A sequential pattern mining model for application workload prediction in cloud environment

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

[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 Agreements (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. For this purpose, the future demand of applications should be determined. Thus, the prediction of the future workload of applications is an essential step before the resource provisioning. To the best of our knowledge, for the first time, this paper proposes a novel Prediction mOdel based on SequentIal paTtern mINinG (POSITING) that considers correlation between different resources and extracts behavioural patterns of applications independently of the fixed pattern length explicitly. Based on the extracted patterns and the recent behaviour of the application, the future demand of resources is predicted. The main goal of this paper is to show that models based on pattern mining could offer novel and useful points of view for tackling some of the issues involved in predicting the application workloads. The performance of the proposed model is evaluated based on both real and synthetic workloads. The experimental results show that the proposed model could improve the prediction accuracy in comparison to the other state-of-the-art methods such as moving average, linear regression, neural networks and hybrid prediction approaches.

1. Introduction

Cloud computing is a computing paradigm based on a pay-as-you-go model that handles the growing needs for computation and stor-age in an efficient and cost-effective manner (Ghorbani et al., 2014; Coutinho et al., 2015). For this purpose, cloud should allocate a suit-able amount of resources according to the current demand of applica-tions. Under-provisioning causes Service Level Agreements (SLA) vio-lation, Quality of Service (QoS) dropping and the customer dissatisfac-tion. 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 (Amiri and Mohammad-Khanli, 2017). Furthermore, the speed of response to the workload changes to achieve the desired performance level is a critical issue for cloud elasticity (Coutinho et al., 2015). Although the important advantage of elasticity is to match the amount of resources allocated to the application with the amount of resources it requires, the time that resources take to be ready to use is a potential problem (Galante and Bona, 2012). Cloud elasticity and dynamic resources allo-cation are based on the virtualization techniques (Hwang et al., 2016). The VM provisioning technologies take several minutes (Jiang et al., 2013). This delay is intolerable for the tasks that need the resources scaling during the computation. It might lead to SLA violation, QoS dropping and finally a reputation loss of cloud. To reduce the delay, there are three approaches. The first approach, VM provisioning tech-nologies, assists to ready new VMs in seconds for the requests (Jiang et al., 2013). The state of the art VM provisioning technologies, such as streaming VM technology (Labonte et al., 2004) and VM cloning (Lagar-Cavilla et al., 2009) cannot decrease time wasting of VM cre-ation (Jiang et al., 2013). The second approach is about to ask all customers to provide a plan of the future resources demand. It is not possible according to the cloud obligations and the lack of customers' knowledge (Jiang et al., 2013). Due to VM technologies and the limita-tions of the customers' knowledge, the future demand prediction is the only practical and effective solution for the fast resources provisioning.

* Corresponding author.
E-mail addresses: maryam.amiri@tabrizu.ac.ir (M. Amiri), khanli@tabrizu.ac.ir (L. Mohammad-Khanli), raffaela.mirandola@polimi.it (R. Mirandola).

Published Journal Article available at: https://doi.org/10.1016/j.jnca.2017.12.015

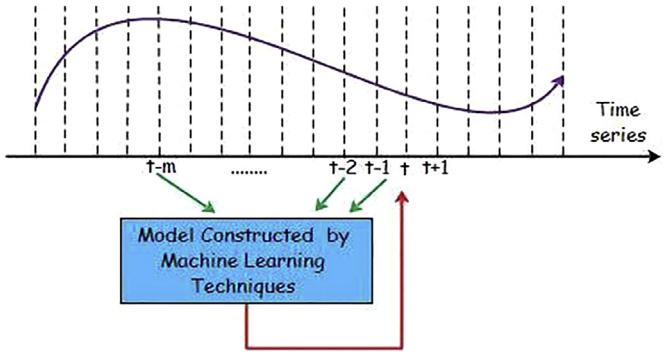


Fig. 1. Employing the sliding window and machine learning techniques on the time series (Amiri and Mohammad-Khanli, 2017).

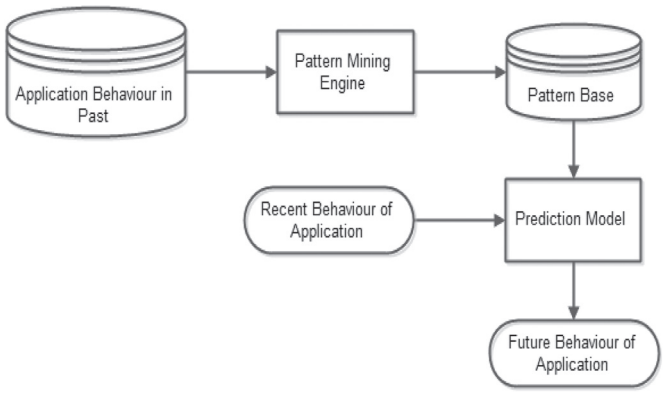


Fig. 2. The scheme of POSITING.

A proactive prediction method predicts the future demand fluctuations in a way that the resource manager has enough time to provide the appropriate resources before occurring the workload burstiness (Amiri and Mohammad-Khanli, 2017).

If the sudden increase of the future demand is predicted, the resource manager scales up the infrastructure and prepares VMs accord-ing 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. The released resources can be used to create new VMs or to allocate them to VMs that need more resources. Indeed, allocated resources are quickly matched with the demand and the rapid elasticity (Mell and Grance, 2011) is accomplished. Thus, SLA is satis-fied, energy wasting is avoided and on demand provisioning is fulfilled for systems implemented by using cloud services.

However, providing cloud services that guarantee dynamic QoS requirements of users and avoid SLA violation is a big challenge. Currently, the services are provisioned and scheduled according to the resources availability, without the guarantee of the expected performance (Singh and Chana, 2015). Therefore, the future demand prediction is an indispensable step for the rapid elasticity implementation and the effective resource provisioning in the dynamic cloud environment (Amiri and Mohammad-Khanli, 2017).

The newest and the most common proposed predictors are based on machine learning techniques (Amiri and Mohammad-Khanli, 2017) such as Neural Network (NN) (Xu et al., 2013; Yang et al., 2014b; Akindele and Samuel, 2013; Jiang et al., 2013; Prevost et al., 2011), Moving Average (MA) and Auto-Regression (AR) (Yang et al., 2014a; Saripalli et al., 2011; Shi et al., 2012; Miu and Missier, 2012; Mat-sunaga and Fortes, 2010a) and Hybrid Prediction Approaches (HPAs)(Jiang et al., 2013; Liu et al., 2015; Chen et al., 2015; Cetinski and Juric, 2015). The machine learning techniques usually model the appli-cation behaviour as a time series. Most of the methods are based on a sliding window with length of m which includes the previous behaviour of the application in the interval of $[t-m\cdots t-1]$ where t is the current time. According to the constructed model and the previous behaviour of the application in the sliding window, the future status in the time t is predicted. In the next step, the sliding window moves rightwards for one position. Fig. 1 shows the time series and the sliding window with length of m. These models cannot extract all 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 work-loads is one of the most important challenges in these methods (Amiri and Mohammad-Khanli, 2017). Moreover, when they are applied to bursty cloud workloads, they have limited accuracy (Di et al., 2014).

The existing prediction models such as NN, Support Vector Machine (SVM) and regression based methods are the discriminative learning approaches. The discriminative learning approaches do not present and understand the behavioural patterns of workloads explicitly. They learn the model parameters to optimize a utility function (such as an error criterion) (Tu, 2007). The model parameters are of little interest to the resources manager. So it seems methods are needed to extract knowledge from the model explicitly and improve the model transparency.

To the best of our knowledge, for the first time, this paper pro-poses a novel Prediction mOdel based on SequentIal paTtern mIN-inG (POSITING) to predict the future demand of cloud applications. As Fig. 2 shows, firstly POSITING considers the application behaviour in the past, investigates the correlation between different resources and extracts the behavioural patterns independently of the fixed pat-tern length explicitly. 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. A comprehensive set of experiments show the effectiveness of POSITING in comparison to the recent proposed predictors. The contributions of this paper are as follows:

- For the first time, POSITING focuses on unearthing the behavioural patterns of the workload variations of cloud applications **explicitly.** Thus, the behavioural patterns of workloads are more readily interpretable by the resources manager.
- Unlike the existing predictors, most of the parameters of POSITING are independent of the model structure. In other words, most of the parameters depend on the characteristics of cloud data centers. So they could be estimated from the domain knowledge of the resources manager easily.
- POSITING investigates the correlation between different resources and provides more understandable results for the resources manager. Thus, the extracted patterns are detailed and could provide more precise results.
- For the first time, this paper presents a comprehensive prediction model based on the pattern mining. The model predicts the status of all allocated resources in prediction time slots based on the extracted behavioural patterns.
- This paper proposes a new optimized representation for patterns. In comparison to the common matrix representation, it needs less memory and provides complete information about patterns.
- This paper introduces a new type of occurrences for patterns, which is called the **latest occurrence.** It is proved that the latest occurrence is more efficient than the other types of occurrences.
- The main goal of this paper is to show that models based on the pattern mining could offer novel and useful points of view for tackling some of the issues involved in predicting the application workloads. We compare POSITING to widely used predictors such as AR, MA, NN and HPA and demonstrate its superior performance.

The rest of the paper is organized as follows: Section 2 reviews related works on the prediction of the application workload in cloud. Primary concepts and definitions are explained in Section 3. Section

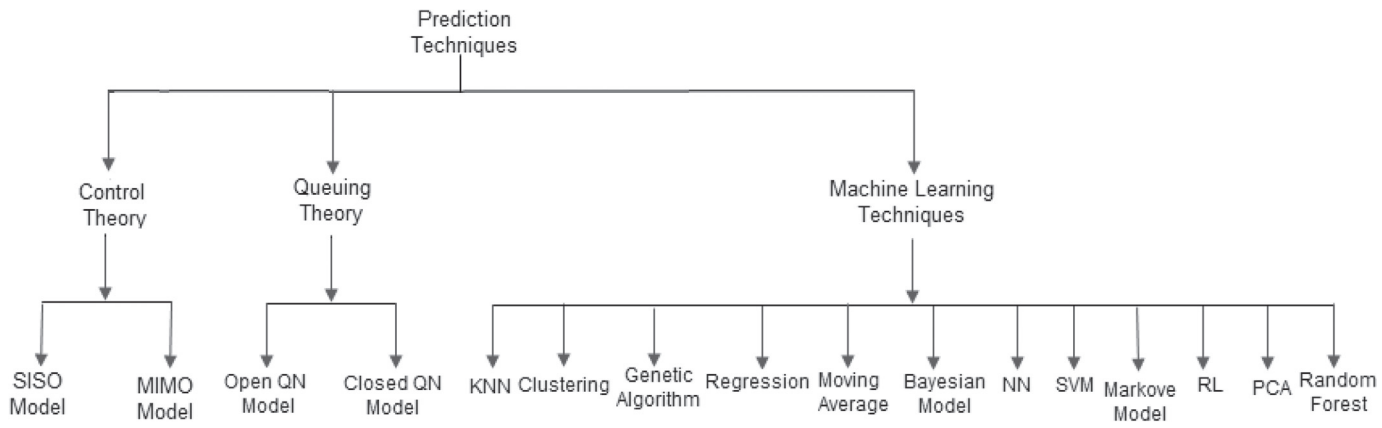


Fig. 3. The Taxonomy of prediction methods (Amiri and Mohammad-Khanli, 2017).

4 describes the pattern mining engine and the extraction of the application behaviour. In section 5, the prediction model, the main core of POSITING, is described. We present the experimental results based on both real and synthetic workloads in Section 6. Finally, the paper is concluded with our future work in Section 7.

2. Related work

In general, according to our previous work in (Amiri and Mohammad-Khanli, 2017), the prediction models proposed for cloud applications could be divided into three groups. As it is shown in Fig. 3, the taxonomy includes control theory, queuing theory and machine learning techniques. In the following subsections, each group is introduced.

2.1. Control theory

In (Liu et al., 2005) a Single Input Single Output (SISO) model maps the CPU share of the application to the inverse of its response time. Q-Cloud proposed in (Nathuji et al., 2010) handles the interference among VMs hosted on a server using the dynamic adjustment of resources allocated to applications. There is a closed loop controller on each server. It maps the resources usage of all VMs to their performance level using a Multi Input Multi Output (MIMO) model. Wu et al. in (Wu et al., 2013) 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 (Zhu et al., 2009). Thus, there is potential of instability. Although the non-linear controllers model the application behaviour accurately, their mathematical computations are complex (Amiri and Mohammad-Khanli, 2017). Some controllers such as fuzzy controllers (Cao et al., 2012; Lama and Zhou, 2013) are based on the rule based approaches. The rules extraction is not easy for the resource management. 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 can be used to predict the performance of applications. These models have parameters such as the requests arrival rate and the average resources requirements of requests that should be specified (Urgaonkar et al., 2008). These parameters can be estimated by solving some equations resulted from the system evaluation. In (Bennani and Menasce, 2005) the response time of the transactional workload and the throughput of batch jobs are modeled by using the open QN and the closed QN respectively. Chalheiros et al. in (Calheiros et al., 2011) propose a mechanism for the VM provision-ing. In this mechanism, a workload analyzer predicts the arrival ratio of requests based on historical data or known workloads. A performance modeler models the system as QN. It predicts the response time, the reject ratio and the resources utilization. If the estimated parameters are below the QoS metrics, the number of VMs allocated to applica-tions is updated. In (Zhang et al., 2007) the CPU demand of different types of transactions is estimated by using the regression based meth-ods. 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 (Amiri and Mohammad-Khanli, 2017).

2.3. Machine learning techniques

The newest proposed approaches are based on machine learning techniques. Garg et al. in (Garg et al., 2014) consider the resources allocation in a data center that includes the non-interactive and the transactional workloads. The proposed method predicts the CPU utilization of transactional applications by NN. Chen et al. in (Chen et al., 2015) propose a system to predict the resources demand. Due to the work-load dynamics in different periods, the base predictors such as the Sec-ond Moving Average Model (SMAM), the Exponential Moving Average method (EMA), the 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. The clustering algorithms are used to optimize the FNN system. Yang et al. in (Yang et al., 2014a) propose a method based on Linear Regression (LR) to predict the number of requests for each cloud service. Accord-ing to the workload fluctuations, the prediction method adjusts itself through the recomputation of parameters of the regression model. Tang et al. in (Tang et al., 2014) propose a bin-packing algorithm that selects a PM for each VM, according to its future memory usage. In addition to allocating enough memory to each VM, the number of PMs should be minimized. The memory usage of VMs is modeled as a random variable. To predict the probability distribution of the future memory demand, the AR model is used. The model parameters are updated for each VM frequently.

Table 1 shows machine learning based methods employed for the workload prediction in some literature. We recommend that readers interested in a comprehensive review of prediction models of cloud applications refer to our previous work in

Table 1
The machine learning based methods and techniques used for the application prediction in different literature.

Methods and Techniques	References
Methods based on Regression and Moving Average	(Yang et al., 2014a), (Akindele and Samuel, 2013), (Jiang et al., 2013), (Liu et al., 2015), (Shi et al., 2012), (Tang et al., 2014)
Bayesian Theory	(Di et al., 2014), (Duan et al., 2009)
K Nearest Neighbor (KNN)	(Akindele and Samuel, 2013)
Random Forest	(Cetinski and Juric, 2015)
Neural Network (NN)	(Xu et al., 2013), (Yang et al., 2014b), (Akindele and Samuel, 2013), (Jiang et al., 2013), (Islam et al., 2012)
Support Vector Machine (SVM)	(Akindele and Samuel, 2013), (Kundu et al., 2012), (Jiang et al., 2013), (Liu et al., 2015), (Matsunaga and Fortes, 2010b)
Markov Model	(Xu et al., 2013), (Zhenhuan et al., 2010), (Khan et al., 2012), (Lu et al., 2015)
Clustering/Classification	(Xu et al., 2013), (Khan et al., 2012), (Bey et al., 2009), (Kundu et al., 2012)
Principal Component Analysis (PCA)	(Gursun et al., 2011)
Reinforcement Learning (RL)	(Xu et al., 2012), (Huang et al., 2014a), (Amiri et al., 2016)
Evolutionary Algorithm (Genetic Algorithm)	(Yang et al., 2014b), (Jiang et al., 2013), (Antonescu et al., 2013)
Fuzzy Logic	(Bey et al., 2009), (Chen et al., 2015)

(Amiri and Mohammad-Khanli, 2017) for more detail.

POSITING has been developed in the direction of improving the existing methods. It is able to extract all the behavioural patterns of workloads independently of the fixed pattern length explicitly. It does not need to make any assumptions about the workload behaviour. So POSITING is a general predictor and could be used for the different types of workloads. The next section describes the foundation and for-mulation of POSITING in detail.

3. Background and primary concepts

Sequential data is data ordered with respect to some indexes. The time series is a type of sequential data, where records are indexed by time (Laxman and Sastry, 2006). Pattern mining of sequential data sets is called Sequential Patten Mining (SPM). In this section, primary concepts and definitions of POSITING are introduced.

3.1. Abstraction representation

To predict the future workload of applications, the behaviour of allocated resources is modeled as a time series (Andreolini et al., 2015). So we encounter multivariate temporal data sets, where the data points are traces of complicated behaviour characterized by time series of resources (Batal et al., 2016). Batal et al. in (Batal et al., 2016) define temporal patterns for time-interval data and employ the value abstrac-tion to handle the complex temporal data. The value abstractions are used to segment the time series based on its values. As Fig. 4 shows the value abstraction converts a numeric time series into a sequence of abstractions $< S_1[st_1, et_1], \dots, S_n[st_n, et_n] >$ where $S_i \in \sum, 1 \le i \le n$ is an abstraction that holds from time st_i to time et_i and $\sum$ is the abstraction alphabet. Therefore the numeric time series of all allocated resources is converted into the symbolic (discretized) time series (Batal et al., 2016).

Let $Status = \{S_1, \dots, S_M\}$ be a set of the abstract values (the abstraction alphabet) and $ResourceType = \{R_1, \dots, R_N\}$ be a set of all the resources allocated to the application. So M is the number of the abstract values and N is the number of allocated resources. Without loss of generality, we define an arbitrary order on *ResourceType*, for example $R_1 < R_2 < \cdots < R_N$. Inspired by the definitions presented in (Batal et al., 2016), we generalize the concepts **state interval** to **event**

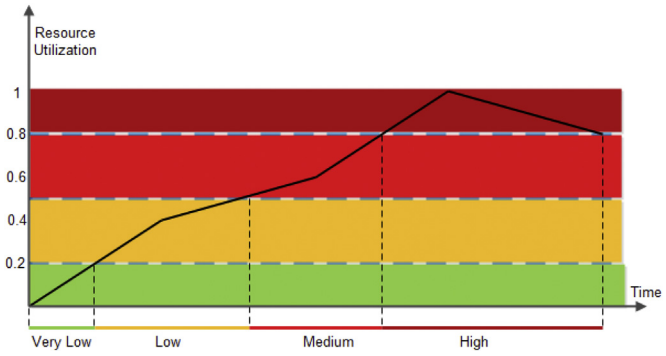


Fig. 4. Converting a time series into a symbolic (discretized) time series by the value abstraction that $\sum = \{Very\ Low, Low, Medium, High\}$ and blue dashed lines show the border of the values (Batal et al., 2016). (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

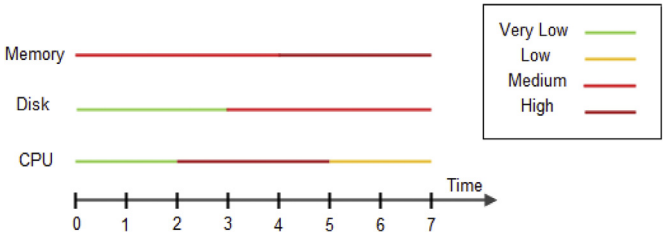


Fig. 5. An example of a multivariate stream with $ResourceType = \{CPU, Memory, Disk\}$ and $Status = \{Very\ Low, Low, Medium, High\}$ (Batal et al., 2016).

and **multivariate state sequence** to **stream** in this paper.

Definition 1. An **event** e_i is defined as a 4-tuple $< r_i, s_i, st_i, et_i >$ 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.

Definition 2. The span of the event $e_i =< r_i, s_i, st_i, et_i >$ is $\Delta e_i = et_i - st_i > \epsilon \ge 0$.

All discretized time series of 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 3. A multivariate **stream** $E =< e_1, e_2, \dots, e_n >$, where n is the index of the latest observed event, is a sequence of events that are ordered according to their start time:

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

Example 1. Fig. 5 shows a multivariate stream E with $ResourceType = \{CPU, Memory, Disk\}$ and $Status = \{Very\ Low, Low, Medium, High\}$. If the order on $ResourceType$ is defined as $CPU < Memory < Disk$, according to Definition 3, $E =< (CPU, Very\ Low, 0,2), (Memory, Medium, 0,3), (Disk, Very\ Low, 0,4), (CPU, High, 2,5), (Disk, Medium, 3,7), (Memory, High, 4,7), (CPU, Low, 5,7) >$.

Definition 4. A **state** is an ordered pair of (r, s), where $r \in ResourceType$ and $s \in Status$.

Definition 5. The Resource-Status (**RS**) is a set of all possible states:

$$RS = \{(r, s) | \forall r \in ResourceType, \forall s \in Status\}.$$

According to Definition 2, the span of each event is at least $\epsilon + 1$ time slots. For the smooth regions of workloads, the span of the events is large. It might lead to the inability to extract all hidden useful patterns. So if the span of events is large, they are decomposed based on the decomposition unit μ. For example the event $(Disk, Medium, 3,7)$

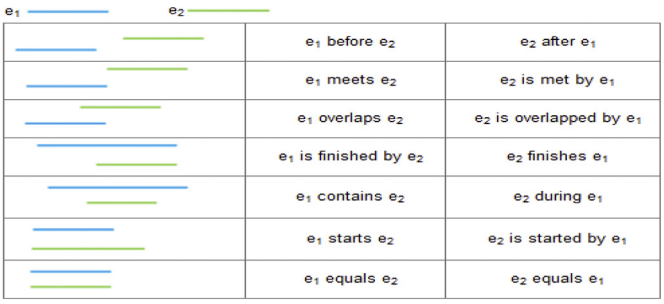


Fig. 6. Temporal relations between two events e_1 and e_2 (Allen, 1984).

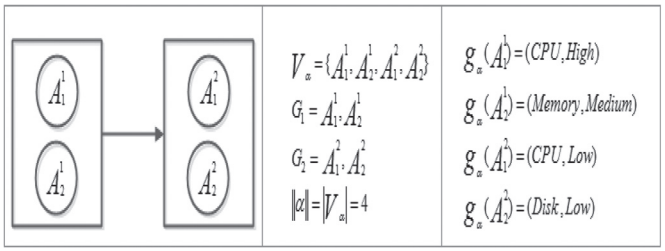


Fig. 7. The graphical representation of the episode $\alpha = (CPU, High)\,(Memory, Medium) \rightarrow (CPU, Low)\,(Disk, Low)$.

with $\mu = 3$ is decomposed into two events (*Disk*, *Medium*, 3,6) and (*Disk*, *Medium*, 6,7). If the span of the event e is L ($\Delta e = L$), based on the decomposition unit μ, $L = k\mu + \theta$ that $k \in \mathbb{N}$ and $0 \le \theta \le \mu - 1$. Since the span of each event should be greater than ϵ (according to Definition 2), then we limit θ as $\epsilon + 1 \le \theta \le \mu - 1$. So we have $\mu \ge \epsilon + 2$. However, after decomposing the event e, the span of the last decomposed event might be less than ϵ. Here, to satisfy Definition 2, the latest and penul-timate decomposed events merge together to create an event with the span $\mu + \theta$.

Lemma 1. *For the event* $e = (r, s, st, et)$: $1 + \epsilon \le \Delta e \le \mu + \epsilon$.[1]

3.2. Temporal relations

Allen in (Allen, 1984) considers 13 possible relations between two events. Fig. 6 shows 7 relations *before*, *meets*, *overlaps*, *is − finished − by*, *contains*, *starts*, *equals* and their inverse (Hoppner; Moskovitch and Shahar, 2009; Batal et al., 2016). If time information of data is not very precise, according to Allen's relations, different patterns could be extracted that they describe similar behaviour of the application work-load (Moerchen, 2006). So if Allen's relations are used, the search space of patterns becomes extremely large (Batal et al., 2016). On the other hand, time and the space complexities of the prediction model should be reasonable in a way that its deployment is affordable (Weingartner et al., 2015). Therefore, inspired by the temporal rela-tions defined in (Batal et al., 2016), we classify Allen's relations into two groups **concurrent** and **consecutive** based on the start time of events.

Definition 6. Given the stream $E =< e_1, \dots, e_n >$, two events e_i and e_j, $1 \le i, j \le n$, are **concurrent** iff $|st_i - st_j| \le \epsilon$.

Definition 7. Given the stream $E =< e_1, \dots, e_n >$, two events e_i and e_j, $1 \le i, j \le n$, are **consecutive** iff $|st_i - st_j| > \epsilon$.

For example, in example 1 for $\epsilon = 0$, events (*CPU*, *Very Low*, 0,2) and (*Memory*, *Medium*, 0,3) are concurrent and events (*Disk*, *Medium*, 3,7) and (*Memory*, *High*, 4,7) are consecutive.

3.3. Episode

In the field of SPM, discovering patterns in sequences is called the episode mining (Han et al., 2007). Mannila et al. in (Mannila et al., 1997) define an episode as a partially ordered collection of events that occur together. So the main goal of the episode mining is to extract episodes that occur frequently in the sequence. Note that we use terms "pattern" and "episode" interchangeably in this paper. The serial episodes are the most appropriate type of episodes for prediction (Toma et al., 2007; Laxman et al., 2008; Fahed et al., 2014; Huang et al., 2014b). In the rest of the paper, the term "episode" is used instead of the term "serial episode" concisely. Inspired by the definition of the episode in (Mannila et al., 1997), a detailed definition of the episode is presented based on our problem domain.

Definition 8. A Concurrent Nodes Group (CNG) $G = A_1, A_2, \dots, A_l$ is a group of nodes that $\forall A_j, A_m \in G, 1 \le j, m \le l$, there is no partial order between A_j and A_m.

Definition 9. 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 $G_1 = A_1^1, A_2^1, \dots, A_{l_1}^1, \dots, G_k = A_1^k, A_2^k, \dots, A_{l_k}^k$ that:

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

Note that the episode α could be represented as a general form $\alpha = G'_1 \rightarrow G'_2 \rightarrow \cdots \rightarrow G'_k$ that means there is a partial order between every state in G'_i $(1 \le i < k)$ and every state in $G'_j (i + 1 \le j \le k)$.

Definition 10. The size of episode α, $\|\alpha\|$, is $|V_\alpha|$.

Example 2. Consider the episode $\alpha = (V_\alpha, \prec_\alpha, g_\alpha)$ in Fig. 7. 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 $\alpha = (CPU, High)\,(Memory, Medium) \rightarrow (CPU, Low)\,(Disk, Low)$. Note that α discloses the correlation between *CPU*, *Memory* and *Disk* explicitly. It declares that if *CPU is High* and *Memory* is *Medium* concurrently, after that the future status of both *CPU* and *Disk* is *Low*.

3.4. Episode occurrence

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 (Mannila et al., 1997).

Definition 11. The episode $\alpha = (V_\alpha, \prec_\alpha, g_\alpha)$ occurs in the stream $E =< e_1, \dots, e_n >$ that $e_i = (r_i, s_i, st_i, et_i), 1 \le i \le n$ if there exists an injective mapping $h : \{A_j^i | \forall i \in \{1, \dots, k\}, \forall j \in \{1, \dots, l_i\}\} \rightarrow \{1,2, \dots, n\}$ such that:

1. $g_\alpha(A_j^i).r = r_{h(A_j^i)}$, $g_\alpha(A_j^i).s = s_{h(A_j^i)}$
2. $\forall A_k^i, A_j^i, 1 \le k, j \le l_i, 1 \le i \le k : |st_{h(A_j^i)} - st_{h(A_k^i)}| \le \epsilon$
3. $\forall i, 1 \le i < k : \min\{st_{h(A_j^{i+1})}\}_{j=1}^{l_{i+1}} - \max\{st_{h(A_j^i)}\}_{j=1}^{l_i} > \epsilon$

[1] The Proof of lemmas, theorems and corollaries could be found in Appendix A.

Example 3. Consider the stream $E =< e_1 = (CPU, High, 0,3), e_2 = (Memory, Medium, 0,4), e_3 = (Network, Low, 0,2), e_4 = (Disk, Medium, 0,3), e_5 = (Network, Medium, 2,5), e_6 = (CPU, 3,5, Low), e_7 = (Disk, Low, 3,5), e_8 = (Memory, Very\ Low, 4,5) >$. For the episode α given in example 2 and $\epsilon = 0$, the injective mapping function h is as follows:

$$h(A_1^1) = 1, h(A_2^1) = 2, h(A_1^2) = 6, h(A_2^2) = 7.$$

So there is an occurrence of the episode α in the stream E.

Lemma 2. *For the episode $\alpha = (V_\alpha, \prec_\alpha, g_\alpha)$:*
$\forall v_1, v_2 \in V_\alpha$ that $v_1, v_2 \in G_i, i \in \{1, \dots, k\}$, if $g_\alpha(v_1).r = g_\alpha(v_2).r$, there is no occurrence for α.

Definition 12. Given the episode α that $|CNG_\alpha| = k$, for each occurrence of α, the **starting interval** of the occurrence of $G_i, (1 \le i \le k)$, $[t_1^i, t_2^i]$, is:

$$t_1^i = \min\{st_{h(A_j^i)}\}_{j=1}^{l_i},\ t_2^i = \max\{st_{h(A_j^i)}\}_{j=1}^{l_i} \tag{3.1}$$

Definition 13. Given the episode α that $|CNG_\alpha| = k$, the **span of the occurrence** of α, $[t_1^\alpha, t_2^\alpha]$, is:

$$[t_1^\alpha, t_2^\alpha] = [\min\{st_{h(A_j^1)}\}_{j=1}^{l_1} = t_1^1, \max\{st_{h(A_j^k)}\}_{j=1}^{l_k} = t_2^k] \tag{3.2}$$

Definition 14. Given the episode α that $|CNG_\alpha| = k$, each occurrence O of α is determined as a sequence of the starting intervals of *CNG*s and the span of O:

$$O = (([t_1^i, t_2^i])_{i=1}^k [t_1^\alpha, t_2^\alpha]) \tag{3.3}$$

Definition 15. *OS et*(α) is a set of all occurrences of the episode α in the stream.

Example 4. Consider the example 3. Based on the injective mapping function h, 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 $[t_1^\alpha, t_2^\alpha] = [0,3]$.

Consider two episodes $\alpha_1 = (CPU, Low)\ (Memory, High) \rightarrow (Network, High)$ and $\alpha_2 = (Memory, High)\ (CPU, Low) \rightarrow (Network, High)$. It is clear that these episodes are equivalent. To get a unique form of episodes, states mapped to nodes of each *CNG* are sorted based on the order defined on *ResourceType*. So if $CPU < Memory$, α_1 shows the unique form.
The sub-episode is a main concept to construct and prune the episodes. This concept is defined formally as follows:

Definition 16. The episode $\beta = (V_\beta, \prec_\beta, g_\beta)$ is a **sub-episode** of the episode $\alpha = (V_\alpha, \prec_\alpha, g_\alpha)$, denoted by $\beta \sqsubseteq \alpha$, iff there exists an injective mapping $F : V_\beta \rightarrow V_\alpha$ such that:

1. $\forall v_1 \in V_\beta : g_\beta(v_1) = g_\alpha(F(v_1))$
2. $\forall v_1, v_2 \in V_\beta$:
   (2.1) if $v_1 \prec_\beta v_2$ then $F(v_1) \prec_\alpha F(v_2)$
   (2.2) if there is no partial order between v_1 and v_2, then there should be no partial order between $F(v_1)$ and $F(v_2)$.

Based on Definition 16, the episode α is a super-episode of the episode β iff $\beta \sqsubseteq \alpha$. Note that $\beta \sqsubseteq \alpha$ means that all states of β and the partial order between them exist in α.

Example 5. Consider the episodes $\alpha = (Memory, High)\ (Disk, Medium) \rightarrow (CPU, Low)\ (Network, Low)$, $\beta = (Disk, Medium) \rightarrow (CPU, Low)$ and $\gamma = (Memory, High)\ (Disk, Medium) \rightarrow (CPU, Low)$. According to Definition 16, we have $\beta \sqsubseteq \gamma \sqsubseteq \alpha$.

Lemma 3. *In each occurrence of the episode α in the stream E, there exists an occurrence of all the sub-episodes of α.*

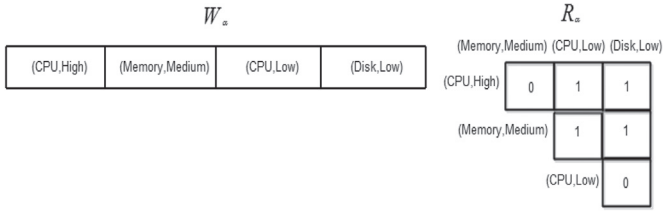


Fig. 8. The matrix representation of the episode $\alpha = (CPU, High)\ (Memory, Medium) \rightarrow (CPU, Low)\ (Disk, Low)$.

3.5. Episode representation

The matrix representation is a common unambiguous representa-tion of time-interval patterns (Hoppner; Moskovitch and Shahar, 2009; Huang et al., 2014b; Batal et al., 2012). In this representation, the episode α is represented by (W_α, R_α) where W_α is an ordered list of states based on their appearance in α and R_α is an upper triangular matrix that includes information of the partial order between each state and all of its following states. If two states are in the same group, there is no partial order between them. So the corresponding entry in the upper triangular matrix is set to 0. If two states are in different groups, there exists the serial relation between them. So the corresponding entry in the upper triangular matrix is set to 1. It could be proved simply that the space complexity of the matrix representation for the episode α is $O(\|\alpha\|^2)$.

Example 6. Consider the episode $\alpha = (CPU, High)\ (Memory, Medium) \rightarrow (CPU, Low)\ (Disk, Low)$. Fig. 8 shows the matrix representation of the episode α. For example, $W_\alpha[1] = (CPU, High)$, $W_\alpha[2] = (Memory, Medium)$ and $R_\alpha(W_\alpha[1], W_\alpha[2]) = 0$.

Definition 17. The **Following Nodes** of the node A_j^i of the episode α, $FN(A_j^i)$, is a sequence of all the nodes of G_i that appear after A_j^i and all the nodes of $G_n, i + 1 \le n \le k$:

$$FN(A_j^i) = ((A_u^i)_{u=j+1}^{l_i}((A_m^n)_{m=1}^{l_n})_{n=i+1}^k) \tag{3.4}$$

Theorem 1. *Given the episode $\alpha = G'_1 \rightarrow \cdots \rightarrow G'_k$ in the form of $\alpha = (W_\alpha, R_\alpha)$, the number of inferable and redundant entries of R_α is:*

$$\sum_{i=1}^{k}(l_i - 1)\left(\sum_{t=i}^{k} l_t - \left(\frac{l_i}{2} + 1\right)\right) \tag{3.5}$$

This paper proposes a new representation for episodes, which is called repreSentAtiVe rEpresentation (SAVE). SAVE is based on the vector.

Definition 18. Given the episode $\alpha = G'_1 \rightarrow \cdots \rightarrow G'_k$, SAVE is composed of one Representative Array (RArray) and k Group Lists (GLists):

- Each entry $i, 1 \le i \le k$ of RArray includes a representative member of G'_i. $RArray[i].x$ is the representative member of the entry i.
- Each entry $i, 1 \le i \le k$ of RArray is linked into one GList. $RArray[i].GList$ is the corresponding GList of the entry i.
- $RArray[i].GList, 1 \le i \le k$ includes all members of G'_i except the representative member.
- $\alpha =< RArray_\alpha >$ means that the episode α is represented in the form of SAVE.

Note that to offer a unique representation for episodes, the representative member of each $G'_i, 1 \le i \le k$ is the first member of G'_i sorted based on the order on *ResourceType*. The other members of G'_i are also inserted in $RArray[i].GList$ based on the order on *ResourceType*.

Example 7. Consider the episode $\alpha = (CPU, High)\ (Memory, Medium)\ (Network, High) \rightarrow (CPU, Low)\ (Disk, Low) \rightarrow (Network, Very\ Low)$. Fig. 9 shows the representation of the episode α in the form of SAVE. Note that $RArray[i].x, 1 \le i \le 3$ is the first member of G'_i.

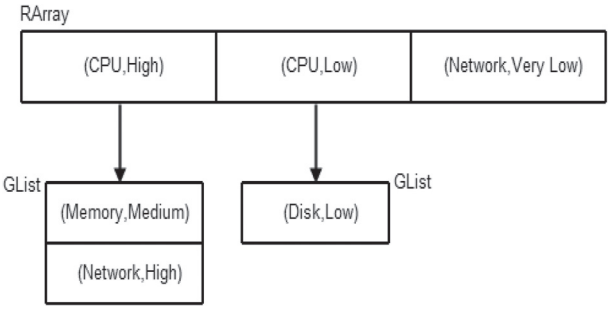


Fig. 9. The representation of the episode $\alpha = (CPU, High)\,(Memory, Medium)\,(Network, High) \rightarrow (CPU, Low)\,(Disk, Low) \rightarrow (Network, Very\ Low)$ in the form of SAVE.

Lemma 4. *The space complexity of SAVE for the episode α is $O(\|\alpha\|)$.*

As Corollary 2 in Appendix A shows SAVE improves the time complexity of the episode processing, in addition to decreasing the memory consumption.

3.6. Frequent episode

A frequent episode occurs often enough in the stream. Given a frequency threshold, the goal of the frequent episodes discovery is to extract all frequent episodes in the stream (Achar et al., 2012a). For this purpose, the frequency of the episode should be defined. The episode frequency is defined in different ways such as the window-based frequency (Mannila et al., 1997), minimal occurrence-based frequency (Mannila et al., 1997), head frequency and total frequency (Iwanuma et al., 2004), Non-Overlapped (NO) frequency (Laxman et al., 2005), non-interleaved frequency (Laxman, 2006) and distinct occurrence-based frequency (Karypis et al., 1999). The simplest and the most efficient algorithm for the frequency counting is the algorithm of the NO frequency (Achar et al., 2012a). So we choose the NO frequency to compute the frequency of episodes. Since the NO frequency is computed based on minimal occurrences, firstly the definition of minimal occurrences is presented.

Definition 19. Given the episode $\alpha = G'_1 \rightarrow \cdots \rightarrow G'_k$, the occurrence O of the episode α, $O = (([t^i_1, t^i_2])^k_{i=1}[t^\alpha_1, t^\alpha_2])$, is a Minimal Occurrence (MO) iff

$$\nexists O' = (([t'^i_1, t'^i_2])^k_{i=1}[t'^\alpha_1, t'^\alpha_2]) \in OS\ et(\alpha)\ such\ that\ (t'^\alpha_1 \neq t^\alpha_1\ or\ t'^\alpha_2 \neq t^\alpha_2)\ and (t^\alpha_1 \leq t'^\alpha_1 < t'^\alpha_2 \leq t^\alpha_2) \quad (3.6)$$

Definition 20. *MOS et(α)* is a set of all minimal occurrences of the episode α in the stream.

Definition 21. An **equivalent class of minimal occurrences** includes all minimal occurrences whose spans are equal.

Lemma 5. *Given the episode α that $|CNG_\alpha| = k$ and $\mu \geq \max(2\epsilon + 1, \epsilon + 2)$, the successive starting intervals of $G_i, 1 \leq i \leq k$ have no overlap.*

Lemma 6. *Given the episode α that $|CNG_\alpha| = k$, $\mu \geq \max(2\epsilon + 1, \epsilon + 2)$ and two occurrences $O, O' \in OS\ et(\alpha)$, if $[u, u']$ is the starting interval of $G_i, 1 \leq i \leq k$ in O, the following starting interval of G_i in O' is $[w, w']$ that $w > 2\epsilon + u$.*

Note that the condition $\mu > 2\epsilon$ is reasonable because the minimum span of events is $\epsilon + 1$ according to Lemma 1. So the decomposition unit of events, μ, could be twice more than the minimum span.

Definition 22. Given the episode α that $|CNG_\alpha| = k$, $\forall O_1, O_2 \in OS\ et(\alpha)$ that $O_1 = (([w^i_1, w^i_2])^k_{i=1}[w^\alpha_1, w^\alpha_2])$ and $O_2 = (([u^i_1, u^i_2])^k_{i=1}[u^\alpha_1, u^\alpha_2])$, $O_1 < O_2$ iff:

$$\exists p, 1 \leq p \leq k\ such\ that\ \forall 1 \leq j < p\ [w^j_1, w^j_2] = [u^j_1, u^j_2]\ and\ w^p_1 < u^p_1 \quad (3.7)$$

Definition 23. Given the episode α that $|CNG_\alpha| = k$, two occurrences $O_1, O_2 \in OS\ et(\alpha)$ that $O_1 = (([w^i_1, w^i_2])^k_{i=1}[w^\alpha_1, w^\alpha_2])$ and $O_2 = (([u^i_1, u^i_2])^k_{i=1}[u^\alpha_1, u^\alpha_2])$ are said to be non-overlapped iff $w^\alpha_2 \leq u^\alpha_1$.

Definition 24. The frequency of the episode α in the stream, *freq*(α), is defined as the cardinality of a maximal non-overlapped set of minimal occurrences of α in the stream. A set of occurrences is said to be non-overlapped if every pair of occurrences in the set is non-overlapped.

Definition 25. $OS\ et^N_M(\alpha) = \{O_1, \ldots, O_L\}$ is a set of all non-overlapped minimal occurrences that O_1 is the first minimal occurrence of α and $O_{i+1}, 1 \leq i < L$ is the first non-overlapped minimal occurrence after O_i. Furthermore, there is no non-overlapped minimal occurrence after O_L.

Lemma 7. *If $O'(\alpha) = \{O'_1, \ldots, O'_F\}$ is a set of non-overlapped minimal occurrences where $O'_i, 1 < i \leq F$ is the first non-overlapped minimal occurrence after O'_{i-1} and $O'_1 \notin OS\ et^N_M(\alpha)$, then for O'_1 with the span of $[w_1, w'_1]$, there is a unique occurrence $O_z \in OS\ et^N_M, 1 \leq z \leq L$ with the span of $[u_z, u'_z]$ where $u_z < w_1 < u'_z < w'_1$. For $O'_i, i > 1$ with the span of $[w_i, w'_i]$, there is a unique occurrence $O_t \in OS\ et^N_M, z < t \leq L$ with the span of $[u_t, u'_t]$ where $u_t < w_i < u'_t < w'_i$ or $(u_t = w_i$ and $u'_t = w'_i)$.*

Theorem 2. *$OS\ et^N_M(\alpha)$ is a maximal non-overlapped set of minimal occurrences of the episode α in the stream: $freq(\alpha) = |OS\ et^N_M(\alpha)|$*

Definition 26. A constraint C is **anti-monotonic** if the episode α satisfying C implies that every sub-episode of α also satisfies C (Pei et al., 2007).

Lemma 8. *Given the episodes α and β and the threshold $\theta \in \mathbb{R}_{\geq 0}$, if $\beta \sqsubseteq \alpha$ and $freq(\alpha) \geq \theta$, then $\theta \leq freq(\alpha) \leq freq(\beta)$ (the anti-monotonic constraint).*

Example 8. *Let $ResourceType = \{CPU, Memory\}$, $Status = \{Low(L), Medium(M), High(H)\}$ and $CPU < Memory$. Consider the stream $E =<$ $(CPU, L, 0,1), (Memory, M, 0,3), (CPU, H, 1,2), (CPU, L, 2,3), (CPU, M, 3,4),$ $(Memory, H, 3,4), (CPU, L, 4,6), (Memory, L, 4,5), (Memory, High, 5,6) >$ and the episode $\alpha = (CPU, Low) \rightarrow (Memory, High)$,. We have $OS\ et(\alpha) =$ $\{([0,0],\ [3,3],\ [0,3]), ([2,2], [3,3], [2,3]), ([0,0], [5,5], [0,5]), ([2,2], [5,5],$ $[2,5]), ([4,4], [5,5], [4,5])\}$, $MOS\ et(\alpha) = \{([2,2], [3,3], [2,3]), ([4,4],$ $[5,5], [4,5])\}$ and $OS\ et^N_M(\alpha) = \{([2,2], [3,3], [2,3]), ([4,4], [5,5], [4,5])\}$. $freq(\alpha) = 2$.*

3.7. Stream representation

The stream can be represented in different forms. In this paper, the stream is represented in the vertical format (Zaki, 2001). In the vertical representation, each $(r, s) \in RS$ is associated with a list whose entries include the starting intervals of that (r, s).

Example 9. Consider the stream of the example 8. Fig. 10 shows the vertical representation of the stream E.

3.8. Pattern tree

In this section, the lexicographic tree of episodes (pattern tree), the main core of POSITING, is described. Based on the set lexicographic order proposed in (Yan et al., 2003), to construct the lexicographic tree, a lexicographic order on RS should be defined. Firstly, different types of the episode extension are defined.

Definition 27. 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:

$$\alpha \oplus (r, s) = G'_1 \rightarrow G'_2 \rightarrow \cdots \rightarrow G'_k \rightarrow (r, s) \quad (3.8)$$

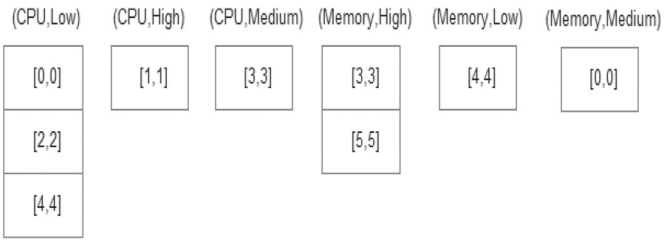


Fig. 10. The vertical representation of the stream of the example 8.

Definition 28. Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_{k-1} \to G'_k$ and $(r,s) \in RS$, the **concurrent extension** of α with (r,s) is:

$$\alpha \odot (r,s) = G'_1 \to G'_2 \to \cdots \to G'_{k-1} \to G'' \quad that\ G'' = G'_k \cup (r,s) \tag{3.9}$$

Definition 29. Given $G_1 = (R_1,S_1)\cdots(R_i,S_i)$ and $G_2 = (R'_1,S'_1)\cdots(R'_j,S'_j)$ that $(R_x,S_x) \in RS, 1 \le x \le i$ and $(R'_y,S'_y) \in RS, 1 \le y \le j$, it is said that $G_1 < G_2$ iff any of the conditions below is true (Yan et al., 2003):

- $\exists h, 0 \le h \le \min(i,j)$ *such that* $(R_h,S_h) < (R'_h,S'_h)$ *and* $\forall r < h$, $(R_r,S_r) = (R'_r,S'_r)$
- $i < j, \forall r, 1 \le r \le i, (R_r,S_r) = (R'_r,S'_r)$

Definition 30. Given two episodes $\alpha = G_1 \to G_2 \to \cdots \to G_i$ and $\beta = G'_1 \to G'_2 \to \cdots \to G'_j$ it is said $\alpha < \beta$ iff any of the conditions below is true (Yan et al., 2003):

- $\exists h, 0 \le h \le \min(i,j)$ *such that* $G_h < G'_h$ *and* $\forall r < h,\ G_r = G'_r$
- $i < j, \forall r, 1 \le r \le i, G_r = G'_r$

Given $(r,s),(r',s') \in RS$ and based on the lexicographic order described in Definition 30, it can be concluded (Yan et al., 2003):

- if $\beta = \alpha \oplus (r,s)$, then $\alpha < \beta$.
- if $\beta = \alpha \oplus (r,s)$ and $\gamma = \alpha \odot (r',s')$, then $\gamma < \beta$.
- if $\beta = \alpha \oplus (r,s)$, $\gamma = \alpha \oplus (r',s')$ and $(r',s') < (r,s)$ then $\gamma < \beta$.
- if $\beta = \alpha \odot (r,s)$, $\gamma = \alpha \odot (r',s')$ and $(r',s') < (r,s)$ then $\gamma < \beta$.

Definition 31. The lexicographic tree is constructed as follows (Yan et al., 2003):

- The root is labeled with $\varnothing$.
- Each node n of the tree is labeled with a state. *Label*(n) is the corresponding label of the node n.
- Each node n of the tree corresponds to an episode. *Pattern*(n) is the corresponding episode of the node n.
- If *Pattern*(n) = α, the corresponding episode of each child of n is either a serial extension or a concurrent extension of α.
- The left sibling is less than the right sibling.

Fig. 11 shows a part of the pattern tree constructed on $RS = \{(R_i,S_j) | R_i \in ResourceType, S_j \in Status, 1 \le i \le N, 1 \le j \le M\}$. Here, the lexicographic order is defined on RS as $(R_1,S_1) < \cdots < (R_1,S_M) < (R_2,S_1) < \cdots < (R_2,S_M) < \cdots < (R_N,S_1) < \cdots < (R_N,S_M)$. The root of the tree is null. All episodes in the tree are generated only by the serial extension or the concurrent extension. For example the episode $((R_1,S_1)(R_2,S_1))$ is generated from the concurrent extension of (R_1,S_1) with (R_2,S_1) and the episode $((R_1,S_1) \to (R_N,S_M))$ is generated from the serial extension of (R_1,S_1) with (R_N,S_M). In the next sections, we will propose algorithms to construct and traverse the tree in more detail. For each node n of the pattern tree, two sets of states are determined to extend *Pattern*(n). Based on Lemma 3 and to avoid generating redundant nodes (nodes with the same episodes), two sets of valid states for the concurrent and the serial extensions are defined as follows:

Definition 32. For each node n of the pattern tree, *SExt*(n) is a set of all the states for the serial extensions of *Pattern*(n):

$$S\,Ext(n) = \{(r,s) | (r,s) \in RS\} \tag{3.10}$$

Definition 33. Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_k$, $LGroup(\alpha) = G'_k$.

Definition 34. For each node n of the pattern tree, *CExt*(n) is a set of valid states for the concurrent extensions of *Pattern*(n).

$$CExt(n) = \{(r,s) | (r,s) \in RS, (r,s) > Label(n) \\ and\ \nexists (r',s') \in LGroup(Pattern(n))\ such\ that\ r' = r\} \tag{3.11}$$

3.9. Gap constraint

Cloud should allocate a suitable amount of resources according to the current demand of applications. Under-provisioning causes SLA violation, QoS dropping and the customer dissatisfaction. This might lead to the loss of customers and a decrease in revenue. On the other hand, overprovisioning wastes energy and resources and it even increases costs like network, cooling and maintenance (Amiri and Mohammad-Khanli, 2017). So, the resource manager should have enough time to provide the appropriate resources before occurring the workload burstiness. Dynamic resources allocation is based on the virtualization techniques (Hwang et al., 2016). The time it takes to instantiate a new VM instance is about 5–15 min (Li et al., 2010). Therefore, based on

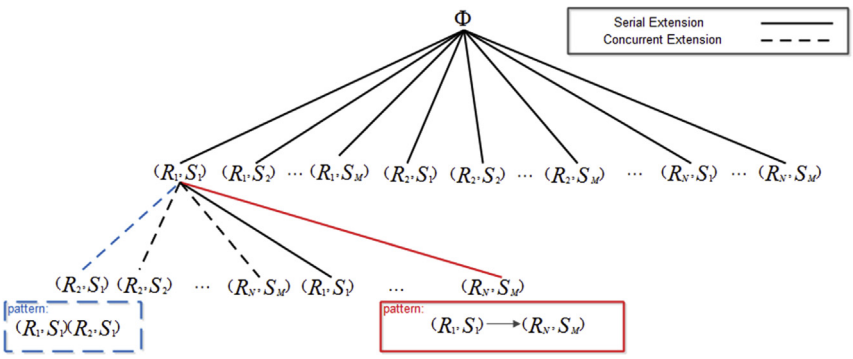


Fig. 11. A part of the lexicographic pattern tree.

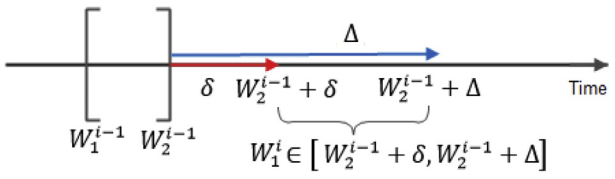


Fig. 12. Gap constraints on an occurrence $O = ([w_1^i, w_2^i]_{i=1}^k [w_1^\alpha, w_2^\alpha])$ of the episode α.

time spent on booting VMs, episodes should be extracted from the application behaviour in a way that SLA is satisfied and energy wasting is avoided. Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_k$ and an occurrence $O = ([w_1^i, w_2^i]_{i=1}^k [w_1^\alpha, w_2^\alpha])$ of α, if the time it takes to instantiate a new VM instance is $\delta(> \epsilon)$ time slots, the starting interval of $G'_i, 1 < i \le k$ should begin after $\delta + w_2^{i-1}$. Thus, the resources manager has enough time to instantiate a new VM instance. On the other hand, if resources are allocated before occurring the workload burstiness for a long time, the energy and resources are wasted. According to the discretion of the resources manager and characteristics of the cloud data center, the gap constraint $\Delta(\ge \delta)$ determines that resources might be allocated at most $\Delta - \delta$ time slots before occurring the workload burstiness. Therefore, the starting interval of G_i should begin before $\Delta + w_2^{i-1}$. As Fig. 12 shows, the valid interval of w_1^i is $[w_2^{i-1} + \delta, w_2^{i-1} + \Delta]$ to satisfy QoS and SLA and avoid wasting energy. Based on the valid intervals, a valid occurrence is defined as follows:

Definition 35. Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_k$ and the internal gaps δ and Δ, an occurrence $O = ([w_1^i, w_2^i]_{i=1}^k [w_1^\alpha, w_2^\alpha])$ of α is a **valid occurrence** iff $\forall i, 1 \le i < k, w_2^i + \delta \le w_1^{i+1} \le w_2^i + \Delta$.

The resources manager defines the parameters Δ and δ based on the characteristics and goals of the cloud data center. δ and Δ are called the minimum internal gap and the maximum internal gap respectively.

3.10. Latest occurrence

To compute the NO frequency of episodes under gap constraints, we should propose a method for finding the minimal occurrences under gap constraints. For this purpose, we introduce a new type of the occurrence for episodes, called the latest occurrence. It could be proved that this type of the occurrence is more efficient than the other types of occurrences because it considers the only one member of each equivalent class of minimal occurrences instead of checking all the members of the equivalent classes.

Based on the definitions of the serial extension and the concurrent extension, if nodes n' and n'' are the serial extension and the concurrent extension of the node n respectively, then we have:

$$\underbrace{Pattern(n')}_{\alpha} = \underbrace{Pattern(n)}_{\beta} \oplus \underbrace{Label(n')}_{x} \tag{3.12}$$

$$\underbrace{Pattern(n'')}_{\gamma} = \underbrace{Pattern(n)}_{\beta} \odot \underbrace{Label(n'')}_{y} \tag{3.13}$$

So without scanning the stream, a maximal non-overlapped set of minimal occurrences of α and γ can be determined by using the join of minimal occurrences of β with minimal occurrences of x and y respectively (Achar et al., 2013).

To present the definition of the latest occurrence, the prefix and the suffix of episodes are defined in a similar way to (Han et al., 2001) firstly. Then concepts of the valid interval for the starting intervals of *CNG*s and the latest prefix occurrence are defined.

Definition 36. Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_k$, $Prefix(\alpha, i)$, $1 \le i \le k$ is $G'_1 \to G'_2 \to \cdots \to G'_i$.

Definition 37. Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_k$, $Suffix(\alpha, i)$, $1 \le i \le k$ is $G'_i \to G'_{i+1} \to \cdots \to G'_k$.

Definition 38. Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_k$, if the starting interval of $G'_i, 1 \le i < k$ is $[t, t']$, then the **Valid Interval** (VI) for the starting interval of G'_{i+1} is $VI([t, t'], i+1) = [t' + \delta, t' + \Delta]$.

Definition 39. Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_k$, in each occurrence $O = (([t_1^i, t_2^i])_{i=1}^k, [t_1^\alpha, t_2^\alpha])$ of α, $[t_1^1, t_2^u], 1 \le u \le k$ is the starting interval of $Prefix(\alpha, u)$.

Definition 40. Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_k$, if for a valid occurrence $O = (([t_1^i, t_2^i])_{i=1}^k, [t_1^\alpha, t_2^\alpha])$ of α there exists no valid occurrence of α such as $Q = (([w_1^i, w_2^i])_{i=1}^{k-1}, [t_1^k, t_2^k], [w_1^\alpha, w_2^\alpha])$ that $\exists j, 1 \le j < k, w_1^j > t_1^j$, it is said that O includes the **latest prefix occurrence**.

Definition 41. Each valid occurrence of the episode α that includes the latest prefix occurrence, is called the **Latest Occurrence (*LO*)**. $LO(\alpha)$ is a set of all the latest occurrences of α.

To prove that the NO frequency of the episodes could be computed based on their *LO*s correctly, the following lemmas and theorem are presented.

Definition 42. Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_k$, each starting interval $[t_1^i, t_2^i]$ of $G'_i, 2 \le i \le k-1$ in the occurrence $O = (([t_1^i, t_2^i])_{i=1}^k, [t_1^\alpha, t_2^\alpha])$ is a valid occurrence of G'_i iff the gap constraints δ and Δ are satisfied between the starting intervals of (G'_i and G'_{i+1}) and (G'_{i-1} and G'_i).

Lemma 9. *Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_k$, an occurrence $O = (([t_1^i, t_2^i])_{i=1}^k, [t_1^\alpha, t_2^\alpha])$ of α is an LO iff $\forall\, i, 1 \le i \le k-1, [t_1^i, t_2^i]$ is the most recent valid occurrence of G'_i.*

Corollary 1. *There is only one occurrence from each equivalent class of minimal occurrences in $LO(\alpha)$.*

Lemma 10. *The first latest occurrence of the episode α is a minimal occurrence of the first equivalent class of the minimal occurrences.*

Lemma 11. *For each episode α, there is at most one member of each equivalent class of minimal occurrences in OS $et_M^N(\alpha)$.*

Theorem 3. *For each episode α, if the latest occurrences of equivalent classes of minimal occurrences are in OS $et_M^N(\alpha)$, then OS $et_M^N(\alpha) \subseteq LO(\alpha)$.*

Definition 43. Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_k$, $LOList(\alpha)$ includes a 4-tuple $(t_2^{k-1}, t_1^k, t_2^k, t_1^\alpha)$ for each occurrence $O = (([t_1^i, t_2^i])_{i=1}^k, [t_1^\alpha, t_2^\alpha]) \in LO(\alpha)$. $LOList(\alpha, i)$ is the $i-th$ member of $LOList(\alpha)$.

4. Foundation and formulation of POSITING

As Fig. 2 shows, the pattern mining engine and the extraction of the application behaviour are two essential steps for the prediction module. In this section, at first, the pattern mining engine is explained. In the following section, the extraction of the recent behaviour of the application is described.

4.1. Pattern discovery

The first step of POSITING is to extract the frequent episodes. For this purpose, the pattern tree is constructed and the frequent episodes are extracted. In section 4.1.1, two algorithms are proposed to identify *LOList* of episodes. Section 4.1.2 presents the main algorithms to extract the episodes for prediction. The frequency threshold of episodes and the valid interval of Δ are discussed in sections 4.1.3 and 4.1.4 respectively.

4.1.1. Computing frequency

Algorithms 1 and 2 are proposed to extract *LOList* of episodes. Algorithm 1 (SSMakeLOList) extracts the *LOList* of episodes using the serial extension. The algorithm receives $LOList(\alpha)$ and $LOList(G)$ that α is an episode and $G \in RS$, and computes $LOList(\beta)$ that $\beta = \alpha \oplus G$ without scanning the stream. Note that $LOList(G)$ is the occurrence list of G in the vertical representation of the stream. The counters i and j traverse the *LOLists* of α and G. Lines 3 to 23 consider for each *LO* of α which *LO*s of G could create an *LO* for β. Lines 5 to 9 check whether the $i-th$ *LO* of α could be the latest prefix occurrence for $j-th$ occurrence of G or not. If it is not, this *LO* of α could not be the latest prefix occurrence for the next *LO*s of G. So the next *LO*s of α are considered. For new *LO*s of α, we start from *LO*s of G that there is no latest prefix occurrence for them. In lines 10 to 12, if an *LO* of α is the latest prefix occurrence for an *LO* of G, the corresponding *LO* of β is generated and inserted in $LOList(\beta)$.

Algorithm 1 SSMakeLOList

Input: LOList(α),LOList(G) % α *is an episode*, $G \in RS$ % $LOList(G,i) = (v_i, v_i)$ % $LOList(\alpha,i) = (x_i, t_i, t'_i, t^{\alpha}_i)$
Output: LOList(β)
1: $i \leftarrow 1$
2: $j \leftarrow 1$;
3: **while** $(i \leq |LOList(\alpha)|)$ **do**
4: **while** $(j \leq |LOList(G)|)$ **do**
5: **if** $(i < |LOList(\alpha)|)$ **then**
6: **if** $(t'_{i+1} + \delta \leq v_j)$ **then**
7: break;
8: **end if**
9: **end if**
10: **if** $(t'_i + \delta \leq v_j \leq t'_i + \Delta)$ **then**
11: *add* $(t'_i, v_j, v_j, t^{\alpha}_i)$ *into* $LOList(\beta)$
12: $j++$
13: **else if** $(v_j > t'_i + \Delta)$ **then**
14: break
15: **else if** $(v_j < t'_i + \delta)$ **then**
16: $j++$;
17: **end if**
18: **end while**
19: **if** $(j > |LOList(G)|)$ **then**
20: break;
21: **end if**
22: $i++$;
23: **end while**
24: **return** $LOList(\beta)$;

Theorem 4. *Given the episode* $\alpha = G'_1 \rightarrow G'_2 \rightarrow \cdots \rightarrow G'_{k-1}$ *and* $G \in RS$, *the algorithm SSMakeLOList finds* $LOList(\beta = \alpha \oplus G)$ *correctly.*

Lemma 12. *Given the episode* α *and* $G \in RS$, *if* $|LOList(\alpha)| = q$ *and* $|LOList(G)| = p$, *then the time complexity of algorithm SSMakeLOList is* $O(p+q)$ *in the worst case and* $O(p)$ *or* $O(q)$ *in the best cases.*

Algorithm 2 (*SCMakeLOList*) extracts *LOList* of episodes using the concurrent extension. The algorithm receives $LOList(\alpha)$ and $LOList(G)$ that α is an episode and $G \in RS$, and computes $LOList(\beta)$ that $\beta = \alpha \odot G$ without scanning the stream. As it is shown in Algorithm 2, there are three cases for $LOList(\alpha)$ and $LOList(G)$. In lines 4 to 7, if the corresponding entries of $LOList(\alpha,i)$ and $LOList(G,j)$ could generate an *LO* of $\beta = \alpha \odot G$, it is inserted in $LOList(\beta)$ and both the counters i and j increase by +1. In lines 8 to 9, if $LOList(G,j)$ occurs after $LOList(\alpha,i)$, then $LOList(\alpha,i)$ should not be considered for the members after $LOList(G,j)$. So the next *LO* of α is checked. In lines 10 and 11, if $LOList(G,j)$ occurs before $LOList(\alpha,i)$, the next *LO*s of G are considered for $LOList(\alpha,i)$.

Theorem 5. *Given the episode* $\alpha = G'_1 \rightarrow G'_2 \rightarrow \cdots \rightarrow G'_k$ *and* $G \in RS$, *the algorithm SCMakeLOList finds* $LOList(\beta = \alpha \odot G)$ *correctly.*

Lemma 13. *Given the episode* α *and* $G \in RS$, *if* $|LOList(\alpha)| = q$ *and* $|LOList(G)| = p$, *then the time complexity of the algorithm SCMakeLOList is* $O(p+q)$ *in the worst case and* $O(\min(p,q))$ *in the best case.*

Algorithm 2 SCMakeLOList

Input: LOList(α),LOList(G) % α *is an episode*, $G \in RS$ % $LOList(G,i) = (v_i, v_i)$ % $LOList(\alpha,i) = (x_i, t_i, t'_i, t^{\alpha}_i)$
Output: LOList(β)
1: $i \leftarrow 1$;
2: $j \leftarrow 1$;
3: **while**$(i \leq |LOList(\alpha)|$ *and* $j \leq |LOList(G))$**do**
4: **if**$(|v_j - t_i| \leq \epsilon$ *and* $|v_j - t'_i| \leq \epsilon$ *and* $x_i + \delta \leq \min(t_i, v_j) \leq x_i + \Delta)$**then**
5: $add(x_i, \min(t_i, v_j), \max(t'_i, v_j), t^{\alpha}_i)$ *into* $LOList(\beta)$;
6: $i++$;
7: $j++$;
8: **elseif**$v_j > t'_i$**then**
9: $i++$;
10: **else**
11: $j++$;
12: **endif**
13: **endwhile**
14: **return**$LOList(\beta)$;

After extracting *LOList* of episodes, the frequency of episodes could be computed. Algorithm 3 receives *LOList* of the episode α and computes its frequency. The correctness of this algorithm is proved by the following lemmas.

Lemma 14. *Given the episode* α, *the first non-overlapped LO after a minimal occurrence in* $LOList(\alpha)$ *is a minimal occurrence.*

Lemma 15. *Given the episode* α, *the algorithm ComputeFreq (algorithm 3) computes* $|OS\,et^N_M(\alpha)|$ *and its time complexity is* $O(|LOList(\alpha)|)$.

Algorithm 3 ComputeFreq

Input: LOList(α) % α *is an episode* % $LoList(\alpha,i) = (x_i, t_i, t'_i, t^{\alpha}_i)$
Output: $freq(\alpha)$
1: $i \leftarrow 1$;
2: $freq \leftarrow 0$;
3: $endT \leftarrow 0$;
4: **while** $(i \leq |LOList(\alpha)|)$ **do**
5: **if** $(t^{\alpha}_i) \geq endT$ **then**
6: $freq++$;
7: $endT \leftarrow t'_i$
8: **end if**
9: $i++$
10: **end while**
11: **return** *freq*;

4.1.2. Closed episodes discovery

Mining frequent episodes might lead to extract a huge number of episodes. To improve the mining efficiency and avoid information loss, a compressed set of episodes, called **closed episodes**, is extracted (Tatti and Cule, 2012).

Definition 44. The episode α is closed iff there is no episode such as β that $\alpha \sqsubset \beta$ and $freq(\alpha) = freq(\beta)$ (Wang and Han, 2004).

Under gap constraints δ and Δ, in each occurrence of the episode α, there is not one occurrence for each sub-episode of α necessarily (Achar et al., 2013). It could be proved that under gap constraints, in each occurrence of α, there is an occurrence for prefixes and suffixes of α. So 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**. In (Achar et al., 2013), a two-step approach is proposed to generate closed episodes under gap constraints. In the first step, candidate closed episodes are extracted. 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 (Yan et al., 2003). Based on this approach, we present Algorithms 4 and 5 to extract closed episodes by the complete traverse of the pattern tree in a depth-first way. At first, in line 2 of Algorithm 4, all the 1-node episodes (denoted by P) are extracted. Then, the pat-tern tree is traversed in a depth-first manner from each of the 1-node episodes using the recursive calls of the algorithm *FindFreqEpisode* (lines 5–6, Algorithm 4). Note that $LOList(p)$ is the corresponding list of p in the vertical representation of the stream. The algorithm *FindFreqEpisode* (Algorithm 5) receives the threshold values θ and *Level*, the episode α and $LOList(\alpha)$, forms the concurrent and serial extensions of α (lines 6 and 20, Algorithm 5) as the episode β and computes $LOList(\beta)$ by calling *SCMakeLOList* and *SSMakeLOList* (lines 7 and 21, Algorithm 5). Then the NO frequency of β is computed by calling *ComputeFreq* (lines 8 and 22, Algorithm 5). If $fre(\beta)$ is above the threshold c (computed based on θ), the tree is traversed further down by calling *FindFreqEpisode* in lines 13 and 27 recursively with β and $LOList(\beta)$ as parameters. When the serial and the concurrent extensions of α are constructed, it is checked (in lines 9 and 23, Algorithm 5) whether any of the super patterns β formed from α has the same frequency as α's or not; if not, we add α to the list of *CandidateClosed* (line 32, Algorithm 5). Note that to avoid enlarging the pattern tree, we could limit the number of *CNG*s of episodes. We define *Level* as the maximum number of *CNG*s of episodes. After extracting the final *CandidateClosed*, a post process-ing step is performed on *CandidateClosed* using a hashing procedure with the frequency as the key (line 8, Algorithm 4) (Yan et al., 2003). In this step, all the candidates with the same frequency are hashed to the same bucket in the hash table. Among the candidate episodes which are hashed to the same bucket, those episodes for which a super-episode with the same frequency is found, are discarded. Finally, a set of all the frequent closed episodes are extracted. Note that the episodes are represented in the form of SAVE to expedite the episode extraction.

Lemma 16. *Given the episode $\alpha = G'_1 \to \cdots \to G'_k$ and $(r, s) \in RS$, the time complexity of SAVE and the matrix representation for the serial/concurrent extension of α with (r, s) is $O(1)$ and $O(\|\alpha\|)$ respectively if the time complexity of duplicating the representation of α is ignored.*

Algorithm 4 AllEpisodes

Global Variables: *CandidateClosed*;
Input: $\theta, Level$ % user-defined threshold, $0 \leq \theta \leq 1$
Output: *ClosedSet*
1: $CandidateClosed \leftarrow \emptyset$;
2: $P \leftarrow \{x | x \in RS \wedge |LOList(x)| > 0\}$;
3: *Sort P Based on Order Defined on RS*;
4: $CandidateClosed \leftarrow CandidateClosed \cup P$
5: **for each** $(p \in P)$ **do**
6: $FindFreqEpisode(\theta, Level, p, LOList(p))$;
7: **end for**
8: $ClosedSet \leftarrow CandidateProcessing(CandidateClosed)$
9: **return** *ClosedSet*

4.1.3. Frequency threshold c

In the previous works such as (Mannila et al., 1997; Toma et al., 2007; Laxman et al., 2008; Fahed et al., 2014; Achar et al., 2012a), the frequency threshold, to identify the frequent episodes, is a user-defined threshold which is selected statically. This might lead to lose some useful episodes for prediction. Since our main goal is prediction, the frequency threshold should be selected intelligently to avoid miss-ing episodes. For each episode α, we select c in Algorithm 5 based on the minimum of frequencies of its states. Therefore, if the minimum of frequencies of states of the episode α is small, c is also small. For example, assume $\alpha = (CPU, Low) \to (Memory, High)$, $|LOList(CPU, Low)| = 40$ and $|LOList(Memory, High), | = 100$. For α, we have $c = \min(40,100) \times \theta$ where $0 \leq \theta \leq 1$. Note that the coefficient θ is the user-defined threshold which is selected statically.

Algorithm 5 FindFreqEpisode

Input: $\theta, Level, \alpha, LOList(\alpha)$; % The threshold c is computed based on θ.
Output: *CandidateClosed*
1: $Flag \leftarrow True$;
2: **if** $(|\alpha.RArray| \leq Level)$ **then**
3: $F_\alpha \leftarrow ComputeFreq(LOList(\alpha))$;
4: $Q \leftarrow CExt(\alpha)$;
5: **for each** $(q \in Q)$ **do**
6: $\beta \leftarrow \alpha \odot q$;
7: $L \leftarrow SCMakeLOList(LOList(\alpha), LOList(q))$;
8: $F \leftarrow ComputeFreq(L)$;
9: **if** $(F = F_\alpha)$ **then**
10: $Flag \leftarrow False$;
11: **end if**
12: **if** $(F > c)$ **then**
13: $FindFreqEpisode(\theta, Level, \beta, L)$;
14: **end if**
15: **end for**
16: **end if**
17: **if** $(|\alpha.RArray| < Level)$ **then**
18: $Q \leftarrow SExt(\alpha)$;
19: **for each** $(q \in Q)$ **do**
20: $\beta \leftarrow \alpha \oplus q$;
21: $L \leftarrow SSMakeLOList(LOList(\alpha), LOList(q))$;
22: $F \leftarrow ComputeFreq(L)$;
23: **if** $(F = F_\alpha)$ **then**
24: $Flag \leftarrow False$;
25: **end if**
26: **if** $(F > c)$ **then**
27: $FindFreqEpisode(\theta, Level, \beta, L)$;
28: **end if**
29: **end for**
30: **end if**
31: **if** $(Flag)$ **then**
32: *add α to CandidateClosed*
33: **end if**

4.1.4. Maximum internal gap

One of the most important goals of cloud is to avoid the energy and resource wasting. If the maximum internal gap, Δ, is big, there are two problems: 1) the next time slot in which the future status will occur, is not specified exactly. So resources might be allocated to applications before the workload burstiness for a long time, which leads to the resources and energy wasting. 2) Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_k, k \geq 2$, suppose $[t_i, t'_i]$ is the starting interval of $G'_i, 1 \leq i \leq k-1$. Since for each event e, we have $\Delta e \geq \epsilon + 1$, some status might exist for resources in $VI([t_i, t'_i], i+1)$. For example, as Fig. 13 shows two events with different status s and s' are observed for the resource r in $VI([t_i, t'_i], i+1)$. Although the states (r, s) and (r, s') could be predicted by the extracted episodes, the episodes could not deter-

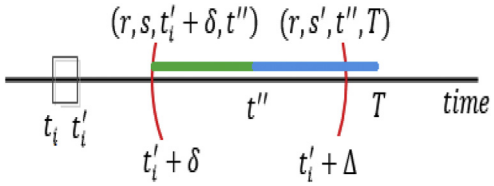


Fig. 13. The valid interval of $VI([t_i,t_i'],i+1)$ of the episode $\alpha = G_1' \to G_2' \to \cdots \to G_k'$, $k \geq 2, 1 \leq i \leq k-1$ under gap constraints and the events observed for the resource r.

mine the order of observing the states. Since the future status of the resources should be predicted precisely, Δ should be defined in a way that at most one event for each resource could be observed. For this purpose, as (4.1) implies Δ should be in the interval of $[\delta, \delta+\epsilon]$:

$$\left.\begin{array}{r}\Delta e \geq \epsilon+1 \\ VI([t_i,t_i'],i+1) = [t_i'+\delta, t_i'+\Delta]\end{array}\right\} \to \frac{\Delta-\delta+1}{\epsilon+1} \leq 1$$
$$\to \Delta \leq \delta+\epsilon \to \Delta \in [\delta,\delta+\epsilon] \qquad (4.1)$$

4.2. Observation extraction

As Fig. 2 shows, the recent behaviour of the application should be determined to predict the future behaviour based on extracted episodes.

Definition 45. 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 46. *Given the observation OB, $|OB| = k$, $1 \leq i \leq k$, $OB[i] = x^i{}_1 \cdots x^i{}_{l_i}$ and $x_l^i \in RS, 1 \leq l \leq l_i$, if e_l^i is the corresponding event of x_l^i, the intersection of $OB[i]$ and $OB[j], 1 \leq i < j \leq k$ is defined as follows:*

$$OB[i] \cap OB[j] = \{x_p^i \mid x_p^i \in OB[i], 1 \leq p \leq l_i \text{ that}$$
$$\forall q, p \leq q \leq l_i, e_1^j.st - e_q^i.st < \delta\}$$
$$OB[j] \cap OB[i] = \{x_p^j \mid x_p^j \in OB[j], 1 \leq p \leq l_j \text{ that}$$
$$\forall q, 1 \leq q \leq p, e_q^j.st - e^i{}_{l_i}.st < \delta\}$$

Lemma 17. *Given the observation OB that $|OB| = k$ and $1 \leq i < k$, if $OB[i] \cap OB[i+1] \neq \emptyset$ then one of three occurrences below is possible:*

1. *The occurrence of the episode $(OB[i] \to OB[i+1] - (OB[i+1] \cap OB[i]))$*
2. *The occurrence of $(OB[i+1])$*
3. *The occurrence of the episode $(OB[i] - (OB[i] \cap OB[i+1]) \to$ states of $OB[i+1]$ that satisfy the gap constraint $\delta)$*

Given the Observation OB that $|OB| = k$, it could be proved simply that $\nexists i,j, 1 \leq i,j \leq k$ such that $OB[i] \subseteq OB[j]$. If the gap constraints δ and Δ are satisfied, there also exists at least one serial relationship between some states of each two consecutive entries.

Lemma 18. *Given the observation OB and $1 \leq k < u \leq |OB|$, if $OB[k] \cap OB[u] = m \neq \emptyset$ and $OB[u] \cap OB[k] = n \neq \emptyset$ then*

1. *m is in the last part of $OB[k]$ and n is in the first part of $OB[u]$.*
2. *$\forall j, k+1 \leq j \leq u-1$, $OB[k] \cap OB[j] \neq \emptyset$ and $OB[j] \cap OB[u] \neq \emptyset$.*
3. *For $\epsilon \geq 2$, a serial relationship might exist between $OB[k+1]$ and $OB[u]$. $\forall j, k+2 \leq j \leq u-1$, there is no serial relationship between $OB[j]$ and $OB[u]$ that is not covered.*

The algorithm *ExtractObservation* in section Appendix B.1, is proposed to extract the observation from the recent history of the stream.

4.2.1. Consistent observation

To predict the future behaviour of applications, the most appropriate episodes should be selected from among extracted episodes based on the last behavioural observation of applications. So the form of the last behavioural observation should be consistent with episodes'. Unlike episodes, it might be no serial relationship under gap constraints between each two consecutive entries of OB necessarily. So we define **consistent** observations.

Definition 47. 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 48. 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.

Algorithm 6 is a recursive function that extracts LCOs of the observation. It has three parameters *Set*, *Suffix* and *Prefix*. *Set* includes LCOs, *Prefix* includes the processed entries of the observation and *Suffix* includes the entries that have not been processed yet. In the first call of the function, *Set* and *Prefix* are empty and *Suffix* includes the observation OB. In each call of the function, the first element of *Suffix* is processed and is added into the end of *Prefix*. After extracting all LCOs, *Suffix* is empty and *Prefix* includes the observation completely. In line 10 of the algorithm, $O_j, 1 \leq j \leq |Prefix|$, the first entry of *Prefix* that intersects with *Suffix*[1] is found. The algorithm considers three cases:

1. The LCOs whose last element is $OB[k], 1 \leq k < j$ (line 13, Algorithm 6): There is no intersection between these LCOs and *Suffix*[1]. Therefore, *Suffix*[1] is inserted in the end of these LCOs (line 14, Algorithm 6).
2. The LCOs whose last element is a subset of O_j (line 15, Algorithm 6): These LCOs could be extended by the concurrent states of *Suffix*[1] (line 16, Algorithm 6).
3. The LCOs whose last element is O_j or O_{j+1}, which is the first entry after O_j (line 17, Algorithm 6): According to Lemma 18, if $\epsilon \geq 2$, then a serial relationship might exist between O_{j+1} and *Suffix*[1]. In this case, according to Lemma 17, three occurrences could occur. So, the LCO is serially extended with the states of *Suffix*[1] that satisfy the gap constraints and two new LCOs are created (lines 18 to 25, Algorithm 6).

According to Lemma 18, the LCOs whose last element is $Prefix[p], j+1 < p \leq |Prefix|$, should not be considered. Note that in line 14 of Algorithm 6, if $x[L]$ is $Prefix[p]$, then there is no state in *Suffix*[1] that satisfies the gap constraints with $x[L]$. So these LCOs don't change. The new generated LCOs are added into a temporary list, *Bag*, which is later emptied into *Set* on coming out of the loop (lines 21 and 25) (Laxman et al., 2005). In line 3, consistent observations that are the longest, are returned.

Table 2
The last events of the stream.

Resource	Status	Start Time
CPU	Low	16
Disk	Low	17
Memory	High	18
CPU	High	19
Network	Medium	20

Example 10. Table 2 *shows the recent events of the stream. Assume* $\Delta = \delta = 3$. Fig. 14 *shows OBs and LCOs that are extracted from the stream (*Table 2*) for different values of* ϵ. *Note that since the span of the event* (*CPU*, *Low*, 16,19) *is 3, then* $\epsilon + 1 \leq 3$.

Lemma 19. *Given the observation OB,* $|OB| = k$ *and* $1 \leq i \leq k$, *if* $T(i)$ *is the number of LCOs that are extracted by processing* $OB[i]$, *then we have:*

$$\begin{cases} T(i) \leq T(i-1) + 2(F(OB[j]) + F(OB[j+1])), & \text{if} \quad i > 1 \\ T(i) = 1, & \text{if} \quad i = 1 \end{cases} \tag{4.2}$$

where $OB[j], 1 \leq j < i$, *is the first element of OB that* $OB[j] \cap OB[i] \neq \emptyset$, *and* $F(OB[p]), 1 \leq p \leq k$, *is the number of extracted LCOs whose last element is* $OB[p]$.

Theorem 6. *The algorithm ExtractLCO (*Algorithm 6*) extracts all LCOs from the observation OB.*

Lemma 20. *Given the observation OB that* $|OB| = T < Level$, *the time complexity of the algorithm ExtractLCO is* $O(T)$ *in the best case and* $O(3^T)$ *in the worst case.*

Lemma 21. *Given the observation OB that* $|OB| = T < Level$, *the number of LCOs extracted from OB by the algorithm ExtractLCO is 1 in the best case and* $O(3^{\frac{T}{2}})$ *in the worst case.*

Algorithm 6 ExtractLCO

Input: Set,Prefix,Suffix;
Output: *LCOs*
1: $Bag \leftarrow \emptyset$;
2: **if** $(suffix = \emptyset)$ **then**
3: return $\{x | x \in Set, \nexists y \in Set \text{ that } x \subseteq y\}$;
4: **end if**
5: **if** $(set = \emptyset)$ **then**
6: $Set \leftarrow suffix\ [1] \cup Set$;
7: $Prefix \leftarrow suffix\ [1]$;
8: $ExtractLCO(Set, Prefix, Suffix[2..|Suffix|])$;
9: **else**
10: $O_j \leftarrow$ The first element of *Prefix* which intersects with $Suffix[1]$;
11: **for each** $(x \in Set)$ **do**
12: $L \leftarrow |x|$;
13: **if** $(x[L] nsube O_j)$ **then**
14: $x[L+1] \leftarrow$ elements of $Suffix[1]$ that satisfy the gap constraints with $x[L]$;
15: **else if** $(x[L] \subset O_j)$ **then**
16: $x[L] \leftarrow x[L] \cup$ elements of $Suffix[1]$ that are concurrent with $x[L]$;
17: **else if** $(x[L] = O_j \text{ or } (\epsilon \geq 2 \text{ and } x[L] = O_{j+1}))$ **then**
18: $x[L+1] \leftarrow$ elements of $Suffix[1]$ that satisfy the gap constraints with $x[L]$;
19: $y[1..L-1] \leftarrow x[1..L-1]$;
20: $y[L] \leftarrow$ elements of $Suffix[1]$ that satisfy the gap constraints with $y[L-1]$;
21: add y to Bag;
22: $z[1..L-1] \leftarrow x[1..L-1]$;
23: $z[L] \leftarrow x[L] - (O_j \cap suffix[1])$;
24: $z[L+1] \leftarrow$ elements of $Suffix[1]$ that satisfy the gap constraints with $z[L]$;
25: add z to Bag;
26: **end if**
27: **end for**
28: **end if**
29: $Set \leftarrow Set \cup Bag$;
30: add $Suffix[1]$ to the end of $Prefix$;
31: $ExtractLCO(Set, Prefix, Suffix[2..|Suffix|])$;

5. Prediction model

In this section, the prediction module of POSITING is presented. Firstly, criteria used to evaluate the selected episodes and data structures used by the prediction module are explained. Then, the main core of the prediction module is presented.

5.1. Criteria for episode evaluation

To predict the future behaviour of the application precisely, we should select the most appropriate episodes based on the recent behaviour of the application. For this purpose, we should measure the strength of the episodes in terms of appropriate criteria. Given the episode $\alpha = G'_1 \rightarrow \cdots \rightarrow G'_k$, we use two criteria to evaluate episodes:

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 more *MatchScore*. The criterion *MatchScore* is defined in (5.1) where *Index* is the index of the start of *LCO* in α. As we will explain in the following section, the selected episodes should be consistent with *LCO*. If α is also consistent with the recent history before *LCO*, then $Index > 1$ and α receives higher *MatchScore*.

$$MatchScore(\alpha) = \frac{|state \in (LCO \cap \alpha)|}{|state \in LCO|} + \frac{Index - 1}{|CNG_\alpha|} \tag{5.1}$$

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 (5.2) (Mannila et al., 1997). It is clear that $freq(\alpha) \leq freq(\gamma)$ based on Lemma 8. So higher *Confidence* implies that occurring β after γ is more probable.

$$Confidence(\alpha) = \frac{freq(\alpha)}{freq(\gamma)} \tag{5.2}$$

5.2. Data structure

We introduce four data structures used by the prediction model. Fig. 15 shows the data structures *Result*, *PredictionStep*, Last Event Group (*LEG*) and *ListFreq*:

- *LEG*: It includes the most recent observed behaviour of resources. The resources that are in the *PrevSection* are sampled in each time slot. The *EventSection* includes events whose start time is in the interval of $[I_1 - 1, I_2 - 1]$. I_2 is the current time, t, and $I_1 = t - \epsilon$. Since the span of each event is at least $\epsilon + 1$ and the start time of events, st, is in the interval of $[I_1 - 1, I_2 - 1]$, we have $t \leq st + \epsilon + 1 \leq t + \epsilon$. It means that status of resources that are in the *EventSection*, is unchanged until the time slot t. So they are not sampled. Note that $EventSection \cap Prev\ Section = \emptyset$ and the union of two sections includes the most recent events of all the resources. LEG will be discussed in more detail in the following section.
- *Result*: It is composed of four parts: *Outcome* includes the predicted status of all the resources. I_1 and I_2 determine the occurrence interval of *Outcome*. Note that *Outcome* could start in each time slot of the interval of $[I_1, I_2]$. To compute the precision of prediction, a *LEG* that is the most consistent with *Outcome* is found in the interval of $[I_1, I_2]$. For this purpose, *Max* is defined as the maximum percentage of *LEG* resources that are consistent with *Outcome*.
- *PredictionStep*: It is a list of all the states that could occur in the next time slot. The list includes possible states (*Outcome*) and their information such as *Confidence* and *MatchScore*.
- *ListFreq*: It is a list of pairs of $(Prefix, Freq)$ that maintains the frequency ($Freq$) of sub-episodes ($Prefix$). The confidence of episodes is computed based on this list.

ε	OB	LCO
0	(CPU,Low) (Disk,Low) (Memory,High) (CPU,High) (Network,Medium)	(CPU,Low)→(CPU,High) (Memory,High) (Disk,Low)→(Network,Medium)
1	(CPU,Low) (Disk,Low) (Memory,High) (CPU,High) (Disk,Low) (Memory,High) (CPU,High) (Network,Medium)	(CPU,Low)(Disk,Low)→(Network,Medium) (Disk,Low)(Memory,High) (CPU,Low)→(CPU,High)(Network,Medium) (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)→(Network,Medium) (CPU,Low)→(CPU,High)(Network,Medium) (CPU,Low)(Disk,Low)(Memory,High) (Disk,Low)(Memory,High)(CPU,High) (Memory,High)(CPU,High)(Network,Medium)

Fig. 14. The observations (*OB*) and the longest consistent observations (*LCO*s) extracted from Table 2 for $\epsilon = 0, 1, 2$.

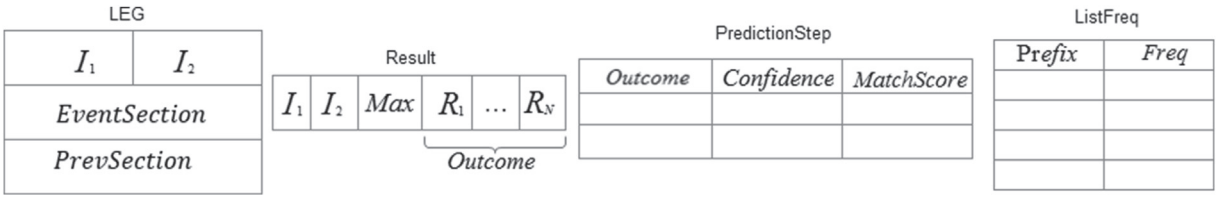


Fig. 15. Data structures used by the prediction model of POSITING.

Lemma 22. *The prediction should be performed in the steps of* $\epsilon + 1$ *time slots.*

5.3. Main core of the prediction model

Algorithm 7 explains the main core of *POSITING* to predict the future behaviour of the applications. The algorithm has three global variables *PredictionCount*, *CorrectCount* and *ResultTable*. *PredictionCount* counts the number of predictions and *CorrectCount* counts the percent-age of resources that have been predicted correctly in each predic-tion slot. *ResultTable* is a list of the data structure *Result*. The function receives three threshold values θ, *Level* and η (to select episodes) and three parameters δ, Δ and ϵ and predicts the future status of resources. T is the time slot in which the future behaviour of the application is predicted for the first time. t is the time slot in which prediction is per-formed for the first time. For prediction, we need to access the recent history of the stream. So, *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 26) 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 last prediction has been performed. *Observation* is found in line 12 and *LCO*s are extracted in line 13. In lines 14 to 16, for extracted *LCO*s, the most appropri-ate episodes are selected for the prediction. The function *Evaluating* in line 17, evaluates the matched episodes and predicts the future behaviour. In lines 18 to 25, 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. In the following section, the functions *EpisodeSelection*, *Evaluating*, *CreateLEG*, *UpdateHistory*, *ComputePrecision* and *UpdateMatchedEpisodes* are explained in more detail.

Algorithm 7 Main Algorithm

Global Variables: *PredictionCount* ← 0; *CorrectCount* ← 0; *ResultTable*: a list of *Results*
Input: $\theta, Level, \eta, \delta, \Delta, \epsilon$;
1: *PatternBase* ← *AllEpisodes*(θ, *Level*);
2: *LEG* ← *Null*
3: $t \leftarrow T - \delta$; % T is the first time slot that is predicted, t is the first time that prediction is performed (the current time)
4: *history* ← *Null*;
5: *MatchedEpisodes* ← *Null*;
6: *ResultTable* ← ∅
7: **while** (1) **do**
8: *history* ← *UpdateHistory*(*history*, *LEG*); % Observed events are added into *history*
9: *LastObservation* ← *FindLastObservation*(); % It finds the most recent events of resources
10: *LastPredicting* ← t; % It is the time that last prediction has been performed
11: *Set* ← *Null*, *Prefix* ← *Null*;
12: *Suffix* ← *ExtractObservation*(*history*, *Level*);
13: *ExtractLCO*(*Set*, *Prefix*, *Suffix*);
14: **for each** (*LCO in Set*) **do**
15: *MatchedEpisodes* ← *MatchedEpisodes* ∪ *EpisodeSelection*(*LCO*, *PatternBase*);
16: **end for**
17: *Evaluating*(*LastObservation*, *MatchedEpisodes*, t, η);
18: *LEG* ← *CreateLEG*(*LastObservation*, *true*, *LEG*);
19: **while** ($LEG.I_1 \leq$ *LastPredicting*) **do**
20: *ComputePrecision*(*LEG*);
21: *UpdateMatchedEpisodes*(*LEG*, *MatchedEpisodes*);
22: *LEG* ← *CreateLEG*(*LastObservation*, *false*, *LEG*);
23: **end while**
24: *ComputePrecision*(*LEG*);
25: *UpdateMatchedEpisodes*(*LEG*, *MatchedEpisodes*);
26: **end while**

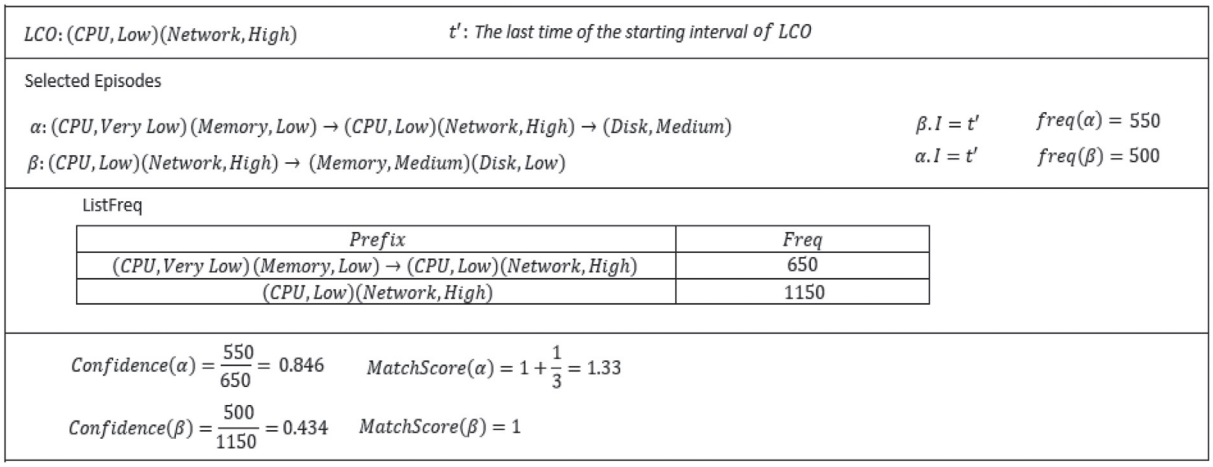


Fig. 16. An example to select and evaluate episodes based on *LCO*.

1. Function *EpisodeSelection*: Based on two defined criteria, the most appropriate episodes should be selected for prediction. We should select episodes that match *LCO*. For this purpose, the correspondence between the entries of *LCO* and the episode should be considered. 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* (Algorithm 9 in section Appendix B.2) is proposed to select these episodes. As Lemma 23 in Appendix A shows, SAVE expedites the episode selection in compared to the matrix representation.

Example 11. Fig. 16 shows *LCO* and *ListFreq*. Assume the last time of the starting interval of *LCO* is t'. Since *LCO* is in the first part of the episode β and is in the middle part of the episode α (suppose the history is consistent with (*CPU*, *Very Low*)(*Memory*, *Low*)), α and β are selected for the prediction. Note that $\alpha.I$ and $\beta.I$ are the matching time of episodes with *LCO*. According to *freq*(α), *freq*(β) and *ListFreq*, the *Confidence* and *MatchScore* of episodes are computed.

2. Function *Evaluating*: The function *Evaluating* (see Algorithm 10 in section Appendix B.3) evaluates the matched episodes and predicts the future behaviour of the application. It receives the *LastObservation*, *MatchedEpisodes*, t that is the current time slot and the threshold value η that is used to select the most confident episodes for prediction. If the future status of a resource cannot be predicted by selected episodes, the Most Recent Event (*MRE*) is extracted for prediction. *MRE* is the most recent events observed in history that match the events of *LastObservation* based on their span. The future status of each resource is predicted as the status which is spontaneously observed after *MRE*. If *MRE* is not found, the observed status of resources in *LastObservation* is considered as the future status. Finally, *Result*, which includes the predicted status of all the resources, is added into *ResultTable*.

Example 12. Consider the example 11 again. Based on Algorithm 10, the confidence of episodes in Fig. 16 and $\eta = 0.8$, the next status of *Disk* would be *Medium* (Fig. 17a). Since the other resources are not predicted by the extracted episodes, their future status is predicted based on *MRE*. Assume the observed status of CPU, memory and network after *MRE* is *Low*, *Low* and *High* respectively. The data structure *Result* in Fig. 17b includes the future status of resources and shows their starting interval is in the interval of $[t + \delta, t + \Delta]$ that t is the current time. Note that the initial value of Max is 0. It is updated while computing the precision of prediction.

3. Function *CreateLEG*: As it has been mentioned, *LEG* includes the most recent events of resources. Pruning and updating the matched episodes and computing the precision of the predicted results are based on *LEG*. The *EventSection* includes events whose start time is in the interval of $[I_1 - 1, I_2 - 1]$. I_2 is set to the current time, t, and $I_1 = t - \epsilon$. The *EventSection* is a list with $\epsilon + 1$ entries. Each entry i, $0 \le i \le \epsilon$ includes events with the start time $I_1 - 1 + i$. Since the span of each event is at least $\epsilon + 1$ and the start time of events, st, is in the interval of $[I_1 - 1, I_2 - 1]$, we have $t \le st + \epsilon + 1 \le t + \epsilon$. It means that status of resources that are in the *EventSection*, is unchanged until the time slot t. So the corresponding resources of these events do not require sampling. On the other hand, based on Lemma 22, the step of the prediction is $\epsilon + 1$. So, the next predic-tion is performed in the time slot $t + \epsilon + 1$. So for this new pre-diction, we have $I_2 = t + \epsilon + 1$ and $I_1 = t + 1$. As it is observed in Algorithm 7, when the prediction is performed *flag* is true to call the function *CreateLEG*. When the *flag* is true (see Algorithm 11 in section Appendix B.4) all the resources are sampled. Note that when *flag* is false, it means that status of resources in *EventSection* is unchanged. So the only resources of *PrevSection* are sampled.

Example 13. Assume the current time slot is 19, $\epsilon = 0$, $\delta = \Delta = 1$ and $\mu = 3$. So the next status of resources is predicted for time slot 20. *LastObservation*, which includes the last events of resources in the stream, is shown in Fig. 18. As Fig. 18 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

PredictionStep

Outcome	Confidence	MatchScore
(Disk,Medium)	0.846	1.33

(a) The data structure *PredictionStep* after calling the function *Evaluating*

Result

I_1	I_2	Max	CPU	Memory	Disk	Network
$t+\delta$	$t+\Delta$	0	Low	Low	Medium	High

(b) The data structure *Result* after calling the function *Evaluating*

Fig. 17. The predicted status of resources based on the *LCO* and extracted episodes in Fig. 16.

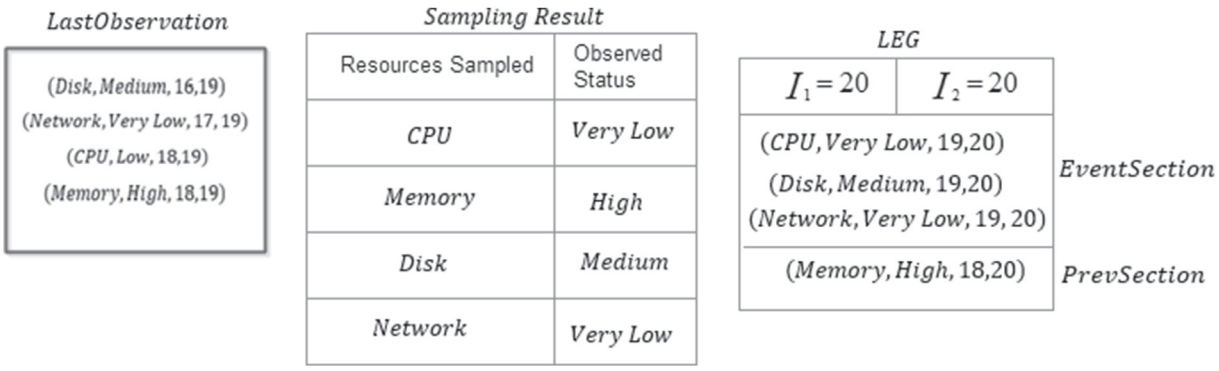


Fig. 18. An example to create *LEG*.

includes the updated last 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 $LEG.I_1 - 1$.

4. Function *UpdateHistory*: The function *UpdateHistory* (see Algorithm 12 in section Appendix B.5) updates *history* based on *LEG*. In the first call of the function, the history is empty. So it is filled with the recent events of the stream. In the following calls, the end time of the most recent observed events is updated (events of *LEG.EventSection*) and the events are added into *history*. For example, all the events of *LEG.EventSection* in the example 13 are added into *history*.
5. Function *ComputePrecision*: The function *ComputePrecision* (see Algo-rithm 13 in section Appendix B.6) prunes the entries of *ResultTable* and updates two counters *PredictionCount* and *CorrectCount*, which are used to compute the precision of prediction. It considers the entries of *ResultTable*. Each entry of *ResultTable* that has predicted the status of resources in the recent time slot is investigated. The function finds a *LEG* that is the most consistent with the predicted status of resources in the interval of $[I_1, I_2]$. Since one time slot has been observed, I_1 of the entry increases by +1. If $I_1 > I_2$, it means that the time interval predicted by the entry has passed. So one prediction has been performed and *PredictionCount* increases by +1. *CorrectCount* also increases by *Max*.

Example 14. Consider the example 13 again. Fig. 19 (a) shows the corresponding entry of *ResultTable* that predicts the time slot 20. It predicts the status of *CPU is Very Low*, *Memory is High*, *Disk is Medium* and *Network is Very Low* in the time slot 20. Since $\delta = \Delta$, there is one LEG in the interval of [20, 20]. According to the observed status of resources (LEG in Fig. 18), the status of all the resources except memory has been predicted correctly. So as Fig. 19 (b) shows Max is set to $\frac{3}{4}$ because three resources are consistent with LEG. As Fig. 19 (c) shows, I_1 of the entry increases by +1. Since $I_1 > I_2$, it means that the time interval predicted by the entry has passed. So *PredictionCount* increases by +1 and *CorrectCount* increases by Max. Since $I_1 > I_2$, this entry is removed from *ResultTable*.

6. Function *UpdateMatchedEpisodes*: The function *UpdateMatchedEpisodes* (see Algorithm 14 in section Appendix B.7) investigates all the matched episodes. The first element after ✻ of the episodes that have predicted the recent time slot is considered. If this element is consistent with *LEG* (the observed events), it is filled with ✻ and the matching time of the episodes is updated based on the start time of events of *LEG*. Finally, the episodes that have been filled with ✻ completely, are removed.

Example 15. Assume Fig. 20 (a) shows episodes used to predict time slot 20 in example 14 and the last time of the starting interval of *LCO* is 18. Note that the first part of the episodes that match *LCO* is ignored (shaded rectangles). α and β predict the status of Memory and Disk is *Low* and *Medium* respectively in the time slot 20 (the first element after ✻ in α and β is (*Memory*, *Low*) and (*Disk*, *Medium*) respectively). According to LEG in Fig. 18, α does not predict the status of memory correctly. So it is removed from *MatchedEpisodes* in the next call of the function *Evaluating* (Fig. 20(b)). The prediction result of β is consistent with LEG. As Fig. 20(c) shows the corresponding entry of predicted result is filled with ✻ and the matching time of β ($\beta.I$) is updated based on the maximum of the start time of events of LEG.

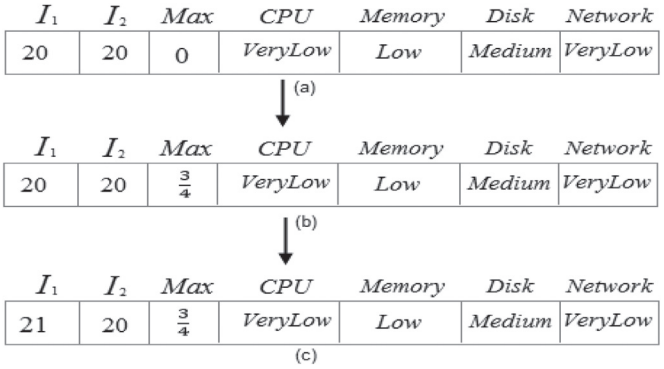


Fig. 19. Updating an entry of *ResultTable* that predicts the time slot 20.

6. Evaluation

In this section, we provide a comprehensive evaluation of POSIT-ING. The prediction precision of POSITING and the effect of the most important parameters on the episodes extraction are considered in this section. In the field of pattern mining, two types of data sets are used for evaluation: real data sets and the synthetic data generated by embedding specific patterns in noise. So we evaluate POSITING on the real and synthetic workloads. There are some parameters for POSITING: δ, Δ, ϵ, μ, *Level*, θ and η. The parameters setting for the evaluation of POSITING is as follows:

- Δ and δ: POSITING is compared to the state-of-the-art predictors such as SMA (Simple Moving Average), LR, NN and HPA. 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 has been mentioned, ϵ should be determined based on the length of sampling intervals. According to our consideration, in most of the real workloads such as Grid Workloads Archive (Iosup et al., 2008) and Google traces (Alam et al., 2015), the resources are sampled every 5 min for each VM. Since the traces are coarse-grained,

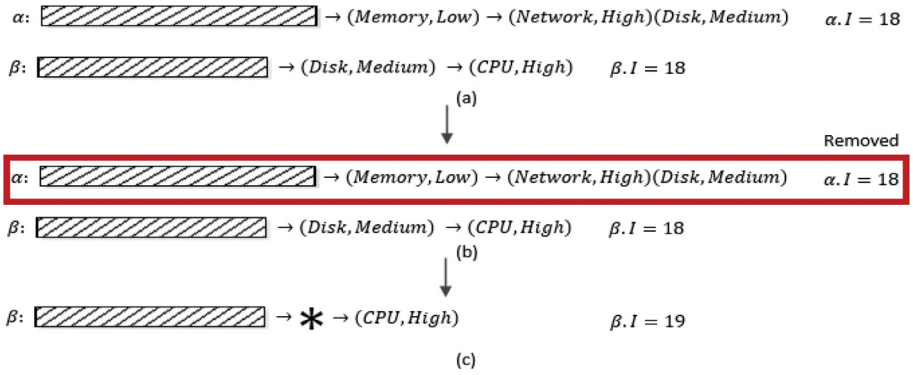


Fig. 20. Updating episodes of *MatchedEpisodes* that predicts the time slot 20.

we set $\epsilon = 0$ in all experiments.

- μ: For smooth workloads, the small values of μ might lead to generate many events, which could increase the duration of the training phase (episodes extraction). On the other hand, the large values of μ might lead to the inability to extract all the hidden useful episodes. We consider the impact of μ on the training phase of POSITING for both real and synthetic workloads.
- θ: It is a threshold value that is used to extract the frequent episodes. It is clear that the small values of θ might lead to identify the huge number of episodes, which might be very time consuming. On the other hand, the large values of θ could lead to loss the useful episodes for prediction. The impact of θ on the training phase is evaluated on both real and synthetic workloads.
- *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.
- η: The episodes are selected for the prediction based on η. The value of η should be selected in a way that an appropriate confident subset of episodes is used for prediction. The small values of η cause many episodes with low confidence to be used for prediction. It could decrease the prediction precision. On the other hand, the large values of η cause less episodes to be used for prediction and the future behaviour to be predicted based on *MRE* frequently. Therefore an appropriate value for η could provide more reliable results. The impact of η on the prediction precision is evaluated on both real and synthetic workloads.

According to the effective utilization reported for resources in some literature such as (Xi et al., 2015; Utilization), we define the abstraction alphabet as Table 3. Note that this alphabet is not unique and more/less abstract values could be defined. Furthermore, without loss of generality, the abstraction alphabet could be defined for each resource exclusively.

POSITING is compared with the state-of-the-art methods such as NN, SMA, LR and HPA. These methods are based on the sliding win-dow. Fig. 21a shows the structure of the sliding window. If the current time slot is t and the length of the sliding window is $h > 0$, each entry x_{ij}, $1 \le i \le N$, $t - 1 \le j \le t - h$ of the window is the status of the resource R_i in the time slot j. Based on the sliding window, each method is imple-mented as follows:

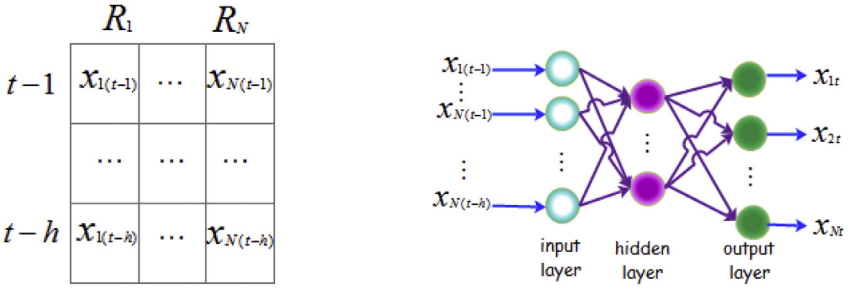
- NN: In most of the literature such as (Jiang et al., 2013; Islam et al., 2012; Amiri et al., 2016), the typical three-layer neural network is used for prediction. The neurons of the input layer take the information of the sliding window as input variables and nodes of the output layer predict the future status of resources. So as Fig. 21b shows, 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 the number of nodes in the hidden layer (*Nodes*) are two effective parameters of NNs that should be selected carefully. 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 (Jiang et al., 2013; Vazquez et al., 2015), SMA is used as a naive predictor for evaluation. For the current time slot t and the resource R_i, $1 \le i \le N$, 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 (Yang et al., 2014a) propose a linear regression model to predict the workload. According to the workload fluctuations in the sliding window, their method adjusts itself through the recomputation of parameters of the regression model. The authors show their method provides more reliable results than common regression based methods. The readers can refer to (Yang et al., 2014a) for more detail.
- HPA: Jiang et al. in (Jiang et al., 2013) 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. To implement this method, we use SMA, NN and LR as predictors. The readers could refer to (Jiang et al., 2013) for more detail.

Since the parameters have a significant effect on the prediction results of the predictors (Amiri and Mohammad-Khanli, 2017), each method is evaluated by using different values of its parameters. For SMA and AR, the length of the sliding window and for NN, the num-ber 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 com-pare POSITING with these methods, the final results predicted by the predictors are converted into the abstract values by using a simple mapping function.

In both real and synthetic workloads, without loss of generality, we assume that each application is encapsulated inside one VM as it is reported in much literature such as (Jiang et al., 2013; Garg et al., 2014; Jheng et al., 2014). The trace of each VM is considered for one month. For each VM, the stream is constructed on the first 15 days of

Table 3
The abstraction alphabet for the abstraction representation.

abstract value	Very Low	Low	Medium	High
range	[0,20%)	[20%, 50%)	[50%, 80%)	[80%, 1]



(a) The structure of the sliding window with length of h

(b) The structure of three-layer NN for the prediction

Fig. 21. The structure of the sliding window and NN.

the month. The closed episodes are extracted from the stream by calling algorithm 4. In the last 15 days of the month, the future behaviour of the application is predicted by calling algorithm 7. Note that when the prediction is performed, counters *CorrectCount* and *PredictionCount* are updated based on the predicted results. Finally, the final precision of POSITING is computed as follows:

$$Precision = \frac{CorrectCount}{PredictionCount} \tag{6.1}$$

As it has been mentioned, the time it takes to instantiate a new VM instance is about 5–15 min. On the other hand, VMs are sampled every 5 min. So three values 1,2 and 3 are evaluated for δ.

6.1. Real workload

Remarkably few workload traces are publicly available. It could also be observed that few traces include information about requested resources, and rarely include network and disk information (Shen et al., 2015). The data set GWA-T-12 Bitbrains[2] contains the performance metrics of 1750 VMs from a distributed data center from Bitbrains, which is a service provider that specializes in managed hosting and business computation for enterprises. The examples of customers are many major banks, credit card operators and insurers. Bitbrains hosts the applications used in the solvency domain.

The workload traces are corresponding to requested and actually used resources in a distributed data center servicing business-critical workloads. The data set focuses on four key types of resources, which can become bottlenecks for business-critical applications: CPU, disk I/O, memory and network I/O. In general, Bitbrains hosts three types of VMs: management servers, application servers, and compute nodes. Management servers are used for the daily operation of the customer environments such as firewalls. Database servers, web servers, and head-nodes (for compute clusters) are classified as the application servers. Compute nodes are mainly used to do simulation and compute-intensive computations, such as Monte-Carlo based financial risk assess-ment (Shen et al., 2015).

For each VM, the performance metrics are sampled every 5 min. The traces include data for 1750 nodes, with over 5000 cores and 20 TB of memory, and operationally include over 5 million CPU hours in 4 operational months. So they are long-term and large-scale time series. Table 4 shows the performance metrics reported for each VM. The resource utilization is the fraction of the used resource that is allocated to a VM (Kaur and Chana, 2014). Before the stream construction, the utilization of resources should be extracted from the traces. According to Table 4, for each time slot t, we compute the resources utilization as follows:

- The traces include the CPU utilization (*CPUU*).
- The memory utilization is computed as $\frac{MemU}{MemP}$

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

Table 4
The performance metrics and their corresponding symbols of traces in GWA-T-12.

No	Metric	Symbol
1	time stamp	*t*
2	CPU core	*CPUCo*
3	CPU capacity	*CPUCa*
4	CPU usage (in terms of MHZ)	*CPUUM*
5	CPU usage (in terms of percentage)	*CPUU*
6	memory provisioned	*MemP*
7	memory used	*MemU*
8	disk read throughput	*DiskRT*
9	disk write throughput	*DiskWT*
10	network received throughput	*NetRT*
11	network transmitted throughput	*NetTT*

- Since the usage of the disk is defined as the sum of the read throughput and the write throughput (VMware), we define the disk utilization as $\frac{DiskRT+DiskWT}{X}$, where X is the maximum of observed usage of the disk.
- Since the usage of the network is defined as the sum of the received throughput and the transmitted throughput (VMware), we define the network utilization as $\frac{NetRT+NetTT}{Y}$, where Y is the maximum of observed usage of the network.

6.1.1. Impact of parameters θ, μ and η

One VM, called VM_R, is selected randomly from GWA-T-12 to evaluate the impact of parameters θ, μ and η:

- Impact of θ: Table 5 shows the impact of θ on the training phase of VM_R for $\mu = 3$ and different values of δ: the number of episodes, candidate closed and closed episodes and time consumed (in sec-onds) to extract the closed episodes. As Table 5 shows, the small values of θ increase the number of extracted episodes (and closed episodes) and the processing time to extract these episodes. Note that the number of closed episodes is much fewer than episodes'. Since the number of the candidate closed episodes for small values of θ increases, the efficiency of the hash table based method used to identify the closed episodes decreases. In the future work, we will focus on presenting a novel efficient method to extract closed episodes directly. Although the small values of θ might lead to extract useful episodes that could predict the infrequent behaviour of the application, the training phase needs more time. So θ should be selected in way that there should be a trade-off between the processing time (to extract the closed episodes) and the prediction precision.
- Impact of μ: To evaluate the impact of μ, we set $\theta = 0.1$. Since $\mu \geq \max(2\epsilon + 1, \epsilon + 2)$ and $\epsilon = 0$, then we have $\mu \geq 2$. Since the parameter μ shows similar behaviour for different values of δ, we show the results only for one of them ($\delta = 3$). Table 6 shows the impact of μ in the interval of [2, 10] on the training phase of

Table 5
Impact of θ on the training phase of POSITING for VM_R ($\mu = 3$).

δ	θ	$\|Episodes\|$	$\|CandidateClosedEpisodes\|$	$\|ClosedEpisodes\|$	$Time(s)$
1	0.1	3826	1774	247	26.1
	0.2	1214	577	133	15
	0.3	203	125	80	7.6
	0.4	101	78	56	7.1
	0.5	50	40	35	2.9
	0.6	28	23	22	2.6
	0.7	23	20	19	2.6
	0.8	20	17	16	2
	0.9	11	10	10	0.89
	1	11	10	10	0.88
2	0.1	4372	2080	261	47.6
	0.2	611	341	136	9.1
	0.3	96	76	64	3.8
	0.4	55	51	47	3.6
	0.5	33	30	27	1.6
	0.6	18	17	17	1.5
	0.7	17	16	16	1.5
	0.8	13	16	13	1.4
	0.9	10	10	10	0.9
	1	10	10	10	0.9
3	0.1	40290	21174	435	210.4
	0.2	10283	4825	176	40.4
	0.3	6920	3149	84	24.3
	0.4	4525	2051	53	16.5
	0.5	4404	1988	41	14.1
	0.6	38	22	3	1.8
	0.7	18	18	18	1.7
	0.8	16	16	16	1.7
	0.9	14	14	14	1.7
	1	10	10	10	0.9

POSITING. Since the real workloads are almost smooth, it is clear that as μ increases, the span of events increases and the number of events decreases subsequently. So the number of episodes decreases as μ increases. It seems that for $\mu = 3$, in addition to the tolerable training time, the behavioural patterns of the workloads are also extracted.

- Impact of η: To evaluate the impact of η on the prediction preci-sion of VM_R, the parameters θ and μ should be determined. Since the main goal of cloud is to avoid SLA violation and QoS dropping, the small values of θ could extract the useful episodes to predict the rare behaviour of the application. Therefore we set $\theta = 0.1$ and $\mu = 3$. Fig. 22 shows the impact of η for different values of δ on the

Table 6
Impact of μ on the training phase of POSITING for VM_R ($\theta = 0.1$ and $\delta = 3$).

μ	$\|Episodes\|$	$\|CandidateClosedEpisodes\|$	$\|ClosedEpisodes\|$	$Time(s)$
2	130980	62917	861	884.2
3	40290	21174	435	210.4
4	2061	1211	161	9.5
5	2662	1387	135	11.6
6	1061	724	111	5.6
7	793	456	105	3.9
8	778	441	93	3.4
9	576	371	89	2.9
10	592	352	89	2.8

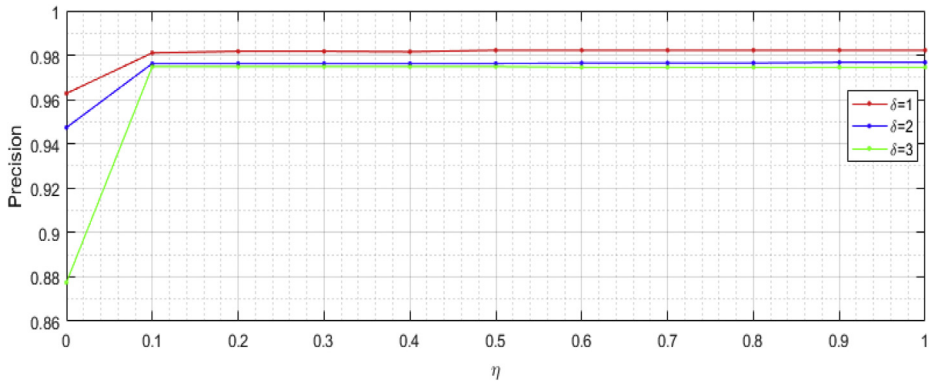
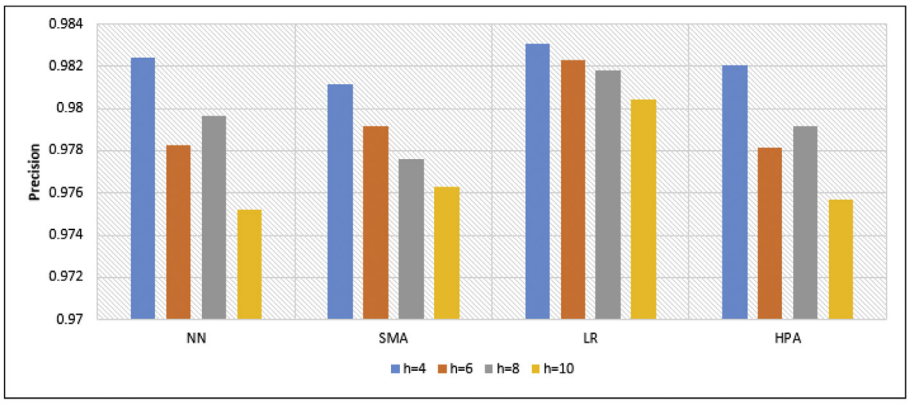
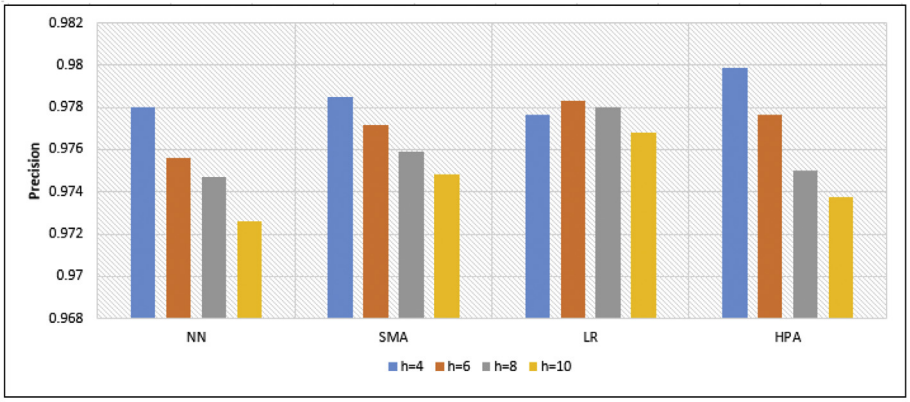


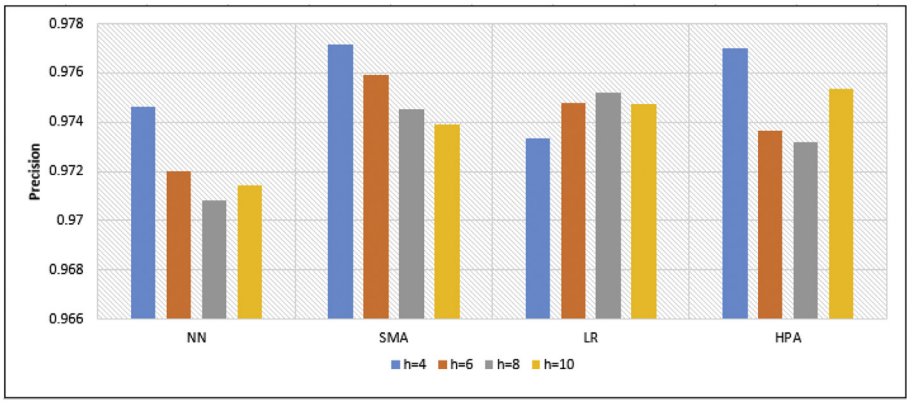
Fig. 22. Impact of η on the prediction precision of VM_R for $\delta = 1, 2, 3$ and $\theta = 0.1$.



(a) The impact of h on precision of the methods for $\delta = 1$



(b) The impact of h on precision of the methods for $\delta = 2$



(c) The impact of h on precision of the methods for $\delta = 3$

Fig. 23. The impact of the length of the sliding window (h) on the precision of the methods to predict the real workloads for different values of δ.

prediction precision of POSITING. $\eta = 0$ means that all the closed episodes, regardless of their confidence, are used for prediction. As η increases a confident sub-set of closed episodes are used for prediction. Since the workload variations of VMs in GWA-T-12 are smooth, there is no significant change in the precision for $\eta \geq 0.1$.

6.1.2. Experimental results

We select 200 VMs from GWA-T-12 randomly using a uniform distribution. To provide a fair comparison between POSITING and the other predictors, the parameters of NN, LR, SMA and HPA should also be determined carefully. The common parameter of all the methods is the length of the sliding window. For NN, the best number of nodes for the hidden layer is also found. Fig. 23 shows the impact of the length of the sliding window (h) on the methods for different values of δ. Note that since the workload variations of VMs in GWA-T-12 are smooth, there is no significant changes in the precision for different values of h. In the evaluation of the synthetic workloads, the significant influence of h on the prediction results will be shown. To compare POSITING with the other methods, we consider the average precision of results predicted by different methods on 200 VMs. According to the impact of θ and μ in Tables 5 and 6 and η in Fig. 22, we set $\theta = 0.1$, $\mu = 3$ and $\eta = 0.8$. Fig. 24 shows the average precision of POSITING, NN, SMA, LR and HPA on the predicted results of 200 VMs for different values of δ. As it is shown in Fig. 24, all the methods provide the reliable results and their precision is more than 0.96. POSITING, NN and HPA provide the most accurate results. LR has the smallest precision in compared to the other methods.

The other public workload traces such as Google workload data set (Reiss et al., 2012), include only CPU and memory characteristics (Shen et al., 2015). The evaluation results in (Shen et al., 2015) show the business-critical workloads are more dynamic than the other classes of hosted workloads. For example, according to results reported

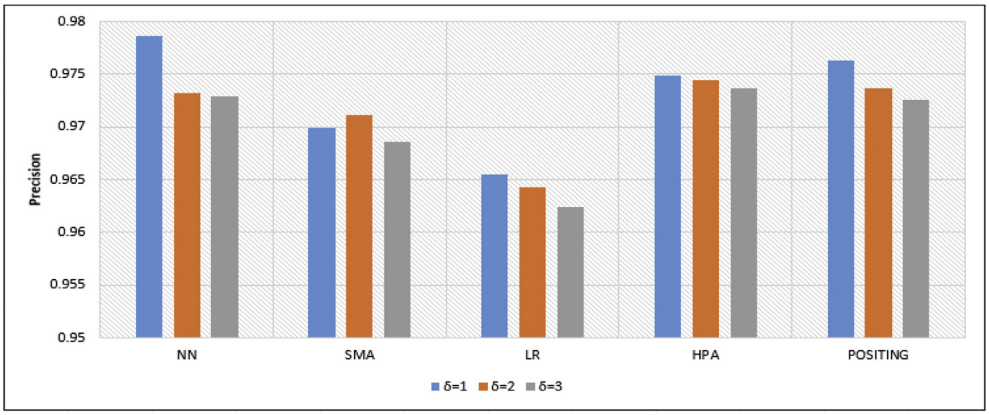


Fig. 24. The average precision of POSITING, NN, SMA, LR and HPA on the predicted results of 200 VMs.

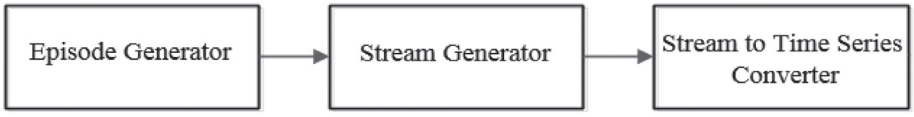


Fig. 25. The schema of the synthetic workload generator.

in (Shen et al., 2015), the actual workload of the Google trace is relatively stable, whereas the results indicate that CPU and memory workloads of GWA-T-12 Bitbrains are very unpredictable for business-critical applications. As Fig. 24 shows POSITING and the other methods could predict the future status of resources accurately. Since the workloads of GWA-T-12 is more dynamic than the other public workloads, it is clear that similar results would be achieved for the other public workloads to evaluate POSITING and the other predictors. Therefore, we generate the synthetic workloads with episodes embedded in noise and compare POSITING with the other predictors.

6.2. Synthetic workload

In the field of the pattern mining, to evaluate the effectiveness and efficiency of the algorithms, extensive experiments are performed on both synthetic and real data sets (Laxman et al., 2005; Laxman, 2006; Achar et al., 2012b). The main goal of the experiments presented in this section is to empirically demonstrate the advantage of POSITING to extract the hidden patterns and predict the future behaviour.

In a similar way to the real workloads, 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 similar to the data generator employed in (Achar et al., 2012b). Fig. 25 shows the general schema of the synthetic workload generator:

- In the first phase, based on the defined *ResourceType* and *Status*, the episode generator generates the episodes such as α that are embedded in the stream. For each episode α, it receives $|CNG_\alpha|$ and generates α randomly in a way that α is consistent with the principles defined in the paper.
- At first, the stream has 8640 empty time slots. The stream gener-ator receives the generated episodes. In a similar way to (Achar et al., 2012b), the list of non-overlapped occurrences of the episodes is generated based on the parameters ϵ and σ, where σ is the gap between each two consecutive *CNG*s. The time between the end of an occurrence and the start of the next occurrence of the episodes is distributed geometrically with a parameter that is generated by

Table 8
The evaluated values of the window length (h) and the number of nodes in the hidden layer (*Nodes*).

The Length of Window (h)	The number of Nodes (*Nodes*)
4	8,10,16,24,32
6	12,18,26,32,40,48
8	16,26,36,46,56,64
10	20,30,40,50,60,70,80

Table 7
The types of synthetic workloads and their embedded episodes.

Embedded episodes	Type of the synthetic workload	Parameters of the episode
α : $(Memory, Low)(Disk, Verylow) \rightarrow (CPU, Low)(Network, High)$	$SWT1\sigma1$	$\sigma = 1, \epsilon = 0$
	$SWT1\sigma2$	$\sigma = 2, \epsilon = 0$
	$SWT1\sigma3$	$\sigma = 3, \epsilon = 0$
β : $(CPU, Low)(Network, Low) \rightarrow (Memory, High), (Disk, Medium) \rightarrow (CPU, High), (Network, Medium)$	$SWT2\sigma1$	$\sigma = 1, \epsilon = 0$
	$SWT2\sigma2$	$\sigma = 2, \epsilon = 0$
	$SWT2\sigma3$	$\sigma = 3, \epsilon = 0$
α : $(Memory, Low)(Disk, Verylow) \rightarrow (CPU, Low)(Network, High)\beta$: $(CPU, Low)(Network, Low) \rightarrow (Memory, High), (Disk, Medium) \rightarrow (CPU, High), (Network, Medium)$	$SWT3\sigma1$	$\sigma = 1, \epsilon = 0$
	$SWT3\sigma2$	$\sigma = 2, \epsilon = 0$
	$SWT3\sigma3$	$\sigma = 3, \epsilon = 0$

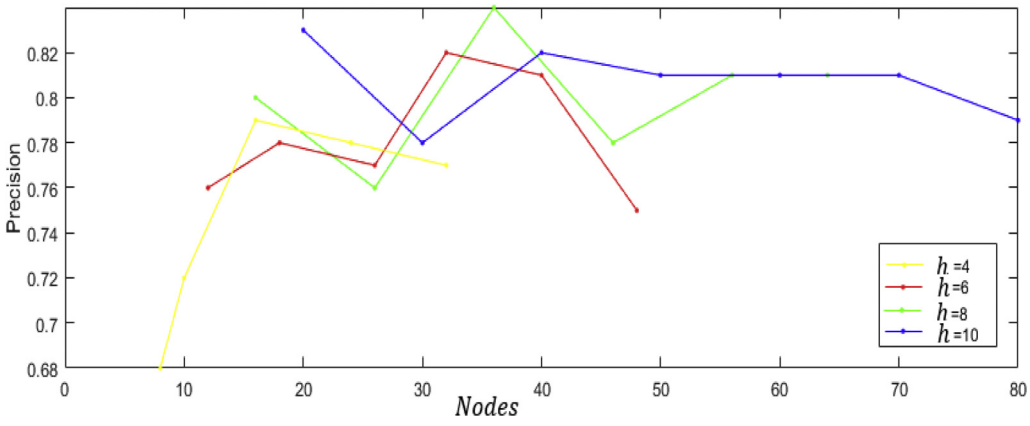


Fig. 26. The impact of h and *Nodes* on the precision of NN for $Trace_{R'}$.

using a uniform distribution randomly. After filling the stream with the episodes based on their occurrence lists, the empty slots of the stream is filled with noise, which is the events that are generated randomly based on *ResourceType* and *Status*. We have to ignore detail due to space limitation. Note that the synthetic stream is completely consistent with the principles defined in the paper.

- Although the constructed stream is the input of POSITING, the input of NN, SMA, LR and HPA is the numerical time series. Therefore, for each resource, the abstract value observed in each time slot of the stream is mapped to a numeric value based on its range randomly. Finally, the stream is converted into N numeric time series where $N = |ResourceType|$.

Most of the literature in the field of episode mining such as (Laxman et al., 2005; Laxman, 2006; Achar et al., 2012b) generate two types of synthetic data sets with embedding separately episodes with different length. We also generate two episodes α and β where $|CNG_\alpha| = 2$ and $|CNG_\beta| = 3$ using the episode generator. Table 7 shows the types of the generated synthetic workloads and their corresponding embedded episodes. To present a more comprehensive evaluation, we generate *SWT*3 in which both the episodes α and β are embedded. Since the valid values of δ could be 1, 2 or 3, we also set these values for σ to evaluate the ability of POSITING to extract episodes under different values of δ. So, for each type of workloads, three experiments with different values of δ are conducted. To avoid chance results, three different traces are generated for each type of workloads. The trace k of the workload type i (*SWTi*) with $\sigma = j$ is called *SWTiσjNk*. Therefore, totally 81 experiments are conducted on the synthetic workloads. Note that due to the length of the sampling intervals, ϵ is set to 0 in all experiments.

6.2.1. Impact of Nodes and h on the predictors

In a similar way to the real workloads, POSITING is compared with NN, SMA, LR and HPA. To demonstrate the influence of parameters on the precision of these methods, we select one trace ($Trace_{R'}$) from 27 generated traces randomly. The length of the sliding window (h) and the number of nodes in the hidden layer (*Nodes*) are two important parameters of NN. The valid interval of *Nodes* is usually defined as $[\frac{O_i}{2}, 2 \times O_i]$ where O_i is the number of nodes in the input layer of NN (Kaastra and Boyd, 1996; Wang, 1994). According to Table 8, we consider values 4, 6, 8 and 10 for h. For each value of h, based on the valid interval, some values for *Nodes* are evaluated. As Fig. 26 shows, h and *Nodes* have the significant influence on the precision of NN for $Trace_{R'}$. Fig. 27 shows the impact of h on the precision of SMA, LR and HPA. So to provide a fair comparison, the best parameters are determined for NN, SMA, LR and HPA in all the experiments.

6.2.2. Impact of parameters θ, μ and η

In this section, the impact of three parameters θ, μ and η on the num-ber of episodes, the processing time and the precision of POSITING is considered. We select the traces from different types of workloads randomly and compare the time consumed to extract the closed episodes of POSITING with time consumed to train NN for these traces.

As Table 9 shows, for each workload type, we select one trace from different types of workloads randomly. For each trace, the best struc-ture of NN should be determined by evaluating different values of h and *Nodes*. We consider the sum of the time consumed to select the parameters of NN as the training time of NN.

To investigate the impact of θ, μ is set to 3. For different values of θ, Table 10 shows the number of episodes, candidate closed and closed episodes and the total processing time of POSITING (*TimeP*). The table also includes the training time of NN (*TimeNN*). According to Table 10, as θ increases the number of episodes, candidate closed episodes and closed episodes decreases. Subsequently, the time required to extract the episodes decreases. It is clear that smaller values of θ could extract the useful episodes that predict the rare behaviour of the application. As Table 10 shows the time required to extract episodes with $\theta = 0.1$ is less than the time consumed for the parameter setting and training of NN in all the traces.

To evaluate the impact of μ, we set $\theta = 0.1$. Since $\mu \geq \max(2\epsilon + 1, \epsilon + 2)$ and $\epsilon = 0$, then we have $\mu \geq 2$. Table 11 shows the impact of μ in the interval of [2, 10] on the training phase of POSITING. Unlike the smooth workloads, there is no clear behaviour of the impact of μ on the selected traces. The increase of μ does not show clear behaviour on the training phase of $Trace_A$. On the other hand, for $\mu \geq 3$, there is no significant change on the training phase of $Trace_B$. For $\mu \geq 8$, there is no significant change on the training phase of $Trace_C$. As Table 11 shows, for all the types of the synthetic workloads, $\mu = 3$ could extract the behavioural patterns of the workloads in the tolerable training time.
So according to Tables 11 and 12, $\mu = 3$ is a good choice for both real and synthetic workloads.

Fig. 28 shows the impact of η on the precision of POSITING for $\theta = 0.1$ and $\mu = 3$. As η decreases, the precision also decreases because more unreliable episodes are used for prediction regardless of their confidence. On the other hand, as η increases more confident episodes are used for prediction. It might also cause the future behaviour to be predicted based on *MRE* frequently. As Fig. 28 shows, it seems that $\eta = 0.8$ is a good choice for all the traces.

Table 12 includes the best structure found for NN, the precision of POSITING (with $\theta = 0.1$, $\mu = 3$ and $\eta = 0.8$) and the best precision of NN. According to Tables 10 and 12, POSITING provides more precise results with less time to extract episodes in compared to NN. According to our experiment results, the time that POSITING takes to predict each time slot is almost 60 ms.[3] So it is appropriate for prediction due to the real-time nature of the cloud environment.

[3] All of the experiments run on a machine with an Intel Core 2 Duo 2.53 GHz processor and 4 GB of RAM.

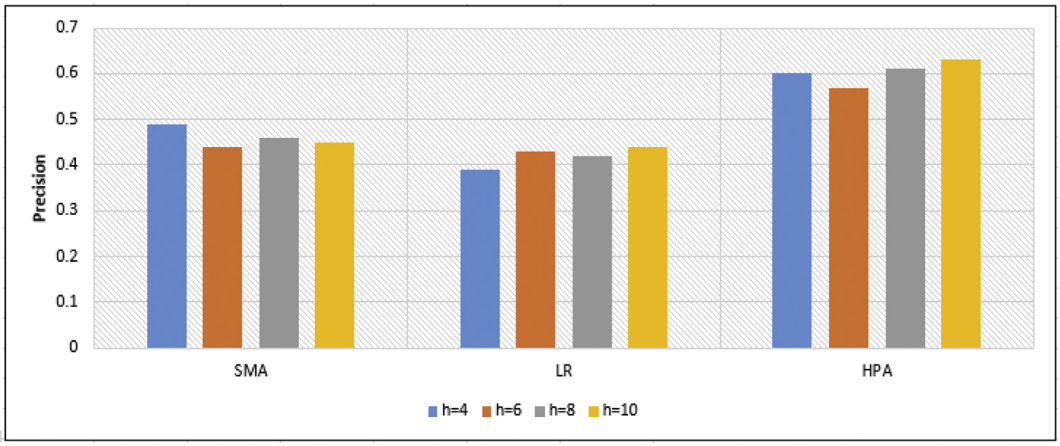


Fig. 27. The impact of h on the precision of SMA, LR and HAP for $Trace_{R'}$.

Table 9
The traces selected from different types of workloads.

Name	Type of synthetic workload	δ
$Trace_A$	$SWT1\sigma1$	$\delta = 3$
$Trace_B$	$SWT2\sigma1$	$\delta = 1$
$Trace_C$	$SWT3\sigma1$	$\delta = 2$

6.2.3. Experimental results

As it has been mentioned, for each workload type, three traces are generated. Due to space limitation, we report the average precision of three traces for each workload type. According to the experiment results in the previous section, we set $\theta = 0.1$, $\mu = 3$ and $\eta = 0.8$.

Synthetic Workloads of Type 1 ($SWT1$): As Table 7 shows, the episode α is embedded with $\sigma = 1, 2, 3$ in this type of workloads. For each workload type of this group, we evaluate the prediction precision for different values of δ. Fig. 29 compares the precision of POSITING with the other methods' for α embedded with different values of σ and valid values of δ. As it is shown in Fig. 29, POSITING outperforms the other methods to predict the future behaviour of resources for different values of δ. It is clear that due to simplicity of SMA and LR, they cannot model the dynamic behaviour of the workloads. The results also show that NN cannot model the application behaviour very well for $\delta = 2$ and 3. HPA is a hybrid approach that weak predictors such as SMA and LR causes its results to be worse than NN. Note that all the methods provide their most precise results when δ is equal to σ. However, POSITING also provides good results when $\sigma \neq \delta$.

Synthetic Workloads of Type 2 ($SWT2$): According to Table 7, the episode β is embedded with $\sigma = 1, 2, 3$ in this type of workloads. The precision of POSITING with the other methods' is compared in Fig. 30

Table 10
The impact of θ on POSITING and NN for the synthetic workloads of different types ($\mu = 3$).

Trace	$TimeNN(s)$	θ	$\lvert Episodes\rvert$	$\lvert CandidateClosedEpisodes\rvert$	$\lvert ClosedEpisodes\rvert$	$TimeP(s)$
$Trace_A$	9856	0.1	169511	127195	42	6716.508
		0.2	4623	3836	158	234.527
		0.3	514	417	79	45.158
		0.4	391	321	70	25.796
		0.5	49	44	24	5.85
		0.6	40	36	21	5.396
		0.7	40	36	21	5.399
		0.8	40	36	21	5.325
		0.9	22	18	12	3.759
		1	9	8	7	1.53
$Trace_B$	16537	0.1	743676	473002	298	15667
		0.2	681603	435283	279	12232.903
		0.3	44972	28290	119	1779.56
		0.4	38526	23500	89	1577.375
		0.5	196	167	25	15.539
		0.6	69	60	14	7.538
		0.7	60	54	10	6.289
		0.8	60	54	10	6.239
		0.9	60	54	10	6.32
		1	58	52	8	6.094
$Trace_C$	36785	0.1	120647	69262	1134	650.4
		0.2	8344	5040	493	118.426
		0.3	1730	1163	245	32.29
		0.4	349	286	115	9.212
		0.5	162	135	67	5.479
		0.6	84	67	46	3.383
		0.7	58	45	34	2.677
		0.8	39	32	26	2.262
		0.9	29	24	20	1.824
		1	23	19	16	1.566

Table 11
Impact of μ on the training phase of POSITING for $Trace_A$, $Trace_B$ and $Trace_c$ ($\theta = 0.1$).

Trace	μ	$\lvert Episodes \rvert$	$\lvert CandidateClosedEpisodes \rvert$	$\lvert ClosedEpisodes \rvert$	$Time(s)$
$Trace_A$	2	135477	109218	1782	5027.755
	3	169511	127195	442	6716.508
	4	98293	77649	1154	3401.059
	5	105317	88275	891	3453.204
	6	107382	87118	903	3865.474
	7	98142	78100	886	2899.272
	8	101336	80229	966	2967.372
	9	106493	85118	891	3177.512
	10	113295	92323	819	3513.268
$Trace_B$	2	3248758	1745792	113563	30468.63
	3	743676	473002	298	15667
	4	743676	473002	298	15684.232
	5	743676	473002	296	15407.071
	6	743676	473002	292	15289.511
	7	743676	473002	292	15258.174
	8	743676	473002	292	15274.727
	9	743676	473002	292	15286.739
	10	743676	473002	292	15279.727
$Trace_C$	2	459717	295821	62920	3754.895
	3	120647	69262	1134	650.4
	4	8912	5948	444	68.01
	5	18786	11895	499	165.179
	6	1388	1062	252	22.666
	7	1140	881	188	17.645
	8	1118	866	174	17.189
	9	1118	866	174	15.848
	10	1118	866	174	15.071

for different values of δ and σ. Fig. 30 shows the prediction results of POSITING is comparable with NN's. For $\delta = 1$, POSITING outper-forms NN significantly. For $\delta = 2$ and 3, results of POSITING are similar to NN's. For different values of δ, SMA and LR provide similar weak results. So HPA cannot provide the good prediction results. Unlike the other predictors, POSITING could provide the reliable results for different values of δ.

Synthetic Workloads of Type 3 ($SWT3$): In this type of workloads, two episodes α and β are embedded with $\sigma = 1, 2, 3$. Fig. 31 shows the precision of POSITING and the other methods for different values of δ and σ. According to Fig. 31, POSITING outperforms the other methods. Since this type of workloads is more dynamic and complicated than workloads of types 1 and 2, the precision of all methods for this type is less than two other types'. The results imply that POSITING could be a

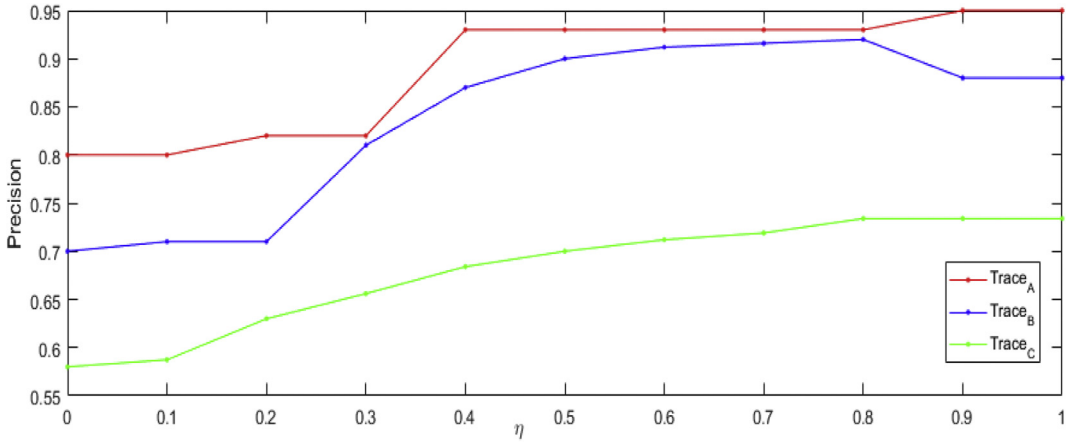
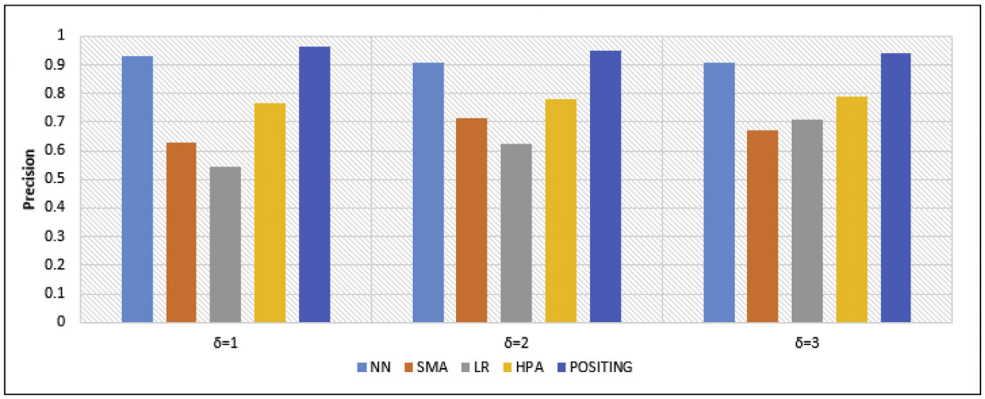


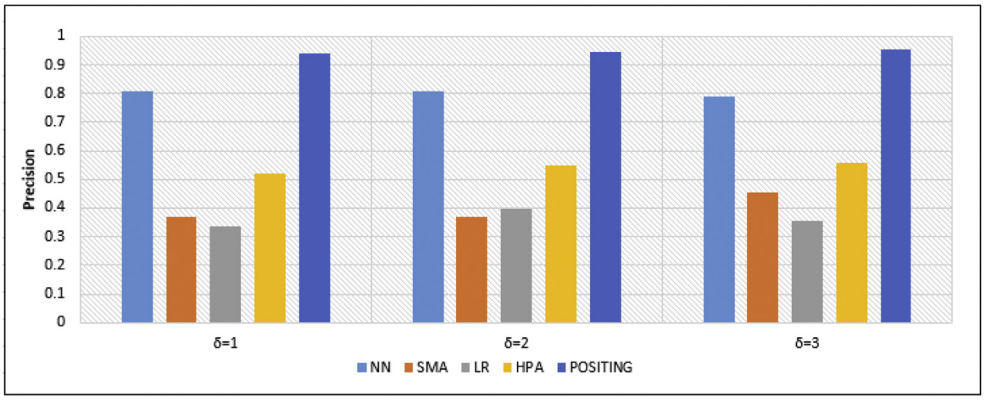
Fig. 28. Impact of η on the prediction precision of $Trace_A$, $Trace_B$ and $Trace_C$ for $\theta = 0.1$.

Table 12
The traces selected from different types of workloads.

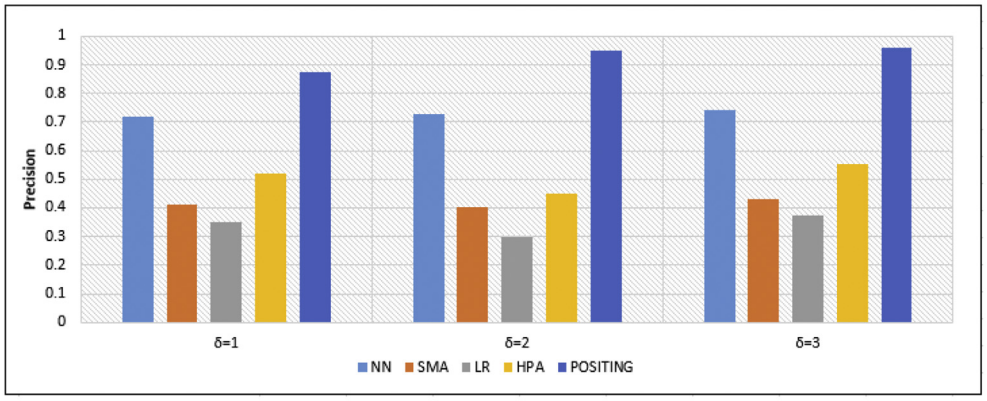
Trace	Precision of POSITING ($\theta = 0.1, \mu = 3, \eta = 0.8$)	Precision of NN	Best Structure of NN:($h, Nodes$)
$Trace_A$	0.93	0.93	(4,16)
$Trace_B$	0.916	0.76	(6,18)
$Trace_C$	0.734	0.47	(10,50)



(a) The prediction precision of POSITING and the other methods for $\sigma = 1$ and $\delta = 1, 2, 3$ on the synthetic workloads of type 1



(b) The prediction precision of POSITING and the other methods for $\sigma = 2$ and $\delta = 1, 2, 3$ on the synthetic workloads of type 1

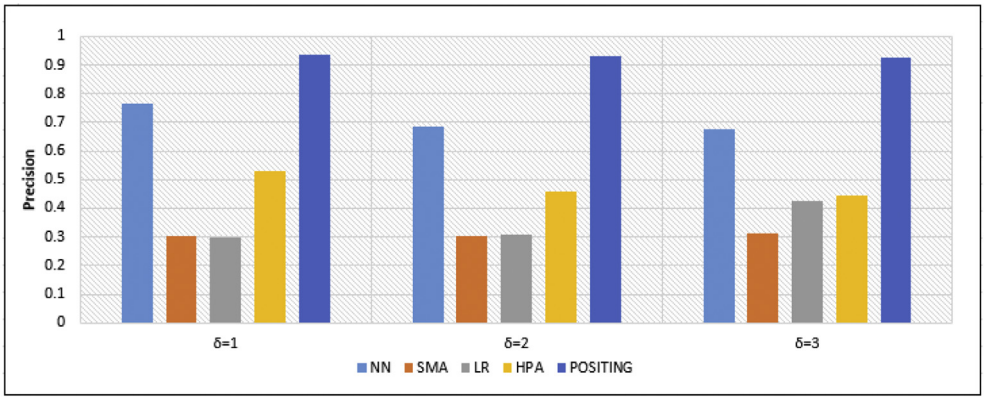


(c) The prediction precision of POSITING and the other methods for $\sigma = 3$ and $\delta = 1, 2, 3$ on the synthetic workloads of type 1

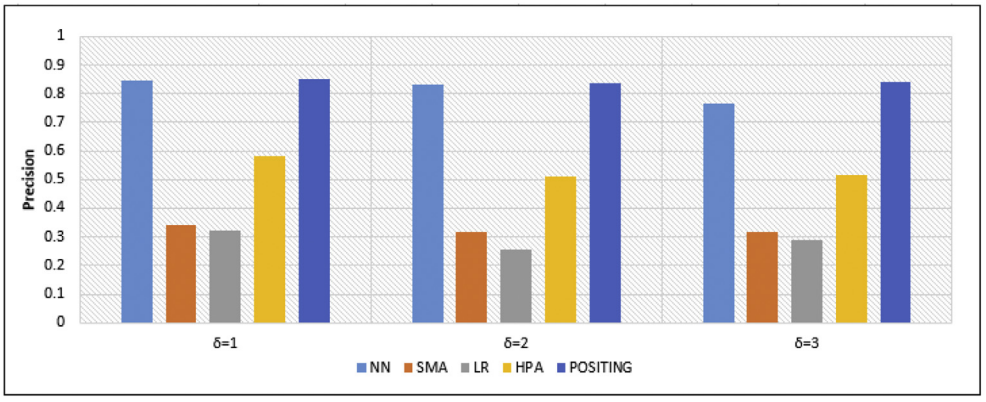
Fig. 29. The prediction precision of POSITING and the other methods for $\sigma = 1, 2, 3$ and $\delta = 1, 2, 3$ on the synthetic workloads of type 1.

good predictor for the dynamic workloads.

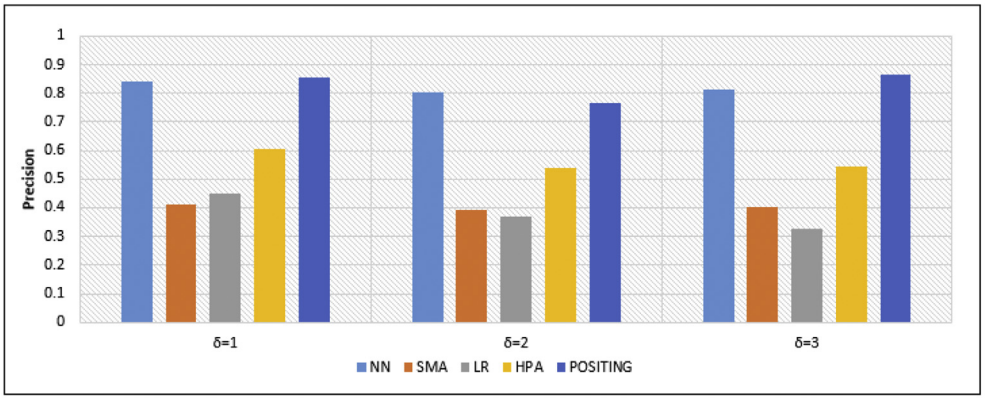
The Results Summary: It is clear that SMA is not a good predictor for dynamic workloads. Although LR is simple, its reliance is based on the oversimplified assumptions of the workload (the linear relationship). Furthermore, LR only considers the workload fluctuations in the sliding window and ignores the correlation between different resources. Therefore LR cannot capture the behavioural changes of applications very well. NN explores the history of the application behaviour for training. Although NN improves the restrictions of LR and SMA, it is not effective for extrapolation, which is very important because the existing data used to train NN might not cover the application behaviour completely. So for behaviour of the application that has not been observed in the past, the output of NN is not reliable. Furthermore, according to experiment results, the parameters setting of NN is very important



(a) The prediction precision of POSITING and the other methods for $\sigma = 1$ and $\delta = 1, 2, 3$ on the synthetic workloads of type 2



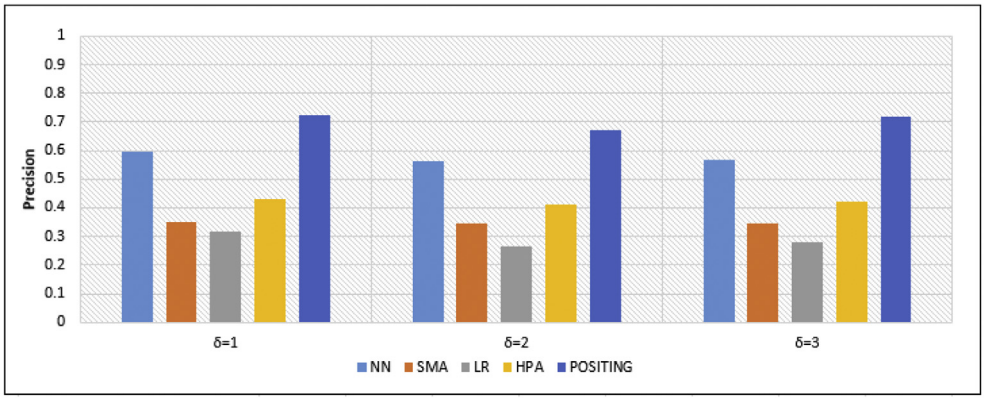
(b) The prediction precision of POSITING and the other methods for $\sigma = 2$ and $\delta = 1, 2, 3$ on the synthetic workloads of type 2



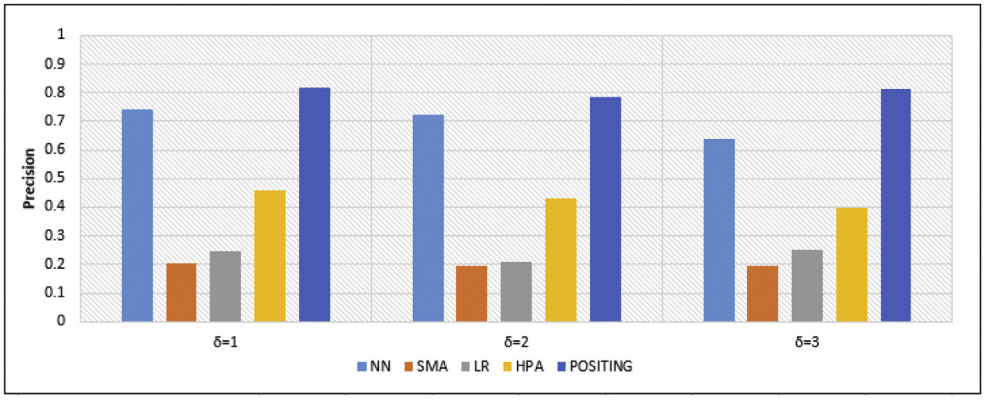
(c) The prediction precision of POSITING and the other methods for $\sigma = 3$ and $\delta = 1, 2, 3$ on the synthetic workloads of type 2

Fig. 30. The prediction precision of POSITING and the other methods for $\sigma = 1, 2, 3$ and $\delta = 1, 2, 3$ on the synthetic workloads of type 2.

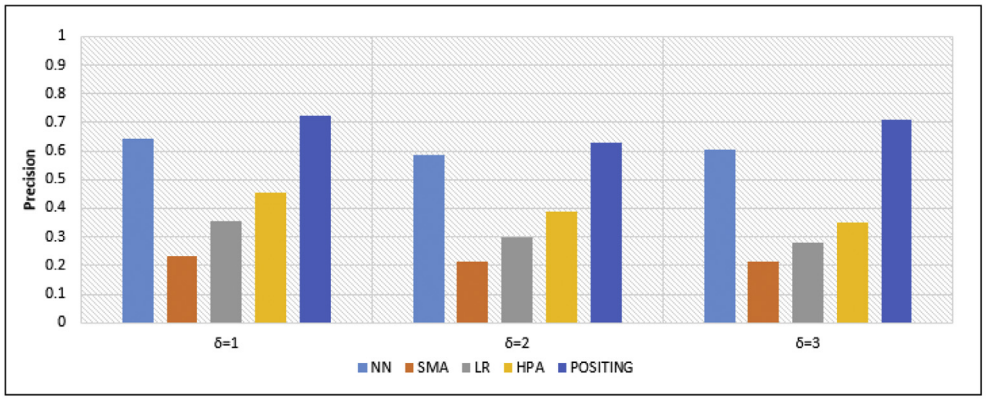
for dynamic workloads. On the contrary, POSITING extracts all the behavioural patterns of workloads independently of the fixed pattern length, observes the recent behaviour of the application and decides whether prediction should be performed based on the extracted confident patterns or *MRE*. Thus, POSITING solves the problem of extrapolation. As the prediction results show, POSITING predicts the future behaviour of the resources reliably. Furthermore, it extracts the interesting trends or patterns of the workload variations explicitly. Thus, the behavioural patterns of workloads are more readily interpretable by the resources manager.



(a) The prediction precision of POSITING and the other methods for $\sigma = 1$ and $\delta = 1, 2, 3$ on the synthetic workloads of type 3



(b) The prediction precision of POSITING and the other methods for $\sigma = 2$ and $\delta = 1, 2, 3$ on the synthetic workloads of type 3



(c) The prediction precision of POSITING and the other methods for $\sigma = 3$ and $\delta = 1, 2, 3$ on the synthetic workloads of type 3

Fig. 31. The prediction precision of POSITING and the other methods for $\sigma = 1, 2, 3$ and $\delta = 1, 2, 3$ on the synthetic workloads of type 3.

7. Conclusion and future work

The future demand prediction is an indispensable step for the rapid elasticity implementation and the effective resource provisioning in the dynamic cloud environment. Most of the prevalent predictors such as NN and LR are based on the fixed pattern length. They cannot extract all 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. For the first time, this paper proposes POSITING that extracts all the behavioural patterns of workloads independently of the fixed pattern length. It focuses on unearthing the interesting trends or patterns of the workload variations explicitly. POSITING investigates the correlation between resources and extracts the corresponding patterns. Thus, the behavioural patterns of workloads are more readily interpretable by the resources manager. The experiment results show that POSITING outperforms the state-of-the-art predictors and provides reliable results.

In the future work, we focus on proposing a new approach to extract the closed episodes directly in a way that closed episodes are extracted

more efficient. Furthermore, the behavioural changes of the application workload might start after extracting the episodes. To adapt to the workload changes, POSITING should be able to be adapted according to the workload variations. For this purpose, we plan to investigate the capabilities of online learning and decreasing the prediction error with time for POSITING.

Acknowledgement

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. Proofs

The proof of all of the theorems, lemmas and corollaries are presented in this appendix. Furthermore, we present some new lemmas that are used to prove the other lemmas and theorems.

Lemma 1. *For the event* $e = (r, s, st, et)$: $1 + \epsilon \leq \Delta e \leq \mu + \epsilon$.

Proof. According to Definition 2, $\Delta e \geq \epsilon + 1$. If $\Delta e > \mu$, the event e is decomposed based on μ : $\Delta e = k\mu + \varsigma$, $k \in \mathbb{N}$. If $\varsigma \leq \epsilon$, then the latest and penultimate decomposed events merge together to create an event e' that $\Delta e' = \mu + \varsigma$. Since $\varsigma \leq \epsilon$, then $\Delta e' \leq \mu + \epsilon$.

Lemma 2. *For the episode* $\alpha = (V_\alpha, \prec_\alpha, g_\alpha)$*:*

$\forall v_1, v_2 \in V_\alpha$ *that* $v_1, v_2 \in G_i, i \in \{1, \ldots, k\}$, *if* $g_\alpha(v_1).r = g_\alpha(v_2).r$, *there is no occurrence for* α.

Proof. The proof is by contradiction: suppose there is at least one occurrence for α. According to Definition 11, nodes of the episode have the corresponding events in the stream such that the partial order of the episode is preserved. So for v_1 and v_2 there exist the corresponding events such as $e' = (r, s', st', et')$ and $e = (r, s, st, et)$ that $st - st' \leq \epsilon$. Since each resource has specified status in each time slot, $et' \leq st$. So $\Delta e' \leq \epsilon$, which is in contradiction to $\Delta e' > \epsilon$.

Lemma 3. *In each occurrence of the episode* α *in the stream* E*, there exists an occurrence of all the sub-episodes of* α.

Proof. According to Definition 16, all states of each sub-episode and the partial order between them exist in α. In each occurrence of α, all states of α have the corresponding events in the stream such that the partial order of α is preserved. So in each occurrence of α, there exists an occurrence of all the sub-episodes of α.

Lemma 23. *Given the episode* $\alpha = (V_\alpha, \prec_\alpha, g_\alpha)$ *in the form of the matrix representation* (W_α, R_α)*:*

1. $\sum_{p=1}^{\|\alpha\|-1} \sum_{q=p+1}^{\|\alpha\|} R_\alpha(W_\alpha[p], W_\alpha[q]) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} l_i l_j$.
2. $\forall A_j^i, i \in \{1, \ldots, k\}, j \in \{1, \ldots, l_i\}$: $\sum_{v \in FN(A_j^i)} R_\alpha(g_\alpha(A_j^i), g_\alpha(v)) = \sum_{t=i+1}^{k} l_t$
3. $\forall A_j^i, i \in \{1, \ldots, k\}, j \in \{1, \ldots, l_i\}$: $|FN(A_j^i)| - \sum_{t=i+1}^{k} l_t = l_i - j$
4. $\sum_{z=2}^{\|\alpha\|} R_\alpha(W_\alpha[1], W_\alpha[z]) \geq 1$

Proof

1. Since each node of $G_i, 1 \leq i < k$ has the consecutive relation with all the nodes of $G_j, i+1 \leq j \leq k$, the number of the consecutive relations for all the nodes of G_i is $\sum_{j=i+1}^{k} l_i l_j$. Since $1 \leq i < k$, then the number of all consecutive relations of the episode α is $\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} l_i l_j$.
2. Since each A_j^i has the consecutive relation with all nodes of $G_t, i+1 \leq t \leq k$, the sum of values 1 in the corresponding row of A_j^i is $\sum_{t=i+1}^{k} l_t$.
3. The number of entries of the corresponding row of $g_\alpha(A_j^i)$ is $|FN(A_j^i)|$. According to the previous item, the number of values 1 in this row is $\sum_{t=i+1}^{k} l_t$. Therefore, the number of values 0 in this row is $|FN(A_j^i)| - \sum_{t=i+1}^{k} l_t$. It is clear that the number of values 0 is equal to the number of nodes $A_z^i, j+1 \leq z \leq l_i$.
4. The Proof is by contradiction: suppose $\sum_{z=2}^{\|\alpha\|} R_\alpha(W_\alpha[1], W_\alpha[z]) = 0$. It means $|CNG_\alpha| = 1$, which is in contradiction to Definition 9.

Lemma 24. *Given the episode* $\alpha = (V_\alpha, \prec_\alpha, g_\alpha)$:
$\forall A_j^i \in V_\alpha, i \in \{1, \ldots, k\}, j \in \{1, \ldots, l_i\}$ *if* $v_1 \in FN(A_j^i)$ *then* $FN(v_1) \subset FN(A_j^i)$

Proof. since $v_1 \in FN(A_j^i)$, $\forall w \in FN(v_1)$, w is also the following node of A_j^i. So $w \in FN(A_j^i)$. Since there is at least one member such as v_1 that $v_1 \in FN(A_j^i)$ and $v_1 \notin FN(v_1)$, so we have $FN(v_1) \subset FN(A_j^i)$. □

Lemma 25. *Given the episode* $\alpha = (W_\alpha, R_\alpha)$, $\forall\ 1 \leq i < j < z \leq \|\alpha\|$: *if* $R_\alpha(W_\alpha[i], W_\alpha[j]) = 0$ *and* $R_\alpha(W_\alpha[i], W_\alpha[z]) = 0$, *then* $R_\alpha(W_\alpha[j], W_\alpha[z]) = 0$

Proof. The proof is by contradiction: suppose $R_\alpha(W_\alpha[j], W_\alpha[z]) = 1$. It means $W_\alpha[z]$ and $W_\alpha[j]$ have the consecutive relation. So $W_\alpha[z]$ appears in the *CNGs* after $W_\alpha[j]$'s. On the other hand, $R_\alpha(W_\alpha[i], W_\alpha[j]) = 0$, which means $W_\alpha[i]$ and $W_\alpha[j]$ are in the same *CNG*. So $R_\alpha(W_\alpha[i], W_\alpha[z]) = 1$, which is in contradiction to $R_\alpha(W_\alpha[i], W_\alpha[z]) = 0$.

Lemma 26. Given the episode $\alpha = (W_\alpha, R_\alpha)$, $\forall\ 1 \leq i < j < z \leq \|\alpha\|$: *if* $R_\alpha(W_\alpha[i], W_\alpha[j]) = 0$ *and* $R_\alpha(W_\alpha[i], W_\alpha[z]) = 1$, then $R_\alpha(W_\alpha[j], W_\alpha[z]) = 1$

Proof. Since $W_\alpha[i]$ and $W_\alpha[j]$ are in the same *CNG* and $W_\alpha[i]$ and $W_\alpha[z]$ are in the sequential *CNGs*, then $W_\alpha[j]$ and $W_\alpha[z]$ are also in the sequential *CNGs*. Therefore, $R_\alpha(W_\alpha[j], W_\alpha[z]) = 1$.

Lemma 27. *Given the episode* $\alpha = (W_\alpha, R_\alpha)$*:*
$\forall A^i_z \in V_\alpha, i \in \{1, \dots, k\}, j < z \le l_i$ *and* $v_1 \in FN(A^i_z) : R_\alpha(g_\alpha(A^i_z), g_\alpha(v_1)) = R_\alpha(g_\alpha(A^i_j), g_\alpha(v_1))$

Proof. According to Definition 17, $A^i_z \in FN(A^i_j)$. Since $v_1 \in FN(A^i_z)$, according to Lemma 24, we have $v_1 \in FN(A^i_j)$. Therefore $R_\alpha(g_\alpha(A^i_j), g_\alpha(A^i_z)) = 0$ and according to Lemmas 25 and 26 $R_\alpha(g_\alpha(A^i_z), g_\alpha(v_1)) = R_\alpha(g_\alpha(A^i_j), g_\alpha(v_1))$.

Theorem 1. *Given the episode* $\alpha = G'_1 \to \cdots \to G'_k$ *in the form of* $\alpha = (W_\alpha, R_\alpha)$*, the number of inferable and redundant entries of* R_α *is:*

$$\sum_{i=1}^{k}(l_i - 1)\left(\sum_{t=i}^{k} l_t - \left(\frac{l_i}{2} + 1\right)\right) \tag{3.5}$$

Proof. According to Lemma 8, all entries of the corresponding rows of $g_\alpha(A^i_j)$, $i \in \{1, \dots, k\}$, $1 < j \le l_i$ could be inferred from $R_\alpha(g_\alpha(A^i_1), g_\alpha(v_1))$ that $v_1 \in FN(A^i_1)$. According to Lemma 4, the number of redundant values 0 for each G_i, $1 \le i \le k$, is:

$$\sum_{t=2}^{l_i}(l_i - t) = \frac{(l_i - 1)(l_i - 2)}{2} \tag{A.1}$$

According to Lemma 23, the number of redundant values 1 for each G_i, $1 \le i \le k$, is:

$$(l_i - 1)\sum_{t=i+1}^{k} l_t \tag{A.2}$$

Based on (A.1) and (A.2), the total number of redundant values 1 and 0 in R_α is:

$$\sum_{i=1}^{k}\left(\frac{(l_i - 1)(l_i - 2)}{2} + (l_i - 1)\sum_{t=i+1}^{k} l_t\right) = \sum_{i=1}^{k}(l_i - 1)\left(\sum_{t=i}^{k} l_t - \left(\frac{l_i}{2} + 1\right)\right) \tag{A.3}$$

Lemma 28. *Given the episode* $\alpha = G'_1 \to \cdots \to G'_k$ *in the form of* $\alpha =< RArray_\alpha >$, $\forall i, 1 \le i \le k : |RArray[i].GList| = l_i - 1$.

Proof. The first member of each $G'_i, 1 \le i \le k$ is inserted in $RArray[i].x$. So the other $l_i - 1$ members of G'_i are inserted in $RArray[i].GList$.

Lemma 4. *The space complexity of SAVE for the episode* α *is* $O(\|\alpha\|)$.

Proof. Based on Lemma 28:

$$\sum_{i=1}^{k}|RArray[i].GList| = \sum_{i=1}^{k}(l_i - 1) = \|\alpha\| - k \tag{A.4}$$

According to Definition 18, $|RArray| = k$. Therefore the space complexity of SAVE is $O(\|\alpha\|)$.

Lemma 29. *Given the stream* $E =< e_1, \dots, e_n >$, $\forall e_i, e_j \in E, 1 \le i < j \le n$, *if* $r_i = r_j$ *and* $st_j = et_i$*, then* ($s_i \ne s_j$ *and* $et_i - st_i \le \mu + \epsilon$*) or* ($s_i = s_j$ *and* $et_i - st_i = \mu$*).*

Proof. According to the abstraction representation of the time series, if $s_i = s_j$, it means that the status of the resource r_i has not changed in the interval of $[st_i, et_j)$. Indeed, the main event is (r_i, s_i, st_i, et_j) that is decomposed based on μ. So $\Delta e_i = \mu$. If $s_i \ne s_j$, the maximum span of the event e_i is $\mu + \epsilon$.

Lemma 5. *Given the episode* α *that* $|CNG_\alpha| = k$ *and* $\mu \ge \max(2\epsilon + 1, \epsilon + 2)$*, the successive starting intervals of* G_i, $1 \le i \le k$ *have no overlap.*

Proof. The proof is by contradiction: suppose there are two starting intervals of G_i such as $[t_1, t_2]$ and $[t', t']$, that there is overlap between them: $t_1 \le t' \le t_2 < t'$. According to Definition 11, $t_2 - t_1 \le \epsilon$ and $t' - t' \le \epsilon$. $\forall e = (r, s, st, et) \in E$ *that* $t_2 < st \le t'$, then there should exist the other events such as $e' = (r, s', st', et')$ such that $s' = s$ and $t_1 \le st' \le t_2$. Since $et' \le st$, if $t' \le st' \le t_2$ then $\Delta e' < \epsilon$. If $t_1 \le st' < t'$ and $et' = st$, according to Lemma 29, $\Delta e' = \mu < 2\epsilon$, which is in contradiction to $\mu > 2\epsilon$. If $t_1 \le st' < t'$ and $et' < st$, there should exist the other event such as $e'' = (r, s'' \ne s, st'', et'')$ where $et' \le st'' < et'' \le st'$. Since $\Delta e' > \epsilon$, we have $et' > t_2$ and $\Delta e'' < \epsilon$. These show that the successive starting intervals of G_i have no overlap.

Lemma 6. *Given the episode* α *that* $|CNG_\alpha| = k$, $\mu \ge \max(2\epsilon + 1, \epsilon + 2)$ *and two occurrences* $O, O' \in OS\ et(\alpha)$*, if* $[u, u']$ *is the starting interval of* G_i, $1 \le i \le k$ *in* O*, the following starting interval of* G_i *in* O' *is* $[w, w']$ *that* $w > 2\epsilon + u$.

Proof. According to Definition 11 for the occurrence O, $\exists A^i_j \in G_i, 1 \le i \le k, j \in \{1, \dots, l_i\}$ that $g_\alpha(A^i_j) = (r, s)$, $h(A^i_j) = a$ and $e_a = (r, s, st = u, et)$. Since $\Delta e_a > \epsilon$, $et > \epsilon + u$. According to Lemma 5, $w > u'$. For the occurrence O', $h'(A^i_j) = b$ such that $e'_b = (r' = r, s' = s, st' = v, et')$, $w \le v \le w'$, $\Delta e'_b > \epsilon$ and $et' > v + \epsilon$. If $st' > et$, there should exist the other event such as $e_m = (r, s_m \ne s, st_m, et_m)$ that $st_m = et$. Since $\Delta e_a > \epsilon$ and $\Delta e_m > \epsilon$, then $w > 2\epsilon + u$. If $st' = et$, according to Lemma 29, $\Delta e_a = \mu$. So $w - u = \mu > 2\epsilon$. □

Lemma 7. *If* $O'(\alpha) = \{O'_1, \dots, O'_F\}$ *is a set of non-overlapped minimal occurrences where* O', $1 < i \le F$ *is the first non-overlapped minimal occurrence after* O'_{i-1} *and* $O'_1 \notin OS\ et^N_M(\alpha)$*, then for* O'_1 *with the span of* $[w_1, w'_1]$*, there is a unique occurrence* $O_z \in OS\ et^N_M$, $1 \le z \le L$ *with the span of* $[u_z, u'_z]$ *where* $u_z < w_1 < u'_z < w'_1$*. For* $O'_i, i > 1$ *with the span of* $[w_i, w'_i]$*, there is a unique occurrence* $O_t \in OS\ et^N_M, z < t \le L$ *with the span of* $[u_t, u'_t]$ *where* $u_t < w_i < u'_t < w'_i$ *or* ($u_t = w_i$ *and* $u'_t = w'_i$*).*

Proof. The proof is by induction. **Base case** for O'_1: Since $O'_1 \notin OS\ et^N_M(\alpha)$ and $O_1 \in OS\ et^N_M(\alpha)$ is the first minimal occurrence of α, $O_1 < O'_1$. Since $O_1 < O'_1$, there are three cases:

1. if $u_1 < w_1 < u'_1 < w'_1$, the lemma is proved.

2. if $u_1 < u'_1 < w_1 < w'_1$, the proof is by contradiction. Suppose $\nexists\ O_z \in OS\ et^N_M(\alpha)$ that $u_z < w_1 < u'_z < w'_1$. It means that $\forall O_p \in OS\ et^N_M(\alpha), 1 \le p \le L$ that $u_p < w_1$, we have $u'_p < w_1$ and $\forall O_q \in OS\ et^N_M(\alpha), 1 \le q \le L$ that $u_q \ge w_1$, we have $u_q > w'_1$. Assume $O_M \in OS\ et^N_M(\alpha), 1 \le M \le L$, is the last occurrence before O'_1 and $O_N \in OS\ et^N_M(\alpha), 1 \le N \le L$, is the first occurrence after O'_1. Since $O_M < O'_1 < O_N$ and there is no overlap between them, then O'_1 is the first non-overlapped minimal occurrence after O_M. So we should have $O'_1 \in OS\ et^N_M(\alpha)$, which is in contradiction to $O'_1 \notin OS\ et^N_M(\alpha)$.
3. if $u_1 < u'_1 = w_1 < w'_1$, according to Definition 23, O_1 and O'_1 are non-overlapped occurrences. So $O'_1 \in OS\ et^N_M(\alpha)$, which is in contradiction to $O'_1 \notin OS\ et^N_M(\alpha)$.

Therefore, for O'_1, $\exists\ O_z \in OS\ et^N_M(\alpha), 1 \le z \le L$ that $u_k < w_1 < u'_k < w'_1$. **Induction step:** Assume it is true for $O'_2, \dots, O'_i$. Then there are two cases:

1. $\exists O_p \in OS\ et^N_M(\alpha), z < p < L$ that $u_p = w_i$ and $u'_p = w'_i$. Since O_{p+1} is the first minimal occurrence after O_p and $[u_p, u'_p] = [w_i, w'_i]$, O_{p+1} is the first minimal occurrence after O'_i. So $[u_{p+1}, u'_{p+1}] = [w_{i+1}, w'_{i+1}]$. Note that if $p = L$, since $[u_p, u'_p] = [w_i, w'_i]$, O'_{i+1} is a non-overlapped minimal occurrence after O_L. This is a contradiction since there is no non-overlapped minimal occurrence after O_L.
2. $\exists O_p \in OS\ et^N_M(\alpha), z < p < L$ that $u_p < w_i < u'_p < w'_i$. If $w'_i < u_{p+1}$, O_{p+1} is the first non-overlapped minimal occurrence after both O_p and w'_i and $[u_{p+1}, u'_{p+1}] = [w_{i+1}, w'_{i+1}]$. If $w'_i > u_{p+1}$, since O'_i is a minimal occurrence, we have $w'_i < u'_{p+1}$. Here, $u_{p+1} < w_{i+1} < u'_{p+1} < w'_{i+1}$ or ($w_{i+1} = u_{p+2}$ and $w'_{i+1} = u'_{p+2}$). Note that O'_{i+1} cannot start in the span of $[u'_{p+1}, u_{p+2}]$ because if $w_{i+1} < u_{p+2}$, then O_{p+2} is not the first minimal occurrence after O_{p+1}.

Now, we should prove that there is an injective mapping between each $O'_z \in O'(\alpha), 1 \le z \le F$ and its corresponding $O_i \in OS\ et^N_M(\alpha), 1 \le i \le L$. Two cases should be considered:

1. The proof is by contradiction: Suppose $\exists O_z \in OS\ et^N_M(\alpha), 1 \le z \le L$ that is mapped into both $O'_i, O'_j \in O'(\alpha), 1 \le i < j \le L$. Since occurrences are non-overlapped, we have $w_i < w'_i < w_j < w'_j$. Since O_z is mapped into both O'_i and O'_j, then $u_z < w_i < w'_i < w_j < u'_z$. It means that O_z is not the minimal occurrence. If $u_z = w_i$ and $u'_z = w'_i$, we have $w_j > u'_k$ that means both O'_i and O'_j are not mapped into O_z.
2. The proof is by contradiction: Suppose both $O_i, O_j \in OS\ et^N_M(\alpha), 1 \le i < j \le L$ are mapped into one $O'_z \in O'(\alpha), 1 \le z \le F$. So O_i and O_j are not non-overlapped or one of them cannot be the minimal occurrence.

Theorem 2. *$OS\ et^N_M(\alpha)$ is a maximal non-overlapped set of minimal occurrences of the episode α in the stream: $freq(\alpha) = |OS\ et^N_M(\alpha)|$*

Proof. Let $O'(\alpha) = \{O'_1, \dots, O'_F\}$ is a set of non-overlapped minimal occurrences that $O'_i, 1 < i \le F$, is the first non-overlapped minimal occurrence after O'_{i-1}. Since $O_1 \in OS\ et^N_M(\alpha)$ is the first minimal occurrence of α, we have $O_1 < O'_1$ or $O_1 = O'_1$. If $O_1 < O'_1$, according to Lemma 7, for each member of $O'(\alpha)$ there is a unique corresponding member of $OS\ et^N_M(\alpha)$. So $F \le L$. If $O_1 = O'_1$, each $O_i \in OS\ et^N_M(\alpha), 2 \le i \le \min(L, F)$, is the first non-overlapped minimal occurrence after both O_{i-1} and O'_{i-1}. Since there is no non-overlapped minimal occurrence after O_L, we have $F \le L$. Therefore, $OS\ et^N_M(\alpha)$ is a maximal non-overlapped set of minimal occurrences of the episode α.

Lemma 8. *Given the episodes α and β and the threshold $\theta \in \mathbb{R}_{\ge 0}$, if $\beta \sqsubseteq \alpha$ and $freq(\alpha) \ge \theta$, then $\theta \le freq(\alpha) \le freq(\beta)$ (the anti-monotonic constraint).*

Proof. Since $\beta \sqsubseteq \alpha$, according to Lemma 3, each occurrence of the episode α includes an occurrence of β. So $\forall O_i \in OS\ et^N_M(\alpha), \exists O'_i \subseteq O_i$ *that* $O'_i \in OS\ et^N_M(\beta)$. Therefore, $|OS\ et^N_M(\alpha)| \le |OS\ et^N_M(\beta)|$ *or* $freq(\alpha) \le freq(\beta)$.

Lemma 9. *Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_k$, an occurrence $O = (([t^i_1, t^i_2])^k_{i=1}, [t^\alpha_1, t^\alpha_2])$ of α is an LO iff $\forall\ i, 1 \le i \le k-1, [t^i_1, t^i_2]$ is the most recent valid occurrence of G'_i.*

Proof. Since this is an "If, and Only If" lemma, we must prove two implications. Proof "Only if" by contradiction: suppose $\exists j, 1 \le j \le k-1$ so that $[t^j_1, t^j_2]$ is not the most recent valid occurrence of G'_j. So there is the other valid occurrence of G'_j such as $[w^j_1, w^j_2]$ that $w^j_1 > t^j_1$. Therefore, there is the other valid occurrence of α such as $Q = (([t^i_1, t^i_2])^{j-1}_{i=1}, ([w^j_1, w^j_2]), ([t^i_1, t^i_2])^k_{i=j+1}, [t^\alpha_1, t^\alpha_2])$. Then $O \notin LO(\alpha)$.

Proof "if" by contradiction: assume $O \notin LO(\alpha)$. It means that there is a valid occurrence of α such as $Q = (([w^i_1, w^i_2])^{k-1}_{i=1}, [t^k_1, t^k_2], [w^\alpha_1, w^\alpha_2])$ that $\exists j, 1 \le j \le k-1, w^j_1 > t^j_1$. So $([t^j_1, t^j_2])$ is not the most recent valid occurrence of G'_j.

Corollary 1. *There is only one occurrence from each equivalent class of minimal occurrences in $LO(\alpha)$.*

Proof. There is at least one occurrence such as O from each equivalent class of minimal occurrences whose starting intervals are the most recent valid occurrences. So we have $O \in LO(\alpha)$. It is clear that there does not exist the other occurrence (such as Q) of this equivalent class that $Q \in LO(\alpha)$ because the starting interval of at least one *CNG* of O is greater than Q's.

Lemma 10. *The first latest occurrence of the episode α is a minimal occurrence of the first equivalent class of the minimal occurrences.*

Proof. The proof is by contradiction: assume the first latest occurrence of α is not a minimal occurrence of the first equivalent class of minimal occurrences. According to corollary 1, there is a member of each equivalent class of minimal occurrences in $LO(\alpha)$. There are two cases: 1) The first latest occurrence is not a minimal occurrence. If an *LO* is not a minimal occurrence, then there exists the other latest occurrence before that. This is a contradiction since it is not the first latest occurrence. So the first latest occurrence of α is a minimal occurrence. 2) It is not a minimal occurrence of the first equivalent class. So according to corollary 1, there are the minimal occurrences of the previous equivalent classes. Then it is not the first member of $LO(\alpha)$.

Lemma 11. *For each episode α, there is at most one member of each equivalent class of minimal occurrences in OS $et_M^N(\alpha)$.*

Proof. The proof is by contradiction: assume there are at least two minimal occurrences of an equivalent class in *OS* $et_M^N(\alpha)$. So there is overlap between them, which is in contradiction to Definition 23. Therefore, there is at most one minimal occurrence of each equivalent class in *OS* $et_M^N(\alpha)$.

Theorem 3. *For each episode α, if the latest occurrences of equivalent classes of minimal occurrences are in OS $et_M^N(\alpha)$, then OS $et_M^N(\alpha) \subseteq LO(\alpha)$.*

Proof. According to Definition 25, *OS* $et_M^N(\alpha) = \{O_1, \dots, O_L\}$. Based on Lemma 10, $O_1 \in LO(\alpha)$. Based on corollary 1, there is one member of each equivalent class in $LO(\alpha)$ and based on Lemma 11, there is at most one member of each equivalent class in *OS* $et_M^N(\alpha)$. On the other hand, an arbitrary member of each equivalent class could be in *OS* $et_M^N(\alpha)$. So if the latest occurrences of equivalent classes of minimal occurrences are in *OS* $et_M^N(\alpha)$, then *OS* $et_M^N(\alpha) \subseteq LO(\alpha)$.

Theorem 4. *Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_{k-1}$ and $G \in RS$, the algorithm SSMakeLOList finds LOList($\beta = \alpha \oplus G$) correctly.*

Proof. To prove this theorem, we focus on the span of *LO*s in *LOList* of episodes. The proof of the theorem includes two parts: 1) Occurrences extracted by the algorithm are *LO*. Given $\alpha = G'_1 \to G'_2 \to \cdots \to G'_{k-1}$ and $G' \in RS$, we have $\beta = G'_1 \to G'_2 \to \cdots G'_{k-1} \to G'$. The proof is by contradiction: there is at least one extracted occurrence of β that is not the latest occurrence. For this occurrence, assume there are corresponding occurrences O_α and O_G of α and G whose span is $[r, r']$ and $[x, x']$ respectively. There are two cases: a) The gap constraints have not been satisfied, which is impossible due to line 10 of the algorithm. b) There is the other *LO* of the episode α such as Q_α with the span $[u, u']$ for the episode α that satisfies the gap constraints for $[x, x']$ and $u \geq r$, $u' > r'$. Otherwise, $[r, r']$ and $[x, x']$ could form an *LO* for β. So, we have:

$$\left.\begin{array}{l} r' + \delta \leq x \leq r' + \Delta \\ u \geq r, u' > r' \\ u' + \delta \leq x \leq u' + \Delta \end{array}\right\} \to r' + \delta < u' + \delta \leq x \leq r' + \Delta < u' + \Delta \tag{A.5}$$

Since $[r, r']$ is before $[u, u']$, $\forall [f, f'] \in LOList(\alpha)$ that $r \leq f \leq u$, we have:

$$r' < f' < u' \text{ and } r' + \delta < f' + \delta < u' + \delta \leq x \leq r' + \Delta < f' + \Delta < u' + \Delta \tag{A.6}$$

Since line 6 of the algorithm is satisfied for O_α, so $[r, r']$ could not be the latest prefix occurrence. 2) All the latest occurrences of β are extracted. The Proof is by contradiction: there is at least an *LO* of β such as O_β, composed of O_α and O_G with spans $[r, r']$ and $[x, x']$ respectively, which is not extracted. Since *LOList*(G) is complete, there are two cases for O_α: a) $[r, r'] \in LOList(\alpha)$: it is checked in the first while loop. If $[x, x']$ is not checked for $[r, r']$, it means that the other latest prefix occurrence has been found for it previously. So $[r, r']$ could not be the latest Prefix occurrence. When $[x, x']$ is checked for $[r, r']$, if $[u, u']$ is after $[r, r']$ in *LOList*(α), then $u' > r'$. If $u' + \delta < x$, then an *LO* with the span $[r, x']$ could not be created. If $x > u' + \Delta$, then $x > r' + \Delta$ and $[r, r']$ could not be the latest prefix occurrence. If $x \leq u' + \Delta$, $[r, x']$ could not also be the latest occurrence for α. So if $[r, r'] \in LOList(\alpha)$, $[r, x']$ is extracted by the algorithm for *LOList*(β) b) If $[r, r'] \notin LOList(\alpha)$, so there is the latest occurrence with the span $[u, r']$ that $u \geq r$. Since r' satisfies the gap constraints with x', the latest occurrence with the span $[u, r']$ also satisfies the gap constraints with $[x, x']$. So there is another valid occurrence with the span $[u, x']$ that $\exists j, 1 \leq j \leq k-2$ that the starting interval of G' in the span $[u, r']$ is greater than its corresponding starting interval in $[r, r']$. So the occurrence with the span $[r, x']$ is not an *LO*.

Lemma 12. *Given the episode α and $G \in RS$, if $|LOList(\alpha)| = q$ and $|LOList(G)| = p$, then the time complexity of algorithm SSMakeLOList is $O(p + q)$ in the worst case and $O(p)$ or $O(q)$ in the best cases.*

Proof. Generally, the time complexity of the algorithm *SSMakeLOList* is $O(k\% \times p + q - f)$, $0 \leq k \leq 100$, $0 \leq f \leq q$. It implies that there is a direct relationship between f and k. It means that when $k\%$ of *LOList*(G) have been traversed by f elements of *LOList*(α), a member of *LOList*(G) is met that should be compared with $q - f$ elements of *LOList*(α). The worst case is when the first element of *LOList*(α) connects to all $p - 1$ elements of *LOList*(G) and the last element of *LOList*(G) connects to no element of *LOList*(α). So the time complexity is $O(p + q)$. The best case is when the first element of *LOList*(α) connects to all members of *LOList*(G) or the first element of *LOList*(G) connects to no element of *LOList*(α). For these cases, the time complexity is $O(p)$ and $O(q)$ respectively.

Theorem 5. *Given the episode $\alpha = G'_1 \to G'_2 \to \cdots \to G'_k$ and $G \in RS$, the algorithm SCMakeLOList finds LOList($\beta = \alpha \odot G$) correctly.*

Proof. The proof of the theorem includes two parts: 1) Occurrences extracted by the algorithm are *LO*. According to the definition of the concurrent extension, we have $\beta = G'_1 \to G'_2 \to \cdots \to (G'_k \cup G = G')$. Since *LOList*($\beta$) is constructed based on *LOList*(α), so all the occurrences extracted by the algorithm satisfy the definition of *LO*. 2) All the latest occurrences of β are extracted. The proof is by contradiction: there is at least an *LO* of β that is not extracted. Since each *LO* of β includes one *LO* of α and one *LO* of G, then it means that *LOList*(α) or *LOList*(G) is not complete or the algorithm could not find this *LO* of β. Since *LOList*(α) and *LOList*(G) are complete and line 4 of the algorithm checks the concurrent extensions of α with G, all possible *LO*s of β are extracted.

Lemma 13. *Given the episode α and $G \in RS$, if $|LOList(\alpha)| = q$ and $|LOList(G)| = p$, then the time complexity of the algorithm SCMakeLOList is $O(p + q)$ in the worst case and $O(\min(p, q))$ in the best case.*

Proof. In the best case, each element of *LOList*(α) matches an element of *LOList*(G) and both the counters i and j increase. So, the loop repeats $\min(p, q)$ times. In the worst case, one element of *LOList*(α) is checked with $t, 1 \leq t \leq p$ elements of *LOList*(G), then one element of *LOList*(G) is checked with $n, 1 \leq n \leq q$ elements of *LOList*(α) and the process is repeated in the same way. Thus, both the *LOList*(α) and *LOList*(β) are considered. So the time complexity is $O(p + q)$.

Lemma 14. *Given the episode α, the first non-overlapped LO after a minimal occurrence in LOList(α) is a minimal occurrence.*

Proof. The proof is by contradiction: assume O is a minimal *LO* of α with span $[x, r']$ ($[r, r']$ is the starting interval of the last *CNG* of α) and Q with

span $[y, u']$ ($[u, u']$ is the starting interval of the last *CNG* of α) is the first non-overlapped occurrence after O which is not minimal. So there is a starting interval for the last *CNG* of α such as $[z, z']$ that $z < u$. Therefore, there is another occurrence such as H with span $[y, z']$ that causes Q not to be minimal. Since $y > r'$, then H is also a non-overlapped occurrence and since $z < u$, then Q could not be the first non-overlapped occurrence after $[x, r']$. So, the first non-overlapped occurrence after O is a minimal occurrence.

Lemma 15. *Given the episode α, the first non-overlapped LO after a minimal occurrence in LOList(α) is a minimal occurrence.*

Proof. Based on theorem 3, we have $OS\ et_M^N(\alpha) \subseteq LOList(\alpha)$. Based on Lemma 10, the first member of $LOList(\alpha)$ is the first member of $OS\ et_M^N(\alpha)$. Since $t_1^\alpha > 0$, so the frequency of α increases by +1. According to lemma 14, the first non-overlapped occurrence after a minimal occurrence in $LOList(\alpha)$ is a minimal occurrence. Thus, the algorithm counts the number of non-overlapped minimal occurrences with the start of the first minimal occurrence. It is equivalent to $|OS\ et_M^N(\alpha)|$. Since the algorithm traverses $LOList(\alpha)$, so its time complexity is $O(|LOList(\alpha)|)$

Lemma 16. *Given the episode $\alpha = G'_1 \to \cdots \to G'_k$ and $(r, s) \in RS$, the time complexity of SAVE and the matrix representation for the serial/concurrent extension of α with (r, s) is $O(1)$ and $O(\|\alpha\|)$ respectively if the time complexity of duplicating the representation of α is ignored.*

Proof. For the serial extension of α with the state $(r, s) \in RS$, a new entry is added to the end of $RArray_\alpha$. It means that $RArray_\alpha[|CNG_\alpha| + 1].x$ is set to (r, s) and $RArray_\alpha[|CNG_\alpha| + 1].GList$ is empty. For the concurrent extension of α with (r, s), this state is added to the end of $RArray_\alpha[|CNG_\alpha|].GList$ easily. On the contrary, in the matrix representation, the matrix should be reconstructed for the extensions because the number of rows and columns of the matrix changes. Furthermore, the entries of the matrix should be filled based on partial order between states. Therefore, if the time complexity of duplicating the representation of α is ignored:

- The time complexity of the pattern extension is $O(1)$ for SAVE.
- Since in the matrix representation, the new row and column added for (r, s) should be filled based on the partial order between (r, s) and the other states, the time complexity of the matrix representation is $O(\|\alpha\|)$.

Lemma 17. *Given the observation OB that $|OB| = k$ and $1 \le i < k$, if $OB[i] \cap OB[i+1] \neq \emptyset$ then one of three occurrences below is possible:*

1. *The occurrence of the episode $(OB[i] \to OB[i+1] - (OB[i+1] \cap OB[i]))$*
2. *The occurrence of $(OB[i+1])$*
3. *The occurrence of the episode $\left(OB[i] - (OB[i] \cap OB[i+1]) \to \text{states of } OB\left[i+1\right] \text{ that satisfy the gap constraint } \delta\right)$*

Proof. Since $OB[i] \cap OB[i+1] \neq \emptyset$, it means that $OB[i]$ has not occurred before $OB[i+1]$ under gap constraints. So The occurrence of the episode $(OB[i] \to OB[i+1] - (OB[i+1] \cap OB[i]))$ or $OB[i+1]$ is possible. If $OB[i] \cap OB[i+1]$ is ignored and the gap constraints are satisfied, then according to Definition 45, $OB[i] - (OB[i] \cap OB[i+1])$ has occurred before states of $OB[i+1]$ that satisfy the gap constraint δ. Therefore there is an occurrence of the episode $\left(OB[i] - \left(OB[i] \cap OB\left[i+1\right]\right) \to \text{states of } OB\left[i+1\right] \text{ that satisfy the gap constraint } \delta\right)$.

Lemma 18. *Given the observation OB and $1 \le k < u \le |OB|$, if $OB[k] \cap OB[u] = m \neq \emptyset$ and $OB[u] \cap OB[k] = n \neq \emptyset$ then*

1. *m is in the last part of $OB[k]$ and n is in the first part of $OB[u]$.*
2. *$\forall j, k+1 \le j \le u-1$, $OB[k] \cap OB[j] \neq \emptyset$ and $OB[j] \cap OB[u] \neq \emptyset$.*
3. *For $\epsilon \ge 2$, a serial relationship might exist between $OB[k+1]$ and $OB[u]$. $\forall j, k+2 \le j \le u-1$, there is no serial relationship between $OB[j]$ and $OB[u]$ that is not covered.*

Proof.

1. According to Definition 46, it is clear that m is in the last part of $OB[k]$ and n is in the first part of $OB[u]$.
2. According to Definition 45, the start times of the corresponding events of $OB[j]$ are between the start time of the second event of $OB[k]$ and the start time of the penultimate event of $OB[u]$. If $OB[j]$ includes the corresponding state of an event of $OB[k]$, it certainly includes the corresponding state of the last event of $OB[k]$. In a similar way, If $OB[j]$ includes the corresponding state of an event of $OB[u]$, it also includes the corresponding state of the first event of $OB[u]$. Therefore, there are some events in $OB[j]$ whose start times are equal to or greater than the start time of the last event of $OB[k]$ or start times are less than or equal to the start time of the first event of $OB[u]$. Since $OB[k] \cap OB[u] \neq \emptyset$, it is clear that $OB[k] \cap OB[j] \neq \emptyset$ and $OB[j] \cap OB[u] \neq \emptyset$.
3. Suppose there is an event such as e in $OB[p]$, $k < p < u$, in a way that e is not in $OB[k]$, x is its start time and there is a serial relationship between it and an event of $OB[u]$. Let A be the maximum of the start times of the corresponding events of m, B be the minimum of the start times of the corresponding events of n and C be the start time of the event of $OB[u]$ that $x + \delta \le C$ (the serial relationship). Since $C - B \le \epsilon$ and $B < A + \delta$, we have $C - \delta < A + \epsilon$. So $x \le C - \delta < A + \epsilon$. It means that e is concurrent with the last event of $OB[k]$ and they are in $OB[k+1]$. It is clear that for $\epsilon \le 1$, there is no event such as e in $OB[k+1]$. So, for $\epsilon \ge 2$, a serial relationship might exist between $OB[k+1]$ and $OB[u]$. Since e might also be in $OB[j]$, the serial relationships between $OB[j]$ and $OB[u]$ are subsets of the serial relationship between $OB[k+1]$ and $OB[u]$.

Lemma 19. *Given the observation OB, $|OB| = k$ and $1 \le i \le k$, if $T(i)$ is the number of LCOs that are extracted by processing $OB[i]$, then we have:*

$$\begin{cases} T(i) \le T(i-1) + 2(F(OB[j]) + F(OB[j+1])), & \text{if } \ i > 1 \\ T(i) = 1, & \text{if } \ i = 1 \end{cases} \tag{4.2}$$

where $OB[j], 1 \le j < i$, is the first element of OB that $OB[j] \cap OB[i] \neq \emptyset$, and $F(OB[p]), 1 \le p \le k$, is the number of extracted LCOs whose last element is $OB[p]$.

To prove the lemma, we focus on the corresponding events of the states of *OB* without loss of generality.

Proof. The proof is by induction. **Base case** for $i = 2$: It is clear that for $i = 2$ there are at most three *LCO*s, which satisfies 4.2. **Induction step**: assume that the number of extracted *LCO*s for $i - 1$ satisfies 4.2. To process $OB[i]$, in the first step, $OB[j]$ is found. There are four cases:

1. The *LCO*s whose last element is $OB[k], 1 \leq k < j$ (line 13, Algorithm 6): There is no intersection between these *LCO*s and $OB[i]$. So $OB[i]$ is inserted in the end of these *LCO*s. Here, the number of *LCO*s is unchanged (line 14, Algorithm 6).
2. The *LCO*s whose last element is a subset of $OB[j]$ (line 15, Algorithm 6): According to Lemma 17, these *LCO*s could be extended by the concurrent states of $OB[i]$ (line 16, Algorithm 6). In this case, the number of extracted *LCO*s is unchanged.
3. The *LCO*s whose last element is $OB[j]$ or $OB[j+1]$ for $\epsilon \geq 2$ (line 17, Algorithm 6): According to Lemma 17, three occurrences are possible. So, two new *LCO*s are created (lines 19–25, Algorithm 6).
4. The *LCO*s whose last element is $OB[p], j+1 < p < i$: According to Lemma 18, these *LCO*s are not considered.

So, if the number of *LCO*s after processing $OB[i-1]$ is $T(i-1)$, they could be divided into three parts A, B and C, where $|A| = m, |B| = n, |C| = r$ and $T(i-1) = m+n+r$. The part A includes the *LCO*s that $OB[i]$ could be added to the end of them if the gap constraints are satisfied. The part B includes the *LCO*s whose last element is $OB[j]$ ($|B| = n = F(OB[j])$). The part C includes the *LCO*s whose last element is $OB[j+1]$ ($|C| = r = F(OB[j+1])$). In the worst case, $\epsilon \geq 2$ and for each *LCO* in the parts B and C, two other *LCO*s are created. Therefore, we have: $T(i) \leq m+n+2n+r+2r \Rightarrow T(i) \leq (m+n+r)+2n+2r \Rightarrow T(i) \leq T(i-1) + 2(F(OB[j]) + F(OB[j+1]))$. Note that for $\epsilon \leq 1$, we have $T(i) \leq T(i-1) + 2F(OB[j])$. □

Theorem 6. *The algorithm ExtractLCO (Algorithm 6) extracts all LCOs from the observation OB.*

Proof. This theorem could be proved simply based on the proof of Lemma 19. According to the proof of Lemma 19, the algorithm considers all the possible situations. So it extracts all *LCO*s from *OB*.

Lemma 20. *Given the observation OB that $|OB| = T < Level$, the time complexity of the algorithm ExtractLCO is $O(T)$ in the best case and $O(3^T)$ in the worst case.*

Proof. Consider Algorithm 6. The best case is when *OB* is an *LCO*. It means that in each repeat, O_j is not found and gap constraints are satisfied between $x[L]$ and $Suffix[1]$. In this case, the time complexity is $O(T)$. The worst case is when the last element of all the extracted *LCO*s is O_j or O_{j+1}. So according to Lemma 19 we have $T(i+1) = 3T(i)$. It means that the time complexity of the algorithm is $O(3^T)$ in the worst case.

Lemma 21. *Given the observation OB that $|OB| = T < Level$, the number of LCOs extracted from OB by the algorithm ExtractLCO is 1 in the best case and $O(3^{\frac{T}{2}})$ in the worst case.*

Proof. Consider Algorithm 6. The best case is when *OB* is an *LCO*. It means one *LCO* is identified by the algorithm. The worst case is when the last element of all the extracted *LCO*s intersects with $Suffix[1]$. This case occurs when there is no intersection between $Suffix[1]$ and $x[L]$ and gap constraints are satisfied. So, $Suffix[1]$ is added to all the extracted *LCO*s. To process $OB[i+1]$ in the next call, if $OB[i] \cap OB[i+1] \neq \emptyset$ and gap constraints are satisfied, according to Lemma 19 we have $T(i+1) = 3T(i)$. This situation occurs when $\forall i, 1 \leq i \leq \frac{T}{2}, OB[2i] \cap OB[2i-1] \neq \emptyset$ and $OB[2i] \cap OB[2i+1] = \emptyset$. It means that there are $\frac{T}{2}$ distinct groups of entries. Therefore, the number of *LCO*s extracted by the algorithm is $O(3^{\frac{T}{2}})$ in the worst case.

Lemma 22. *The prediction should be performed in the steps of $\epsilon + 1$ time slots.*

Proof. If the prediction is performed in the time slot t, then the starting interval of the predicted group is $[t+\delta, t+\Delta]$. Since the span of each event is at least $\epsilon + 1$, a new event might occur after the time slot $t + \delta + \epsilon + 1$. On the other hand, δ is the time it takes to instantiate a new VM instance. So we should predict the future behaviour at the time slot $t + \delta + \epsilon + 1 - \delta = t + \epsilon + 1$. □

Lemma 30. *The time complexity of SAVE and the matrix representation to check the correspondence between the entries of LCO and the episode is $O(\|LCO\|)$ and $O(\|LCO\|^2)$ respectively.*

Proof. Note that *LCO* is in the representation form of episodes. In SAVE, each *CNG* of the episode is inserted in $RArray_\alpha$ based on the order defined on resources. In lines 7 and 10 in Algorithm 9, the correspondence between the entries of *LCO* and the episode is considered. It is clear that the time complexity of SAVE for checking this correspondence is $O(\|LCO\|)$. On the contrary, the time complexity for the matrix representation is $O(\|LCO\|^2)$ because the partial order between all the states should be checked in the matrix. □

Corollary 2. *SAVE expedites the episode processing in compared to the matrix representation.*

Proof. There are two steps in which episodes are processed directly: 1) traversing the pattern tree and 2) the episode selection for prediction. As Lemmas 16 and 30 show SAVE expedites the episode extensions in traversing the pattern tree and the episode selection for prediction in compared to the matrix representation. □

Appendix B. Algorithms

In this appendix, we present all the functions called by the algorithm *Main* (algorithm 7) in a canonical form and explain them in detail.

Appendix B.1. Function ExtractObservation

Algorithm 8 is proposed to extract the observation from the recent history of the stream. It receives the recent history of the stream and *Level*. Since the goal is to predict the future behaviour of the application, the algorithm returns an observation whose length is at most $Level - 1$. *Group* is a group of concurrent events. *EventStartGroup* is the index of the latest event of *Group* in *List*. *Max* and *Min* are the maximum and minimum of the start time of events of *Group*. In lines 8 to 21, the events before the current event are considered whether could be in the same *Group* or not. In lines 22 to 25, the events after the current event are considered to complete *Group*. Lines 26 to 34 consider whether *Group* could be inserted in *OB* under the gap constraint Δ. Finally, in lines 39 to 41, the corresponding states of events are returned as the observation.

Algorithm 8 ExtractObservation

Input: Recent History of the Stream, *Level*;
Output: *Observation OB′*

1: $List \leftarrow$ Recent History of the stream;
2: $OB \leftarrow \emptyset$;
3: $EventStartGroup \leftarrow |List| + 1$;
4: $Group \leftarrow \emptyset$;
5: $i \leftarrow |List| + 1$;
6: $count \leftarrow 1$;
7: **while** $(i > 1$ *and* $|OB| < Level)$ **do**
8: **if** $(Group = \emptyset)$ **then**
9: $j \leftarrow EventStartGroup - 1$;
10: $i - -$;
11: **while** $(j >= i)$ **do**
12: **if** $(List[j].st - List[i].st \leq \epsilon)$ **then**
13: add from $List[j]$ to $List[i]$ into *Group*;
14: $Max \leftarrow List[j].st$;
15: $Min \leftarrow List[i].st$;
16: $EventStartGroup \leftarrow j$;
17: break;
18: **else**
19: $j - -$;
20: **end if**
21: **end while**
22: **else if** $(Group \neq \emptyset$ *and* $Max - List[i-1].st \leq \epsilon)$ **then**
23: add $List[i-1]$ into *Group*;
24: $Min \leftarrow List[i-1].st$;
25: $i - -$;
26: **else if** $(Group \neq \emptyset$ *and* $Max - List[i-1] > \epsilon)$ **then**
27: **if** $(OB = \emptyset)$ *or* $Min\ Of\ StartTime(OB[count-1]) - Max \leq \Delta)$ **then**
28: $OB[count] \leftarrow Group$;
29: $count++$;
30: $Group \leftarrow \emptyset$;
31: **else**
32: break;
33: **end if**
34: **end if**
35: **end while**
36: **if** $(Group \neq \emptyset$ *and* $((OB = \emptyset)$ *or* $Min\ Of\ StartTime(OB[count-1]) - Max \leq \Delta))$ **then**
37: $OB[count] \leftarrow Group$;
38: **end if**
39: **for** $(i = |B|; i \geq 1; i - -)$ **do**
40: $OB'[|B| - i + 1] \leftarrow$ corresponding states of $OB[i]$;
41: **end for**
42: **return** OB';

Appendix B.2. Function EpisodeSelection

The function receives *LCO* and *PatternBase*. *PatternBase* is a set of extracted patterns. Each pattern includes one episode and its information such as *frequency*, *MatchScore* and *Confidence*. For each matched pattern, the last time of the starting interval of the last entry of *LCO* is also maintained. Each entry $freq[i], 1 \leq i \leq |LCO|$ includes the frequency of $LCO[i..|LCO|]$. As it was implied *ListFreq* is a list of pairs of $(Prefix, Freq)$ that *Prefix* is the Prefix of episodes and *Freq* is its frequency. This list is used to compute the confidence of episodes. In lines 4 to 26 of the algorithm, $\forall k, 2 \leq k \leq |LCO|$ the episodes whose left hand side matches $LCO[k..|LCO|]$ are selected. At the same time, lines 13 and 14 update the frequency of subsequences of *LCO*. In lines 27 to 46, episodes whose left hand side matches $LCO[1..|LCO|]$ are considered. Lines 37 to 39 update the frequency of *LCO*. In lines 47 to 60, episodes whose middle part matches *LCO* and left hand side is consistent with the history of the stream are considered. In lines 48 to 52, for each of these episodes, a new entry is inserted in *MatchedEpisodes*. Lines 53 to 60 update *ListFreq*. In lines 61 to 71, *ListFreq* is updated based on the frequency of episodes whose last part matches *LCO*. Finally, the function *ComputeConfidence* computes the confidence of episodes without scanning *PatternBase* to find the frequency of the episodes. It could be proved that the algorithm finds unique episodes and computes their confidence correctly. The detail of the function *ComputeConfidence* is omitted due to space limitation.

Algorithm 9 EpisodeSelection

Input: *LCO*,*PatternBase*; % *LCO* is the longest Consistent Observation
Output: *MatchedEpisodes*; % A set of selected episodes

freq is an array of $|LCO|$ integers that is initialized by 0;
ListFreq is a list of pairs of (*Prefix*, *Freq*), which is empty firstly;
$MatchedEpisodes \leftarrow \emptyset$;
for $(j = |LCO|; j > 1; j--)$ **do**
  for $(i = 1; i \leq |PatternBase|; i++)$ **do**
    $A \leftarrow False$,$B \leftarrow False$;
    if $(PatternBase[i].Episode[1..|LCO| - j + 1] = LCO[j..|LCO|])$ **then**
      $A \leftarrow True$;
    end if
    if $(PatternBase[i].Episode[|PatternBase[i].Episode| - (|LCO| - j)..|PatternBase[i].Episode|] = LCO[j..|LCO|])$ **then**
      $B \leftarrow True$;
    end if
    if $(A \text{ or } B)$ *and* $freq[j] < PatternBase[i].freq$ **then**
      $freq[j] \leftarrow PatternBase[i].freq$;
    end if
    if (A) **then**
      if $(PatternBase[i].Episode.FindIndex(LCO) = 0)$ **then**
        define the new pattern r;
        $r \leftarrow PatternBase[i]$;
        $r.MatchScore \leftarrow \frac{\sum_{k=j}^{|LCO|} LCO[k]}{|states \in LCO|}$;
        $r.I \leftarrow$ the last time of the starting interval of the last entry of *LCO*;
        add r to *MatchedEpisodes*;
      end if
    end if
  end for
end for
for $(j = 1; j \leq |PatternBase|; j++)$ **do**
  $Index \leftarrow PatternBase[j].Episode.FindIndex(LCO)$;
  $A \leftarrow False$,$B \leftarrow False$;
  if $(Index = 1)$ **then**
    $A \leftarrow True$;
  end if
  if $(Index > 1$ *and* $Index + 1 = |PatternBase[j].Episode| - |LCO|)$ **then**
    $B \leftarrow True$;
end if
if $(A \text{ or } B)$ **then**
    if $(freq[1] < PatternBase[j].freq)$ **then**
      $freq[1] \leftarrow PatternBase[i].freq$;
    end if
  end if
  if $(A$ *and* $|PatternBase[j].Episode| > |LCO|)$ **then**
    define the new pattern r;
    $r \leftarrow PatternBase[j]$;
    $r.MatchScore \leftarrow 1$;
    $r.I \leftarrow$ the last time of the starting interval of the last entry of *LCO*;
    add r to *MatchedEpisodes*;
  else if $(Index > 1$ *and* $Index + |LCO| - 1 < PatternBase[j].Episode|$ *and* $HistoryConsistent(PatternBase[j].Episode, Index - 1))$ **then**
    define the new pattern r;
    $r \leftarrow PatternBase[j]$;
    $r.MatchScore \leftarrow 1 + \frac{Index-1}{|PatternBase[j].Episode|}$;
    $r.I \leftarrow$ the last time of the starting interval of the last entry of *LCO*;
    add r to *MatchedEpisodes*;
    $K \leftarrow ListFreq.FindIndex(PatternBase[j].Episode[1..Index - 1])$;
    if $(K \neq 0)$ **then**
      if $(ListFreq[K].freq < PatternBase[j].freq)$ **then**
        $ListFreq[K].Freq \leftarrow PatternBase[j].freq$;

```
57:       end if
58:     else
59:       add (PatternBase[j].Episode[0..Index − 1], PatternBase[j].freq) into ListFreq;
60:     end if
61:   else if (B) then
62:     for each (x ∈ ListFreq) do
63:       if (x.Prefix = PatternBase[j].Episode[Index − |x.Prefix|..Index − 1]) then
64:         if (x.Freq < PatternBase[j].freq) then
65:           x.Freq ← PatternBase[j].freq;
66:           break;
67:         end if
68:       end if
69:     end for
70:   end if
71: end for
72: Compute Confidence
(MatchedEpisodes,ListFreq);
73: return Matched Episodes;
```

Appendix B.3. Function Evaluating

The main goal of Algorithm 10 is to predict the future status of resources. In lines 4 to 24, the algorithm considers the matched episodes. In line 5, it checks whether the matching time of the episode is equal to $t-1$ or not. If the matching time of the episode is equal to $t-1$, then the episode could be used for prediction. In line 6, the index of the last ✻ in the episode is determined (*Index*). In lines 8 to 20, *Episode*[*Index* + 1] of *MatchedEpisodes*[*i*] (the first entry after the last ✻) is searched in *PredictionStep*. If *PredictionStep* includes it, the maxi-mum values of *Confidence* and *MatchScore* are maintained for it. Otherwise, the corresponding entry of *MatchedEpisodes*[*i*].*Episode*[*Index* + 1] is inserted in *PredictionStep*. In lines 21 to 23 and 25 to 27, the matched episodes whose prediction results have not been consistent with the observed behaviour, are determined and deleted. In lines 28 and 29, the valid interval of the future status of resources is determined. In lines 30 and 31, based on the threshold η the most confident episodes are maintained and sorted in *PredictionStep*. In line 34, the most confident status of resources is considered as prediction results. In lines 36 to 38, for resources that are not in *PredictionStep*, the future sta-tus is predicted based on *MRE*. If *MRE* is not found, according to line 40, the last observed status of resources is considered as the future status.

Algorithm 10 Evaluating

Input: *LastObservation*, *MatchedEpisodes*, t, η;
Output: It predicts the future status of resources;

```
1: DeleteList ← ∅;
2: PredictionStep ← ∅;
3: MRE ← ∅;
4: for (i = 1; i ≤ |MatchedEpisodes|; i++) do
5:   if (MatchedEpisodes[i].I + 1 = t) then
6:     Index ← MatchedEpisodes[i].Episode.FindLastIndex(✻);
7:       L ← |MatchedEpisodes[i].Episode| − Index;
8:       for (k = Index + 1; k ≤ Index + min(L, 1); k++) do
9:         j ← PredictionStep.Outcome.FindIndex(MatchedEpisodes[i].Episode[k]);
10:          if (j > 0) then
11:            PredictionStep[j].Confidence ← max(PredictionStep[j].Confidence, MatchedEpisodes[i].Confidenece);
12:            PredictionStep[j].MatchScore ← max(PredictionStep[j].MatchScore, MatchedEpisodes[i].MatchScore);
13:          else
14:            create a new entry e for PredictionStep;
15:            e.Episode ← MatchedEpisodes[i].Episode
16:            e.Confidence ← MatchedEpisodes[i].Confidence;
17:            e.MatchScore ← MatchedEpisodes[i].MatchScore;
18:            add e to PredictionStep;
19:          end if
20:        end for
21:   else if (MatchedEpisodes[i].I + Δ < t) then
22:       add i to DeleteList;
23:   end if
24: end for
```

25: **for** ($j = |DeleteList|; j \geq 0; j--$) **do**
26: Delete $MatchedEpisodes[DeleteList[j]]$;
27: **end for**
28: $Result.I1 = t + \delta$;
29: $Result.I2 = t + \Delta$;
30: $PredictionStep \leftarrow$ Rows of $PredictionStep$ with corresponding $Confidence \geq \eta$;
31: Sort $PredictionStep$ in descending order based on $Confidence$, If some entries have the equal $Confidence$, sort them based on $MatchScore$;
32: **for each** ($Resource\ x \in Result.Outcome$) **do**
33: **if** ($x \in PredictionStep.Outcome$) **then**
34: $Result.Outcome[x] \leftarrow$ The first status of x in $PredictionStep.Outcome$;
35: **else**
36: $MRE \leftarrow$ The most recent event in history that matches the status and the span of the event of x in $LastObservation$;
37: **if** ($MRE \neq \emptyset$) **then**
38: $Result.Outcome[x] \leftarrow$ The observed status of the resource x after MRE;
39: **else**
40: $Result.Outcome[x] \leftarrow$ the status of the resource x in $LastObservation$
41: **end if**
42: **end if**
43: **end for**
44: add $Result$ into $ResultTable$;

Appendix B.4. Function CreateLEG

Pruning and updating the matched episodes and computing the precision of the predicted results are based on *LEG*. In Algorithm 7, when the prediction is performed, *flag* is true. So in line 4 of Algorithm 11, *LastObservation* is copied in the *PrevSection* of *LEG*. In line 6, the status of resources is sampled in the next time slot. It means that t increases by +1. In line 7, the interval of *LEG* is determined. In lines 8 to 30, the corresponding events of the sampled resources in *PrevSection* are considered. In lines 10 to 16, if the status of resources is not unchanged, it means that there are new events. Therefore they are omitted from *PrevSection* and added to the list L that will be inserted in *EventSec* later. In lines 17 to 27, the events that have the large span are decomposed into two events based on the composition unit μ. The *PrevSection*, *LastObservation* and the list L are also updated. In lines 27 to 29, if there is no change in the corresponding event of the resource of e_R ($e_R.Resource$), the end time of the event is update in the *LastObservation* and *history*. In lines 31 to 32, the oldest entry of $LEG.EventSec$ is deleted and L is inserted in its last entry. Note that the function *CreateLEG* is called in lines 18 and 22 of Algorithm 7. Since the step of the prediction is $\epsilon + 1$, the time slots between two predictions, t and $t + \epsilon + 1$, are processed in lines 19 to 23 of Algorithm 7. So calling the function *CreateLEG* is performed by the *false* value of *flag*.

Algorithm 11 CreateLEG

Input: $LastObservation, flag, LEG$;
Output: LEG; % It Updates LEG
1: $SampledRes \leftarrow \emptyset$;
2: $L \leftarrow \emptyset$;
3: **if** ($flag$) **then**
4: $LEG.PrevSection \leftarrow LastObservation$;
5: **end if**
6: $SampledRes \leftarrow$ Sample resources whose corresponding events are in $LEG.PrevSection$ in the next time slot; % the current time increases by +1 slot. It means $t \leftarrow t + 1$;
7: $LEG.I_1 \leftarrow t - \epsilon$, $LEG.I_2 \leftarrow t$;
8: **for each** ($(Resource, Status) \in SampledRes$) **do**
9: $e_R \leftarrow$ event $e = (r, s, st, et)$ in $LEG.PrevSection$ that $e.r = Resource$;
10: **if** ($Status \neq e.s$) **then** % It checks the change of status of resources
11: Create the new event E=(r,s,st,et);
12: $E.r \leftarrow Resource$;
13: $E.s \leftarrow Status$;
14: $E.st \leftarrow t - 1$, $E.et \leftarrow t$;
15: add E to L;

16: delete the corresponding event of *Resource* in *LEG.PrevSection*;
17: **else if** $((t-1)-(\mu+\epsilon)=e_R.st)$ **then**
18: Update e_R as $(e_R.Resource, e_R.Status, t-1-\epsilon, t)$ in *LEG.PrevSection* and *LastObservation*;
19: Decompose $(e_R.Resource, e_R.Status, t-1-(\mu+\epsilon), t)$ into two events $A=(e_R.Resource, e_R.Status, t-1-(\mu+\epsilon), t-\epsilon-1)$ and $B=(e_R.Resource, e_R.Status, t-1-\epsilon, t)$;
20: Remove the event A from *LEG.PrevSection* and insert it into *history*;
21: **if** $(B.st = LEG.I_2 - 1)$ **then** 22: add the event B into L;
23: delete the corresponding event of $B.Resource$ in *LEG.PrevSection*;
24: **else**
25: add the event B into *history*;
26: **end if**
27: **else if** $((t-1)-(\mu+\epsilon) < e_R.st)$ **then**
28: Update *et* (end time) of the corresponding event of $e_R.Resource$ in *LastObservation* and *history*;
29: **end if**
30: **end for**
31: $LEG.EventSec[0..\epsilon-1] \leftarrow LEG.EventSec[1..\epsilon]$;
32: $LEG.EventSec[\epsilon] \leftarrow L$;
33: **return** LEG;

Appendix B.5. Function UpdateHistory

Algorithm 12 updates *history* based on *LEG*. In lines 1 to 2, if *history* is empty, *history* is filled with the recent events of the stream. If *history* is not empty, in lines 4 to 9, the end time of the most recent observed events is updated (events of *LEG.EventSection*) and the events are added into *history*.

Algorithm 12 UpdateHistory

Input: *history*, *LEG*;
Output: It updates *history*
1: **if** $(history = \emptyset)$ **then**
2: $history \leftarrow$ recent events of stream;
3: **else**
4: **for** $(i = 0; i \leq \epsilon; i++)$ **do**
5: **for each** $(event\ x \in LEG.EventSection[i])$ **do**
6: $x.et \leftarrow LEG.I_2$;
7: insert x into *history*;
8: **end for**
9: **end for**
10: **end if**

Appendix B.6. Function ComputePrecision

Algorithm 13 updates two counters *PredictionCount* and *CorrectCount*, which are essential to compute the prediction precision, and prunes the entries of *ResultTable*. In lines 2 to 20, the entries of *ResultTable* are considered. In line 3, each entry of *ResultTable* that has predicted the status of resources in the recent time slots is investigated. In lines 5 to 9, the counter *sum* counts the number of resources whose status is consistent with *LEG*. In lines 10 to 12, $ResultTable[i].Max$ is updated based on *sum* to find a *LEG* that is the most consistent with $ResultTable[i].Outcome$ in the interval of $[ResultTable[i].I_1, ResultTable[i].I_2]$. Since one time slot has been observed, $ResultTable[i].I_1$ increases by +1 in line 13. In lines 14 to 18, if $ResultTable[i].I_1 > ResultTable[i].I_2$, it means that the time interval predicted by $ResultTable[i]$ has passed. So one prediction has been performed and *PredictionCount* increases by +1. Furthermore, *CorrectCount* increases by $ResultTable[i].Max$. Finally, in lines 21 to 23, the entries of *ResultTable* whose time interval has passed, are removed.

Algorithm 13 ComputePrecision

Input: *LEG*;
Output: It updates two counters *PredictionCount* and *CorrectCount*;
1: *DeleteList* $\leftarrow \emptyset$;
2: **for** $(i = 1; i \leq |ResultTable|; i++)$ **do**
3: **if** $(ResultTable[i].I_1 \leq LEG.I_1 \leq ResultTable[i].I_2)$ **then**
4: $sum \leftarrow 0$;
5: **for each** $(Resouce\ R \in ResultTable[i].Outcome)$ **do**
6: **if** (state $(R, ResultTable[i].Outcome[R])$ occurs in *LEG*) **then**
7: $sum \leftarrow sum + 1$;
8: **end if**
9: **end for**
10: **if** $(\frac{sum}{|ResourceType|} > ResultTable[i].Max)$ **then**
11: $ResultTable[i].Max \leftarrow \frac{sum}{|ResourceType|}$;
12: **end if**
13: $ResultTable[i].I_1 \leftarrow ResultTable[i].I_1 + 1$;
14: **if** $(ResultTable[i].I_1 > ResultTable[i].I_2)$ **then**
15: $PredictionCount++$;
16: $CorrectCount \leftarrow CorrectCount + ResultTable[i].Max$;
17: add i into *DeleteList*;
18: **end if**
19: **end if**
20: **end for**
21: **for** $(i = |DeleteList|; i > 0; i--)$ **do**
22: Delete $ResultTable[DeleteList[i]]$;
23: **end for**

Appendix B.7. Function UpdateMatchedEpisodes

Algorithm 14 shows the function used to update the matched episodes. In lines 2 to 13, all the matched episodes are considered. In lines 3 and 4, the first element after ✻ of episodes that have predicted the events of the recent time slots, is considered. In lines 5 to 7, if this element is consistent with *LEG*, the element is filled with ✻ and the matching time of the episode is updated based on the start times of events of *LEG*. Finally, the episodes that have been filled with ✻ completely, are deleted in lines 14 to 16.

Algorithm 14 UpdateMatchedEpisodes

Input: *LEG*, *MatchedEpisodes*;
Output: It updates *MatchedEpisodes*;
1: *DeleteList* $\leftarrow \emptyset$;
2: **for** $(i = 1; i \leq |MatchedEpisodes|; i++)$ **do**
3: **if** $(MatchedEpisodes[i].I + \delta \leq LEG.I_1 - 1 \leq MatchedEpisodes[i].I + \Delta)$ **then**
4: $j \leftarrow MatchedEpisodes[i].Episode.FindIndex(✻)$;
5: **if** $(MatchedEpisodes[i].Episode[j+1])$ is consistent with *LEG* **then**
6: $MatchedEpisodes[i].Episode[j+1] \leftarrow ✻$;
7: $MatchedEpisodes[i].I \leftarrow$ max of start times of events of *LEG* that are consistent with $MatchedEpisodes[i].Episode[j+1]$;
8: **if** $(j + 1 = |MatchedEpisodes[i].Episode|)$ **then**
9: add i to *DeleteList*;
10: **end if**
11: **end if**
12: **end if**
13: **end for**
14: **for** $(i = |DeleteList|; i > 0; i--)$ **do**
15: delete $MatchedEpisodes[DeleteList[i]]$;
16: **end for**

References

Achar, A., Ibrahim, A., Sastry, P., 2013. Pattern-growth based frequent serial episode discovery. Data Knowl. Eng. 87, 91–108. https://doi.org/10.1016/j.datak.2013.06.005. 0169–023X http://www.sciencedirect.com/science/article/pii/S0169023X13000724.

Achar, A., Laxman, S., Sastry, P.S., 2012. A unified view of the apriori-based algorithms for frequent episode discovery. Knowl. Inf. Syst. 31 (2), 223–250. https://doi.org/10.1007/s10115-011-0408-2 0219–3116.

Achar, A., Laxman, S., Viswanathan, R., Sastry, P.S., 2012. Discovering injective episodes with general partial orders. Data Min. Knowl. Discov. 25 (1), 67–108. https://doi.org/10.1007/s10618-011-0233-y 1384–5810.

Akindele, A.B., Samuel, A.A., 2013. Predicting cloud resource provisioning using machine learning techniques. In: 2013 26th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), pp. 1–4 Vancouver, Canada.

Alam, M., Shakil, K.A., Sethi, S., 2015. Analysis and Clustering of Workload in Google Cluster Trace Based on Resource Usage. arXiv preprint arXiv:1501.01426, http://arxiv.org/abs/1501.01426.

Allen, J.F., 1984. Towards a general theory of action and time. Artif. Intell. 23 (2), 123–154. 0004–3702 https://doi.org/10.1016/0004-3702(84)90008-0. http://www.sciencedirect.com/science/article/pii/0004370284900080.

Amiri, M., Mohammad-Khanli, L., 2017. Survey on prediction models of applications for resources provisioning in cloud. J. Netw. Comp. Appl. 82, 93–113. 1084–8045 https://doi.org/10.1016/j.jnca.2017.01.016. http://www.sciencedirect.com/science/article/pii/S1084804517300231.

Amiri, M., Feizi-Derakhshi, M.R., Mohammad-Khanli, L., 2016. IDS fitted Q improvement using fuzzy approach for resource provisioning in cloud. J. Intell. Fuzzy Syst. Prepr. Prepr. 1–12. https://doi.org/10.3233/JIFS-151445. http://content.iospress.com/articles/journal-of-intelligent-and-fuzzy-systems/ifs151445.

Andreolini, M., Colajanni, M., Pietri, M., Tosi, S., 2015. Adaptive, scalable and reliable monitoring of big data on clouds. J. Parallel Distrib. Comp. 79–80, 67–79. 0743–7315 https://doi.org/10.1016/j.jpdc.2014.08.007. http://www.sciencedirect.com/science/article/pii/S074373151400149X.

Antonescu, A.F., Robinson, P., Braun, T., 2013. Dynamic SLA management with forecasting using multi-objective optimization. In: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 457–463 Ghent, Belgium.

Batal, I., Fradkin, D., Harrison, J., Moerchen, F., Hauskrecht, M., 2012. Mining recent temporal patterns for event detection in multivariate time series data. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12. ACM, Beijing, China, pp. 280–288. https://doi.org/10.1145/2339530.2339578. 978-1-4503-1462-6 http://doi.acm.org/10. 1145/2339530.2339578.

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

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

Bey, K.B., Benhammadi, F., Mokhtari, A., Guessoum, Z., 2009. CPU load prediction model for distributed computing. In: 2009 Eighth International Symposium on Parallel and Distributed Computing, pp. 39–45. https://doi.org/10.1109/ISPDC.2009.8 Lisbon, Portugal.

Calheiros, R.N., Ranjan, R., Beloglazov, A., Rose, C.A.F.D., Buyya, R., 2011. Cloud Sim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. 41 (1), 23–50. https://doi.org/10.1002/spe.995 0038–0644.

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

Cetinski, K., Juric, M.B., 2015. AME-WPC: advanced model for efficient workload prediction in the cloud. J. Net. Comp. Appl. 55, 191–201. https://doi.org/10.1016/j.jnca.2015.06.001.

Chen, Z., Zhu, Y., Di, Y., Feng, S., 2015. Self-adaptive prediction of cloud resource demands using ensemble model and subtractive-fuzzy clustering based fuzzy neural network. Comp. Intell. Neurosci. https://doi.org/10.1155/2015/919805 1687–5265.

Coutinho, E.F., de Carvalho Sousa, F.R., Rego, P.A.L., Gomes, D.G., de Souza, J.N., 2015. Elasticity in cloud computing: a survey. Ann. Telecommun. Ann. des Télécommun. 70 (7), 289–309. https://doi.org/10.1007/s12243-014-0450-7 1958–9395.

Di, S., Kondo, D., Cirne, W., 2014. Google host load prediction based on Bayesian model with optimized feature combination. J. Parallel Distrib. Comp. 74 (1), 1820–1832. https://doi.org/10.1016/j.jpdc.2013.10.001 0743–7315.

Duan, R., Nadeem, F., Wang, J., Zhang, Y., Prodan, R., Fahringer, T., 2009. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '09. IEEE Computer Society, Shanghai, China, pp. 339–347. https://doi.org/10.1109/CCGRID.2009.58. 978-0-7695-3622-4.

Fahed, L., Brun, A., Boyer, A., 2014. Episode Rules Mining Algorithm for Distant Event Prediction Technical Report hal-01062542. HAL.

Galante, G., Bona, L. C. E. d., 2012. A survey on cloud computing elasticity. In: 2012 IEEE Fifth International Conference on Utility and Cloud Computing, pp. 263–270. https://doi.org/10.1109/UCC.2012.30 Chicago, IL, USA.

Garg, S.K., Toosi, A.N., Gopalaiyengar, S.K., Buyya, R., 2014. SLA-based virtual machine management for heterogeneous workloads in a cloud data center. J. Net. Comp. Appl. 45, 108–120. 1084–8045 https://doi.org/10.1016/j.jnca.2014.07.030. http://www.sciencedirect.com/science/article/pii/S1084804514001787.

Ghorbani, M., Wang, Y., Xue, Y., Pedram, M., Bogdan, P., 2014. Prediction and control of bursty cloud workloads: a fractal framework. In: Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis, CODES '14. ACM, New Delhi, India, pp. 12:1–12:9. https://doi.org/10.1145/2656075.2656095. 978-1-4503-3051-0 http://doi.acm.org/10.1145/2656075.2656095.

Gursun, G., Crovella, M., Matta, I., 2011. Describing and forecasting video access patterns. In: 30th IEEE International Conference on Computer Communications (INFOCOM 2011), pp. 16–20 Shanghai, China.

Han, J., Pei, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C., 2001. Prefix span: mining sequential patterns efficiently by Prefix-projected pattern growth. In: Proceedings of the 17th International Conference on Data Engineering. , pp. 215–224. http://hanj.cs.illinois.edu/pdf/span01.pdf.

Han, J., Cheng, H., Xin, D., Yan, X., Aug 2007. Frequent pattern mining: current status and future directions. Data Min. Knowl. Discov. 15 (1), 55–86. https://doi.org/10.1007/s10618-006-0059-1. 1573–756X https://doi.org/10.1007/s10618-006-0059-1.

F. Hoppner. Learning temporal rules from state sequences. In Seventeenth International Joint Conference on Artificial Intelligence, IJCAI-01.

Huang, D., He, B., Miao, C., 2014. A survey of resource management in multi-tier web applications. IEEE Commun. Surv. Tutor. 16 (3), 1574–1590. https://doi.org/10.1109/SURV.2014.010814.000601553–877X.

Huang, P., Liu, C.J., Yang, X., Xiao, L., Chen, J., 2014. Wireless spectrum occupancy prediction based on partial periodic pattern mining. IEEE Trans. Parallel Distrib. Syst. 25 (7), 1925–1934. https://doi.org/10.1109/TPDS.2013.283 1045–9219.

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

Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., Epema, D.H., 2008. The grid workloads archive. Future Gen. Comp. Syst. 24 (7), 672–686. 0167–739X https://doi.org/10.1016/j.future.2008.02.003. http://www.sciencedirect.com/science/article/pii/S0167739X08000125.

Islam, S., Keung, J., Lee, K., Liu, A., 2012. Empirical prediction models for adaptive resource provisioning in the cloud. Future Gen. Comp. Syst. 28 (1), 155–162. https://doi.org/10.1016/j.future.2011.05.027 0167–739X.

Iwanuma, K., Takano, Y., Nabeshima, H., 2004. On anti-monotone frequency measures for extracting sequential patterns from a single very-long data sequence. In: IEEE Conference on Cybernetics and Intelligent Systems, 2004, vol. 1, pp. 213–217. https://doi.org/10.1109/ICCIS.2004.1460414 Singapore, Singapore.

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

Jiang, Y., Perng, C.-S., Li, T., Chang, R.N., 2013. Cloud analytics for capacity planning and instant VM provisioning. IEEE Trans. Net. Serv. Manag. 10 (3), 312–325. Kaastra, I., Boyd, M., 1996. Designing a neural network for forecasting financial and economic time series. Neurocomputing 10 (3), 215–236. 0925–2312 https://doi.org/10.1016/0925-2312(95)00039-9. http://www.sciencedirect.com/science/article/pii/0925231295000399 Financial Applications, Part {II}.

Karypis, G., Joshi, M.V., Kumar, V., 1999. Universal formulation of sequential patterns Technical Report TR 99–9021. Department of Computer Science, University of Minnesota, Minneapolis.

Kaur, P.D., Chana, I., 2014. A resource elasticity framework for qos-aware execution of cloud applications. Future Gen. Comp. Syst. 37, 14–25. 0167-739X https://doi.org/10.1016/j.future.2014.02.018. http://www.sciencedirect.com/science/article/pii/S0167739X14000430.

Khan, A., Yan, X., Tao, S., Anerousis, N., 2012. Workload characterization and prediction in the cloud: a multiple time series approach. In: 2012 IEEE Network Operations and Management Symposium, pp. 1287–1294 Maui, HI, USA.

Kundu, S., Rangaswami, R., Gulati, A., Zhao, M., Dutta, K., 2012. Modeling virtualized applications using machine learning techniques. In: Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments, VEE '12. ACM, pp. 3–14. https://doi.org/10.1145/2151024.2151028. 2151028.

Labonte, F., Mattson, P., Thies, W., Buck, I., Kozyrakis, C., Horowitz, M., 2004. The stream virtual machine. In: Proceedings of 13th International Conference on Parallel Architecture and Compilation Techniques, PACT '04, pp. 267–277. https://doi.org/10.1109/PACT.2004.1342560 Antibes Juan-les-Pins, France, ISBN: 1089-795X.

Lagar-Cavilla, H.A., Whitney, J.A., Scannell, A.M., Patchin, P., Rumble, S.M., de Lara, E., Brudno, M., Satyanarayanan, M., 2009. Snow flock: rapid virtual machine cloning for cloud computing. In: Proceedings of the 4th ACM European Conference on Computer Systems, EuroSys '09. ACM, pp. 1–12. https://doi.org/10.1145/1519065. 1519067. Nuremberg, Germany.

Lama, P., Zhou, X., 2013. 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.

Laxman, S., 2006. Discovering frequent episodes: fast algorithms, connections with HMMs and generalizations PhD Thesis. Department of Computer Science, University of Victoria, Bangalore, India.

Laxman, S., Sastry, P.S., 2006. A survey of temporal data mining. Sadhana 31 (2), 173–198. https://doi.org/10.1007/BF02719780 0973–7677.

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

Laxman, S., Tankasali, V., White, R.W., 2008. Stream prediction using a generative model based on frequent episodes in event sequences. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '08. ACM, Las Vegas, Nevada, USA, pp. 453–461. https://doi.org/10.1145/1401890.1401947. 978-1-60558-193-4 http://doi.acm.org/10.1145/1401890.1401947.

Li, A., Yang, X., Kandula, S., Zhang, M., 2010. Cloudcmp: comparing public cloud providers. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, IMC '10. ACM, Melbourne, Australia, pp. 1–14. https://doi.org/10.1145/1879141.1879143. 978-1-4503-0483-2 http://doi.acm.org/10.1145/1879141.1879143.

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

Liu, C., Shang, Y., Duan, L., Chen, S., Liu, C., Chen, J., 2015. Optimizing workload category for adaptive workload prediction in service clouds. In: Proceedings of the 13th International Conference on Service-oriented Computing (ICSOC 2015). Springer-Verlag, Goa, India, pp. 87–104. Berlin Heidelberg.

Lu, C.T., Chang, C.W., Chang, J.S., 2015. VM scaling based on Hurst exponent and Markov transition with empirical cloud data. J. Syst. Soft. 99, 199–207 0164–1212.

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

Matsunaga, A., Fortes, J.A.B., 2010. 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, pp. 495–504. https://doi.org/10.1109/CCGRID.2010.98.

Matsunaga, A., Fortes, J.A.B., 2010. 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, pp. 495–504. https://doi.org/10.1109/CCGRID.2010.98.

Mell, P., Grance, T., 2011. NIST Special Publication 800-145. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

Miu, T., Missier, P., 2012. Predicting the execution time of workflow activities based on their input features. In: Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, SCC '12. IEEE Computer Society, Salt Lake City, UT, USA, pp. 64–72. https://doi.org/10.1109/SC.Companion.2012.21.

Moerchen, F., 2006. Algorithms for time series knowledge mining. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '06. ACM, Philadelphia, PA, USA, pp. 668–673. https://doi.org/10.1145/1150402.1150485. 1-59593-339-5.

Moskovitch, R., Shahar, Y., 2009. Medical temporal-knowledge discovery via temporal abstraction. In: Proceedings of 2009 AMIA Annual Symposium, pp. 452–456 San Francisco, CA.

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

Pei, J., Han, J., Wang, W., 2007. Constraint-based sequential pattern mining: the pattern-growth methods. J. Intell. Inf. Syst. 28 (2), 133–160. https://doi.org/10.1007/s10844-006-0006-z 0925–9902.

Prevost, J.J., Nagothu, K., Kelley, B., Jamshidi, M., 2011. Prediction of cloud data center networks loads using stochastic and neural models. In: 2011 6th International Conference on System of Systems Engineering, pp. 276–281 Albuquerque, New Mexico, USA.

Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A., 2012. Heterogeneity and dynamicity of clouds at scale: google trace analysis. In: Proceedings of the Third ACM Symposium on Cloud Computing, SoCC '12. ACM, San Jose, California, pp. 7:1–7:13. https://doi.org/10.1145/2391229.2391236. 978-1-4503-1761-0, http://doi.acm.org/10.1145/2391229.2391236.

Saripalli, P., Kiran, G.V.R., Shankar, R.R., Narware, H., Bindal, N., 2011. Load prediction and hot spot detection models for autonomic cloud computing. In: Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing. IEEE Computer Society, Melbourne, Victoria, Australia, pp. 397–402. https://doi.org/10.1109/ucc.2011.66.

Shen, S., Beek, V. v., Iosup, A., May 2015. Statistical characterization of business-critical workloads hosted in cloud data centers. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 465–474. https://doi.org/10.1109/CCGrid.2015.60.

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

Singh, S., Chana, I., 2015. QoS-aware autonomic resource management in cloud computing: a systematic review. ACM Comp. Surv. 48 (3) https://doi.org/10.1145/2843889.

Tang, Z., Mo, Y., Li, K., Li, K., 2014. Dynamic forecast scheduling algorithm for virtual machine placement in cloud computing environment. The J. Supercomput. 70 (3), 1279–1296. https://doi.org/10.1007/s11227-014-1227-5 0920–8542.

Tatti, N., Cule, B., 2012. Mining closed strict episodes. Data Min. Knowl. Discov. 25 (1), 34–66. https://doi.org/10.1007/s10618-011-0232-z 1384–5810.

Toma, T., Abu-Hanna, A., Bosman, R.-J., 2007. Discovery and inclusion of {SOFA} score episodes in mortality prediction. J. Biomedical Inf. 40 (6), 649–660. https://doi.org/10.1016/j.jbi.2007.03.007. 1532–0464 http://www.sciencedirect.com/science/article/pii/S153204640700024X Intelligent Data Analysis in Biomedicine.

Tu, Z., 2007. Learning generative models via discriminative approaches. In: 2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007). IEEE Computer Society, Minneapolis, Minnesota, USA.

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

Utilization. http://h17007.www1.hpe.com/device_help/HPJ3298A/utilization.htm. (Accessed 06 06 2017) .

Vazquez, C., Krishnan, R., John, E., 2015. Time series forecasting of cloud data center workloads for dynamic resource provisioning. J. Wirel. Mob. Net. Ubiquitous Comp. Depend. Appl. JoWUA 6 (3), 87–110.

VMware vRealize Operations Manager 6.0.1 Documentation Center. https://pubs.vmware.com/vrealizeoperationsmanager-6/index.jsp#com.vmware.vcom.core.doc/GUID-41603CD6-453B-4E26-A237-34E733BAB00C.html. Accessed 7 June 2017.

Wang, C., 1994. A Theory of Generalization in Learning Machines with Neural Network Applications PhD thesis, Philadelphia, PA, USA. AAI9521138.

Wang, J., Han, J., 2004. BIDE: efficient mining of frequent closed sequences. In: Proceedings of the 20th International Conference on Data Engineering, ICDE '04. IEEE Computer Society, Boston, Massachusetts, USA. 0-7695-2065-0 http://dl.acm.org/citation.cfm?id=977401.978142.

Weingartner, R., Brascher, G.B., Westphall, C.B., 2015. Cloud resource management: a survey on forecasting and profiling models. J. Net. Comp. Appl. 47, 99–106.

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

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

Xu, C.-Z., Rao, J., Bu, X., 2012. URL: a unified reinforcement learning approach for autonomic cloud management. J. Parallel Distrib. Comp. 72 (2), 95–105. https://doi.org/10.1016/j.jpdc.2011.10.003. 0743–7315 http://www.sciencedirect.com/science/article/pii/S0743731511001924.

Xu, D.Y., Yang, S.L., Liu, R.P., 2013. A mixture of HMM, GA, and Elman network for load prediction in cloud-oriented data centers. J. Zhejiang Univ. SCIENCE C 14 (11), 845–858. https://doi.org/10.1631/jzus.C1300109 1869–1951.

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

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

Yang, Q., Peng, C., Zhao, H., Yu, Y., Zhou, Y., Wang, Z., Du, S., 2014. A new method based on PSR and EA-GMDH for host load prediction in cloud computing system. The J. Supercomputing 68 (3), 1402–1417. https://doi.org/10.1007/s11227-014-1097-x 0920–8542.

Zaki, M.J., 2001. Spade: an efficient algorithm for mining frequent sequences. Mach. Learn. 42 (1–2), 31–60. https://doi.org/10.1023/A:1007652502315 0885–6125.

Zhang, Q., Cherkasova, L., Smirni, E., 2007. 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, p. 27. https://doi.org/10.1109/ICAC.2007.1 Jacksonville, Florida, USA.

Zhenhuan, G., Xiaohui, G., Wilkes, J., 2010. PRESS: PRedictive elastic ReSource scaling for cloud systems. In: Proceedings of the 6th International Conference on Network and Service Management, CNSM 2010, pp. 9–16. https://doi.org/10.1109/CNSM.2010.5691343 Niagara Falls, Canada.

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