EHMM-CT: An Online Method for Failure Prediction in Cloud Computing Systems

Weiwei Zheng[1], Zhili Wang[1], Haoqiu Huang[1], Luoming Meng[1], and Xuesong Qiu[1]
[1] State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications
Beijing, China
[e-mail: {zhengweiwei, zlwang, francesco, lmmeng, xsqiu}@bupt.edu.cn]

Received November 10, 2015; revised July 5, 2016; accepted July 29, 2016; published September 30, 2016

Abstract

The current cloud computing paradigm is still vulnerable to a significant number of system failures. The increasing demand for fault tolerance and resilience in a cost-effective and device-independent manner is a primary reason for creating an effective means to address system dependability and availability concerns. This paper focuses on online failure prediction for cloud computing systems using system runtime data, which is different from traditional tolerance techniques that require an in-depth knowledge of underlying mechanisms. A 'failure prediction' approach, based on Cloud Theory (CT) and the Hidden Markov Model (HMM), is proposed that extends the HMM by training with CT. In the approach, the parameter ω is defined as the correlations between various indices and failures, taking into account multiple runtime indices in cloud computing systems. Furthermore, the approach uses multiple dimensions to describe failure prediction in detail by extending parameters of the HMM. The likelihood and membership degree computing algorithms in the CT are used, instead of traditional algorithms in HMM, to reduce computing overhead in the model training phase. Finally, the results from simulations show that the proposed approach provides very accurate results at low computational cost. It can obtain an optimal tradeoff between 'failure prediction' performance and computing overhead.

Keywords: Online failure prediction, cloud theory, hidden Markov model, cloud computing systems

A preliminary result of this paper was presented at IEEE ISCC 2015, July 06-09, Larnaca, Cyprus. This version includes a concrete analysis. This work was supported by NSFC (No.61501044).

http://dx.doi.org/10.3837/tiis.2016.09.004 **ISSN : 1976-7277**

1. Introduction

A large number of data centers are built in modern cloud computing systems. In general, these data centers are constructed using commodity servers, where commodity components are developed by different manufacturers. Due to the highly complex nature of the underlying infrastructure connected with the components, these systems thus incur a high risk of encountering failures and exceptions [1][3][12]. Data centers as the mainstay to efficiently meet the demand for service providers' cloud-based service raise the challenges of high availability and scalability. Such data centers usually carry a number of high-performance computing (HPC) applications, which distribute and replicate data on several nodes in order to meet stringent Quality of Service (QoS) requirements regarding the high availability of systems. As such, failures are inevitable and may lead to catastrophic consequences in the whole system. Specifically, failures in system nodes can abort applications, which usually span various nodes, resulting in little forward progress [2]. When the dynamics of the systems are also examined, susceptibility to cascading failures is revealed, which in particularly serious cases causes the entire system to be affected.

Failures that play a crucial role must be handled promptly to ensure system survivability and reliability in cloud computing systems. An appropriate method to avoid failures is to predict them, sensing the occurrence of anomalous behavior. Accurate and timely predictions also can mitigate the effect of failures by taking proper recovery actions before failures occur [12]. Liang et al. [2] have shown that the capability of predicting the time/location of the next failure, though not perfect, can considerably boost benefits of other runtime fault tolerance techniques.

As an innovative approach to further enhance system dependability and availability, failure prediction anticipates failures before occurring, and performs preventive strategies or reduces time-to-repair by preparation for imminent failures [8]. Failure prediction does not intend to identify the root cause of problems, rather its goal is to evaluate the current system state and to estimate the possibility of the failure occurring within a short future.

Current techniques for failure prediction and anomaly detection mainly focus on pattern matching and statistical analysis schemes, where failures or anomalies are considered as deviations from the normal behavior and are modeled regarding variability in systems [9][8]. Specifically, online machine learning and data mining are exploited to model behavior of systems, using error sequences or symptom-specific features, such as CPU and memory utilization. As one of the classic pattern matching models, the Hidden Markov Model (HMM) has been successfully applied to various pattern recognition tasks, for example, speech recognition, genetic sequence analysis as well as dependable computing including intrusion detection, fault diagnosis, and network traffic models [10].

Although HMMs have been extended for different scenarios and requirements, these extensions are not appropriate for the symptom monitoring-based failure prediction approach [10]. Additionally, HMM and its extensions usually apply machine learning techniques to model training and pattern identification. The computational complexity of the training is high because each step of the long iterative process re-estimates all the parameters numerically. To address the problem this paper exploits Cloud Theory (CT) because CT has been proven useful for solving the uncertain transitions between quantitative values and qualitative terms in training prediction models [11][17].

This paper focuses on online failure prediction that evaluates a current state of systems and makes a short-term failure prediction, supported by runtime symptoms monitoring-based

methods. In this work, the time-varying characteristics of the system are referred to as indices and form the basis of the failure prediction. Specifically, this paper proposes a failure prediction approach based on an extended HMM and CT (EHMM-CT). This method extends HMM and trains the model with CT. By analyzing the runtime indices that represent the states of the systems, the proposed EHMM-CT models the behavior of the systems. The main contributions of this paper are summarized as follows.

(1) It proposes an EHMM-CT for online failure prediction. Parameters in the model are extended to multiple dimensions so as to perform failure prediction based on various runtime indices. Moreover, a new parameter ω is defined to indicate the weight correlations between different indices and failures.

(2) It exploits CT to train the model. In the training algorithm, it evaluates system states by state division and then estimates the probability distributions based on maximum membership degree and likelihood. The proposed training algorithm can reduce the computational complexity substantially.

(3) The results from the simulations show the feasibilities and effectiveness of EHMM-CT. Most importantly, the execution time of the proposed failure prediction is greatly reduced regarding the promised prediction performance metric.

The rest of this paper is organized as follows. Section 2 shows a short review of HMM and CT, respectively. Then, the proposed EHMM-CT is presented in detail in Section 3. Section 4 provides the online failure prediction model design. Section 5 describes the evaluation environment and presents the performance of proposed EHMM-CT. Section 6 presents the related work. Finally, conclusions and future work are highlighted in Section 7.

2. Preliminaries

In this section, a brief overview of HMMs and CT is presented, respectively. In particular, for the CT, it shows the calculation process of likelihood in detail that can be applied directly to the EHMM-CT.

2.1 Review of Hidden Markov Models

HMMs are an extension to the Discrete Time Markov Chain (DTMC), which consists of three elements: i) a state set $S=\{s_i\}(i=1,...,N)$ containing N states; ii) a square matrix $\Lambda=\{a_{ij}\}(i,j=1,...,N)$ defining transition probabilities between the states; and iii) a vector $\pi=\{\pi_{ij}\}$ specifying the initial state probabilities. Additionally, HMMs are also determined by other two quantities, the symbol set *O* and the emission probability distribution *B*. Specifically, $O=\{o_i\}(i=1,...,P)$ is a finite, countable set, containing *P* different symbols. $\mathrm{B}=\{b_{ij}\}(i=1,...,N;j=1,...P)$ is a stochastic matrix, where b_{ij} is the probability for emitting symbol o_j, given that the stochastic process is in state s_i. For better readability, b_{ij} may sometimes be denoted by $b_i(o_j)$. In HMMs, only the outputs can be measured from outside, and the state of the stochastic DTMC process is hidden from the observer. Therefore, an HMM is usually described by a tuple of five elements HMM={S, O, A, B, π}, simply expressed by λ = {A, B, π} [10].

Standard HMM algorithms such as Baum-Welch and Forward-Backward [10] are adopted for model training (i.e., adjusting λ from a set of training sequences) as well as for efficiently computing sequence likelihood. Nonetheless, these algorithms are based on machine learning and estimate parameters by repeated recursions and iterations, which may need extensive computation and timely overhead.

2.2 Cloud Theory Overview

CT is a model describing the transition between quantitative data and qualitative term. It has been successfully applied to many areas [11][4][16][17][13], including for example, natural language processing, data mining, decision analysis, intelligent control, and image processingThe details of CT are presented below.

Cloud Model [11]. Let *U* be the universe of a discourse set, and *C* be the qualitative term associated with *U*. The membership degree of any element *x* in *U* to the qualitative term *C*, denoted by *membership(x,C)*, is a random number with a stable tendency taking the values in [0, 1]. Thus, the distribution of *x* in *U* is defined as *cloud model*, and each *x* is called a *cloud drop*.

Normal Cloud Model [11]. If *x* in *U* satisfies: i) the distribution of *x* in *U* is a *cloud model*; ii) $x \sim N(Ex, En'^2)$where $En'^2 \sim N(En, He^2)$, and iii) the membership degree of *x* to the qualitative term *C* is $membership(x,C) = e^{\frac{(x-Ex)^2}{2(En')^2}}$, then the distribution of *x* in *U* is defined as *normal cloud*.

Feature Vector of Cloud. A *normal cloud* can be expressed by a tuple of independent parameters *C=(Ex,En,He)*, called the *feature vector*. In the *feature vector*, the expected value *Ex* represents the overall level of the cloud model, while the entropy *En* presents the discrete degree, and hyper-entropy *He* denotes the uncertainty.

Cloud Generator [4][13]. *Cloud generators* are models realizing the transition between quantitative values and qualitative concept, and consist of *forward cloud generator* (CG) and *backward cloud generator* (CG^{-1}).

Given a *cloud model C=(Ex,En,He)*, the CG is used to generate several cloud drops $drop(x_i, membership(x_i, C))$based on the *feature vector*; while CG^{-1} extracts the *feature vector* of *cloud model* from *cloud drops*.

Likelihood. Define likelihood $likelihood(C_1, C_2)$ between two given *cloud models* $C_1 = (Ex_1, En_1, He_1)$ and $C_2 = (Ex_2, En_2, He_2)$ as the total distances of drops in the clouds, whose range is [0,1]. The details are shown in **Algorithm. 1**.

Algorithm 1. Calculation process of likelihood.

Input: Clouds $C_1 = \{Ex_1, En_1, He_1\}$ and $C_2 = \{Ex_2, En_2, He_2\}$.

Output: $likelihood(C_1, C_2)$.

Generating *n* drops x_i based on cloud $C_1$by *GC*.

for each drop x_i **do**

Calculating $membership(x_i, C_2)$;

/*The membership degree of x_i to cloud C_2. */

end

$likelihood(C_1, C_2) = \frac{1}{n}\sum_{i=1}^{n} membership(x_i, C_2);$

/* Getting the average of the membership degrees. */

return $likelihood(C_1, C_2);$

3. EHMM-CT: An Extension of Hidden Markov Model

The proposed approach builds on the fundamental assumption that characteristic patterns of symptoms identify the failure-prone behavior. The goal for failure prediction is to evaluate the current system state by taking into account the risk that a failure occurs within a short interval in the future, regardless of the fault causing the failure.

EHMM-CT is a novel extension of HMMs, which is appropriate for failure prediction in cloud computing systems. Approaches in CT are introduced to model training, which can significantly reduce the amount and complexity of computation. EHMM-CT aims to model system behavior using historical runtime data. It estimates the probability of the failure occurrence using system state matching. The details of EHMM-CT are shown as follows.

3.1 What Is New about EHMM-CT

The proposed EHMM-CT is designed for failure prediction in cloud computing systems. EHMM-CT extends the HMMs presented in 2.1 in three aspects.

(1) Parameters in EHMM-CT, which represent the system states and possibility distributions, are extended to multiple dimensions so as to characterize failure prediction based on multiple runtime indices.

(2) For failure prediction, EHMM-CT uses the new parameter ω to express correlations between multiple indices and the occurrence of failures.

(3) A new training algorithm based on CT is exploited to reduce the number of computations and the time cost for training.

3.2 Parameters

The first extension builds on the fact that data collection for representing the states of systems may involve various indices, such as CPU utilization, memory utilization, and I/O requests. This paper, to characterize different states of these indices, extends the state set from a vector to a matrix $S = \{s_{ij}\}_{sn \times N}$, where *N* represents the number of indices, and *sn* is the maximum size of index states, thus s_{ij} presents the *i* th state of the *j* th index. For better readability, $S_t = \{s_1^t, ..., s_N^t\}$ is defined as the system state at the *t* th slot.

Also, a cube $\mathbf{A} = \{A_k\}_N$ is defined as the transition probabilities between states, where $A_k = \{a_{i,j}\}_{sn_k \times sn_k}$ is a stochastic matrix, satisfying: $\sum_{j=1}^{sn_k} a_{ij} = 1$, where sn_k is the size of a state set for index *k*.

A $sn \times N$ matrix of initial state probabilities $\pi = \{\pi_{ij}\}$ has to be specified. For each column *j* of π, $\sum_{i=1}^{sn_j} \pi_{ij} = 1$ must be satisfied.

A description of the stochastic process of EHMM-CT follows. An initial state for each index *k* is chosen in light of the probability distribution defined by π. Starting from the initial state, the process transits from one state to the next according to the transition probabilities defined by $\mathbf{A}$. Therefore, EHMM-CT satisfies the termed *Markov assumptions*, which can be expressed by the following equation:

$$a_{ij} = P\{s_k^{t+1} = s_{jk} \mid s_k^t = s_{ik}, s_k^{t-1}, ..., s_k^0\} = P\{s_k^1 = s_{jk} \mid s_k^0 = s_{ik}\} \quad (1)$$

where $i, j = 1, ..., sn_k; k = 1, ...N$.

The characteristics of *Markov assumptions* mean that: since the transition probability only depends on the immediate state, it has no memory of the states the process has travelled through. It is assumed that for a certain index, each state could reach any other state by one step.

In this work, it instantiates the set of symbols $\mathbf{O} = \{o_i\}$. Specifically, in cloud computing systems, the collected indices can be divided into three degrees: *normal*, *alarm* and *failure*, where *normal* represents that the runtime indices are in the normal condition; *alarm* denotes the abnormal of system resources and is relevant to a set of symptoms. *Alarm* can be classified in detail by the severity as *critical* alarm, *major* alarm, *minor* alarm, and *warning*. *Alarm* generally implies that part of the system indices are in abnormal but may not affect the running of the system [7]. *Failure* is the observation when the system is in malfunction. Hence, the observation set of the EHMM-CT is defined as $\mathbf{O} = \{N, A_{warning}, A_{minor}, A_{major}, A_{critical}, F\}$. Also, it defines O_t as the observations at the *t* th slot.

Moreover, a stochastic triple cube $\mathbf{B} = \{B_i\}_N$ is defined. For each element $B_k = \{b_{ij}\}_{sn_k \times P}$, the row *i* represents a probability distribution for state s_{ik}. Specifically, b_{ij} is the probability for emitting symbol o_j, given that the stochastic process is in state s_{ki}, i.e., $b_{ij} = P\{O_t = o_j \mid s_i^t = s_{ik}\}$. Hence, B_k has dimensions of $sn_k \times P$ and satisfies $\sum_{j=1}^{P} b_{ij} = 1$, where *P* is the size of symbols set $\mathbf{O}$.

For requirements of failure prediction in cloud computing systems, $\omega = \{\omega_1, ..., \omega_N\}$ is defined to represent correlations between multiple indices and the occurrence of failures, where the *i* th element of the vector is the weight probability of index *i* causing a failure. It constrains that

$$\begin{cases} \omega_i = P\{O_t = F \mid s_t^i = s_{*i}\} \\ \sum_{i=1}^{N} \omega_i = 1 \end{cases} \quad (2)$$

In Eq. (2), s_{*i} represents the value space of s_i^t and corresponds to the i th column of S, i.e., $s_{*i} \in \{s_{1i}, ..., s_{sn_i i}\}$. Thus, $P\{O_t = F \mid s_t^i = s_{*i}\}$ is the weight probability of index *i* causing a failure.

In summary, the EHMM-CT is exactly defined by EHMM-CT$= \{S, \mathrm{A}, \mathrm{B}, \pi, \omega\}$, simply expressed by $\lambda = \{S, \mathrm{A}, \mathrm{B}, \omega\}$.

4. Failure Prediction Model Design

In this section, the authors present how the model EHMM-CT $=\{S,A,B,\pi,\omega\}$ is trained and used for failure prediction. The EHMM-CT is called "hidden" since it is assumed that only the generated symbols can be observed and that the state of the stochastic process is hidden from the observer. In the case of failure prediction, the "hidden" can be mapped to the fundamental concepts of faults and symptoms.

- *Faults* are by definition unobserved. Thus, they are corresponding to the hidden *states*.
- *Symptoms* correspond to the manifestation of hidden states, which are *observation symbols*.

The proposed failure prediction approach consists of two phases: model training and failure prediction. The former mainly addresses how to adjust the model parameters so as to model the behavior of systems, whereas the latter focuses on the online failure prediction. **Fig. 1** shows the detail.

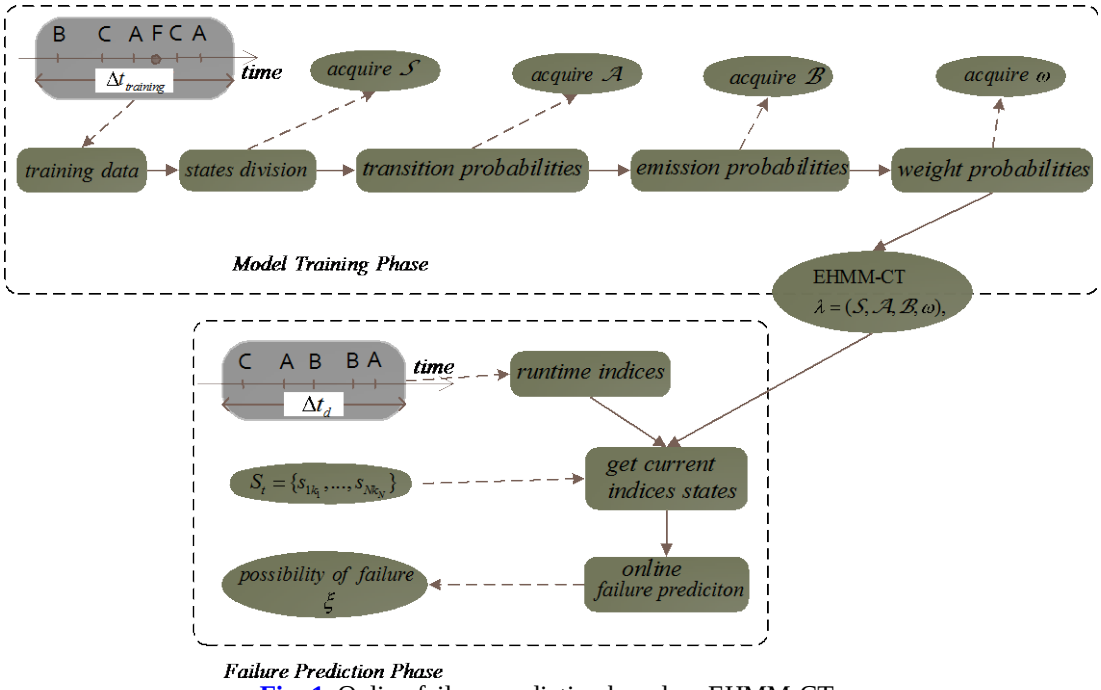

Fig. 1. Online failure prediction based on EHMM-CT.

4.1 Model Training Phase

In the model training phase, we focus on the estimation of model parameters reflecting the features of the runtime indices and behavior of the system. Instead of the standard HMM training algorithms (Forward-Backward and Baum-Welch which acquire the parameters through repeated iterations and recursions [10]), approaches in CT are used to estimate the parameters. By CT, the *feature vector* associated with the statistical characteristics of the training samples can be extracted. Then, the parameter set of EHMM-CT is estimated via the maximum *membership degree* and the maximum *likelihood* in CT and represents the "characteristic vector" associated with the training samples distribution.

4.1.1 State Division

In the state division, the model aims to instantiate the state set $\mathcal{S}$. In particular, it defines each element in $\mathcal{S}$ corresponding to a cloud model, called *state cloud*. Firstly, the number and

boundaries of divided *state clouds* for each index are calculated by estimating the index values (see later step (a)-(c)); and then the *feature vector* for each *state cloud* is extracted (see later step (d)). The details are as follows.

(a) Calculating the gradient for each index *j*, based on the range of index values. The gradient is defined as:

$$\theta_j = \frac{Ej_{max} - Ej_{min}}{\delta_j}, j = 1,2,...,N \tag{3}$$

Where Ej_{max}, Ej_{min} and δ_j are the maximum value, the minimum value, and the alarm trigger value of index *j*, respectively. In Eq. (3), the gradient represents the relative range of index values and reflects the deviation of the index value. It is calculated by dividing the index value range by the alarm trigger value of the index. The gradient, rather than the value range, can be exploited to characterize the diversity of the value range for various indices.

(b) Getting $\lceil \theta_j \rceil$ as the number of *state clouds*, sn_j, for index *j*, and *sn* is the maximum of the state cloud numbers. Acquiring the state interval Λ_j for each index *j* as follows:

$$\Lambda_j = \frac{Ej_{max} - Ej_{min}}{sn_j} = \frac{Ej_{max} - Ej_{min}}{\lceil \theta_j \rceil}, j = 1,2,...,N \tag{4}$$

The state interval corresponds to the difference of *Ex* for each state cloud and is defined as the average interval for each state. Thus, in Eq. (4), it is calculated by dividing the index value range by the number of state clouds.

(c) Estimating the state boundaries for each index *j*: since the range of index values is $[Ej_{min}, Ej_{max}](j = 1,2,...,N)$, the boundary of the *l* th state for index *j* is:

$$\begin{cases} Rj_{min}^{l} = Ej_{min} + (l-1) \cdot \Delta_j \\ Rj_{max}^{l} = Ej_{min} + l \cdot \Delta_j \end{cases}; 1 \le l \le sn_j \tag{5}$$

Thus, according to Eq. (5), the range of state *l* for index *j* is $[Rj_{min}^{l}, Rj_{max}^{l})$.

(d) Extracting the *feature vectors* of *state clouds in* $C_j = \{c_1,...,c_{sn_j}\}$ for each index *j*. Then, for each element c_l in C_j, the *feature vecto*r is:

- Expectation of c_l, denoted by Ex_l, is the average of index values in the boundary:

$$Ex_l = \frac{Rj_{min}^{l} + Rj_{max}^{l}}{2}, 1 \le l \le sn_j; \tag{6}$$

- Entropy of c_l, denoted by En_l, which presents the discrete degree, is calculated as follows:

$$En_l = \frac{Rj_{min}^{l} + Rj_{max}^{l}}{2 \times sn_j}, 1 \le l \le sn_j; \tag{7}$$

- Hyper-entropy of c_l, denoted by He_l, represents the randomness of the index in the boundary. A random value η between 0 and 1 is assigned to He_l. The random value may not be significant because the bigger the random value is, the greater, the error

of the expectation is and this may lead to an uncertain result. Thus He_l is

$$He_l = \eta \tag{8}$$

A failure prone state for each index *j*, denoted by S_j^*, is selected from the generated state sets above. The failure prone state for each index *j* is the one whose feature vector deviates most from the normal, and thus the possibility of triggering failures is the maximum.

4.1.2 Transition Probability

The cloud likelihood represents the similarity between clouds and is quantified by a value between 0 and 1. It is acquired by adding up the distances of drops from clouds (as shown in **Algorithm 1**). In this model, since the states of the index are presented by state clouds, the cloud likelihood can be exploited when estimating the state transition probabilities. For index *k*, the transition probability distribution A_k is calculated by

$$a_{ij} = \begin{cases} likelihood(c_i, c_j), & if \ i \neq j \\ 1 - \sum_{1 \le h \le sn_k, h \neq i} likelihood(c_i, c_h), & if \ i = j \end{cases} \tag{9}$$

where a_{ij} is the probability of state s_{ik} transiting to state s_{jk}, and $likelihood(c_i, c_j)$ is the likelihood of clouds c_i and c_j (see Section 2.2) and its range is [0, 1].

4.1.3 Emission Probability

The emission probability indicates the possibility of getting a certain observed symbol at the specified state. In EHMM-CT, the *membership degree* in CT is exploited to describe the emission probability. The bigger the *membership degree* is, the more the fitting is presented between the observation and states (i.e., the more possibilities to output the observation at the corresponding state), and vice versa.

Therefore, it calculates the *membership degrees* for the index data collected at different slots. Then, the observation probability is defined as the average of *membership degrees*. The formal process is shown in **Algorithm. 2**.

Algorithm 2. Estimating the emission probability distribution.

Input: observed symbols $O=\{o_1,...,o_N\}$ and a state matrix $S=\{s_{ij}\}$

Output: emission probability distribution B

for each index *j=1:N* **do**
  for each state *i = 1:sn* **do**
    for each observation o_k (k=1:P) in **O do**

```
membership(o_k, s_ij);
/*Getting the membership degree between the observed symbol and state i. */
symbol_degree = get_symbol_degree(o_k);
/* Calculating the classification of the observed symbol and */
/* the range of the symbol_degree is {N, A_warning, A_minor, A_major, A_critical, F}. */
total(symbol_degree)+= membership(o_k, s_ij);
count(symbol_degree)+=1;
```

/* Adding up the membership degrees by *symbol_ degree*. */
end
end
$b(k,symbol\_degree,j) = total(symbol\_degree) / count(symbol\_degree)$;
/* Computing the average of the membership degrees. */
end
return $b(k,symbol\_degree,j)$;

4.1.4 Weight Probability

The correlations between indices and failure occurrences are presented in three aspects: *DF* (Dispersion Frame) [15], *FDI* (Failure Dispersion Index), and *Rate*. For each index *j*, a *DF* is the interval time (number of slots) between successive alarms, symbolized by $T = (\Delta\tau_1, \Delta\tau_2, ...)$; *FDI* is defined as the number of failures $N = (n_1, n_2, ...)$ observed in a *DF*; and *Rate* is defined as the ratio of failure encountered DFs over all DFs, that is, *Rate* represents the correlation between the alarms of index *j* and failure occurrences and is expressed by γ_j. **Fig. 2** shows how to obtain the *DF*, *FDI*, and *Rate*, respectively.

Therefore, the relationship among the index weight ω, *DF*, *FDI*, and *Rate* is presented as follows:

$$\omega_j \propto \frac{\overline{N_j}\gamma_j}{\overline{T_j}} \tag{10}$$

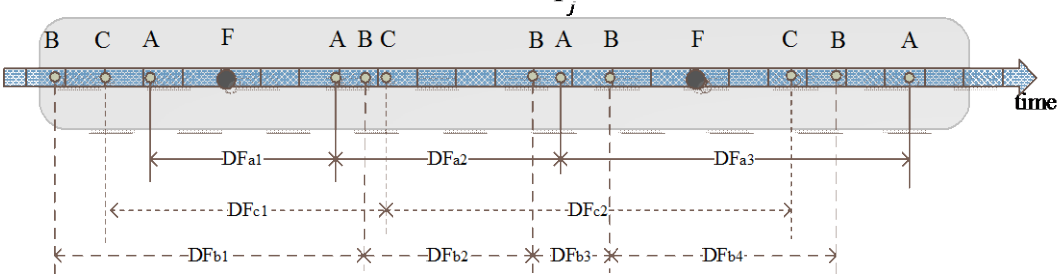

Fig. 2. Method to determine indices weights. For example, the values of *DF*, *ADI*, and *Rate* for index A, are $T_a = [4,6,9]$, $N_a = [1,0,1]$, $\gamma_a = 2/3$, respectively; For index B and C, $T_b = [8,4,2,6]$, $N_b = [1,0,0,1]$, $\gamma_a = 2/4$; $T_c = [8,10]$, $N_c = [1,1]$, $\gamma_c = 1$.

Where $\overline{N_j}$ and $\overline{T_j}$ are the average of *DF* and *FDI* for index *j*, respectively.

Finally, we normalize the weight probability, according to the formula (10). Thus, the weight probability $\omega = \{\omega_1, ..., \omega_N\}$ satisfies:

$$\omega_i = \frac{\sigma_i}{\sum_{j=1}^{N}\sigma_j} \tag{11}$$

where $\sigma_k = \frac{\overline{N_k}\gamma_k}{\overline{T_k}}, k = 1, ..., N.$

4.2 Failure Prediction Phase

4.2.1 Preprocess of Runtime Indices

In this section, the current states of indices, named *index states*, are evaluated and represented as clouds. In particular, it extracts the feature vector of *index state* for each index based on collected runtime data matrix $X = \{x_{ij}\}_{T\times N}$. Note that, for the matrix, the row represents the relative collected time of the runtime data, i.e., the bigger the row index is, the fresher the data is.

To evaluate the timeliness of runtime system indices, a constant is defined as the time impact factor *TIF* to differentiate the impacts of index data collected at different slots on failures. Recently collected data for failure prediction is more valuable than previously obtained data- see later equation (12).

For the collected runtime indices data $X = \{x_{ij}\}_{T\times N}$, the feature vector of index states is estimated based on CG^{-1} while considering each index sample x_{ij} as a cloud drop. The process of the evaluation is shown as follows.

(a) For row *i*, $X_i = (x_{i1},...,x_{iN})$ in *X*, estimating the membership degree μ_i using:

$$\mu_i = e^{-\frac{(T-i)}{T}\times \ln TIF} \tag{12}$$

In Eq. (12), μ_i is defined for the index data collected at slot *i* to reflect the impacts of index timeliness on failure prediction. In particular, a bigger *i* (i.e., the collected time is more closer) will lead to a higher μ_i, which increases weight x_{ij} when calculating the average value $\overline{x_j}$ - see (b).

(b) Averaging the values of index *j* as the expectation *Ex* of index state IS_j, where $\overline{x_j} = \frac{1}{T}\sum_{i=1}^{T} x_{ij} \times (TIF + \mu_{ij})$ and $Ex = (Ex_1,...,Ex_N) = (\overline{x_1},...,\overline{x_N})$.

(c) For each cloud drop (x_{ij}, μ_{ij}), evaluating the membership degree to the index state, and using the average as the entropy *En* of current state:

$$\begin{cases} En_{ij} = \dfrac{|x_{ij} - Ex_j|}{\sqrt{-\dfrac{\ln \mu_{ij}}{TIF}}} \\ En_j = \dfrac{1}{T}\sum_{i=1}^{T} En_{ij} \end{cases} \tag{13}$$

(d) Calculating the normal variance of each entropy En_j as the hyper-entropy He_j:

$$He_j = \sqrt{\frac{1}{T-1}\sum_{i=1}^{T}(En_{ij} - En_j)^2} \tag{14}$$

Thus, the current index states are denoted by $IS = \{is_1,...,is_N\}$, where $is_j = (Ex_j, En_j, He_j)$.

4.2.2 Failure Probability Evaluation

In the failure probability evaluation, the model needs to estimate the possibility of the failure occurrence based on the trained EHMM-CT and current index states $IS = \{is_1, ..., is_N\}$ acquired in Section 4.2.1. The details are as follows.

(a) For each index *j*, the likelihood between the index state is_j and every state $s_{\bar{y}}$ in trained model is evaluated, and then the matched state is estimated.

$$state(is_j) = \max_{\bar{y}} \{likelihood(is_j, s_{\bar{y}})\} \tag{15}$$

(b) For each index *j*, calculating the transition probability P_j that the current state transits to the failure prone state within prediction time τ:

$$P_j = P(S(t+\tau) = s_j^* \mid S(t) = state(is_j)) \tag{16}$$

(c) Getting the weight average of the failure triggering probabilities $P = \{P_j\}_N$ with the weight probability $\omega = \{\omega_1, ..., \omega_N\}$ as the probability of the failure occurrence.

$$\xi = \sum_{j=1}^{N} \omega_j \times P_j \tag{17}$$

(d) The failure is predicted if the probability ξ exceeds the failure prediction threshold ϑ.

5. Experiments and Analysis

5.1 Evaluation environment

Online prediction of eventually upcoming failures is shown in **Fig. 3**. If a prediction is performed at time *t*, it would like to know whether during $[t+\Delta t_l, t+\Delta t_l+\Delta t_p]$ a failure will occur or not, based on the trained prediction model and data within the data window Δt_d [15].

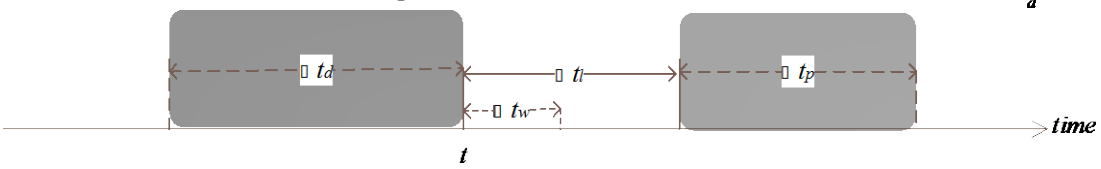

Fig. 3. Online failure prediction: *t* is the present time; Δt_d represents the data window size; Δt_l denotes the lead-time; Δt_w is the warning time and Δt_p symbolizes the prediction period.

The performance of the proposed approach is evaluated via MATLAB with runtime index data collected from our laboratory-wide cloud computing system. The system contains 27 high-performance and connected computing servers dedicated to computational research. The computing servers are virtualized into a cloud computing resource pool. Multiple virtual machines are created using the virtual resource carrying parallel or cooperative applications, such as distributed data mining algorithms and research simulations. In the simulations, performance indices of the cloud computing system like CPU utilization, memory utilization, and I/O rate, are extracted for the failure prediction. A failure is defined as the event when a system ceases to fulfill its specification [15]. **Fig. 4** presents the number of alarms per slot (which is equal to the measurement period of the system, i.e., 30s) in part of test data. As

shown in the figure, the amount of alarms per slot varies heavily, and there is a correlation between the occurrence of failures (represented by triangles in **Fig. 4**) and alarms. **Fig. 5** is a histogram of time-between-failures (TBF). It can be seen from the histogram of TBF that the distribution of failure is wide-spread. There is no periodicity evident about the failure occurrence. **Fig. 4** and **Fig. 5** show that, in the simulations, the causality between alarms and failures can be directly observed, and the distribution of failures is random and not uniform.

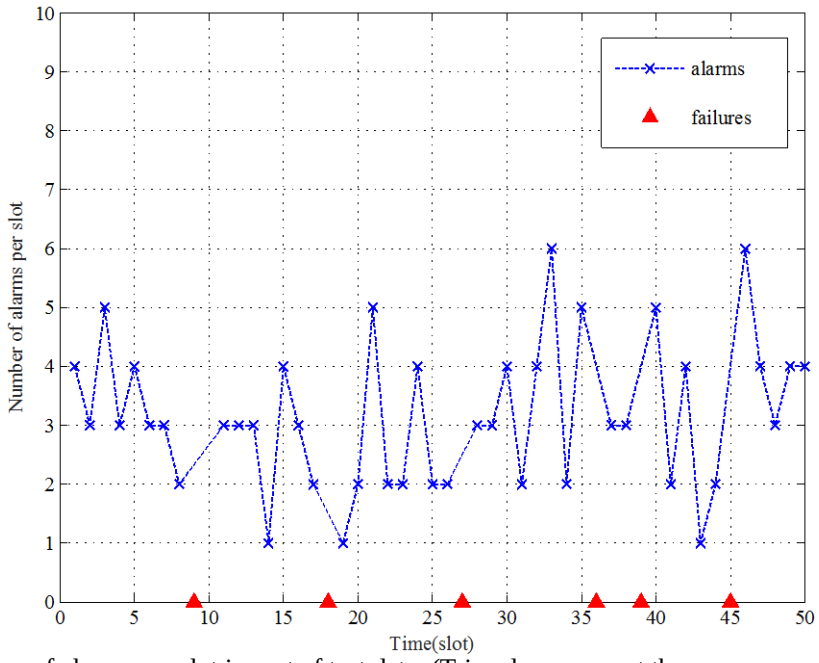

Fig. 4. Number of alarms per slot in part of test data. (Triangles represent the occurrences of failures)

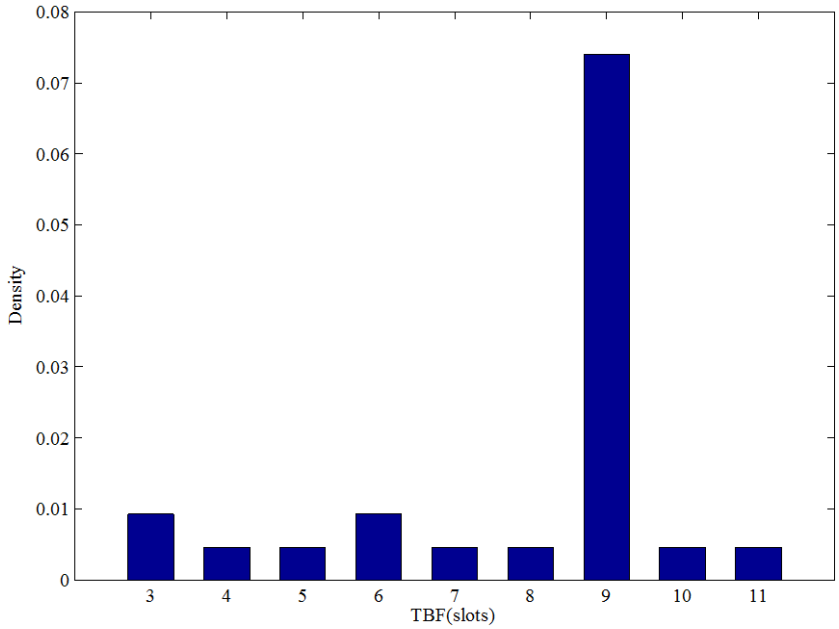

Fig. 5. Time between failures (TBF).

5.2. Performance

5.2.1 Feasibilities

In this section, the execution scenario is designed and the feasibilities of the EHMM-CT for failure prediction are verified. The steps of the simulation are as follows.

(a) Dividing the states and estimating the feature vectors of the states for each index based on the training data as described in Section 4.1.1. **Fig. 6** shows the state clouds of a randomly chosen index. The index data is divided into four states, based on the value range.

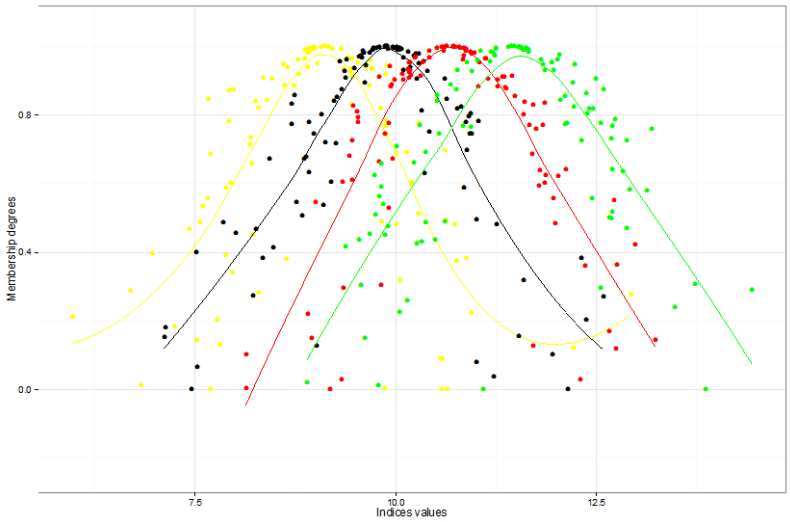

Fig. 6. Generation of state clouds.

(b) The state transition probabilities and alarm emission probabilities are evaluated, based on the divided states and feature vectors. **Fig. 7** illustrates the hidden states and the corresponding probability distribution generated above.

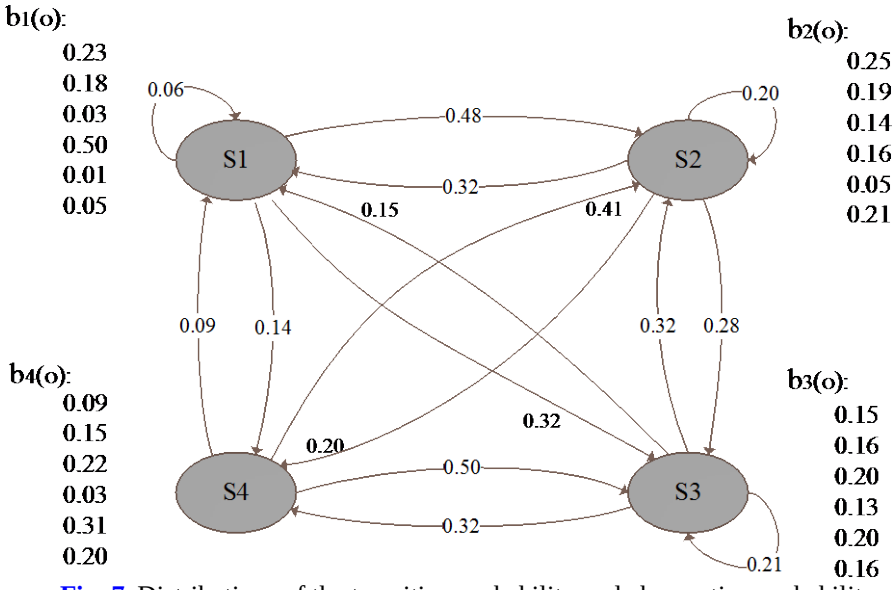

Fig. 7. Distributions of the transition probability and observation probability.

In **Fig. 7**, the ellipses represent the hidden states. The line with the arrow between the ellipse and the value on the line present the state transition direction and the corresponding transition probability. The labels near the states are the alarm emission probabilities in terms of the alarm severity order. For example $b_1(o)$ represents the individual possibility to get *normal*, *warning*, *minor*, *major* and *critical* at state S_1 which are 0.23, 0.18, 0.03, 0.50, 0.01 and 0.05, respectively. As such, the trained EHMM-CT can be generated.

(c) Getting the runtime index states based on the data within the data window. Then, evaluating the matching degree between the index states and elements in the state set of the trained model, and obtaining the possibility of the matched state triggering a failure. **Table 1** is the probability distributions of state matching and triggering a failure for a certain index.

Table 1. Probability distributions of the state matching and triggering a failure.

States	S_1	S_2	S_3	S_4
Likelihood between states	0.13	0.33	0.12	0.20
Possibility of a failure	0.0065	0.1155	0.0024	0.0032

(d) Adding up all the probabilities of each index triggering a failure, and comparing with the threshold ϑ. Finally, a failure can be predicted if the threshold is exceeded.

5.2.2 Effectiveness

For evaluating the performance of the proposed EHMM-CT, metrics in [15] including *precision*, *recall* and *F-measure* are introduced. A perfect failure prediction is to achieve a one-to-one matching between predicted and actual failures which results in *precision=recall=1* and *fpr=0* [8].

Additionally, there is an inverse proportionality between high *recall* and high *precision*, which means that improving *recall*, in most cases, leads to a lower *precision* and vice versa.

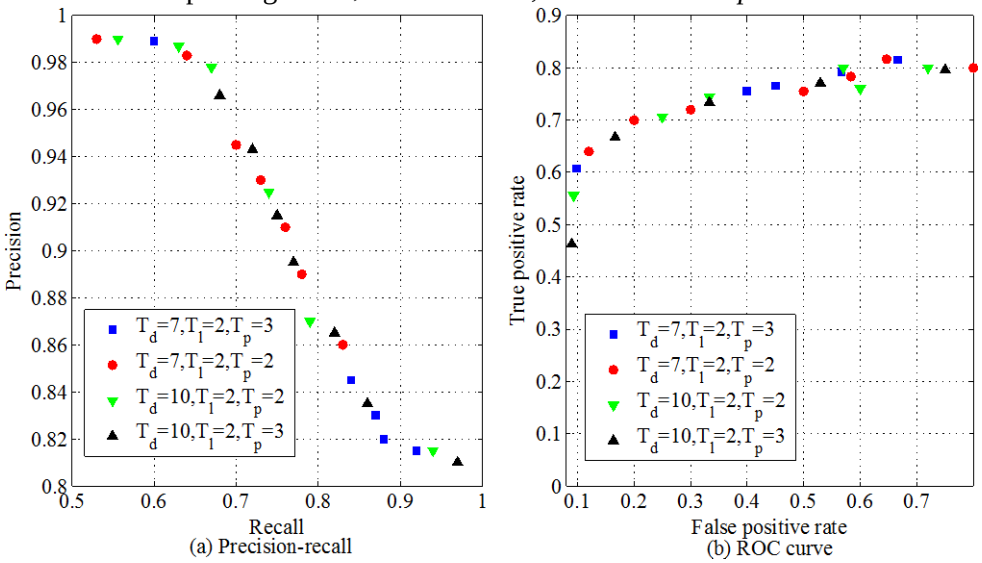

Fig. 8. Failure prediction performance of the EHMM-CT approach. The various symbols are different parameter settings.

Then, the same inverse proportionality is for *recall* and *fpr*. The proposed EHMM-CT provides a customizable failure prediction threshold ϑ for classification. By varying the threshold, the inverse proportionality can be controlled. **Fig. 8** is the *precision/recall* plot and *recall/fpr* (so-called ROC plot), which are plotted by fluctuating the value of parameters like Δt_d, Δt_l and Δt_p. The proposed approach provides a customizable threshold ϑ by which the trade-offs, e.g., between *precision* and *recall*, can be controlled.

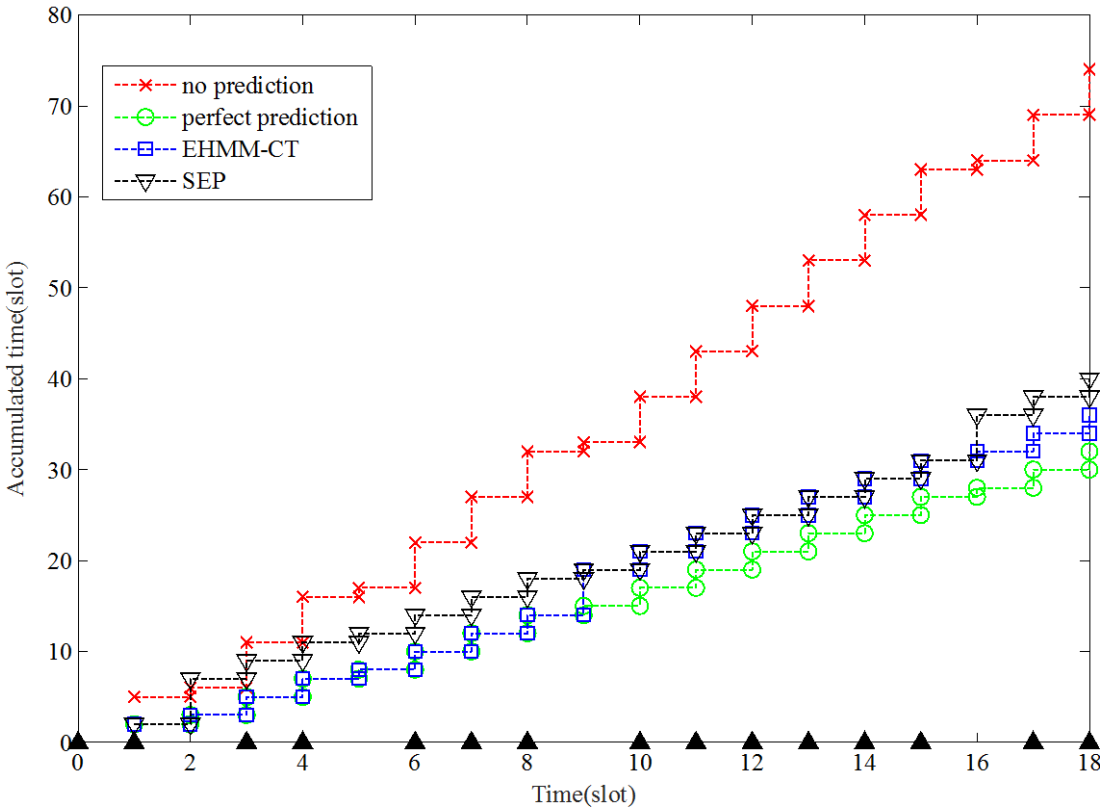

Fig. 9. Accumulated cost for each technique. Cost of 1 has been assigned to true positives, 2 to false positives, and 5 to false negative predictions. Triangles represent failures that occurred in the data set.

Fig. 9 illustrates the accumulated runtime cost for the system. It shows a run of 18 slots containing 14 failures, denoted by black triangles. For comparison, we added two curves: "perfect prediction" and "no prediction". The former refers to the case where a "perfect failure predictor" is to predict all failures without any mis-prediction. The latter refers to a system with neither a predictor nor any reaction scheme in place; once a failure occurs, execution costs increase by 5, which is the overhead for false negatives. We use the SEP approach proposed in [15] as a comparison.

As shown in **Fig. 9**, the performance curve of proposed EHMM-CT is closed to the cost curve of perfect prediction, and can predict most of the failures (out of 14 failures, 13 were predicted), which reduces the cost for handling failures. As time goes on, the accumulated execution cost of the proposed approach is less than the SEP in [15], and the EHMM-CT performs advantages in prediction accuracy.

Table 2 is the summary of performance compared with HSMM in [8] and FABM in [18], taking into account the impact of thresholds on the performance of EHMM-CT. The table shows the prediction performance with different threshold settings. These settings are based on the statistical characteristics of the prediction probabilities outputted by the EHMM-CT during training, including the average (AVE), the minimum (MIN) and the median (MED) of the prediction probabilities. As shown in the table, EHMM-CT with different settings provides better prediction performance than HSMM. In particular, EHMM-CT with MED achieves the maximum *F-measure* of 0.8058 with *precision*=0.8348, *recall*=0.7787. Note that the EHMM-CT allows employing customizable thresholds by which the trade-off between *precision* and *recall* can be adjusted for the various requirements. For example, for systems where once a failure occurs the processing cost is heavy, it is necessary to set lower thresholds to get a higher *recall* and to prevent the omission of failures. Thus the setting as MIN is preferred; whereas for a system where the occurrence of failures is frequent, and a higher *precision* is required, it is appropriate to set as AVE.

Table 2. Summary of prediction results.

Prediction approaches	***Threshold setting***	***Precision***	***Recall***	***F-measure***
EHMM-CT	MIN	0.7800	0.8300	0.8042
	AVE	0.9231	0.7059	0.8000
	MED	0.8348	0.7787	0.8058
HSMM	--	0.8520	0.6570	0.7419
FABM	--	0.8390	0.9750	0.9020

The EHMM-CT can be trained without complex preprocessing of training data, and the computations of the model parameters estimation can be significantly reduced, compared with other prediction techniques. **Fig.10** is the plot of average *fpr* versus the average training time about EHMM-CT, HSMM in [8] and FABM in [18]. As shown in the figure, the FABM achieves the best average fpr (the average fpr is about 0.05) with the most training time (about 9 slots). The HSMM gets the worst prediction performance regarding average fpr but a faster convergence (about 7 slots) than the FABM. The EHMM-CT achieves the fastest convergence, that is, getting an acceptable prediction result (the average fpr is less than 0.1) with the least training time (about 5 slots). These can be explained because, for the FABM as well as the HSMM, they make a failure prediction using machine learning techniques to model training and pattern identification. Because each step of the long iterative process re-estimates all the parameters numerically, the computational complexity of the training for FABM and HSMM is higher, thus incurring longer training time. Among the three prediction approaches, EHMM-CT obtains an optimal trade-off between outstanding failure prediction performance and reduced computing complexity, which means that the time is significantly reduced under the promise of an acceptable prediction performance.

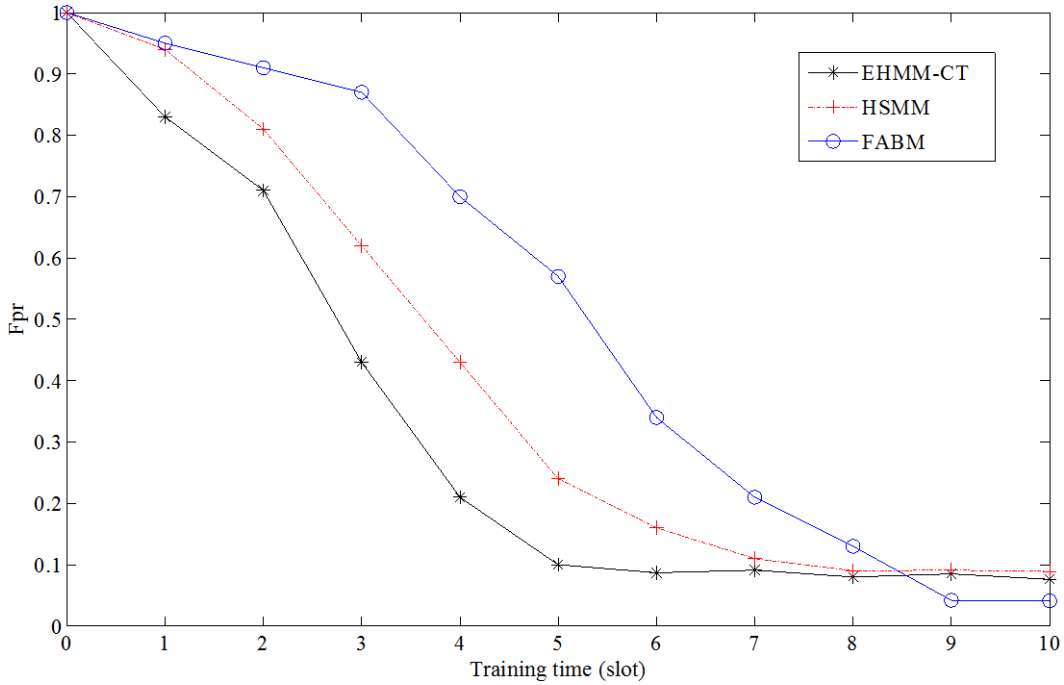

Fig. 10. *Fpr* versus training time for each prediction approach.

6. Related Work

In recent years, failure prediction and anomaly diagnosis have been paid special attention by ISPs and the network research community to provide better network management and get more resilient network systems [9]. In this paper, only those failure prediction and anomaly diagnosis that are closer to the proposed solution and that mainly inspired the approach followed are discussed.

In [6], finite-state machines (FSMs) are used to model correlations between network alarm sequences occurred during and before failures. Specifically, a probabilistic finite-state machine model is built by considering known network fault based on historical data collected during its occurrences.

In pattern matching approaches, online machine learning techniques are usually used to model the behavior of monitored data [5][6][14][9]. These approaches can be distinguished as symptoms monitoring techniques for predicting failures (as the proposed EHMM-CT does) and error monitoring mechanisms. Baldoni et al. [12] proposed an online prediction in safety-critical systems, where they monitored network traffic only to perform failure prediction, and the HMM was exploited to create a state recognizer. Salfner et al. [8] presented an error monitoring failure prediction technique that uses Hidden Semi-Markov Model (HSMM) to recognize error patterns, which can lead to failures. Hoffman et al. [6] proposed two error monitoring based approaches, one of which resorts to a Discrete Time Markov Model and the other which employs function.

As the cloud computing system evolves, most of the methods mentioned above require significant recalibration and "retraining". By continuously tracking the behavior of the system, the recalibration and failure detection become more automated, which provides vital support to autonomic fault management [7]. Therefore, based on these autonomic technologies, the EHMM-CT to profile the states of systems using historical runtime data is proposed. In the

EHMM-CT, the system behavior is modeled as the hidden state and failure events as a part of the symptom set is also considered. A new training algorithm is proposed in this paper for our EHMM-CT, and then the trained model can be used to predict failure in cloud computing systems.

7. Conclusions

This work has shown the online failure prediction approach, EHMM-CT, in cloud computing systems. EHMM-CT is a novel extension of HMMs, which is appropriate for failure prediction in cloud computing systems. Approaches in CT are introduced to model training, which can reduce the amount and complexity of computation substantially. EHMM-CT aims to model system behavior using historical runtime data. It estimates the probability of the failure occurrence by system state matching. Its performance is evaluated via our extensive simulations. Results from these simulations have shown that our approach for predicting failure in cloud computing systems is effective. Moreover, the optimal tradeoff between failure prediction performance and computational complexity are provided, showing the excellent performance regarding this tradeoff.

Still many extensions and refinements of our approach are possible, in particular for a universal and adaptive threshold setting mechanism in cloud computing systems. Moreover, as a promising fault tolerance technique, a resource reallocation exploiting results of the failure prediction will be a part of our future investigations.

References

[1] R. Jhawar and V. Piuri, *Computer and Information Security Handbook*, 2nd Edition, Elsevier, Waltham, 2013. Article (CrossRef Link)

[2] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R. Sahoo, "BlueGene/L failure analysis and prediction models," in *Proc. of IEEE Conf. on Dependable Systems and Networks*, pp. 425-434, June, 2006. Article (CrossRef Link)

[3] X. Wang, H. Sun, T. Deng, and J. Huai, "On the tradeoff of availability and consistency for quorum systems in data center networks," *Computer Networks*, vol. 76, pp. 191-206, January, 2015. Article (CrossRef Link)

[4] C. Liu, D. Li, Y. Du, and X. Han, "Some statistical analysis of the normal cloud model," *Information and Control*, vol. 34, no. 2, pp. 236-239+248, 2005. Article (CrossRef Link)

[5] S. Fu and C.-Z. Xu, "Quantifying temporal and spatial correlation of failure events for proactive management," in *Proc. of 26th IEEE Int. Symposium on Reliable Distributed Systems*, pp. 175-184, October, 2007. Article (CrossRef Link)

[6] G. A. Hoffmann, F. Salfner, and M. Malek, "Advanced failure prediction in complex software systems," *Technical Report*, 2004. Article (CrossRef Link)

[7] R. Chaparadza, N. Tcholtchev, and V. Kaldanis, "How Autonomic Fault-Management Can Address Current Challenges in Fault-Management Faced in IT and Telecommunication Networks," *Access Networks*, vol. 63, pp. 253-268, 2011. Article (CrossRef Link)

[8] F. Salfner and M. Malek, "Using hidden semi-markov models for effective online failure prediction," in *Proc. of 26th IEEE Int. Symposium on Reliable Distributed Systems*, pp. 161-174, October, 2007. Article (CrossRef Link)

[9] A. K. Marnerides, A. Schaeffer-Filho, and A. Mauthe, "Traffic anomaly diagnosis in Internet backbone networks: A survey," *Computer Networks*, vol. 73, pp. 224-243, November, 2014. Article (CrossRef Link)

[10] F. Salfner, "Modeling event-driven time series with generalized hidden semi-Markov models," *Technical Report*, 2006. Article (CrossRef Link)

[11] D. Li and C. Liu, "Study on the universality of the normal cloud model," *Engineering Science*, vol. 6, no. 8, pp. 28-34, 2004. Article (CrossRef Link)
[12] R. Baldoni, L. Montanari, and M. Rizzuto, "On-line failure prediction in safety-critical systems," *Future Generation Computer Systems*, vol. 45, pp. 123-132, April, 2015. Article (CrossRef Link)
[13] H. S. Huang and R. C. Wang, "Subjective trust evaluation model based on membership cloud theory," *Journal of Communication*, vol. 29, no. 4, pp.13-19, 2008. Article (CrossRef Link)
[14] P. Casas, J. Mazel, and P. Owezarski, "UNADA: Unsupervised network anomaly detection using sub-space outliers ranking," *NETWORKING*, vol. 6640, pp. 40-51, May, 2011. Article (CrossRef Link)
[15] F. Salfner, M. Schieschke, and M. Malek, "Predicting failures of computer systems: A case study for a telecommunication system," in *Proc. of 20th IEEE Int. Symposium in Parallel and Distributed Processing*, April, 2006. Article (CrossRef Link)
[16] H. L. Li, C. H. Guo, and W. R. Qiu,. "Similarity measurement between normal cloud models," *Acta Electronica Sinica*, vol. 39, no. 11, pp. 2561-2567, 2011. Article (CrossRef Link)
[17] S. B. Zhang, C. X. Xu, and Y. J. An, "Study on the Risk Evaluation Approach Based on Cloud Model," *Chinese Journal of Computers*, vol. 42, no. 1, pp. 92-68, 2013. Article (CrossRef Link)
[18] H. J. Abed, A. Al-Fuqaha, B. Khan, and A. Rayes, "Efficient failure prediction in autonomic networks based on trend and frequency analysis of anomalous patterns," *International Journal of Network Management*, vol. 23, no. 3, pp. 186-213, 2013. Article (CrossRef Link)

Weiwei Zheng is a Ph.D. candidate from State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications (BUPT), Beijing, China. She received her M.Sc. from BUPT. Her main research interests include fault-tolerant communication networks, cloud computing systems and web services.

Zhili Wang is an Associate professor from State Key Laboratory of Networking and Switching Technology, BUPT, Beijing, China. He received a PhD in Computer Science from Beijing University of Posts and Telecommunications in 2005. His research interests include web services, quality of service (QoS), fault tolerance and cloud network service resiliency.

Haoqiu Huang is a Ph.D. candidate from State Key Laboratory of Networking and Switching Technology, BUPT, Beijing, China. He received his M.Sc. from Shenyang Ligong University. His main research interests include network management and communications software.

Luoming Meng is a professor and Ph.D. supervisor in BUPT. He graduated from Tsinghua University in 1987. He is the director of Communications Software Technical Committee of China Institute of Communications and the chairman of the National Network Management Standards Study Group. His research interests include communication network, network management, and communications software.

XueSong Qiu received a PhD in Computer Science from Beijing University of Posts and Telecommunications in 1999. Currently, he is a professor of BUPT and deputy director of State Key Laboratory of Networking and Switching Technology. He published about 150 SCI/EI index papers and received 13 national and provincial scientific and technical awards. He was selected as a winner of Beijing Higher Education Young Elite Teacher Plan in 2005. He is a Web Chair in 2016 IEEE/ACM IWQoS. His research interests include communication network and communication software, Internet routing and applications, smart service provisioning in mobile social network, network management.