CPU demand of different types of transactions is estimated by using the regression based methods. The estimated values are used to parameterize QN. QN determines the resources requirements of multi-tier applications according to the workload fluctuations.

Although QN needs no training phase, it is very sensitive to the parameterization. The precise estimation of parameters such as the arrival ratio and the service time of requests is expensive and difficult. Assuming the specified probability distributions for some parameters is not reasonable due to the dynamic nature of cloud. Furthermore, some mechanisms are needed to adapt QN models to incorporate the new behavioural patterns immediately [5].

2.3. Machine learning techniques

The newest proposed approaches are based on machine learning techniques [5]. Most of these methods need a training phase to learn the application behaviour. The method proposed in [27] predicts the CPU utilization of transactional applications by NN. Chen et al. in [15] propose a system to predict the resources demand. Due to the workload dynamics in different periods, the base predictors such as the Second Moving Average Model (SMAM), the Exponential Moving Average method (EMA), the Auto-Regression (AR) model and the Trend Seasonality Model (TSM) are selected. The output of the base predictors is sent to a Fuzzy Neural Network (FNN) which improves the accuracy of the prediction results. Yang et al. in [12] propose a method based on LR to predict the number of requests for each cloud service. According to the workload fluctuations, the prediction method adjusts itself through the recomputation of parameters of the regression model.

In general, the prediction accuracy of these methods is based on the behavioural similarities of the application in the training and the test phases. If the application behaviour in the test phase is not correlated with one in the training phase, the predicted results are not reliable and the training phase should be repeated so that the model can adapt to the workload dynamics [5].

2.4. SPM

In [17], for the first time, we propose a predictor based on SPM to predict the future demand of cloud applications. This predictor, POSITING, is able to extract all the behavioural patterns of workloads independently of the fixed pattern length explicitly. POSITING investigates the correlation between resources and extracts the corresponding patterns. So, the behavioural patterns of workloads are readily interpretable by the resources manager. Furthermore, it does not need to make any assumptions about the workload behaviour. So, POSITING could be used for the different types of workloads. As the experiment results in [17] show, POSITING outperforms the state-of-the-art predictors and provides reliable results. However, if the behavioural changes of the application workload start after the pattern extraction, it is not able to adapt to the workload changes. On the other hand, since consumption of the computation resources of the prediction model should be reasonable [5], storing the entire data and then processing them are impractical.

In this paper, we investigate the problem of online learning for POSITING. We develop POSITING and propose RELENTING. RELENTING updates the consistency criterion of the off-line patterns and learns the new behavioural patterns. Both online and off-line learned patterns are used for prediction. So, in addition to providing the accurate prediction results, RELENTING is able to adapt to the workload changes rapidly. In the next section, the foundation of POSITING is discussed in more detail.

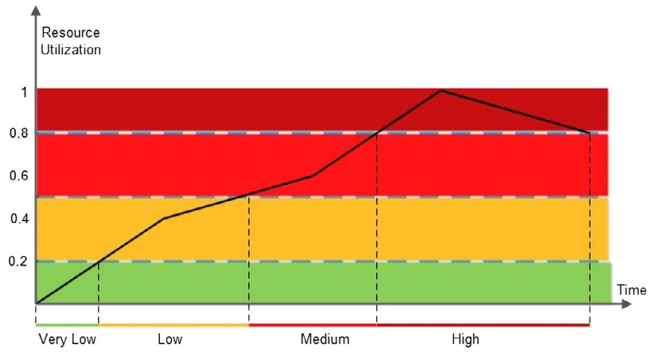


Fig. 2. Converting a time series into a symbolic (discretized) time series by the value abstraction that $\sum = \{Very\ Low, Low, Medium, High\}$ and blue dashed lines show the border of the values [17]. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

3. An overview of POSITING

In this section, POSITING is described briefly. Firstly, the background concepts such as **event**, **stream** and **observation** are introduced. Then, the core of POSITING, the pattern extraction in the off-line mode, is explained concisely. We recommend that readers refer to [17] for a more detailed description of POSITING.

3.1. Background concepts

As Fig. 2 shows, POSITING converts the numeric time series of all the resources allocated to the application into a se quence of abstractions $\langle S_1[st_1, et_1], \ldots, S_n[st_n, et_n]\rangle$ where $S_i \in \sum, 1 \leq i \leq n$ is an abstraction that holds from time st_i to time et_i and $\sum$ is the abstraction alphabet. Let $Status = \{S_1, \ldots, S_M\}$ be a set of the abstract values (the abstraction alphabet) and $ResourceType = \{R_1, \ldots, R_N\}$ be a set of all the resources allocated to the application. Without loss of generality, we define an arbitrary order on $ResourceType$, for example $R_1 < R_2 < \cdots < R_N$.

Definition 1. An **event** e_i is defined as a 4-tuple $\langle r_i, s_i, st_i, et_i\rangle$ that means the abstract value of $r_i \in ResourceType$ is $s_i \in Status$ from the start time st_i to the end time et_i. The span of the event $e_i = \langle r_i, s_i, st_i, et_i\rangle$ is $\Delta e_i = et_i - st_i > \epsilon \geq 0$.

All the discretized time series of the resources are represented as a multivariate stream. Note that the value of ϵ depends on the length of sampling intervals. In coarse grained sampling, ϵ is set to small values. For fine grained sampling, ϵ could be set to larger values.

Definition 2. A multivariate **stream** $E = \langle e_1, e_2, \ldots, e_n\rangle$, where n is the index of the latest observed event, is a sequence of events that are ordered according to their start time:

$\forall e_i, e_j \in E$ that $1 \leq i < j \leq n : (st_i < st_j)$ or $(st_i = st_j$ and $r_i < r_j)$.

Definition 3. A **state** is an ordered pair of (r, s), where $r \in ResourceType$ and $s \in Status$. The Resource-Status (**RS**) is a set of all the possible states: $RS = \{(r, s)|\forall r \in ResourceType, \forall s \in Status\}$.

If the span of events is large, they are decomposed based on the decomposition unit μ. For example the event $(Disk, Medium, 3, 7)$ with $\mu = 3$ is decomposed into two events $(Disk, Medium, 3, 6)$ and $(Disk, Medium, 6, 7)$. However, after decomposing the event e, the span of the last decomposed event might be less than ϵ. Here, to satisfy Definition 1, the latest and penultimate decomposed events merge together.

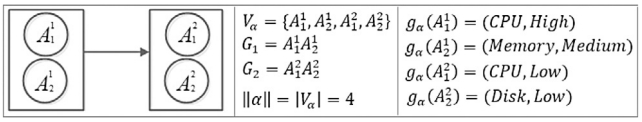


Fig. 3. The graphical representation of the episode $\alpha = (CPU, High)(Memory, Medium) \rightarrow (CPU, Low)(Disk, Low)$ [17].

Inspired by the temporal relations defined in [28], we define two types of relations between events: **concurrent** and **consecutive**.

Definition 4. Given the stream $E = \langle e_1, \ldots, e_n\rangle$, two events e_i and $e_j, 1 \leq i, j \leq n$, are **concurrent** iff $|st_i - st_j| \leq \epsilon$ and are **consecutive** iff $|st_i - st_j| > \epsilon$.

An episode is a partially ordered collection of events that occur together [29]. Note that we use terms "pattern" and "episode" interchangeably in this paper.

Definition 5. A Concurrent Nodes Group (CNG) $G = A_1 A_2 \ldots A_l$ is a group of nodes that $\forall A_j, A_m \in G, 1 \leq j, m \leq l$, there is no partial order between A_j and A_m.

Definition 6. The episode α is defined as a directed acyclic graph $\alpha = (V_\alpha, \prec_\alpha, g_\alpha)$, where V_α is a set of nodes, $\prec_\alpha$ is a partial order on V_α and $g_\alpha : V_\alpha \rightarrow RS$ is a function that maps each node into one state. The episode α is composed of $k(>1)$ CNGs in the form of $G_1 = A_1^1, A_2^1, \ldots, A_{l_1}^1, \ldots, G_k = A_1^k, A_2^k, \ldots, A_{l_k}^k$ that:

1. $|G_i| = l_i$
2. $V_\alpha = \{A_1^1, \ldots, A_{l_1}^1, A_1^2, \ldots, A_{l_2}^2, \ldots, A_1^k, \ldots, A_{l_k}^k\}$
3. $\forall A_j^i \in G_i, \forall A_n^m \in G_m, 1 \leq i < m \leq k, j \in \{1, \ldots, l_i\}, n \in \{1, \ldots, l_m\} : A_j^i \prec_\alpha A_n^m$
4. $|CNG_\alpha| = k$
5. $G_i' = \{(r, s) \in RS | g_\alpha(v) = (r, s),\ \forall v \in G_i\}$.

The episode α could be represented as a general form $\alpha = G_1' \rightarrow G_2' \rightarrow \cdots \rightarrow G_k'$.

Example 1. Consider the episode $\alpha = (V_\alpha, \prec_\alpha, g_\alpha)$ in Fig. 3. The set V_α contains four nodes. As it is shown, the function g_α maps the nodes into the states and $A_1^1 \prec_\alpha A_1^2$, $A_1^1 \prec_\alpha A_2^2$, $A_2^1 \prec_\alpha A_1^2$ and $A_2^1 \prec_\alpha A_2^2$. As a simple graphical notation, this episode is represented as $\alpha = (CPU, High)(Memory, Medium) \rightarrow (CPU, Low)(Disk, Low)$.

Informally, the occurrence of an episode in the stream means that the nodes of the episode have the corresponding events in the stream such that the partial order of the episode is preserved [29]. We define $freq(\alpha)$ as the number of the Non-Overlapped (NO) minimal occurrences [30] of the episode α.

Example 2. Consider the stream $E = \langle e_1 = (CPU, High, 0, 3), e_2 = (Memory, Medium, 0, 4), e_3 = (Network, Low, 0, 2), e_4 = (Disk, Medium, 0, 3), e_5 = (Network, Medium, 2, 5), e_6 = (CPU, 3, 5, Low), e_7 = (Disk, Low, 3, 5), e_8 = (Memory, Very\ Low, 4, 5)\rangle$. For $\epsilon = 0$, there is an occurrence of the episode α given in Example 1 in the stream E.

Definition 7. Given the episode α that $|CNG_\alpha| = k$, each occurrence O of α is determined as a sequence of the starting intervals of CNGs: $O = (([t_1^i, t_2^i])_{i=1}^k)$.

Example 3. Consider Example 2. The starting intervals of the occurrence of G_1 and G_2 are $[t_1^1, t_2^1] = [0, 0]$ and $[t_1^2, t_2^2] = [3, 3]$ respectively. The span of the occurrence is $[0, 3]$.

Dynamic resources allocation is based on the virtualization techniques [31,32]. Based on time spent on booting VMs, patterns should be extracted from the application behaviour in a way that SLA is satisfied and energy wasting is avoided. Given the episode $\alpha = G'_1 \rightarrow G'_2 \rightarrow \cdots \rightarrow G'_k$ and an occurrence $O = ([w_1^i, w_2^i]_{i=1}^k)$ of α, if the time it takes to instantiate a new VM instance is $\delta(>\epsilon)$ time slots and the gap constraint $\Delta(\geq\delta)$ determines that resources might be allocated at most $\Delta - \delta$ time slots before occurring the workload burstiness, then the valid interval of w_1^i, $1 < i \leq k$ is $[w_2^{i-1} + \delta, w_2^{i-1} + \Delta]$ to satisfy QoS and SLA and avoid wasting energy. δ and Δ are called the minimum internal gap and the maximum internal gap respectively. As we explained in [17], Δ should be in the interval of $[\delta, \delta + \epsilon]$ to provide the precise prediction results.

As Fig. 1 shows, the recent behaviour of the application should be determined to predict the future behaviour based on extracted episodes.

Definition 8. An observation OB is a list of states which describe the recent status of resources allocated to the application. It satisfies three conditions below:

- The states of each entry of the list OB, $OB[i]$, $1 \leq i \leq |OB|$ are corresponding to events that are concurrent.
- The states of each entry are in ascending order based on the start time of their corresponding events. The entries are in ascending order based on the minimum start time of their events.
- There are at least two states in each two consecutive entries whose corresponding events are not concurrent. The corresponding events of states of each two consecutive entries satisfy the gap constraint Δ.

Definition 9. Given the observation OB, an observation OB' that $OB' \subseteq OB$ is a consistent observation of OB iff there is a serial relation under the gap constraints between each two consecutive entries.

Definition 10. If OB' is a consistent observation of OB and there is no other consistent observation of OB such as OB'' that $OB' \subset OB''$, then OB' is the Longest Consistent Observation (LCO) of OB.

Example 4. Assume $\Delta = \delta = 3$, the current time slot is 21 and the recent events of the stream are as follows:

$\langle \ldots, (CPU, Low, 16, 19), (Disk, Low, 17, 21),$
$(Memory, High, 18, 21),$
$(CPU, High, 19, 21), (Network, Medium, 20, 21)\rangle.$

Fig. 4 shows OBs and LCOs that are extracted from the stream for different values of ϵ. Note that since the span of the event $(CPU, Low, 16, 19)$ is 3, then $\epsilon + 1 \leq 3$.

3.2. The pattern extraction in the off-line mode

The main step of POSITING is to extract the frequent patterns. For this purpose, the pattern tree is constructed and the frequent patterns are extracted [33].

Definition 11. Given the episode $\alpha = G'_1 \rightarrow G'_2 \rightarrow \cdots \rightarrow G'_k$ and $(r, s) \in RS$, the **serial extension** of α with (r, s) is:

$$\alpha \oplus (r, s) = G'_1 \rightarrow G'_2 \rightarrow \cdots \rightarrow G'_k \rightarrow (r, s). \tag{3.1}$$

Definition 12. Given the episode $\alpha = G'_1 \rightarrow G'_2 \rightarrow \cdots \rightarrow G'_{k-1} \rightarrow G'_k$ and $(r, s) \in RS$, the **concurrent extension** of α with (r, s) is:

$$\alpha \odot (r, s) = G'_1 \rightarrow G'_2 \rightarrow \cdots \rightarrow G'_{k-1} \rightarrow G'' \text{ that } G'' = G'_k \cup (r, s). \tag{3.2}$$

Mining frequent episodes might lead to extract a huge number of patterns. To improve the mining efficiency and avoid information loss, a compressed set of episodes, called **closed episodes**, is extracted [34]. Under gap constraints δ and Δ, if there is no other episode such as β that α is its prefix or suffix and $freq(\alpha) = freq(\beta)$, then α is a **closed episode**. The pattern tree is constructed based on the serial and concurrent extensions. Firstly, POSITING extracts candidate closed episodes by the complete traverse of the pattern tree in a depth-first way. In the next step, candidate closed episodes are considered and closed episodes are determined by using a hashing procedure with the frequency as the key. To avoid enlarging the pattern tree, we could limit the number of CNGs of episodes. We define $Level$ as the maximum number of CNGs of episodes.

4. RELENTING

When the stream continuously grows, old episodes might become obsolete while new episodes might emerge [35]. To identify the new episodes, storing data and then processing them are not practical due to the real time nature of the cloud environment [36,37] and the consumption of the computation resources. Furthermore, the delay of the learning phase might lead to drastic loss in cloud [5]. So the efficient online approaches are required to identify the new behaviour of the application from the growing stream. Briefly, the online mining process should be fast and responsive along with reasonable time and space complexities. For this purpose, we propose RELENTING, which is inspired by the types of the human memory. There are two major categories of the memory: the long-term memory and the short-term memory. The long-term memory holds information that is related to the past happenings. On the contrary, the short-term memory is responsible for storing new information temporarily [38]. As Fig. 1 shows there are two pattern bases. The off-line pattern base is corresponding to the long-term memory and it stores the episodes extracted from the application behaviour observed in a long period. The online pattern base is corresponding to the short-term memory and it stores the episodes extracted from the new behaviour of the application. As long-term memories are much more complex than short-term ones, the off-line pattern base also includes the closed episodes and their NO frequency. On the contrary, the online pattern base includes the episodes and approximate information about their occurrences.

RELENTING is composed of two main steps: (1) Updating the off-line pattern base (2) Extracting the episodes from the new behaviour. In Section 4.1, the update of the off-line pattern base is explained. The main procedure of online learning is described in Section 4.2. Finally, the prediction module of RELENTING is presented in Section 4.3.

4.1. The update of the off-line pattern base

We should measure the strength of the episodes to predict the future behaviour of the application, in terms of appropriate criteria. Given the episode $\alpha = G'_1 \rightarrow G'_2 \rightarrow \cdots \rightarrow G'_k$, POSITING employs two criteria to evaluate it:

1. *MatchScore*: It determines that how much the episode α matches the longest consistent observation LCO. If the episode includes more states of LCO, it receives higher *MatchScore*. The criterion *MatchScore* is defined in (4.1)

ϵ	OB	LCO
0	(CPU, Low) | $(Disk, Low)$ | $(Memory, High)$ | $(CPU, High)$ | $(Network, Medium)$	$(CPU, Low) \rightarrow (CPU, High)$ $(Disk, Low) \rightarrow (Network, Medium)$ $(Memory, High)$
1	(CPU, Low) $(Disk, Low)$ | $(Disk, Low)$ $(Memory, High)$ | $(Memory, High)$ $(CPU, High)$ | $(CPU, High)$ $(Network, Medium)$	$(CPU, Low)(Disk, Low) \rightarrow (Network, Medium)$ $(CPU, Low) \rightarrow (CPU, High)(Network, Medium)$ $(Disk, Low)(Memory, High)$ $(Memory, High)(CPU, High)$
2	(CPU, Low) $(Disk, Low)$ $(Memory, High)$ | $(Disk, Low)$ $(Memory, High)$ $(CPU, High)$ | $(Memory, High)$ $(CPU, High)$ $(Network, Medium)$	$(CPU, Low)(Disk, Low) \rightarrow (Network, Medium)$ $(CPU, Low) \rightarrow (CPU, High)(Network, Medium)$ $(Disk, Low)(Memory, High)(CPU, High)$ $(Memory, High)(CPU, High)(Network, Medium)$ $(CPU, Low)(Disk, Low)(Memory, High)$

Fig. 4. The observations (OB) and the longest consistent observations (LCOs) extracted from the stream in the Example 4 for $\epsilon = 0, 1, 2$ [17].

where *Index* is the index of the start of *LCO* in α. According to the procedure of the episode selection proposed in [17], we have $Index \geq 1$. If α is also consistent with the recent history before *LCO*, then $Index > 1$ and α receives higher *MatchScore*.

$$MatchScore(\alpha) = \frac{|state \in (LCO \cap \alpha)|}{|state \in LCO|} + \frac{Index - 1}{|CNG_\alpha|}. \quad (4.1)$$

2. *Confidence*: it measures the reliability of the inference made by the episode. For two sub-episodes $\gamma = G'_1 \rightarrow \cdots \rightarrow G'_i$ and $\beta = G'_{i+1} \rightarrow \cdots \rightarrow G'_k$ where $1 \leq i < k$, *Confidence* could be interpreted as the conditional probability of occurring β, having occurred γ. $Confidence(\alpha)$ is computed as (4.2) [29]. It is clear that $freq(\alpha) \leq freq(\gamma)$. So higher *Confidence* implies that occurring β after γ is more probable.

$$Confidence(\alpha) = \frac{freq(\alpha)}{freq(\gamma)}. \quad (4.2)$$

The criterion *MatchScore* considers only the recent observation of the application and the criterion *Confidence* is computed based on the application behaviour in the past. None of the criteria investigates whether the episode could appropriately reflect the current behaviour of the application or not. For this purpose, RELENTING employs a new criterion for the episode, called *Consistency*, which is computed based on the results predicted by the episode.

Definition 13. Given the longest consistent observation *LCO* and the episode $\alpha = G'_1 \rightarrow G'_2 \rightarrow \cdots \rightarrow G'_k$, if $Prefix(\alpha, i) = G'_1 \rightarrow \cdots \rightarrow G'_i$, $1 \leq i \leq k$, matches *LCO*, then $Antecedent(\alpha, LCO)$ is $Prefix(\alpha, i)$ and $Consequent(\alpha, LCO)$ is $Suffix(\alpha, i+1) = G'_{i+1} \rightarrow \cdots \rightarrow G'_k$.

For each G'_i, $1 \leq i \leq k$, of the episode $\alpha = G'_1 \rightarrow \cdots \rightarrow G'_k$ extracted in the off-line mode, two counters $Cons_i$ and $Incons_i$ are defined. Given the longest consistent observation *LCO*, if α matches *LCO*, the counters are updated as follows:

- For all *CNG*s of $Antecedent(\alpha, LCO)$, the counter *Cons* increases by $+1$.
- If $Consequent(\alpha, LCO) = G'_p \rightarrow \cdots \rightarrow G'_k$, $1 < p \leq k$, for each G'_j, $p \leq j \leq k$, the counter $Cons_j$ increases by $+1$ if $G'_p \rightarrow \cdots \rightarrow G'_j$ is consistent with the future behaviour of the application. Otherwise, the counter $Incons_j$ increases by $+1$.

Now, we could define the criterion *Consistency*. For each G'_i, $1 \leq i \leq k$, of the episode $\alpha = G'_1 \rightarrow \cdots \rightarrow G'_k$ extracted in the off-line mode, $Consistency(G'_i)$ measures how much G'_i matches the application behaviour after the sub-episode $\beta = G'_1 \rightarrow \cdots \rightarrow G'_{i-1}$ is observed.

Definition 14. Given the episode $\alpha = G'_1 \rightarrow \cdots \rightarrow G'_k$ extracted in the off-line mode, $Consistency(G'_i)$, $1 \leq i \leq k$, is defined as follows:

$$Consistency\left(G'_i\right) = \begin{cases} \frac{Cons_i}{Cons_i + Incons_i} & if\ Cons_i + Incons_i \neq 0 \\ -1 & if\ Cons_i + Incons_i = 0 \end{cases}. \quad (4.3)$$

Lemma 1. *Given the episode $\alpha = G'_1 \rightarrow \cdots \rightarrow G'_k$ extracted in the off-line mode and $1 \leq i < k$, $Consistency(G'_i) \geq Consistency(G'_{i+1})$.*[1]

Lemma 2. *Given the episode $\alpha = G'_1 \rightarrow \cdots \rightarrow G'_k$ extracted in the off-line mode, $Incons_1 = 0$.*

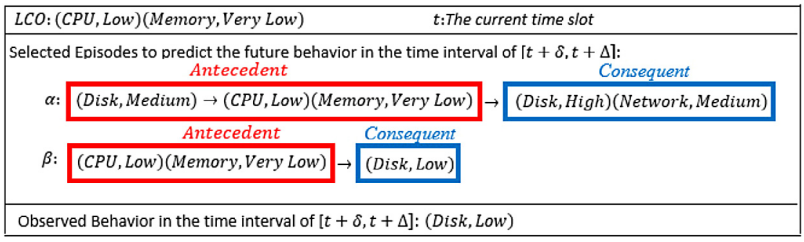
Example 5. Fig. 5(a) shows *LCO* and the episodes extracted to predict the future behaviour in the interval of $[t + \delta, t + \Delta]$ where t is the current time slot. *LCO* is in the first part of β and in the middle part of α. As the figure shows *Antecedent* and *Consequent* of episodes are determined based on *LCO* (red and blue boxes). As it is shown in Fig. 5(b)(A) the counters *Cons* of $Antecedent(\alpha, LCO)$ and $Antecedent(\beta, LCO)$ increase by $+1$. If the future behaviour of resources in the interval of $[t + \delta, t + \Delta]$ is $(Disk, Low)$, the counter *Incons* of $Consequent(\alpha, LCO)$ and the counter *Cons* of $Consequent(\beta, LCO)$ increase by $+1$ according to Fig. 5(b)(B).

4.2. The procedure of the online learning

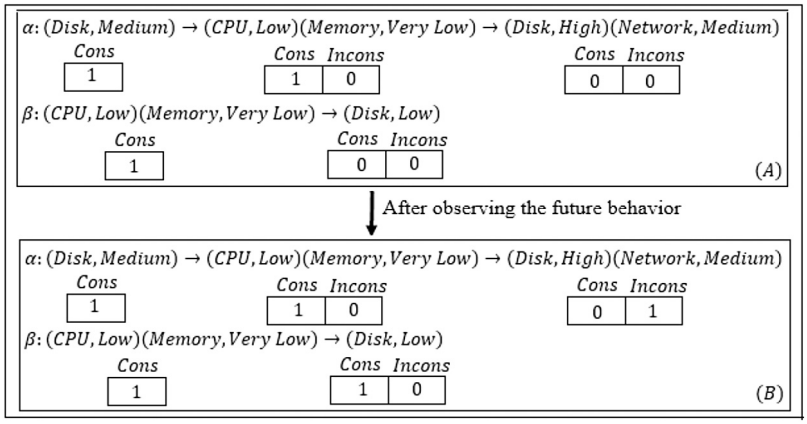
One of the most important advantages of the episode mining is to model online learning. A straight algorithm to learn the new behaviour of the application is based on the approach proposed in [35]. Ao et al. in [35] propose MESELO (Mining frEquent Serial Episode via Last Occurrence) for online frequent episode mining. It stores all the minimal occurrences of episodes [29]. As a new event emerges, all the new minimal occurrences of episodes are determined by appending the new event to the last occurrence of episodes. In the following example, we show how this approach could be employed to model the online learning for POSITING.

Example 6. Assume the online learning starts from the time slot 20, $A, B, C \in RS$ and $\delta = 2$, $\Delta = 3$, $\epsilon = 1$. Fig. 6(a) shows a time series and the corresponding events of A, B and C. Fig. 6(b) shows the constructed episodes and their occurrences [17]. Note that each occurrence of the episode α is summarized in a 4-tuple (x, t, t', t''), where x is the end of the starting interval of the penultimate CNG of α, $[t, t']$ is the starting interval of the last CNG and t'' is the start time of the occurrence. As Fig. 6(b) shows, when the event $(A, 20, 23)$ emerges, the episode A is constructed. With coming $(B, 22, 24)$, the episode $A \rightarrow B$ and its corresponding occurrence are created. Furthermore, the episode B is created. When the event

[1] The proof of lemmas and theorems could be found in Appendix C.

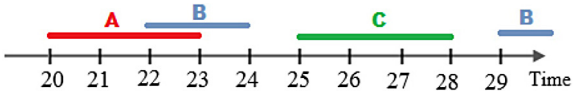


(a) *LCO* and the episodes extracted for prediction.



(b) The update of counters *Cons* and *Incons* of the episodes.

Fig. 5. An example of the episode selection and the update of counters *Cons* and *Incons* based on *LCO* and the future behaviour. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



(a) The time series and the corresponding events.

A	$A \rightarrow B$	B	$A \rightarrow B \rightarrow C$	$B \rightarrow C$	C
(20,20,20,20)	(20,22,22,20)	(22,22,22,22) (29,29,29,29)	(22,25,25,20)	(22,25,25,22)	(25,25,25,25)

(b) The new episodes constructed by appending the new events to the occurrences of the previous episodes.

Fig. 6. An example of the online learning of the episodes based on the method proposed in [35].

$(C, 25, 28)$ comes, three episodes $A \rightarrow B \rightarrow C$, $B \rightarrow C$ and C are constructed. With coming $(B, 29, 30)$, none of the episodes could be extended with it. The new occurrence of B is inserted in its occurrence list.

Although MESELO is able to identify the frequent episodes by online learning, the memory consumption for storing all of the episodes and their occurrences is a critical challenge. Since, time and space complexities of the prediction model should be reasonable in cloud [39], this approach is not appropriate.

Theorem 1. *The number of the episodes constructed by appending the coming events to the existing episodes, grows exponentially.*

In addition to the exponential growth of constructed episodes, this method learns the application behaviour completely even if the application behaviour has not changed significantly. On the contrary, according to the real time nature of cloud, the predictor should learn the new behaviour of the application quickly with reasonable consumption of the resources. To learn the new behaviour of the application and avoid generating repetitious episodes, RELENTING only focuses on learning the new behaviour of the application. The procedure of online learning in RELENTING is composed of two steps: (1) generating episodes implying the new behaviour of the application and (2) extending the generated episodes based on the observed behaviour. These steps are explained in more detail in the following subsections.

4.2.1. Generating episodes in the online mode

To provide the reliable prediction, the selected episodes, in addition to having high *Confidence*, should cover the current behaviour of the application. When there is not such an episode for prediction, it means that there is no confident consistent episode:

Definition 15. Given the episode $\alpha = G'_1 \rightarrow G'_2 \rightarrow \cdots \rightarrow G'_k$ and the longest consistent observation *LCO*, if $\underline{Consequent}(\alpha, LCO) = Suffix(\alpha, i)$, $2 \leq i \leq k$ and $Confidence(\alpha) \times \sqrt{|Consistency(G'_i)|} \geq \omega$, where $\omega \in [0, 1]$ is a user-defined threshold, then α is a coNsIstent ConfidEnt (NICE) episode.

If there is no NICE episode, RELENTING notices that the current behaviour of the application does not match the extracted episodes. So the episodes describing the new behaviour should be identified. RELENTING learns the new behaviour from *LCO*. It extracts the continuous sub-episodes from *LCO* and extends them with the events observed in the valid interval of the sub-episodes.

(a) The data structure of *LearnedEpisode*.

(b) The data structure of *LCO*.

Fig. 7. The data structures used by RELENTING to extract new episodes.

Fig. 7a shows the data structure *LearnedEpisode*, which includes the new learned episode and information about it:

- *Episode*: The episode learned in the online mode
- *Ref* : The number of references to *Episode*
- $[I_1, I_2]$: The valid interval of the next *CNG* of *Episode* for the serial extension
- *Last*: The time of the latest occurrence of *Episode*.

The data structure *ListLearnedEpisodes* is a list of all the *LearnedEpisodes*. As it was mentioned in Definition 10, an *LCO* is the longest observation that satisfies the gap constraints. Fig. 7b shows the data structure of *LCO*. Since online learning is based on *LCO*, the data structure *LCO* includes *Episode* and *Time*, whose *Episode* is the longest consistent observation and *Time* is the last time of the starting interval of the last group of *Episode*. The data structure *LCOList* is a list of all *LCO*s extracted from the observation.

RELENTING extracts all of the continuous sub-episodes of *LCO*s by the algorithm *ExtractEpisodeFromLCO* (see Algorithm 1). The function receives *LCOList*, extracts the continues sub-episodes from each *LCO* of *LCOList* and adds them to *ListLearnedEpisodes*. In line 4, it is checked whether a continues sub-episode of *LCO.Episode* exists in *ListLearnedEpisodes* or not. If it does, in lines 6 to 11, the corresponding entry of the sub-episode is updated: *Ref* increases by +1, if the sub-episode is finished by the last group of *LCO.Episode*, then the interval of $[I_1, I_2]$ is updated based on *LCO.Time* and *Last* is set to *LCO.Time*. If the sub-episode does not exist in *ListLearnedEpisodes*, then a new entry is created for it and added to *ListLearnedEpisodes* in lines 13 to 21. Note that since *LCO.Time* is the last time of the starting interval of the last group of *LCO.Episode*, $[I_1, I_2]$ is only updated for the sup-episodes finished by the last group of *LCO.Episode*.

Algorithm 1 ExtractEpisodeFromLCO

Input: *LCOList* % *LCOList* is a list of *LCO*s extracted from the observation
Output: % The function adds the new episodes to *ListLearnedEpisodes*
1: **for** (*LCO in LCOList*) **do**
2: **for** ($i = 1; i \leq |LCO.Episode|; i++$) **do**
3: **for** ($j = min(i+1, |LCO.Episode|); j \leq |LCO.Episode|; j++$) **do**
4: $index \leftarrow IndexOf(LCO.Episode[i..j], ListLearnedEpisodes)$;
5: **if** ($index > 0$) **then**
6: $ListLearnedEpisodes[index].Ref++$;
7: **if** ($j = |LCO.Episode|$) **then**
8: $ListLearnedEpisodes[index].I_1 \leftarrow LCO.Time + \delta$;
9: $ListLearnedEpisodes[index].I_2 \leftarrow LCO.Time + \Delta$;
10: **end if**
11: $ListLearnedEpisodes[index].Last \leftarrow LCO.Time$;
12: **else**
13: define new *LearnedEpisode R*;
14: $R.Episode \leftarrow LCO.Episode[i..j]$;
15: **if** ($j = |LCO.Episode|$) **then**
16: $R.I_1 \leftarrow LCO.Time + \delta$;
17: $R.I_2 \leftarrow LCO.Time + \Delta$;
18: **end if**
19: $R.Last \leftarrow LCO.Time$;
20: $R.Ref++$;
21: $add(R, ListLearnedEpisodes)$;
% add *R* to *ListLearnedEpisodes*
22: **end if**
23: **end for**
24: **end for**
25: **end for**

Example 7. Assume $LCO.Episode = (CPU, Low) \rightarrow (Memory, High) \rightarrow (Disk, Medium) \rightarrow (Memory, Low)$ and $LCO.Time = t'$. Table 1 shows the continuous sub-episodes extracted from *LCO*. These sub-episodes are inserted in *ListLearnedEpisodes*. Note that *Last* of all the sub-episodes is set to t' and $[I_1, I_2]$ of sub-episodes 3, 5, 6 and 7 is set to $[t' + \delta, t' + \Delta]$. It means that sub-episodes 3, 5, 6 and 7 are extended serially by events observed in the interval of $[t' + \delta, t' + \Delta]$.

Lemma 3. *The number of continuous sub-episodes extracted from each LCO is* $O(Level^2)$.

4.2.2. Extending the episodes learned in the online mode

After extracting the new episodes by the algorithm *Extract EpisodeFromLCO*, these episodes are serially extended by the events observed in their valid intervals. For this purpose, all of the valid combinations of concurrent events are extracted from the concurrent events group.

Lemma 4. *The number of the possible combinations of each concurrent events group is* $O(2^{|ResourceType|})$.

The algorithm *LearningProcedure* (see Algorithm 2) serially extends the learned episodes with the concurrent events observed in their time intervals. The function receives a list of all the valid combinations of the concurrent events, called *CombList*. In line 1, all episodes that could be extended serially in the current time slot are determined. In lines 3 to 20, the serial extensions of the episodes with the concurrent events groups are considered. In lines 5, the episode *R* is extended with the concurrent events group *Comb*. In line 6, the extended episode is searched in *ListLearnedEpisodes*. If this episode exists in *ListLearnedEpisodes*, its corresponding entry is updated in lines 8 to 10. Otherwise, in lines 12 to 17, a new *LearnedEpisode* is constructed and added to a temporary list, *Bag*, which is later emptied into *ListLearnedEpisodes* on coming out of the loop [30]. Note that *Last* is the time of the latest occurrence of the episode. So it is set to t. Furthermore, since the procedure of online learning should be rapid and its resources consumption should be reasonable , the episodes are serially extended once. So the valid interval of the extended episodes is set to $[-1, -1]$.

Example 8. Consider Example 7 again. Assume $t' = 20$, $\delta = \Delta = 3$, $\epsilon = 0$ and $t = 24$. So sub-episodes 3, 5, 6 and 7 in Table 1 are extended by the concurrent events group. If the observed concurrent events group is $G' = (CPU, Medium)(Network, High)$, there are three combinations $(CPU, Medium)$, $(Network, High)$ and $(CPU, Medium)(Network, High)$. Table 2 shows the serial extension of the sub-episodes by all the combinations of G'.

Algorithm 2 LearningProcedure

Input: *CombList* % *CombList* is a list of all the valid combinations of the event group
Output: % The function extends the episodes of *ListLearnedEpisodes* serially
1: $UpdatingEpisodes \leftarrow \{R | R \in ListLearnedEpisodes \text{ that } R.I_1 + 1 \leq t \leq R.I_2 + 1\}$; % t is the current time
2: $Bag \leftarrow \varnothing$;
3: **for each** ($Comb \in CombList$) **do**
4: **for each** ($R \in UpdatingEpisodes$) **do**

Table 1
The continuous sub-episodes extracted from *LCO* of Example 7.

No	Sub-episode	No	Sub-episode
1	$(CPU, Low) \rightarrow (Memory, High)$	4	$(Memory, High) \rightarrow (Disk, Medium)$
2	$(CPU, Low) \rightarrow (Memory, High) \rightarrow (Disk, Medium)$	5	$(Memory, High) \rightarrow (Disk, Medium) \rightarrow (Memory, Low)$
3	$(CPU, Low) \rightarrow (Memory, High) \rightarrow (Disk, Medium) \rightarrow (Memory, Low)$	6	$(Disk, Medium) \rightarrow (Memory, Low)$
7	$(Memory, Low)$	–	–

Table 2
The serial extension of sub-episodes in Example 7 by the concurrent events group $G' = (CPU, Medium)(Network, High)$.

No	Episode
1	$(CPU, Low) \rightarrow (Memory, High) \rightarrow (Disk, Medium) \rightarrow (Memory, Low) \rightarrow (CPU, Medium)$
2	$(CPU, Low) \rightarrow (Memory, High) \rightarrow (Disk, Medium) \rightarrow (Memory, Low) \rightarrow (Network, High)$
3	$(CPU, Low) \rightarrow (Memory, High) \rightarrow (Disk, Medium) \rightarrow (Memory, Low) \rightarrow (CPU, Medium)(Network, High)$
4	$(Memory, High) \rightarrow (Disk, Medium) \rightarrow (Memory, Low) \rightarrow (CPU, Medium)$
5	$(Memory, High) \rightarrow (Disk, Medium) \rightarrow (Memory, Low) \rightarrow (Network, High)$
6	$(Memory, High) \rightarrow (Disk, Medium) \rightarrow (Memory, Low) \rightarrow (CPU, Medium)(Network, High)$
7	$(Disk, Medium) \rightarrow (Memory, Low) \rightarrow (CPU, Medium)$
8	$(Disk, Medium) \rightarrow (Memory, Low) \rightarrow (Network, High)$
9	$(Disk, Medium) \rightarrow (Memory, Low) \rightarrow (CPU, Medium)(Network, High)$
10	$(Memory, Low) \rightarrow (CPU, Medium)$
11	$(Memory, Low) \rightarrow (Network, High)$
12	$(Memory, Low) \rightarrow (CPU, Medium)(Network, High)$

```
5:        r' ← R.Episode ⊕ Comb;
6:        Index ← IndexOf(r', listLearnedEpisodes);
7:        if (Index ≠ 0) then
8:            ListLearnedEpisodes[Index].Ref + +;
9:            ListLearnedEpisodes[Index].Last ← t;
10:           ListLearnedEpisodes[Index].[I1, I2] ← [−1, −1];
11:       else
12:           define new LearnedEpisode R'';
13:           R''.Episode ← r';
14:           R''.Ref + +;
15:           R''.Interval ← [−1, −1];
16:           R''.Last ← t;
17:           add(R'', Bag);          % add R'' to Bag
18:       end if
19:   end for
20: end for
21: ListLearnedEpisodes ← ListLearnedEpisodes ∪ Bag;
```

4.3. The prediction model

Since RELENTING is based on POSITING, the algorithm *Main* of RELENTING is similar to POSITING's. It (see Algorithm 3) predicts the future status of resources. The algorithm has six global variables t, *PredictionCount*, *CorrectCount*, *ResultTable*, *ListLearnedEpisodes* and *OffLinePatternBase*. t is the current time. *PredictionCount* counts the number of predictions and *CorrectCount* counts the percentage of resources that have been predicted correctly in each prediction slot. *ResultTable* is a list of the data structure *Result*, which includes the prediction results [17]. *ListLearnedEpisodes* is a list of *LearnedEpisodes*. The function receives four threshold values θ, *Level*, ω and Ω (to select episodes) and four parameters δ, Δ, ϵ and μ. T is the time slot in which the future behaviour of the application is predicted for the first time. In line 3, t is set to the time slot in which prediction is performed for the first time. *history* defined in line 4 includes the recent history of the stream of the application. In each repeat of the while loop (lines 7 to 28) the future behaviour of the resources is predicted. In line 8, *history* is updated based on the new observed events. In line 9, *LastObservation* includes the most recent events of all the resources. *LastPredicting* determines the time slot in which the latest prediction has been performed. *Observation* is found in line 12 and *LCO*s are extracted in line 13. In lines 14 to 18, for extracted *LCO*s, the most appropriate episodes are selected for the prediction. Note that *MatchedCons* and *MatchedAnt* maintain *Consequent* and *Antecedent* of the matched episodes respectively. The function *Evaluating* in line 19, evaluates the matched episodes and predicts the future behaviour. In lines 20 to 27, based on observed events, the matched episodes are pruned and the precision of the prediction is computed. If the prediction results derived by a matched episode is consistent with the future behaviour, the corresponding entry of the episode is fill with ∗. Otherwise, the episode is omitted. Since the functions *EpisodeSelection*, *Evaluating*, *CreateLEG* and *UpdateMatchedEpisodes* have been modified, we only present these functions in detail. The readers could refer to [17] for more detail of the other functions.

Algorithm 3 Main Algorithm

```
Global Variables: t; OffLinePatternBase; PredictionCount ← 0;
    CorrectCount ← 0; ResultTable: a list of Results; ListLearnedEpisodes:
    a list of LearnedEpisodes;
Input: θ, Level, ω, Ω, δ, Δ, ϵ, μ;
1:  OffLinePatternBase ← AllEpisodes(ϵ, δ, Δ, θ, Level);
2:  LEG ← ∅
3:  t ← T − δ;          % T is the first time slot that is predicted, t is the first
    time in which prediction is performed (the current time)
4:  history ← ∅;
5:  MatchedCons ← ∅, MatchedAnt ← ∅;
6:  ResultTable ← ∅
7:  while (1) do
8:      history ← UpdateHistory(history, LEG);          % Observed events
    are added into history
9:      LastObservation ← FindLastObservation(history);          % It finds
    the most recent events of resources in history
10:     LastPredicting ← t;          % It is the time in which the latest
    prediction has been performed
11:     Set ← ∅, Prefix ← ∅;
12:     Suffix ← ExtractObservation(ϵ, Δ, history, Level);
13:     ExtractLCO(ϵ, δ, Δ, Set, Prefix, Suffix);
14:     for each (LCO in Set) do
15:         A, C ← EpisodeSelection(LCO, OffLinePatternBase)
16:         MatchedCons ← MatchedCons ∪ C;
17:         MatchedAnt ← MatchedAnt ∪ A;
18:     end for
19:     Evaluating(LastObservation, MatchedCons, MatchedAnt, history,
    δ, Δ, ω, Ω, Set);
20:     LEG ← CreateLEG(μ, ϵ, LastObservation, LEG, LastPredicting,
    history);
21:     while (LEG.I1 ≤ LastPredicting) do
22:         ComputePrecision(LEG);
23:         UpdateMatchedEpisodes(LEG, MatchedAnt, MatchedCons);
24:         LEG ← CreateLEG(μ, ϵ, LastObservation, LEG,
    LastPredicting, history);
25:     end while
26:     ComputePrecision(LEG);
27:     UpdateMatchedEpisodes(LEG, MatchedAnt, MatchedCons);
28: end while
```

1. Function *EpisodeSelection*: episodes that match *LCO* are selected for prediction. Since the main goal is to predict the future behaviour, two groups of episodes are selected: (1) the episodes that

LCO is in their first part and (2) the episodes that *LCO* is in their middle part and their first part is consistent with the events observed before *LCO*. The function *EpisodeSelection* proposed in [17] is used to select these episodes. According to line 15 in algorithm *Main*, the function *EpisodeSelection* splits the matched episodes and returns their *Antecedent* and *Consequent* separately.

Example 9. Consider Example 5 again. If α and β are the only episodes that match *LCO*, the function *EpisodeSelection* splits them and returns their *Antecedent* (A) and *Consequent* (C) as follows:

$A = \{(Disk, Medium) \rightarrow (CPU, Low)(Memory, VeryLow),$
$(CPU, Low)(Memory, Very\ Low)\}$
$C = \{(Disk, High)(Network, Medium), (Disk, Low)\}.$

2. Function *Evaluating*: The function *Evaluating* evaluates the matched episodes and predicts the future behaviour of the application. It receives the *LastObservation*, *MatchedCons* and *MatchedAnt*, *history*, the parameters δ and Δ, the threshold values ω and Ω that are used to select NICE episodes and online episodes respectively and *Set* that is a set of *LCO*s. If the future status of resources cannot be predicted by NICE episodes extracted in the off-line mode, the prediction is performed by the episodes extracted in the online mode. In addition, the new episodes are identified and added to the online pattern base (see Algorithm 5 in Appendix A.1).

Example 10. Consider Example 5 again. Assume α and β are the only episodes that match *LCO*. So the function *Evaluating* investigates them to predict the future behaviour. Fig. 8(A) shows the counters *Cons* and *Incons* of the episodes before calling the function *Evaluating*. According to Fig. 8(B), when the function *Evaluating* is called, the counters *Cons* of *Antecedent*(α, *LCO*) and *Antecedent*(β, *LCO*) increase by +1. Note that the counters of *Consequent*(α, *LCO*) and *Consequent*(β, *LCO*) are updated after the future behaviour of resources is observed. Based on the frequencies of episodes, *ListFreq* and *freq* (each entry *freq*[i], $1 \leq i \leq |LCO|$ includes the frequency of *LCO*[$i..|LCO|$]) in Fig. 8(C), the criteria *Consistency*, *Confidence* and *MatchScore* are computed for states predicted by the episodes. Fig. 8(D) shows the data structure *PredictionStep*, which is a list of all the states that could occur in the next time slot and information about them [17]. It is clear that although α matches the past behaviour of the application (high *Confidence*), it is not consistent with the current behaviour (low *Consistency*). On the contrary, the episode β is more consistent with the current behaviour compared to the past behaviour. These imply that the application behaviour has changed after extracting the off-line pattern base. If $\omega = 0.8$, there is no NICE episode for prediction. So prediction is performed by the episodes extracted in the online mode.

3. Function *PredictionByOnlineLearning*: To predict the future behaviour of the application using the online learned episodes, the most credible episodes that match *LCO* are selected. For this purpose, we define a criterion that measures credibiliTy of the online episodes fOr Prediction (*TOP*). To define *TOP*, two important factors *Ref* and *Last* of *LearnedEpisode* are investigated. It is clear that the episodes with higher *Ref* and *Last* closer to the current time slot are more credible. The criterion *TOP* is defined as follows:

Definition 16. Given the online episode $\alpha = G'_1 \rightarrow G'_2 \rightarrow \cdots \rightarrow G'_k$ and the current time slot t, $TOP(\alpha, t)$ is defined as follows:

$$TOP(\alpha, t) = 1 - \frac{1}{1 + \alpha.Ref(0.9)^{t-\alpha.Last}}. \quad (4.4)$$

The algorithm *PredictionByOnlineLearning* (Algorithm 4) predicts the future status of resources based on the online learned episodes. It receives *LastObservation*, *history*, *LCOList*, which is a list of *LCO*s extracted from the observation and Ω, which is a threshold value to select episodes based on their *TOP*. The function predicts the future status of resources based on the online learned episodes. In lines 2 to 19, all the extracted *LCO*s are processed. In lines 4 to 6, it is checked whether *LCO* could be used to predict the status of the resources in the time slot t or not. In lines 8 to 9, all the suffixes of *LCO.Episode* are determined and the learned episodes including them are found by calling the function *SearchOnlineEpisodes* (see algorithm 6 in Appendix A.2). In lines 10 to 17 of Algorithm 4, *PredictionList* is filled with the future status of resources and the maximum values of *Ref* and *Last*. In lines 20 and 21, based on the threshold Ω, the most credible episodes are maintained and sorted in *PredictionList*. In line 24, the most credible status of resources is considered as prediction results. In lines 26 to 28, for resources that are not in *PredictionList*, the future status is predicted based on *MRE* (Most Recent Event) [17]. If *MRE* is not found, according to line 30, the latest observed status of resources is considered as the future status.

Algorithm 4 PredictionByOnlineLearning

Input: *LCOList*, Ω, *LastObservation*, *history*;
Output: *Outcome*; % *Outcome* includes the future status of resources
1: *PredictionList* $\leftarrow \varnothing$; % *PredictionList* is a list of *LearnedEpisodes*
2: **for** ($k = 1$; $k \leq |LCOList|$; $k++$) **do**
3: *LCO* $\leftarrow$ *LCOList*[k];
4: **if** (*LCO.Time* $+ 1 \neq t$) **then**
5: Continue;
6: **end if**
7: **for** ($i = 1$; $i \leq |LCO.Episode|$; $i++$) **do**
8: $S \leftarrow$ *LCO.Episode*[$i..|LCO.Episode|$];
9: $L \leftarrow$ *SearchOnlineEpisodes*(S);
10: **for** ($h = 1$; $h \leq |L|$; $h++$) **do**
11: *Index* $\leftarrow$ *IndexOf*($L[h]$, *PredictionList*);
12: **if** (*Index* $= 0$) **then**
13: *add*((*L*[*h*].*FirstGroup*(), *L*[*h*].*Ref*, [−1, −1], *L*[*h*].*Last*), *PredictionList*) % add (*L*[*h*].*FirstGroup*(), *L*[*h*].*Ref*, [−1, −1], *L*[*h*].*Last*) to *PredictionList*
14: **else**
15: *PredictionList*[*Index*] $\leftarrow$ (*L*[*h*].*FirstGroup*(), max(*PredictionList*[*Index*].*Ref*, *L*[*h*].*Ref*), [−1, −1], max(*PredictionList*[*Index*].*Last*, *L*[*h*].*Last*));
16: **end if**
17: **end for**
18: **end for**
19: **end for**
20: *Sort*(*PredictionList*, *Descending*, *Top*) % Sort *PredictionList* in descending order based on the criterion *TOP*
21: *DeleteEntriesTOP*(*PredictionList*, Ω); % delete entries of *PredictionList* with *TOP* $< \Omega$
22: **for each** (*Resource* $x \in$ *Result.Outcome*) **do**
23: **if** ($x \in$ *PredictionList*) **then**
24: *Outcome*[x] $\leftarrow$ *FirstStatus*(x, *PredictionStep.Outcome*) % It returns the first status of x in *PredictionList*
25: **else**
26: *MRE* $\leftarrow$ *FindMRE*(x, *LastObservation*, *history*) % It returns the most recent event in *history* that it matches the status and the span of the event of x in *LastObservation*;
27: **if** (*MRE* $\neq \varnothing$) **then**
28: *Outcome*[x] $\leftarrow$ *NextObservedStatus*(x, *MRE*, *history*) % It returns the observed status of the resource x after *MRE*
29: **else**
30: *Outcome*[x] $\leftarrow$ *FindStatus*(x, *LastObservation*) % It returns the status of the resource x in *LastObservation*
31: **end if**
32: **end if**
33: **end for**
34: return *Outcome*;

Example 11. Assume *LCO.Episode* $= (CPU, Low) \rightarrow (Network, High)$, *LCO.Time* $= 4999$, the current time slot is 5000

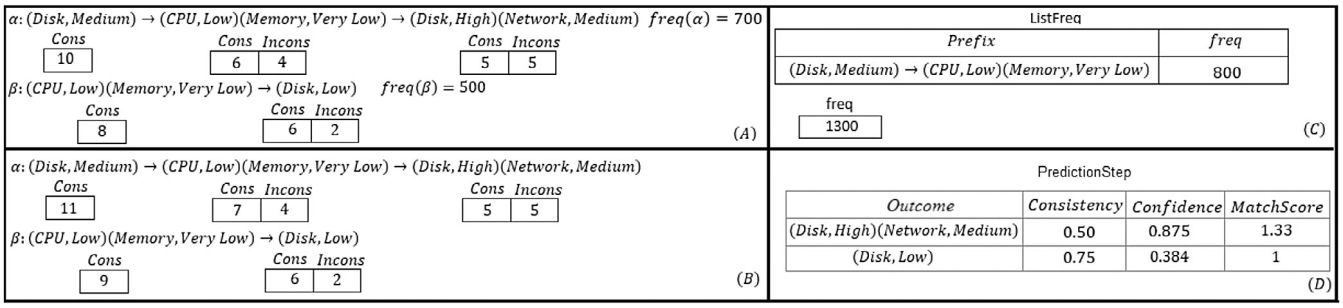


Fig. 8. Evaluating the matched episodes in Example 5.

and there is no NICE episode for prediction (Fig. 9(A)). Fig. 9(B) shows *ListLearnedEpisodes*. Since α matches *Suffix*(*LCO.Episode*, 2) and β matches *Suffix*(*LCO.Episode*, 1), they are selected for prediction. So the first groups of *Consequent*(α, *LCO*) and *Consequent*(β, *LCO*) are inserted in *PredictionList*. As Fig. 9(C) shows, *TOP*(α, 5000) = 0.9 and *TOP*(β, 5000) = 0.62. Note that since α.*Last* is closer to the current time slot, it is more credible for prediction. If $\Omega = 0.8$, then the corresponding entry of β is omitted from *PredictionList* and α determines the next status of *CPU* is *Medium*. The status of the other resources is predicted by *MRE*.

4. Function *CreateLEG*: The data structure *LEG* includes the latest event group of resources [17]. As it was mentioned, the episodes learned in the online mode are serially extended by the concurrent events group. So, at first, all of the valid combinations of concurrent events of *LEG* are identified. In the next step, the episodes are serially extended by these combinations. For this purpose, the function *CreateLEG* proposed in [17] is modified as algorithm 7 in Appendix A.3.

Example 12. Assume the current time slot is 19, $\epsilon = 0, \delta = \Delta = 1$ and $\mu = 3$. So the next status of resources is predicted for the time slot 20. *LastObservation*, which includes the latest events of resources in the stream, is shown in Fig. 10. As Fig. 10 shows, in the first call of the function *CreateLEG*, all the resources are sampled. So the current time slot is 20 now. Based on the status of sampled resources, *LEG* includes the latest updated events of resources. *LEG.EventSection* includes events whose start time is in the interval of [*LEG*.$I_1 - 1$, *LEG*.$I_2 - 1$] and *LEG.PrevtSection* includes events whose start time is before *LEG*.$I_1 - 1$. As the figure shows, *CombList* includes all the valid combinations of the events of *LEG.EventSection*.

5. Function *UpdateMatchedEpisodes*: As it was mentioned, in RELENTING, there are two counters *Cons* and *Incons* for episodes extracted in the off-line mode. When an episode such as α matches *LCO*, the counters *Cons* of *Antecedent*(α, *LCO*) increase by +1. The counters *Cons* and *Incons* of *Consequent*(α, *LCO*) are updated based on the correspondence between the predicted behaviour and the observed behaviour. So the function *UpdateMatchedEpisodes* is modified in a way that counters of *Consequent* of the selected episodes are updated based on the observed behaviour of the application (see Algorithm 8 in Appendix A.4).

Example 13. Assume $\epsilon = 0$, $\Delta = \delta = 1$ and Fig. 11(A) and (B) show *LCO* and NICE episodes used to predict the time slot 222. Fig. 11(C) and (D) show *MatchedAnt* and *MatchedCons* of the selected episodes. It is clear that α and β predict the status of *CPU* and *Memory* is *Low* and *Medium* respectively in the time slot 222. Note that the counters *Cons* of *Antecedent*(α, *LCO*) and *Antecedent*(β, *LCO*) increase by +1 (the red numbers). Assume the status of *CPU* and *Memory* is *Low* and *High* respectively in the time slot 222. It means that β does not predict the status of *Memory* correctly. So the counter *Cons* of *Consequent*(α, *LCO*) and the counters *Incons* of *Consequent*(β, *LCO*) increase by +1 (the red numbers). Since the prediction result of α is consistent with *LEG*, the first element after $*$ in *Consequent* of α ((*CPU*, *Low*)) is filled with $*$ and the matching time of α ($\alpha.I$) is set to 221 (Fig. 11(D)). Furthermore, β is removed from *MatchedAnt* and *MatchedCons* in the next call of the function *UpdateMatchedEpisodes*.

5. Evaluation

In [17], we evaluate POSITING on real and synthetic workloads comprehensively and investigate the impact of different parameters on it. According to our evaluation results in [17], the real workloads are smooth and there is no significant change in the prediction precision for different parameters settings. So, in this paper, the main focus of evaluation is on the synthetic workloads. We generate complicated synthetic and real workloads to evaluate the ability of RELENTING for online learning. Furthermore, the effect of the important parameters such as ω and Ω on RELENTING is considered for synthetic workloads. There are some parameters for RELENTING: $\delta, \Delta, \epsilon, \mu, Level, \theta, \omega$ and Ω. The parameters setting for the evaluation of RELENTING is as follows:

- Δ and δ: RELENTING is compared to the state-of-the-art predictors such as SMA (Simple Moving Average), AR, NN, HPA and LV (Last Value). Since these methods predict the status of resources in one certain time slot, we have to set $\delta = \Delta$ to provide the comparable results. The values of δ depend on the time spent on booting VMs.
- ϵ: As it was mentioned, ϵ should be determined based on the length of sampling intervals. Since the real workloads [40,41] are coarse-grained and the synthetic workloads are also generated in a similar way to them, we set $\epsilon = 0$ in all the experiments.
- μ: We evaluate the impact of μ on the training phase of POSITING for both real and synthetic workloads in [17]. According to the evaluation results, $\mu = 3$ is a good choice for synthetic and real workloads.
- θ: It is a threshold value that is used to extract the frequent episodes. We evaluate the impact of θ on the training phase of POSITING for both real and synthetic workloads in [17]. According to the evaluation results, for $\theta = 0.1$ there is a good trade-off between the processing time and the prediction precision.
- *Level*: To avoid enlarging the pattern tree, *Level* limits the length of episodes. However, a larger value of *Level* could lead to extract more useful episodes. So we choose the mediocre value 6 for it in all the experiments.
- ω: The NICE episodes learned in the off-line mode are selected for prediction based on ω. The small values of ω cause many episodes with low confidence or low consistency to

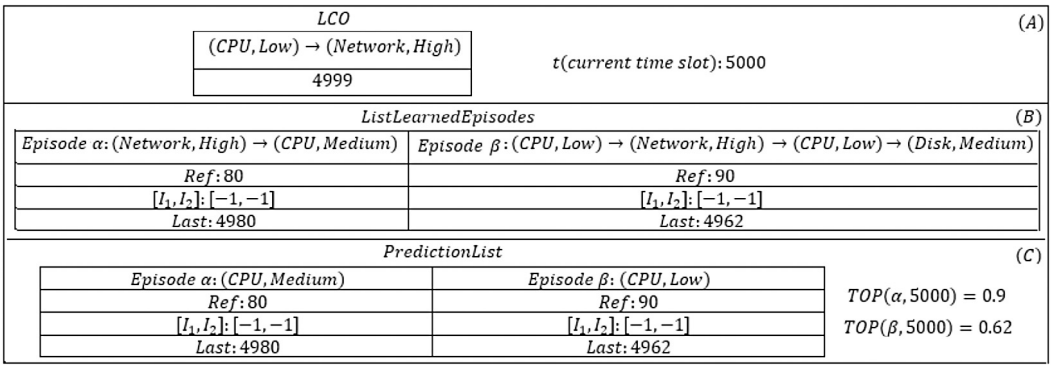


Fig. 9. An example to predict the future status based on the online learned episodes.

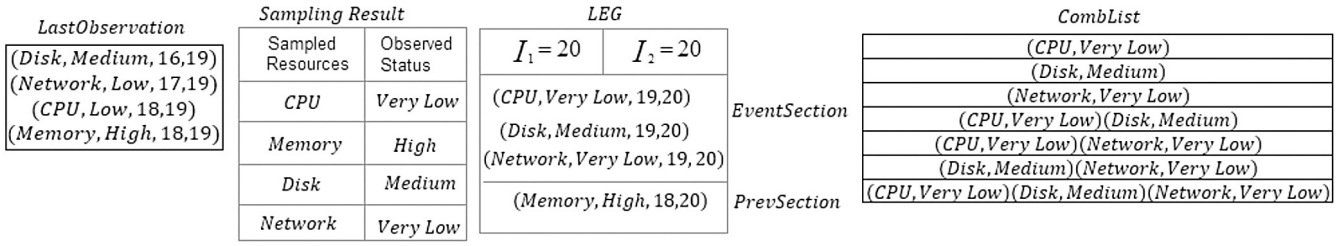


Fig. 10. An example to create *LEG* and extract all the combinations of the concurrent events.

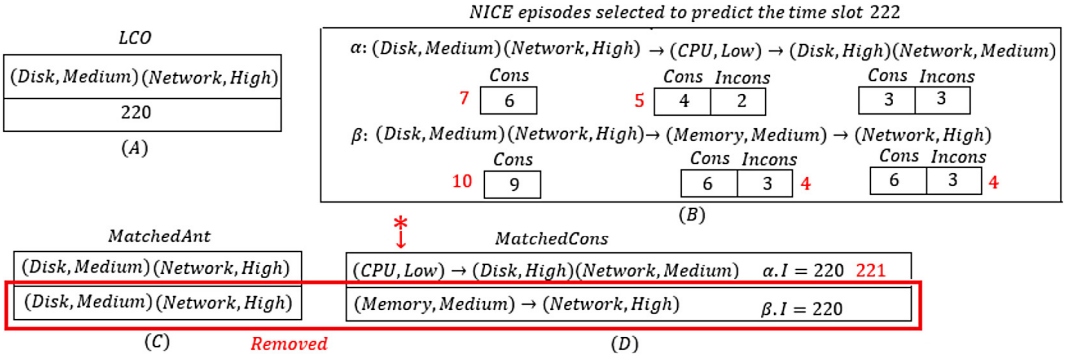


Fig. 11. Updating NICE episodes selected to predict the time slot 222.

be used for prediction. It could decrease the prediction precision. On the other hand, the large values of ω cause less NICE episodes to be used for prediction. So the application behaviour is learned frequently even if it has not changed significantly. Therefore, an appropriate value for ω could provide a good trade-off between the resources consumption of online learning and the prediction precision. The impact of ω on the number of online episodes is investigated on synthetic workloads (see Appendix B).

- Ω: The online learned episodes are selected for prediction based on Ω. The small values of Ω cause many online episodes with low credibility to be used for prediction. It could decrease the prediction precision. On the other hand, the large values of Ω cause less online episodes to be used for prediction and the future behaviour to be predicted based on *MRE* frequently when there is no NICE episode. Therefore, an appropriate value for Ω could provide more reliable results. The impact of Ω on the prediction precision is evaluated on synthetic workloads (see Appendix B).

According to the effective utilization reported for resources in some literature such as [42,43], we define the abstraction alphabet as Table 3 [17]. Note that this alphabet is not unique. In a similar way to [17], RELENTING is also compared with the state-of-the-art methods such as NN, SMA, LR, HPA and LV. These methods are based on the sliding window. If the current time slot is t and the length of the sliding window is $h > 0$, each entry $x_{ij}, 1 \leq i \leq N, t-1 \leq j \leq t-h$ of the window is the status of the resource R_i in the time slot j. Based on the sliding window, each method is implemented as follows:

Table 3
The abstraction alphabet for the abstraction representation [17].

Abstract value	Very low	Low	Medium	High
Range	[0, 20%)	[20%, 50%)	[50%, 80%)	[80%, 1]

- NN: In most of the literature such as [6,9,10], the typical three-layer neural network is used for prediction. The neurons of the input layer take the information of the sliding window as input variables and nodes of the output layer predict the future status of resources. So the number of the nodes of the input and the output layers is $h \times N$ and N respectively. The length of the sliding window (h) and

Table 4
The types of basic synthetic workloads and their embedded episodes.

Embedded episodes	Type of the synthetic workload
α : $(Memory, Low)(Disk, Verylow) \rightarrow (CPU, Low)(Network, High)$ β : $(CPU, Low)(Network, Low) \rightarrow (Memory, High)(Disk, Medium) \rightarrow$ $(CPU, High), (Network, Medium)$	$SWT1$ $SWT2$
α : $(Memory, Low)(Disk, Verylow) \rightarrow (CPU, Low)(Network, High)$ β : $(CPU, Low)(Network, Low) \rightarrow (Memory, High)(Disk, Medium) \rightarrow$ $(CPU, High), (Network, Medium)$	$SWT3$

the number of nodes in the hidden layer (*Nodes*) are two effective parameters of NNs that they should be selected carefully [17]. To select the parameters, for each pair of (h, *Nodes*), the average precision of 5 runs of NN is considered.

- SMA: In much literature such as [6,44], SMA is used as a naive predictor for evaluation. For the current time slot t and the resource R_i, $1 \leq i \leq N$, SMA predicts the next status of R_i based on the sliding window as $x_{it} = \frac{\sum_{j=1}^{h} x_{i(t-j)}}{h}$.
- LR: Yang et al. in [12] propose a linear regression model to predict the workload. According to the workload fluctuations in the sliding window, their method adjusts itself through the re-computation of parameters of the regression model. It means that this method has the ability of online learning. The authors show their method provides more reliable results than common regression based methods.
- HPA: Jiang et al. in [6] propose a hybrid approach for the future demand prediction of VMs and the capacity planning. They use several prediction models to predict the future workload. The results predicted by different methods are merged by a weighted linear combination strategy. The initial weights of predictors are equal. According to the prediction error of the methods, the weights are updated. Updating weights based on the prediction error implies that the method adjusts itself according to the workload variations. To implement this method, we use SMA, NN and LR as predictors.
- LV: the LV predictor is perhaps the simplest prediction method. The LV predictor assumes that the most recent observed behaviour is representative of the future application behaviour [45]. Indeed, *LV* is a type of *SMA* that h is set to 1.

Since the parameters have a significant effect on the prediction results of the predictors [5], each method is evaluated by using different values of its parameters and the best parameters are determined to provide a fair comparison. For SMA and AR, the length of the sliding window and for NN, the number of nodes in the hidden layer and the length of the sliding window are considered. Note that these methods receive the numerical time series and predict the future status of resources as numeral. To compare RELENTING with these methods, the final results predicted by the predictors are converted into the abstract values by using a simple mapping function [17].

Without loss of generality, we assume that each application is encapsulated inside one VM as it is reported in much literature such as [6,27,46]. In a similar way to the real workloads used to evaluate POSITING in [17], the trace of each VM is generated for one month. For each VM, the stream is constructed on the first 15 days of the month. The closed episodes are extracted from the stream by calling algorithm *AllEpisodes* [17]. In the last 15 days of the month, the future behaviour of the application is predicted by calling algorithm 3. Note that when the prediction is performed, counters *CorrectCount* and *PredictionCount* are updated based on the predicted results [17]. Finally, the final precision of POSITING is computed as follows:

$$Precision = \frac{CorrectCount}{PredictionCount}. \tag{5.1}$$

The time it takes to instantiate a new VM instance is about 5–15 min [47]. On the other hand, in a similar way to the real workloads used in [17], we assume VMs are sampled every 5 min. So three values 1,2 and 3 are evaluated for δ. Section 5.1 explains how the complicated synthetic and real workloads are generated. Section 5.2 compares the precision of RELENTING with the other methods' on the synthetic and real workloads. Finally, the ability of RELENTING to learn the new behaviour of the application is considered in Section 5.3.

5.1. Complicated real and synthetic workloads

In this section we explain how the complicated workloads are generated from the real and synthetic workloads.

The complicated real workloads: Since the workloads of the data set GWA-T-12[2] Bitbrains are more dynamic than the other public workloads [48], in a similar way to [17], we use these workloads to generate the complicated real workloads. The complicated real workloads are a combination of the different real workloads. To generate a complicated real workload, three real workloads are selected randomly, two cut-points are randomly chosen in the interval of [2500, 3100] and [4200, 6500] respectively and the real workloads merge together based on the cut-points. For each value of δ, we generate three complicated real workloads.

The complicated synthetic workloads: In a similar way to the data set GWA-T-12 Bitbrains, the synthetic workloads are generated for one month with the sampling intervals of 5 min. So there are 8640 time slots for each generated trace. The synthetic workload generator used in this paper is the same as the workload generator employed in [17]. Table 4 shows the types of the generated basic synthetic workloads and their corresponding embedded episodes. In [17], we define σ as the gap between each two consecutive *CNG*s and perform the comprehensive experiments on the different values of σ. Since the main focus of RELENTING is on online learning, σ is set to 1 in all the synthetic workloads. We generate complicated workloads, which are a combination of the different types of the basic workloads. To generate the complicated workloads, the basic workloads are generated, cut-points are randomly selected in the interval of [4321, 7000] because we want to simulate a situation that the application behaviour changes after the episode extraction, and the basic workloads merge together based on the cut-points. Table 5 describes the complicated synthetic workloads. For example, $VMComb1$ is composed of the workload type $SWT1$ until the time slot 4360 and $SWT2$ after that. As the table shows, eight complicated workloads are generated by using the basic workloads. For each type of workloads, three experiments with different values of δ are conducted.

5.2. Experimental results

In this section, the precision of RELENTING on the complicated workloads is compared to the other methods'. According to the

[2] These traces can be accessed at http://gwa.ewi.tudelft.nl/datasets/Bitbrains.

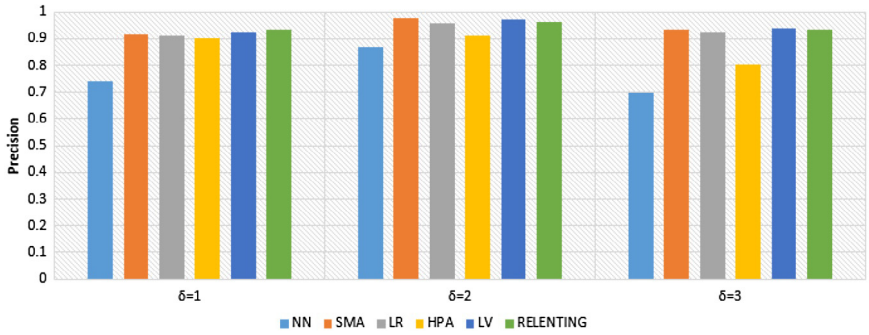


Fig. 12. The average precision of RELENTING and the other methods for $\delta = 1, 2, 3$ on the complicated real workloads.

Table 5
The complicated synthetic workloads generated to evaluate RELENTING.

Name	Cut-point	Description
*VMComb*1	4360	$SWT1 \rightarrow SWT2$
*VMComb*2	4580	$SWT1 \rightarrow SWT3$
*VMComb*3	5614	$SWT2 \rightarrow SWT1$
*VMComb*4	5398	$SWT2 \rightarrow SWT3$
*VMComb*5	6312	$SWT3 \rightarrow SWT1$
*VMComb*6	6097	$SWT3 \rightarrow SWT2$
*VMComb*7	5000,6800	$SWT1 \rightarrow SWT3 \rightarrow SWT2$
*VMComb*8	5200,6700	$SWT2 \rightarrow SWT3 \rightarrow SWT1$

experiment results reported in [17] and Appendix B, we set $\theta = 0.1$, $\mu = 3$, $\omega = 0.8$ and $\Omega = 0.8$ in all the experiments.

The complicated real workloads: For each value of δ, we generate three complicated real workloads and report only the average precision due to space limitation. Fig. 12 shows the average precision of RELENTING, NN, SMA, LR, HPA and LV on the predicted results of the complicated real workloads for different values of δ. As it is shown in the figure, the precision of all the methods is more than 0.7. RELENTING, SMA, LR and LV provide the most accurate results. As the results show, since the real workloads are smooth [17], the complicated real workloads are also almost smooth. So the simple predictors such as SMA, LR and LV could also provide the reliable results. NN has the smallest precision compared to the other methods. It also causes the precision of HPA to decrease. It implies that NN should be retrained to adapt to the workload changes even if the changes are soft.

*VMComb*1: This trace is composed of two different workload types $SWT1$ and $SWT2$. According to Table 5, the behavioural pattern of this trace changes after the time slot 4360 completely. Fig. 13 shows the prediction precision of RELENTING and the other methods. Since data used to train NN does not cover the application behaviour after the training phase, NN could not provide reliable results. LR, LV, SMA and HPA provide more reliable results than NN. As it is observed, RELENTING provides the most accurate results for this sudden change of behavioural patterns.

*VMComb*2: This trace is composed of two different workload types $SWT1$ and $SWT3$. According to Table 5, a new behavioural pattern occurs after the time slot 4580. Fig. 14 shows the prediction precision of RELENTING and the other methods. Unlike *VMComb*1, the application behaviour after the training phase doest not change completely. So NN provides more reliable results for *VMComb*2 compared to *VMComb*1. Like *VMComb*1, LR, LV, SMA and HPA provide better results compared to NN. The precision of RELENTING is better than the other methods'.

*VMComb*3: This trace is composed of two different workload types $SWT2$ and $SWT1$. According to Table 5, the behavioural pattern β changes to the pattern α after the time slot 5614. Fig. 15 shows the prediction precision of RELENTING and the other methods. Since the behavioural pattern of *VMComb*3 in the training phase changes to a simpler pattern, NN provides better results compared to *VMComb*1. The other methods provide the similar results and RELENTING outperforms them.

*VMComb*4: This trace is composed of two different workload types $SWT2$ and $SWT3$. According to Table 5, in addition to the behavioural pattern β, the pattern α occurs after the time slot 5398. Fig. 16 compares the prediction precision of RELENTING with the other methods'. Compared to *VMComb*3, in this trace, the behavioural pattern does not change completely. So NN provides better results for *VMComb*4 compared to *VMComb*3. Furthermore, the figure implies the predictors such as SMA, LV and LR, which are based on the recent behaviour of the application, do not always provide more precise results than NN. As the figure shows, RELENTING provides the most precise results for different values of δ.

*VMComb*5: According to Table 5, in this trace, the behavioural pattern β vanishes after the time slot 6312 and only the episode α occurs. Fig. 17 shows the prediction precision of all the predictors. Due to the smoothness of the workload in the test phase, the predictors SMA, LR, LV and HPA provide more precise results than NN. Since the application behaviour in the training phase is different from the test phase's, NN could not provide reliable results. According to the figure, RELENTING outperforms all the predictors.

*VMComb*6: This trace is composed of two different workload types $SWT3$ and $SWT2$. As Table 5 shows, in this trace, the behavioural pattern α vanishes after the time slot 6097 and only the episode β occurs. Fig. 18 shows the precision of RELENTING and the other predictors. Since the workload variations of this trace are more than *VMComb*5's , the prediction precision of SMA, LR, LV and HPA is less for this trace compared to *VMComb*5. Therefore, for this trace, due to the similarity between the application behaviour in the test and the training phases, NN provides more precise results than SMA, LV, HPA and LR. RELENTING also outperforms all the predictors.

*VMComb*7: This trace is composed of three workload types $SWT1$, $SWT3$ and $SWT2$. As Table 5 shows, in addition to α, the behavioural pattern β occurs after the time slot 5000. After the time slot 6800, the episode α vanishes and only the episode β occurs. Fig. 19 shows the prediction precision of the predictors. Since the application behaviour in a short period of the test phase is completely different from the application behaviour in the training phase, NN provides the most precise results after RELENTING. Due to high workload dynamics in the test phase, the predictors SMA, LR, LV and HPA could not provide reliable results.

*VMComb*8: This trace is composed of three workload types $SWT2$, $SWT3$ and $SWT1$. According to Table 5, in addition to β, the behavioural pattern α occurs after the time slot 5200. After the time slot 6700, the episode β vanishes and only the episode α occurs. Fig. 20 compares the prediction precision of the predictors. Since the workload fluctuations in the test phase of this trace

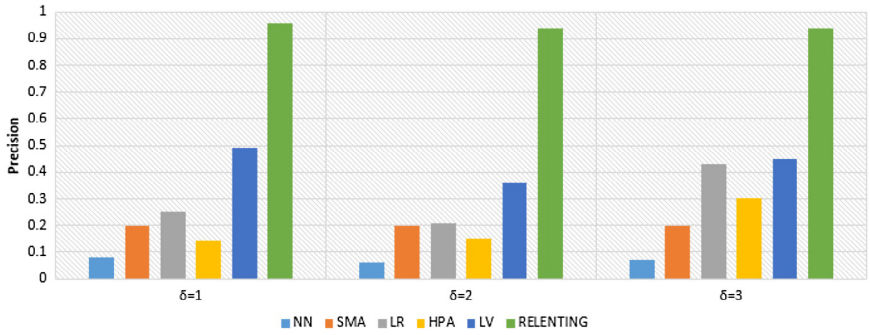


Fig. 13. The prediction precision of RELENTING and the other methods for $\delta = 1, 2, 3$ on *VMComb*1.

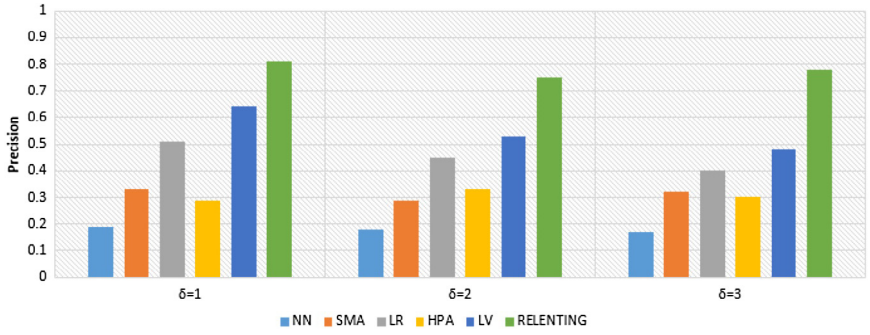


Fig. 14. The prediction precision of RELENTING and the other methods for $\delta = 1, 2, 3$ on *VMComb*2.

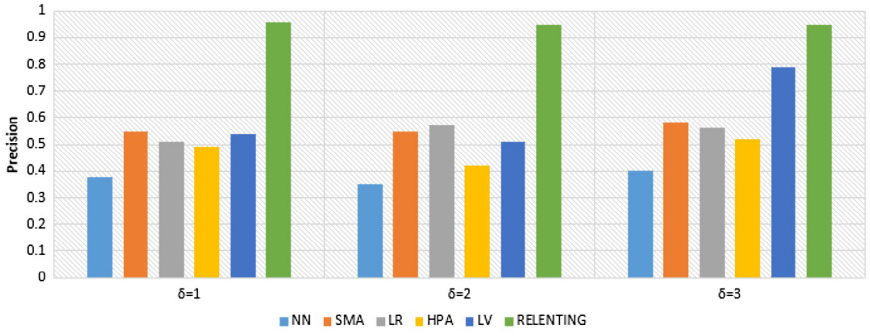


Fig. 15. The prediction precision of RELENTING and the other methods for $\delta = 1, 2, 3$ on *VMComb*3.

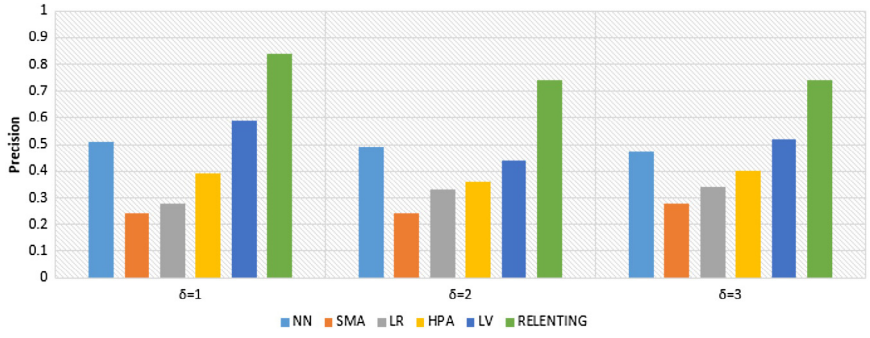


Fig. 16. The prediction precision of RELENTING and the other methods for $\delta = 1, 2, 3$ on *VMComb*4.

are less than *VMComb*7's , the predictors SMA, LR, LV and HPA provide more reliable results. NN provides the same precise results as *VMComb*7's. Furthermore, RELENTING outperforms the other predictors.

The Results Summary: According to the experiment results, we summarize the following points about the predictors:

- NN is a discriminative learning approach. So it needs much training data [49]. As the experiment results show, if data

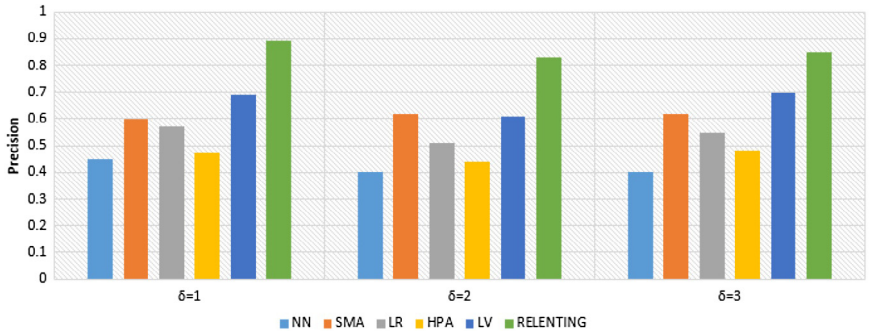


Fig. 17. The prediction precision of RELENTING and the other methods for $\delta = 1, 2, 3$ on *VMComb*5.

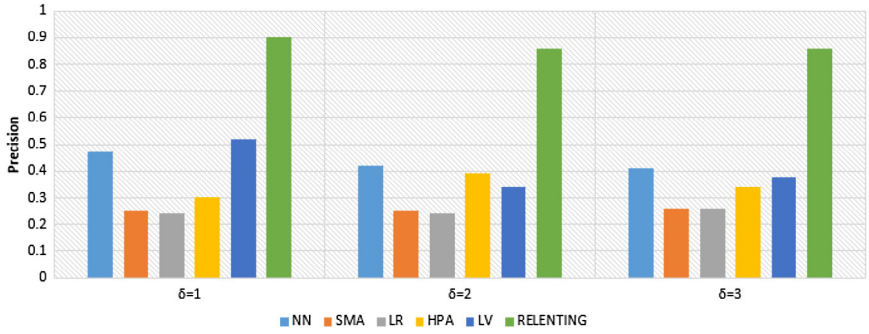


Fig. 18. The prediction precision of RELENTING and the other methods for $\delta = 1, 2, 3$ on *VMComb*6.

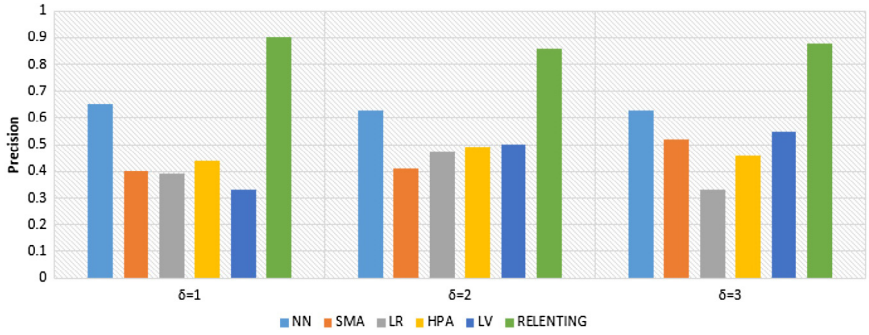


Fig. 19. The prediction precision of RELENTING and the other methods for $\delta = 1, 2, 3$ on *VMComb*7.

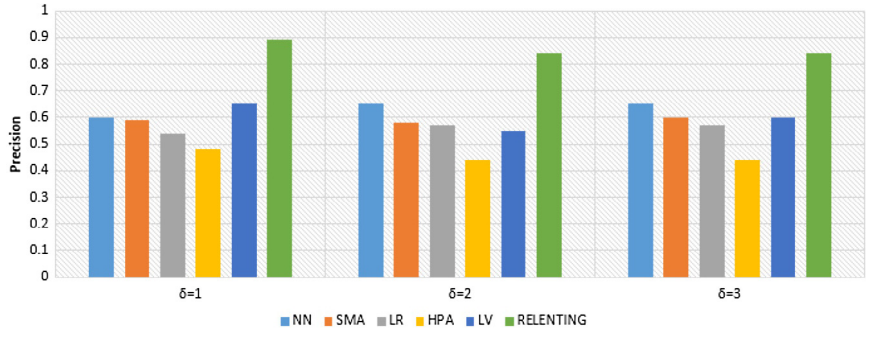


Fig. 20. The prediction precision of RELENTING and the other methods for $\delta = 1, 2, 3$ on *VMComb*8.

used to train NN does not cover the application behaviour after the training phase, NN could not provide the reliable results. To adapt to the workload changes, NN should be retrained using new data. For this purpose, sufficient training data should be gathered, which might be time consuming. Furthermore, the delays of the parameters setting

for NN and its learning phase might lead to drastic loss in cloud.

- SMA and LV do not consider the application behaviour in the past and only focus on the recent behaviour of the application. According to the experiment results, SMA and LV predict the complicated real workloads accurately. However, they do not provide reliable results for *VMComb*1, *VMComb*6 and *VMComb*7. All of these synthetic traces are composed of *SWT*2 in their test phase. Since the episode β leads to generate more dynamic workloads compared to the episode α, it is evidence that these predictors are not appropriate for dynamic workloads.
- LR adjusts itself through the re-computation of the parameters of the regression model according to the workload fluctuations in the sliding window. Although LR has the ability of learning, its reliance is based on the oversimplified assumptions of the workload (the linear relationship). Furthermore, LR considers only the workload fluctuations in the sliding window and ignores the correlation between different resources. According to the experiment results, although LR predicts the smooth workloads (complicated real workloads) reliably, it cannot capture the behavioural changes of dynamic workloads (complicated synthetic workloads) very well.
- HPA Updates its weights based on the prediction error. It implies that HPA adjusts itself according to the workload variations. Since HPA is a hybrid approach, a weak predictor could lead to unreliable results. According to the experiment results, in the best case, the prediction results of HPA are the same precise as its best predictor's.
- In addition to the advantages of POSITING such as extracting the behavioural patterns of workloads independently of the fixed pattern length explicitly and considering the correlation between different resources, RELENTING tackles the inability of POSITING to adapt to the behavioural changes. As the experiment results show, RELENTING predicts the future behaviour of the resources reliably for both of the smooth workloads (complicated real workloads) and the dynamic workloads (complicated synthetic workloads). It learns the new behavioural patterns and updates the closed episodes extracted in the off-line mode.

5.3. The ability of RELENTING for online learning

To investigate the ability of RELENTING to learn the new behaviour of the application, the closed episodes describing the new behaviour are extracted by calling the algorithm *AllEpisodes* in [17] and compared with the episodes learned by RELENTING. Note that the algorithm *AllEpisodes* extracts the closed episodes and RELENTING learns both of the offline closed episodes and the online episodes. We introduce a criterion, called **Covering Percentage** (*CP*), to measure how much the episodes learned by RELENTING cover the new behaviour of the application. A higher value of *CP* indicates the more ability of RELENTING to learn the behavioural patterns of the application.

Definition 17. Given the pattern base *PB* and the closed episode $\alpha = G'_1 \rightarrow \cdots \rightarrow G'_k$, if the continuous sub-episode $G'_i \rightarrow G'_{i+1} \rightarrow \cdots \rightarrow G'_j$, $1 \leq i < j \leq k$, is an episode of *PB*, then each G'_t, $i \leq t \leq j$, is absorbed by *PB*. *Absorb*(α, *PB*) implies the number of *CNG*s of α that they are absorbed by *PB*.

Definition 18. If *ClosedEpisodes* is a set of closed episodes extracted from the test part of the trace, $|ClosedEpisodes| = P$ and *OfflineOnlineEpisodes* is a set of the offline and the online episodes learned by RELENTING, then *CP* is defined as follows:

$$CP = \frac{\sum_{i=1}^{P} \frac{Absorb(ClosedEpisodes[i], OfflineOnlineEpisodes)}{|CNG_{ClosedEpisodes[i]}|}}{P}. \quad (5.2)$$

Table 6 shows the ability of RELENTING to learn the new behaviour of the synthetic workloads (*CP*), the number of the closed episodes extracted from the test part of the traces ($|ClosedEpisodes|$), the number of the episodes extracted in the online mode ($|OnlineEpisodes|$) and the number of the closed episodes extracted in the offline mode ($|OfflineEpisodes|$). As it is observed, for all the synthetic traces and all the values of δ, RELENTING is able to learn the application behaviour very well. Since RELENTING extracts the closed episodes in the offline mode and the episodes in the online mode, it is clear that $|ClosedEpisodes|$ is much fewer than $|OnlineEpisodes| + |OfflineEpisodes|$. Furthermore, *ClosedEpisodes* are extracted for $\theta = 0.1$ meanwhile RELENTING extracts *OnlineEpisodes* independently of their frequency. Therefore, it might extract episodes with low frequencies.

According to the experiment results, RELENTING provides the reliable prediction results and learns the new behavioural patterns in reasonable time and space complexities. So, an efficient resources management plan could fulfil SLA and prevent the resources wasting [5]. Thus, the energy consumption and the operational cost decrease. It is clear that the reduction of the energy consumption leads to decrease carbon emissions, which could facilitate green cloud computing [50]. Furthermore, based on the workload prediction and SLA, an energy optimization technique could balance the energy efficiency and performance requirements [51].

6. Conclusion and future work

The future demand prediction is an indispensable step for the rapid elasticity implementation and the effective resource provisioning in cloud. Most of the prevalent predictors such as NN and LR cannot extract all the useful patterns whose length is less/more than the fixed length. Our previous predictor, POSITING, considers the correlation between different resources and extracts behavioural patterns of applications independently of the fixed pattern length explicitly. Although the cloud environment is dynamic and is changing continuously, POSITING and the other predictors are not able to adapt to the workload changes. To tackle this issue, we develop POSITING and propose RELENTING, which learns the behavioural changes of the application and improves its knowledge about the application behaviour. Thus, in addition to the advantages of POSITING such as accurate and interpretable results, RELENTING fulfils adaptability. To model online learning, RELENTING compares the predicted results with the observed behaviour and updates the criterion *Consistency* of episodes extracted in the off-line mode. If there is no NICE episode for prediction, based on the recent and the current observed behaviour of the application, the new behavioural patterns are identified. RELENTING employs both of the patterns learned in the online and the off-line modes for prediction. The experiment results show that RELENTING outperforms the state-of-the-art predictors and learns the new behavioural patterns very well.

Although RELENTING identifies the new behavioural patterns, it does not provide the precise information about episodes. For example, RELENTING only determines the number of references to episodes, not their NO frequency. Thus, recognizing the closed episodes in the online mode is not possible. We plan to improve RELENTING in a way that it could provide more precise information about the episodes. Furthermore, some criteria could be defined to identify and prune the inefficient online episodes. Thus, time and space complexities of RELENTING are improved. RELENTING could also be customized for different fields due to its ability for

Table 6
The ability of RELENTING to learn the new behaviour of synthetic workloads for different values of δ.

δ	*Trace*	|*ClosedEpisodes*|	|*OnlineEpisodes*|	|*OfflineEpisodes*|	*CP*
1	*VMComb*1	462	2421	363	1
	*VMComb*2	636	5247	637	1
	*VMComb*3	789	4201	979	1
	*VMComb*4	195	2539	462	0.996
	*VMComb*5	485	3234	330	1
	*VMComb*6	570	2913	523	1
	*VMComb*7	126	4267	637	1
	*VMComb*8	340	2788	979	1
2	*VMComb*1	397	3559	523	1
	*VMComb*2	539	5728	460	1
	*VMComb*3	741	5022	1488	1
	*VMComb*4	178	3564	408	0.966
	*VMComb*5	351	4215	331	1
	*VMComb*6	545	4426	430	1
	*VMComb*7	131	4957	460	1
	*VMComb*8	337	3802	1488	1
3	*VMComb*1	416	5258	294	1
	*VMComb*2	656	7566	442	1
	*VMComb*3	1620	8289	821	0.999
	*VMComb*4	284	6209	452	0.933
	*VMComb*5	447	7258	447	1
	*VMComb*6	620	6779	691	1
	*VMComb*7	921	7115	442	1
	*VMComb*8	450	7590	821	1

online leaning and extracting the patterns explicitly. For example, in [52], for the deployment of replica servers in virtual content distribution networks, the correlations between the node pairs could be extracted explicitly. Furthermore, the dynamic placement problem for replica servers could be considered based on online learning.

Acknowledgment

The GWA-T-12 Bitbrains traces are provided by Bitbrains IT Services Inc., which is a service provider that specializes in managed hosting and business computation for enterprises. We thank the GWA team and all those who have graciously provided the data for us.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.future.2018.04.044.

References

[1] D. Petcu, J.L. Vzquez-Poletti, European Research Activities in Cloud Computing, Cambridge Scholars Publishing, 2012, p. 350. ISBN 1443835072, 9781443835077.

[2] B. Gupta, D.P. Agrawal, S. Yamaguchi, Handbook of Research on Modern Cryptographic Solutions for Computer and Cyber Security, first ed., IGI Global, Hershey, PA, USA, 2016. ISBN 1522501053, 9781522501053.

[3] M. Al-Ayyoub, S. AlZu'bi, Y. Jararweh, M.A. Shehab, B.B. Gupta, Accelerating 3d medical volume segmentation using gpus, Multimedia Tools Appl. (ISSN: 1573-7721) 77 (4) (2018) 4939–4958. http://dx.doi.org/10.1007/s11042-016-4218-0.

[4] N.R. Herbst, S. Kounev, R. Reussner, Elasticity in cloud computing: what it is, and what it is not, in: The 10th International Conference on Autonomic Computing ICAC 2013, San Jose, CA, USA, 2013.

[5] M. Amiri, L. Mohammad-Khanli, Survey on prediction models of applications for resources provisioning in cloud, J. Netw. Comput. Appl. (ISSN: 1084-8045) 82 (2017) 93–113. http://dx.doi.org/10.1016/j.jnca.2017.01.016.

[6] Y. Jiang, C.-S. Perng, T. Li, R.N. Chang, Cloud analytics for capacity planning and instant VM provisioning, IEEE Trans. Network Serv. Manage. 10 (3) (2013) 312–325.

[7] V. Chang, The business intelligence as a service in the cloud, Future Gener. Comput. Syst. (ISSN: 0167-739X) 37 (2014) 512–534. http://dx.doi.org/10.1016/j.future.2013.12.028.

[8] G. Sun, D. Liao, D. Zhao, Z. Sun, V. Chang, Towards provisioning hybrid virtual networks in federated cloud data centers, Future Gener. Comput. Syst. (ISSN: 0167-739X) (2017). http://dx.doi.org/10.1016/j.future.2017.09.065.

[9] K. Cetinski, M.B. Juric, AME-WPC: Advanced model for efficient workload prediction in the cloud, J. Netw. Comput. Appl. 55 (2015) 191–201. http://dx.doi.org/10.1016/j.jnca.2015.06.001.

[10] M. Amiri, M.R. Feizi-Derakhshi, L. Mohammad-Khanli, IDS fitted Q improvement using fuzzy approach for resource provisioning in cloud, J. Intell. Fuzzy Syst. 32 (1) (2017) 229–240. http://dx.doi.org/10.3233/JIFS-151445.

[11] P. Altevogt, W. Denzel, T. Kiss, Cloud modeling and simulation, in: Encyclopedia of Cloud Computing, Wiley-IEEE Press, ISBN: 978-1-118-82197-8, 2016.

[12] J. Yang, C. Liu, Y. Shang, B. Cheng, Z. Mao, C. Liu, L. Niu, J. Chen, A cost-aware auto-scaling approach using the workload prediction in service clouds, Inf. Syst. Front. (ISSN: 1387-3326) 16 (1) (2014) 7–18. http://dx.doi.org/10.1007/s10796-013-9459-0.

[13] P. Shi, H. Wang, G. Yin, F. Lu, T. Wang, Prediction-based federated management of multi-scale resources in cloud, Adv. Inf. Sci. Serv. Sci. 4 (6) (2012) 324–334.

[14] A. Matsunaga, J.A.B. Fortes, On the use of machine learning to predict the time and resources consumed by applications, in: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE Computer Society, Melbourne, Victoria, Australia, 2010, pp. 495–504. http://dx.doi.org/10.1109/CCGRID.2010.98.

[15] Z. Chen, Y. Zhu, Y. Di, S. Feng, Self-adaptive prediction of cloud resource demands using ensemble model and subtractive-fuzzy clustering based fuzzy neural network, Comput. Intell. Neurosci. (ISSN: 1687-5265) 2015 (2015). http://dx.doi.org/10.1155/2015/919805.

[16] S. Di, D. Kondo, W. Cirne, Google hostload prediction based on Bayesian model with optimized feature combination, J. Parallel Distrib. Comput. (ISSN: 0743-7315) 74 (1) (2014) 1820–1832. http://dx.doi.org/10.1016/j.jpdc.2013.10.001.

[17] M. Amiri, L. Mohammad-Khanli, R. Mirandola, A sequential pattern mining model for application workload prediction in cloud environment, J. Netw. Comput. Appl. (ISSN: 1084-8045) 105 (2018) 21–62. http://dx.doi.org/10.1016/j.jnca.2017.12.015.

[18] X. Liu, X. Zhu, S. Singhal, M. Arlitt, Adaptive entitlement control of resource containers on shared servers, in: 2005 9th IFIP/IEEE International Symposium on Integrated Network Management, 2005. IM 2005, Nice, France, 2005, pp. 163–176, http://dx.doi.org/10.1109/INM.2005.1440783.

[19] R. Nathuji, A. Kansal, A. Ghaffarkhah, Q-clouds: managing performance interference effects for qos-aware clouds, in: Proceedings of the 5th European Conference on Computer Systems, in: EuroSys '10, ACM, Paris, France, 2010, pp. 237–250. http://dx.doi.org/10.1145/1755913.1755938.

[20] H. Wu, W. Zhang, J. Zhang, J. Wei, T. Huang, A benefit-aware on-demand provisioning approach for multi-tier applications in cloud computing, Front. Comput. Sci. (ISSN: 2095-2228) 7 (4) (2013) 459–474. http://dx.doi.org/10.1007/s11704-013-2201-8.

[21] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, K. Shin, What does control theory bring to systems research?, SIGOPS - Oper. Syst. Rev. (ISSN: 0163-5980) 43 (1) (2009) 62–69. http://dx.doi.org/10.1145/1496909.1496922.

[22] J. Cao, W. Zhang, W. Tan, Dynamic control of data streaming and processing in a virtualized environment, IEEE Trans. Autom. Sci. Eng. 9 (2) (2012) 365–376.

[23] P. Lama, X. Zhou, Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee, in: Proceedings of IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), San Francisco, California, US, 2013.

[24] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, T. Wood, Agile dynamic provisioning of multi-tier Internet applications, ACM Trans. Auton. Adapt. Syst. (ISSN: 1556-4665) 3 (1) (2008) 1–39. http://dx.doi.org/10.1145/1342171.1342172.

[25] M.N. Bennani, D.A. Menasce, Resource allocation for autonomic data centers using analytic performance models, in: Proceedings of the Second International Conference on Automatic Computing, in: ICAC '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 229–240. http://dx.doi.org/10.1109/icac.2005.50.

[26] Q. Zhang, L. Cherkasova, E. Smirni, A regression-based analytic model for dynamic resource provisioning of multi-tier applications, in: Proceedings of the Fourth International Conference on Autonomic Computing, ICAC '07, pp. 27–27, Jacksonville, Florida, USA, 2007. http://dx.doi.org/10.1109/ICAC.2007.1.

[27] S.K. Garg, A.N. Toosi, S.K. Gopalaiyengar, R. Buyya, SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter, J. Netw. Comput. Appl. (ISSN: 1084-8045) 45 (2014) 108–120. http://dx.doi.org/10.1016/j.jnca.2014.07.030.

[28] I. Batal, G.F. Cooper, D. Fradkin, Jr., J. Harrison, F. Moerchen, M. Hauskrecht, An efficient pattern mining approach for event detection in multivariate temporal data, Knowl. Inf. Syst. (ISSN: 0219-1377) 46 (1) (2016) 115–150. http://dx.doi.org/10.1007/s10115-015-0819-6.

[29] H. Mannila, H. Toivonen, A. Inkeri Verkamo, Discovery of frequent episodes in event sequences, Data Min. Knowl. Discov. (ISSN: 1573-756X) 1 (3) (1997) 259–289. http://dx.doi.org/10.1023/A:1009748302351.

[30] S. Laxman, P.S. Sastry, K.P. Unnikrishnan, Discovering frequent episodes and learning hidden markov models: A formal connection, IEEE Trans. Knowl. Data Eng. (ISSN: 1041-4347) 17 (11) (2005) 1505–1517. http://dx.doi.org/10.1109/TKDE.2005.181.

[31] K. Hwang, X. Bai, Y. Li, M. Shi, W.G. Chen, Y. Wu, Cloud performance modeling and benchmark evaluation of elastic scaling strategies, IEEE Trans. Parallel Distrib. Syst. 27 (1) (2016) 130–143. http://dx.doi.org/10.1109/TPDS.2015.2398438.

[32] S. Gupta, B.B. Gupta, Xss-secure as a service for the platforms of online social network based multimedia web applications in cloud, Multimedia Tools Appl. (ISSN: 1573-7721) 77 (4) (2018) 4829–4861. http://dx.doi.org/10.1007/s11042-016-3735-1.

[33] X. Yan, J. Han, R. Afshar, CloSpan: Mining: Closed sequential patterns in large datasets, in: Proceedings of the 2003 SIAM International Conference on Data Mining, San Francisco, CA, USA, pp. 166–177, 2003. http://dx.doi.org/10.1137/1.9781611972733.15.

[34] N. Tatti, B. Cule, Mining closed strict episodes, Data Min. Knowl. Discov. (ISSN: 1384-5810) 25 (1) (2012) 34–66. http://dx.doi.org/10.1007/s10618-11-0232-z.

[35] X. Ao, P. Luo, C. Li, F. Zhuang, Q. He, Online frequent episode mining, in: 2015 IEEE 31st International Conference on Data Engineering, pp. 891–902, 2015, http://dx.doi.org/10.1109/ICDE.2015.7113342.

[36] M. Ibtihal, E.O. Driss, N. Hassan, Homomorphic encryption as a service for outsourced images in mobile cloud computing environment, Internat. J. Cloud Appl. Comput. 7 (2) (2017) 27–40. http://dx.doi.org/10.4018/IJCAC.2017040103.

[37] M.A. Alsmirat, Y. Jararweh, I. Obaidat, B.B. Gupta, Internet of surveillance: a cloud supported large-scale wireless surveillance system, J. Supercomput. (ISSN: 1573-0484) 73 (3) (2017) 973–992. http://dx.doi.org/10.1007/s11227-16-1857-x.

[38] D. Chiras, Human Biology, Jones & Bartlett Learning, ISBN: 9780763783457, 2011.

[39] R. Weingartner, G.B. Brascher, C.B. Westphall, Cloud resource management: A survey on forecasting and profiling models, J. Netw. Comput. Appl. 47 (2015) 99–106.

[40] M. Alam, K.A. Shakil, S. Sethi, Analysis and clustering of workload in google cluster trace based on resource usage, CoRR, abs/1501.01426, 2015.

[41] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, D.H. Epema, The grid workloads archive, Future Gener. Comput. Syst. (ISSN: 0167-739X) 24 (7) (2008) 672–686. http://dx.doi.org/10.1016/j.future.2008.02.003.

[42] S. Xi, C. Li, C. Lu, C.D. Gill, M. Xu, L.T.X. Phan, I. Lee, O. Sokolsky, RT-open stack: Cpu resource management for real-time cloud computing, in: 2015 IEEE 8th International Conference on Cloud Computing, pp. 179–186, 2015. http://dx.doi.org/10.1109/CLOUD.2015.33.

[43] Utilization. http://dx.doi.org/h17007.www1.hpe.com/device_help/HPJ3298A/utilization.htm (Accessed: 06-06-17).

[44] C. Vazquez, R. Krishnan, E. John, Time series forecasting of cloud data center workloads for dynamic resource provisioning, J. Wirel. Mobile Netw. Ubiquitous Comput. Dep. Appl. (JoWUA) 6 (3) (2015) 87–110.

[45] R. Sarikaya, C. Isci, A. Buyuktosunoglu, Runtime application behavior prediction using a statistical metric model, IEEE Trans. Comput. (ISSN: 0018-9340) 62 (3) (2013) 575–588. http://dx.doi.org/10.1109/TC.2012.25.

[46] J. Jheng, F. Tseng, H. Chao, L. Chou, A novel vm workload prediction using grey forecasting model in cloud data center, in: The International Conference on Information Networking 2014 (ICOIN2014), 2014, pp. 40–45, http://dx.doi.org/10.1109/ICOIN.2014.6799662.

[47] A. Li, X. Yang, S. Kandula, M. Zhang, CloudCmp: Comparing public cloud providers, in: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, in: IMC '10, ACM, Melbourne, Australia, ISBN: 978-1-4503-0483-2, 2010, pp. 1–14. http://dx.doi.org/10.1145/1879141.1879143.

[48] S. Shen, V. v. Beek, A. Iosup, Statistical characterization of business-critical workloads hosted in cloud datacenters, in: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2015, pp. 465–474, http: //dx.doi.org/10.1109/CCGrid.2015.60.

[49] B. Liu, G.I. Webb, Generative and discriminative learning, in: C. Sammut, G.I. Webb (Eds.), Encyclopedia of Machine Learning, Springer US, Boston, MA, ISBN: 978-0-387-30164-8, 2010, pp. 454–455. http://dx.doi.org/10.1007/978-1-387-30164-8_332.

[50] J. Shuja, R.W. Ahmad, A. Gani, A.I. Abdalla Ahmed, A. Siddiqa, K. Nisar, S.U. Khan, A.Y. Zomaya, Greening emerging IT technologies: techniques and practices, J. Internet Serv. Appl. (ISSN: 1869-0238) 8 (1) (2017) 9. http://dx.doi.org/10.1186/s13174-017-0060-5.

[51] J. Shuja, K. Bilal, S.A. Madani, M. Othman, R. Ranjan, P. Balaji, S.U. Khan, Survey of techniques and architectures for designing energy-efficient data centers, IEEE Syst. J. (ISSN: 1932-8184) 10 (2) (2016) 507–519. http://dx.doi.org/10.1109/JSYST.2014.2315823.

[52] G. Sun, V. Chang, G. Yang, D. Liao, The cost-efficient deployment of replica servers in virtual content distribution networks for data fusion, Inform. Sci. (ISSN: 0020-0255) 432 (2018) 495–515. http://dx.doi.org/10.1016/j.ins.2017.08.021.