ARF-Predictor: Effective Prediction of Aging-Related Failure Using Entropy

Pengfei Chen, Yong Qi, *Member, IEEE*, Xinyi Li, Di Hou, and Michael Rung-Tsong Lyu, *Fellow, IEEE*

Abstract—Even well-designed software systems suffer from chronic performance degradation, also known as "software aging", due to internal (e.g., software bugs) or external (e.g., resource exhaustion) impairments. These chronic problems often fly under the radar of software monitoring systems before causing severe impacts (e.g., system failures). Therefore, it is a challenging issue how to timely predict the occurrence of failures caused by these problems. Unfortunately, the effectiveness of prior approaches are far from satisfactory due to the insufficiency of aging indicators adopted by them. To accurately predict failures caused by software aging which are named as Aging-Related Failure (ARFs), this paper presents a novel entropy-based aging indicator, namely Multidimensional Multi-scale Entropy (MMSE) which leverages the complexity embedded in runtime performance metrics to indicate software aging. To the best of our knowledge, this is the first time to leverage entropy to predict ARFs. Based upon MMSE, we implement three failure prediction approaches encapsulated in a proof-of-concept prototype named ARF-Predictor. The experimental evaluations in a Video on Demand (VoD) system, and in a real-world production system, AntVision, show that ARF-Predictor can predict ARFs with a very high accuracy and a low *Ahead-Time-To-Failure (ATTF)*. Compared to previous approaches, ARF-Predictor improves the prediction accuracy by about 5 times and reduces $ATTF$ even by 3 orders of magnitude. In addition, ARF-Predictor is light-weight enough to satisfy the real-time requirement.

Index Terms—Software aging, performance degradation, multi-scale entropy, failure prediction, availability

1 Introduction

Software is becoming the backbone of modern society. Especially with the development of cloud computing, more and more software systems are deployed in the cloud and work in a distributed way. Two common characteristics of those software systems, namely long-running and high complexity, increase the risks of resource exhaustion and faults. With the accumulation of faults or resource consumption, software systems may suffer from chronic performance degradation, failure rate/probability increase and even crashes. This phenomenon is known as "software aging" [1], [2], [3], [4], [5] or "Chronics" [6].

Software aging has been extensively studied for two decades since it was first quantitatively analyzed in AT&T lab in 1995 [7]. This phenomenon has been widely observed in variant software systems such as cloud computing infrastructure (e.g., Eucalyptus) [8], [9], virtual machine monitors (VMMs) [10], [11], operating systems [1], [12], Java Virtual Machines (JVMs) [5], [13], web servers [4], [14] and so on. Currently, the fundamental mechanism of software aging has been partially uncovered [15]. Moreover, a common point is that software aging is a complex process influenced by many factors such as software bugs [15], [16], resource utilization [1], [2], [17], workload [18], [19], [20], [21], etc. In a long-running software system, resource exhaustion is not an uncommon phenomenon. For instance, the disk space is exhausted due to continuous logging. As the degree of software aging increases, software performance degrades gradually, resulting in QoS (e.g., response time) violations. When the QoS is lower than a preset threshold, the software system steps into an unserviced state which is also called "pseudo failure" [20]. For instance, for a web server, when the response time exceeds a preset threshold (e.g., 20 seconds), the system is unable to provide normal service any more. As the accumulation of new requests, the system goes to failure quickly. The unplanned outage caused by software aging in enterprise systems especially in cloud platforms can lead to considerable revenue loss. A recent survey shows that the IT downtime on an average can reach 14 hours of downtime per year, leading to $26.5 billion lost [22]. Therefore, prediction and counteracting the failures caused by software aging are of essence for building a reliable software system.

An efficient and commonly used counteracting software aging strategy is "software rejuvenation" [3], [4], [5], [23], which proactively recovers the system before the occurrence of failures to a completely or partially new state by cleaning the internal state. The benefit of rejuvenation strategies heavily relies on the time conducting rejuvenation actions. Frequent rejuvenation actions may cause a negative impact on the system availability due to the non-ignorable planned downtime or overhead caused by such actions. Instead, an ideal rejuvenation strategy is to recover the system when the system gets near to a failure . The failure caused by software aging is referred to as the "Aging-Related Failure" (ARF)

- *P. Chen, Y. Qi, X. Li and D. Hou are with the Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China. E-mail: chenpengfei@outlook.com, {qiy, houdi}@mail.xjtu.edu.cn, xingga.li@stu.xjtu.edu.cn.*
- *M.R. Lyu is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong. E-mail: lyu@cse.cuhk.edu.hk.*

Manuscript received 4 Aug. 2015; revised 26 June 2016; accepted 14 Aug. 2016. Date of publication 30 Aug. 2016; date of current version 6 July 2018. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TDSC.2016.2604381

[24]. Different from sudden crashes with no warnings, e.g., segment faults or hardware failures, ARF is a kind of "chronics" [6] which means some durable features can help us predict system crashes. However, to effectively predict such failures, we confront the following three challenges:

- Different from fail-stop problems, non-crash problems caused by software aging, where the server does not crash but fails to process the request compliant with the SLA constraints, have no observable and sufficient symptoms to indicate them. These problems often fly under the radar of monitoring systems. Hence, finding out an indicator to reveal the hidden software aging state becomes the first challenge.
- The internal changes (e.g., memory leak) and external changes (e.g., workload variation) make the running system extraordinarily complex. Hence, the running system can be described neither by a simple linear model nor by a single performance metric. How to handle the complexity and multi-dimension in the aging indicator is the second challenge.
- Fluctuations or noises may be involved in collected performance metrics due to the highly dynamic property of the running system. And the emerging cloud computing exacerbates the dynamics due to its elasticity and flexibility (e.g., VM creation and deletion). These noises may confuse the distinction between transient anomaly and ARF leading to additional rejuvenation cost. Thus, how to mitigate the influence of noise and keep the prediction approach noise-resilient is the third challenge.

The ARF prediction procedure is constructed on two bases. The first basis is an aging indicator. The other one is the prediction approach built on the aging indicator. Hence, an efficient aging indicator is the cornerstone of ARF prediction. However, the aging indicator is overlooked in previous work. To the best of our knowledge, very few studies focus on this topic. A commonly used aging indicator in previous studies [1], [13], [21], [25], [26] is *trend* embedded in the performance metrics. *trend* is adopted to predict the onset of software aging. If a downtrend or an uptrend is found, aging occurs. However, *trend* shows a weak power to predict ARFs. Because it aims to capture the long-term behavior of a software system rather than the short-term changes, which hinders the wide usage of *trend* in real-time ARF prediction. What is worse, there may be no significant increasing/decreasing trends in performance metrics when a failure occurs. This paper focuses on predicting ARFs rather than detecting the onset of software aging. To address the aforementioned challenges and the drawbacks of current studies, we conjecture that an ideal aging indicator should have *Monotonicity* property to reveal the hidden aging state, *Integration* property to comprehensively describe aging process, and *Stability* property to tolerate system fluctuations. In previous study [16], the authors stated that a concurrent bug was also the reason of software aging. Due to the uncertainty of the concurrent bug, the aging process may not have the monotonicity property and the ARF becomes unpredictable. In this paper, we restrict our approach to detecting such ARFs that have the feature of gradual degradation. Nevertheless, the aging problems caused by concurrent bugs can be resolved with the approaches coming from the effort of software engineer (e.g., [27]).

In this paper, we are the first to propose a novel entropy-based aging indicator, namely Multidimensional Multi-scale Entropy (MMSE). According to our experiments and practice, MMSE is capable of reflecting the hidden software aging state. MMSE is a complexity oriented and model-free indicator without deterministic linear or non-linear model assumption. In addition, the multi-scale feature mitigates the influence of system fluctuations and the multi-dimension feature makes MMSE effective in describing software aging. As we will see, MMSE satisfies the three properties namely *Stability*, *Monotonicity* and *Integration*, which we conjecture that an ideal aging indicator should have. Based upon MMSE, we develop three ARF prediction approaches encapsulated in a proof-of-concept prototype named ARF-Predictor.

The experimental evaluations in a VoD system deployed in a controlled environment and in a real production system, AntVision,[1] show that ARF-Predictor has a strong power to predict ARFs with a high accuracy and a small *ATTF*. Compared to previous approaches, ARF-Predictor increases the prediction accuracy by about 5 times and reduces the *ATTF* significantly even by 3 orders of magnitude. The contribution of this paper is three-fold:

- We demonstrate that entropy increases with software aging. Moreover, the empirical results show that this tends to be the case.
- We first put forward a novel aging indicator named MMSE. MMSE employs the complexity embedded in multiple runtime performance metrics to measure software aging, and leverages multi-scale and multi-dimension integration to tolerate system fluctuations, which makes MMSE satisfy the key properties, namely *Stability*, *Monotonicity* and *Integration*.
- We design and implement a proof-of-concept prototype named ARF-Predictor, and evaluate our approach in a real-world system with convincing results.

The rest of this paper is organized as follows. We demonstrate the motivations of this paper in Section 2. Section 3 demonstrates the proposed MMSE aging indicator. Section 4 describes the detailed design of ARF-Predictor including metric selections and ARF prediction approaches. Section 5 shows the evaluation results and comparisons to previous approaches. In Section 6, we state the related work briefly. Section 7 concludes this paper.

2 Motivation

The accuracy of Aging-Rriented Failure prediction approaches heavily relies on aging indicators. Moreover, the prediction result determines the rejuvenation cost. If the rejuvenation actions are always triggered at the time close to failures, the rejuvenation cost will tend to be optimal. But unfortunately, prior prediction approaches based upon explicit aging indicators such as free memory [1], [14], [26], memory usage [2], [5], [28], used swap space [1], [4], response time [25], [29] and so on do not function well

1. www.antvision.net

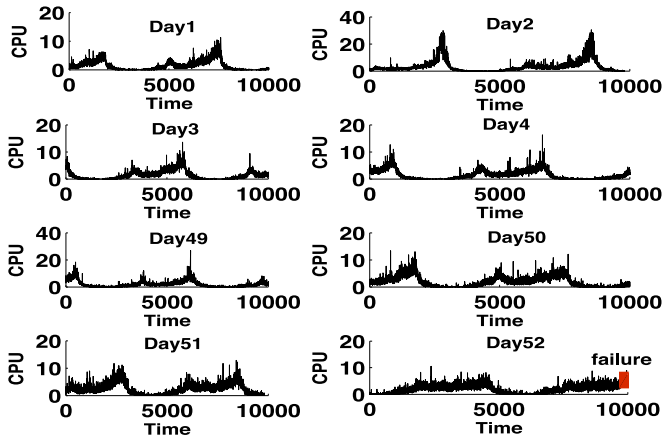


Fig. 1. The CPU utilization of a real VoD system. In this figure, we only show the CPU utilization of the first four days and the last four days.

especially in the face of dynamic workloads. They either miss some failures or mistake some normal states as failures. The insufficiency of previous indicators motivates us to seek new indicators.

As mentioned in [20], the aging level of a software system over time is denoted by $D(t), t > 0$. The failure at time t is equivalent to the aging level that exceeds a critical level D_f at time t [20]. Therefore, a threshold is always required to set on $D(t)$ in order to predict ARFs. However, $D(t)$ is always a hidden process. Suppose I is an aging indicator. I not only can indicate the onset of aging but also the aging level. To some extent, the degradation path of $I(t) \approx$ the degradation path of $D(t)$. Thus, we can set a threshold on $I(t)$ to predict ARFs. Traditionally, the threshold is always set on explicit aging indicators. We refer to the performance metrics which can be observed by common monitor tools as explicit aging indicators. For instance, take CPU utilization as an aging indicator. The CPU utilization increases with the aging level. When it exceeds 90 percent, an ARF occurs. However, it is not always the case. The explicit observations do not always accurately reveal the hidden aging levels.

For example, a real-world campus Video on Demand (VoD) system which is in charge of sharing movies amongst students runs for 52 days until a failure occurs. By manually investigating the reason of failure, we attribute it to an ARF as the quality of video broadcasting degrades gradually and no exceptions are thrown out in a console or in a log file. During the system running, the CPU utilization is recorded. Fig. 1 shows the CPU utilization of the first four days ($Day1, Day2, Day3, Day4$) and the last four days ($Day49$, $Day50, Day51, Day52$). A failure happened on Day52. However, the CPU utilization at the failure point seems normal (lower than 10 percent), meaning that the CPU utilization does not increase monotonically with the aging level in Helix Server. Thus, it is difficult to predict failures using this metric as an aging indicator. Moreover, we observe that other indicators such as RealMemoryFree [26], usedSwapSpace [1], [12], [26], Resident Set Size (RSS)[17] and Heap Usage Size (HUS) [17] indeed have an increasing/decreasing trend. But they do not satisfy a monotonicity property, meaning that it is difficult to determine when an ARF occurs. Therefore, we need a novel aging indicator.

Conjecture. According to the above observation, we provide a high level abstraction of the properties that an ideal aging indicator should have. *Monotonicity:* As proposed in [30], the aging effect is not reversible without external intervention. Moreover, it depends on the clock time [30]. Thus, software aging is a gradual deterioration process, which is a common assumption in prior articles, e.g., [20]. Given an aging indicator $I(t)$, we have $I(t) \leq I(t+1)$ or $I(t) \geq I(t+1)$ for every t. As the most essential property, the monotonicity is fundamental to predict an ARF. But it is extraordinarily difficult to find such an aging indicator subject to a monotonic constraint due to system noises and fluctuations. *Stability:* To robustly indicate software aging, the indicator should be tolerant to the noises and fluctuations involved in the runtime performance metrics. Namely, the indicator changes with time smoothly. The variance of $I(t)$ during an interval $[t_1, t_2]$, $t_1 < t_2$, is small. *Integration:* As software aging is a complex process affected by multiple factors, the indicator should cover these influences from multiple data sources, which means it is the fusion of multiple runtime metrics. In other words, $I(t) = f(X_1(t), X_2(t), \ldots, X_n(t))$, where $X_i(t)$ denotes a performance metric (e.g., CPU utilization). These properties may not be complete, and any new property which can strengthen the prediction power of aging indicators can be complemented. In reality, it is very difficult to find such an aging indicator strictly satisfying these properties. But we can find a workaround.

3 Multidimensional Multi-Scale Entropy

To predict ARFs, the first step is to find an appropriate aging indicator satisfying the three properties mentioned in Section 2. As claimed in [14] and [31], the performance metrics exhibit chaotic behaviors during software aging. They show that the complexity increases along with the aging levels. A well-known measurement of system complexity is the classical Shannon entropy [32]. However, Shannon entropy is only concerned with the instant entropy at a specific time point. It cannot capture the temporal structures of one time series completely leading to statistical characteristic loss and even false judgment.

To resolve this problem, Multi-scale Entropy (MSE) is proposed by Costa et al. [33]. MSE is used to quantify the amount of structures (i.e., complexity) embedded in the real-world time series. A structure reveals the correlation in the time series. For example, let $X(t)$ be a time series. If $X(t-3)$ is highly correlated with $X(t)$ (e.g, the Pearson correlation coefficient of $X(t)$ and $X(t-3)$ is high), then this pair of correlation is called a structure. In other words, $X(t)$ can be described by $X(t-3)$. But multiple structures may co-exist in a time series, namely $X(t-1), X(t-2), \ldots, X(t-i), \ldots$ or the linear or non-linear combinations of these variables are all possibly correlated with $X(t)$. In that case, it is extremely difficult to figure out an individual linear or nonlinear model to describe $X(t)$. Actually, $X(t)$ is a combination of multiple models or processes embedded in the original time series. These models or processes involve different structures. For example, suppose $X(t)$ is composed by two models. One model involves two explanatory variables: $X(t-1)$ and $X(t-2)$. While the other one involves another two explanatory variables: $X(t-4)$ and $X(t-6)$. The complexity of a time series increases with the

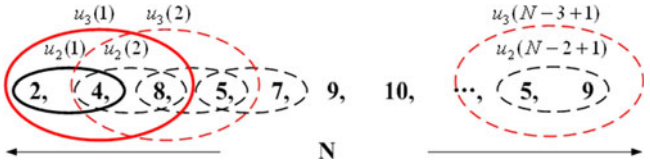


Fig. 2. The segmentation procedure when $m = 2$ and $m = 3$.

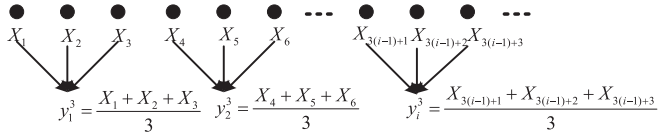


Fig. 3. The coarse-graining procedure when $\tau = 3$.

number of structures. Thus, it becomes difficult to conduct prediction on a time series when the number of structures is large. As software aging is a gradual deterioration process with time, the observed time series contain multiple structures ranging from short-term correlations to long-term correlations. Therefore, MSE is a potential indicator to indicate and measure software aging.

MSE calculates the sample entropy [34] of a coarse-grained univariate time series. The rational of coarse-graining procedure is to construct temporal scales. Thus, a time series without structures would exhibit a significant entropy decrease with an increasing time scale. Sample entropy [34] which stems from approximate entropy [34] is introduced to quantitatively and numerically measure the regularity or complexity of a time series with limited length. Sample entropy achieves this by calculating the repetitive patterns in a time series. Given a positive number m, a random variable X and a time series $\mathbf{X} = \{X(1), X(2), \ldots, X(N)\}$ with length N, $\mathbf{X}$ is partitioned into consecutive segments. Each segment is represented by an m-length vector: $u_m(t) = \{X(t), X(t+1), \ldots, X(t+m-1)\}, 1 \le t \le N-m+1$, where m could be recognized as the embedded dimension and recommended as $m = 2$ [34]. Fig. 2 shows the segmentation procedure when $m = 2$ and $m = 3$ in a sample time series. Let $n_i^m(r)$ denote the number of segment, $u_m(j)$, that are similar to $u_m(i)$ satisfying $d(u_m(i), u_m(j)) \le r, i \ne j$, where $i \ne j$ guarantees that self-matches are excluded, r is a preset threshold indicating the tolerance level for two segments to be considered similar and recommended as $r = 1.5 * \sigma$ [34], where σ is the standard deviation of the original time series. $d(\cdot) = max\{|X(i+k) - X(j+k)| : 1 \le k \le m-1\}$ represents the maximum of the absolute values of differences between $u_m(i)$ and $u_m(j)$, measured by Euclidean distance adopted in this paper. If $d(\cdot) \le r$, then $u_m(i)$ and $u_m(j)$ are similar. Let $C_i^m(r) = \frac{n_i^m(r)}{N-m}$ represent the probability that any segment $u_m(j)$ is close to segment $u_m(i)$, where $N-m$ denotes all the other segments in the time series. The average of $C_i^m(r)$ is expressed as

$$\Phi^m(r) = \frac{\sum_i^{N-m+1} C_i^m(r)}{N-m+1}. \tag{1}$$

$\Phi^m(r)$ denotes the probability that any two segments are within the tolerant level r of each other. Eckmann et al. further improved $\Phi^m(r)$ using $lnC_i^m(r)$ to substitute $C_i^m(r)$ [33]. To ensure $\Phi^{m+1}(r)$ is defined in any particular N-length time series, $N-m+1$ is changed to $N-m$ in $\Phi^m(r)$. Finally, the sample entropy is formalized as

$$S_E(m, r, N) = -ln\frac{\Phi^{m+1}(r)}{\Phi^m(r)}. \tag{2}$$

A lower $S_E(m, r, N)$ means more regular or lower complexity, while a higher $S_E(m, r, N)$ means less regular or higher complexity. Sample entropy has been widely used (e.g., [35], [36]) in the time series with modest data length as it is less dependent on time series length and yields more consistent complexity estimators. However, due to the structural complexity at different scales, sample entropy is not enough to capture these hidden characters. Therefore, MSE is proposed [37]. MSE extends the conventional sample entropy from one scale to multiple scales with an additional coarse-graining procedure.

Suppose τ is the scale factor, the consecutive coarse-grained time series Y^τ is constructed in the following two steps: **i**). Divide the original time series $\mathbf{X}$ into consecutive and non-overlapping windows of length τ; **ii**). Average the data points inside each window. Finally we get $Y^\tau = \{y_j^\tau | : 1 \le j \le \lfloor \frac{N}{\tau} \rfloor\}$ and each element of Y^τ is defined as

$$y_j^\tau = \frac{\sum_{i=(j-1)\tau+1}^{j\tau} X(i)}{\tau}, 1 \le j \le \lfloor \frac{N}{\tau} \rfloor. \tag{3}$$

When $\tau = 1$, Y^τ degenerates to the original time series $\mathbf{X}$. Then MSE of the original time series $\mathbf{X}$ is obtained by calculating the sample entropy of Y^τ at all scales. Fig. 3 demonstrates the coarse-graining procedure when $\tau = 3$.

However, the conventional MSE is designed for single dimensional analysis. While software aging process shows complexity not only in temporal scale but also in spatial scale (multivariate). Currently, MSE doe not satisfy the property *Integration* of an aging indicator. To this end, we extend MSE to MMSE via several modifications.

Modification 1. The collected multi-dimensional performance metrics usually have different scales and numerical ranges. For example the CPU utilization metric stays in the range of $0 \sim 100$ percent while the total memory utilization may vary in the range $1 \sim 4$ GB. Thus, the distance between two segments may be biased by the performance metrics with large numerical ranges, which further results in MSE bias. To avoid that bias, we normalize all the performance metrics to a unified numerical range, namely $0 \sim 1$. Suppose X is a Nxp data matrix where p is the number of performance metrics, N is the length of the data window and each column of X denotes the time series of one particular performance metric, then X is normalized in the following way:

$$X'_{ji} = \frac{X_{ji} - min(X_i)}{max(X_i) - min(X_i)}, 1 \le i \le p, 1 \le j \le N. \tag{4}$$

Modification 2. In MSE algorithm, we quantify the similarity between two segments via maximum norm [35] of two scalar numbers. A novel quantification approach is necessary when MSE is extended to MMSE. Each element in the

maximal norm pair: $max\{|X(i+k)-X(j+k)| : 1 \le k \le m-1\}$ such that $X(i+k)$ is replaced by a vector $\mathbf{X}(i+k)$ where each element represents the observation of one specific performance metric at time $i+k$. Thus the scalar norm is transformed to the vector norm. The embedded dimension m should also be vectorized when the analysis shifts from single dimension to multiple dimensions. The vectorization brings a nontrivial problem in the calculation procedure of sample entropy that is how to obtain $\Phi^{m+1}(r)$. Assume that the embedding vector $\mathbf{m} = (m_1, m_2, \ldots, m_p)$ denotes the embedded dimensions for p performance metrics respectively. A new embedding vector $\mathbf{m}^+$ which has one additional dimension compared to $\mathbf{m}$ can be obtained in two ways. According to the embedding theory mentioned in [38], $\mathbf{m}^+$ can be achieved by adding one additional dimension to only one specific embedded dimension in $\mathbf{m}$, which leads to p different alternatives. $\mathbf{m}^+$ can be any one of the set $\{(m_1, m_2, \ldots, m_k+1, \ldots, m_p), 1 \le k \le p\}$. $\Phi^{\mathbf{m}^+}(r)$ is calculated in a naive way or a rigorous way, both of which are depicted in detail in [35]. The other approach is very simple and intuitional, that is, adding one additional dimension to every embedded dimension in $\mathbf{m}$. There is only one alternative for $\mathbf{m}^+$, namely $\{(m_1+1, m_2+1, \ldots, m_k+1, \ldots, m_p+1), 1 \le k \le p\}$. This simple approach implies that each embedded dimension is identical, which may be a strong constraint. However, compared to the former approach, the latter one has negligible computation overhead and works well in our experiment. The former approach will be discussed in our future work.

Modification 3. In MSE algorithm, the threshold r is set as $r = 0.15 * \sigma$. In MMSE algorithm, we need a single number to represent the variance of the multi-dimensional performance data in order to apply it directly in the similarity calculation procedure. Here we employ the total variance denoted by $tr(\mathbf{S})$, which is defined as the trace of the covariance $\mathbf{S}$ of the normalized multi-dimensional performance data, to replace σ.

Modification 4. We argue that an ideal aging indicator should be expressed as a single number in order to be readily used in failure prediction. The output of the conventional MSE is a vector of entropy values at multiple scales. We need to use a holistic metric to integrate all the entropy values at multiple scales. Thus a composed entropy (CE) is proposed. Let T denote the number of scales and the vector $\mathbf{E} = (e_1, e_2, \ldots, e_T)$ denote the entropy value at each scale respectively. Then CE is defined as the euclidean norm of the entropy vector $\mathbf{E}$,

$$CE = \sqrt[2]{\sum_{i=1}^{T} e_i^2}, \tag{5}$$

CE could be regarded as the euclidean distance between $\mathbf{E}$ and a "zero" entropy vector which consists of 0 entropy values. A "zero" entropy vector represents an ideal system state meaning, that the system runs in a healthy state without any fluctuations. Thus the more $\mathbf{E}$ deviates from a "zero" entropy vector, the worse the system performance is. It is worth noting that CE is not the unique metric which can integrate the entropy values at all scales. Other metrics also have the potential to be aging indicators. For example, the average of $\mathbf{E}$ is another alternative, although we observe that it has a consistent result with CE.

For the sake of clarity, we demonstrate the pseudo code of MMSE calculation procedure in Algorithm 1. Does MMSE satisfy the proposed three properties, namely *Monotonicity*, *Stability* and *Integration*? First of all, we show that entropy caters to $Monotonicity$ during software aging via empirical verifications. As MMSE is built on Shannon entropy [32], the properties of Shannon entropy are inherited by MMSE. For example, both of them are the measurements of complexity. Therefore, we only show that the system Shannon entropy increases with the degree of software aging. In this paper, we aim to predict ARFs before a failure occurs. Therefore, we are only concerned about two aging states, namely the normal working state (s_w) and the failure state (s_f). The hidden aging process dominates the observed performance metrics. For instance, if the aging process shows a degradation trend, the observed performance metrics such as the response time have a decreasing trend correspondingly; if the complexity of the aging process increases, then certain observed performance metrics do so. The empirical verification is shown in Appendix A. This result tells us that the system entropy tends to increase when the failure probability (p_f) is smaller than the probability of working state (p_w). In most situations, the system cannot provide acceptable services or goes to failure very soon once $p_w < p_f$. When $p_w < p_f$, the $Monotonicity$ property of entropy in software aging tends to be the case.

Algorithm 1. MMSE Calculation Procedure

Input m: the embedded dimension; T: the number of scales; N: the length of data window; $\mathbf{X}$: a NXp data matrix where each p denotes the number of performance metrics and each column $X_i, 1 \le i \le p$ denotes the time series of one specific performance metric with length N.
Output: The aging degree metric CE

```
Normalize the original time series into the range [0,1]
  according to equation (4)
// Preset the similarity threshold r
S = Cov(X') // Cov denotes the matrix covariance
r = tr(S) // tr denotes the trace of a particular matrix
for τ = 1; τ = T; τ++ do
    // Coarse-graining procedure
    for i = 1; i = p; i++ do
        for j = 1; j = ⌊N/τ⌋; j++ do
            Y_ji = (Σ_{k=(j-1)τ+1}^{jτ} X'_ki) / τ
        end for
    end for
    E(τ) = ExtendedSampleEntropy(m, r, Y)
    // The similarity calculation between two
    // segments has been extended from scalar
    // to vector in ExtendedSampleEntropy(·)
end for
CE = ∛(Σ_{i=1}^{T} E(i)^2)
```

We apply MSE to the CPU utilization data series shown in Fig. 1. Fig. 4 shows the entropy values of the first four days (i.e., $Day1, Day2, Day3, Day4$) and the last four days (i.e., $Day49, Day50, Day51, Day52$). We observe that the

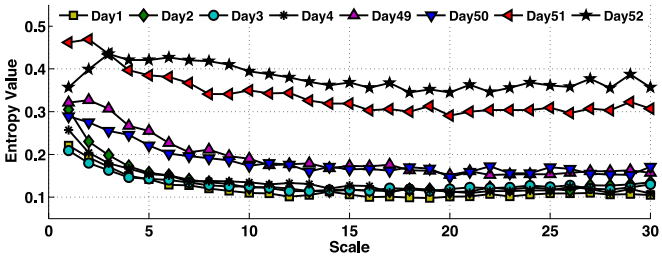


Fig. 4. The entropy values of a real VoD system at 30 scales.

entropy values of the last four days are much larger than the ones of the first four days nearly at all scales. Especially, the entropy values of Day52 when the system failed are significantly higher than others. Actually, the similar entropy increase phenomenon can also be observed in previous articles such as Fig. 3 in [31] and Fig. 5 in [39]. Although we cannot give the precise entropy values of these data due to lack of their detail, we can intuitively observe that the system goes from a steady state to a fluctuation state, which implies an entropy increase. However, the strict monotonicity may be biased by external interventions e.g., garbage collection.

Due to the inherent "multi-scale" nature of MMSE, the *Stability* property is obtained. Via multi-scale transformation, some noises are filtered out or smoothed. In addition, the combination of entropy values at multiple scales further mitigates the influence of noises. MMSE is a fusion of MSE on multiple performance metrics (e.g., Memory utilization) and can readily cover more metrics. Hence, MMSE satisfies the *Integration* property. Based upon MMSE, we have implemented several approaches to predicting ARFs. To evaluate the effectiveness of these approaches, we design and implement a proof-of-concept prototype named ARF-Predictor. The details of ARF-Predictor will be depicted in next section.

4 System Design

The architecture of ARF-Predictor is shown in Fig. 5. ARF-Predictor mainly contains four modules: data collection, metric selection, MMSE calculation, and failure prediction. The data collection module collects runtime performance metrics from multiple data sources including applications, processes and operating systems. The collected data are stored in a time series database named *InfluxDB* [40]. Amongst the raw performance metrics, collinearity is thought to be common meaning that some metrics are redundant. What is worse, a significant overhead is caused if all of these performance data are analyzed. Thus, it is necessary to select a subset of the original metrics without major loss of quality. The selected metric subset is fed into the MMSE calculation module to calculate the sample entropy at multiple scales in real time. Then the entropy values are leveraged to predict an ARF by the failure prediction module. We will describe the details in the following parts.

4.1 Metric Selection

To get rid of the collinearity amongst the high-dimensional performance metrics and to reduce computational overhead, we select a subset of metrics which can be used as a

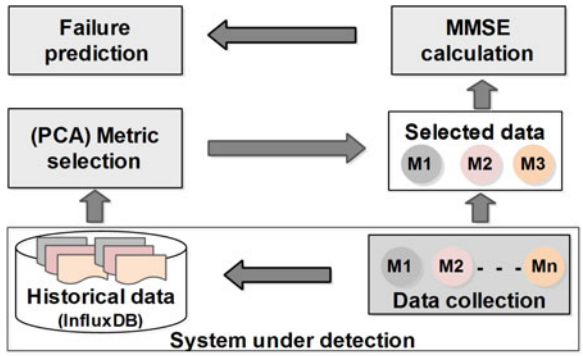


Fig. 5. The architecture of ARF-Predictor.

surrogate for a full set of metrics without significant loss of information. Assuming there are M metrics, our goal is to select the best subset of any size k from 1 to M. To this end, Principal Component Analysis (PCA) variable selection method is introduced.

As a classical multivariate analysis approach, PCA is always used to transform orthogonally a set of variables which may be correlated to a set of variables which are linearly uncorrelated (i.e., principal component, or PC). Let $\mathbf{X}$ denote a column centered nxM matrix, where M denotes the number of metrics, n denotes the number of observations. Via PCA, the matrix $\mathbf{X}$ could be reconstructed approximately by k PCs, where $k \ll M$. Mathematically, $\mathbf{X}$ is transformed into a new nxk matrix of principal component scores $\mathbf{T}$ by a loading or weight kxM matrix $\mathbf{W}$ if keeping only the k principal component, namely $\mathbf{T} = \mathbf{X}\mathbf{W}^T$ where each column of $\mathbf{T}$ is called a PC. The loading factor $\mathbf{W}$ can be obtained by calculating the eigenvector of $\mathbf{X}^T\mathbf{X}$ or via singular value decomposition (SVD) [41]. In this paper, we adopt the former method. $\mathbf{X}^T\mathbf{X}$ is first transformed to a diagonal matrix Λ. The values on the diagonal line of Λ are the eigenvalues. The eigenvectors corresponding to the k-largest eigenvalues constitute $\mathbf{W}$. Instead, we leverage PCA to select variables rather than to reduce dimensions.

In order to achieve this goal, we first introduce a well-defined numerical criterion in order to rank the subset of variables. Three kinds of criteria, namely RM [42], [43], Generalized Coefficient of Determination (GCD) [43], [44] and RV [43], are proposed in previous literatures. RM and RV are defined as matrix correlations between an nXM data matrix and projections of its columns on some subspace of R^n which is spanned by k original variables. While GCD is a measurement of the proximity of two subspaces spanned by different variable sets. In this paper, GCD is a measurement of similarity between the principal subspace spanned by the k specified PCs and the subspace spanned by a given p-variable subset of the original M-variable data set. By default, the specified PCs are usually the first k PCs and the number of variables and PCs is the same ($k = p$). In other words, RM and RV involve all the PCs (i.e., usually n PCs for an nXM matrix) while GCD only involves k PCs. Thus, the computational overhead of GCD is lower than the ones of RM and RV especially when n is large and k is small. Moreover, we observe that the effectiveness of GCD is comparable with the ones of RM and RV when local search heuristics (e.g., simulated annealing) are adopted according to the experimental results in [42]. These two points are also

empirically verified in our experiments. Therefore, we choose GCD [43], [44] as a criterion. The detailed description of GCD can be found in [43].

Then we need a search algorithm to select the best p-variable subset of the full data set. In this paper, we adopt a heuristic simulated annealing algorithm to search for the best p-variable subset. The algorithm is described in detail in [42]. In brief, an initial p-variable subset is fed into the simulated annealing algorithm, then the GCD criterion value is calculated. Further, a subset in the neighborhood[2] of the current subset is randomly selected. The alternative subset is chosen if its GCD criterion is larger than the one of the current subset. However, to avoid getting stuck in local optima, we choose a worse alternative subset with a probability $e^{\frac{ac-cc}{t}}$ if the GCD criterion of the alternative subset (ac) is smaller than the one of the current subset (cc), where t denotes the temperature and decreases throughout the iterations of the algorithm. The algorithm stops when the number of iterations exceeds the preset threshold. The merit of the simulated annealing algorithm is that the best p-variable subset can be obtained with a reasonable computation overhead, even the number of variables may be very large. With the well-defined GCD criterion and the simulated annealing search algorithm, we can reduce the high-dimensional runtime performance metrics (e.g., 76) to a very low-dimensional data set (e.g., 5) with very little information loss. Therefore, the MMSE overhead will be mitigated significantly. The selected metrics are fed into the MMSE calculation module to calculate *CE* using the proposed MMSE algorithm in real time.

4.2 ARF Prediction Based upon MMSE

Based upon the proposed aging indicator MMSE, it is easy to design algorithms to predict ARFs in real time. According to the survey [45], there are three kinds of approaches including *time series analysis*, *threshold-based* and *machine learning* to detecting the onset of aging or predicting the occurrence of ARFs. In this paper, we only discuss the time series and threshold-based approaches, and leave the machine learning approach to our future work. But before that we need to determine a sliding window in order to calculate MMSE in real time. As mentioned in previous work [34], $\lfloor \frac{N}{\tau} \rfloor$ should stay in the range 10^m to 30^m. Thus the sliding window heavily depends on the scale factor τ. In previous studies [33], [35], the authors usually set the scale factor τ in the range $1 \sim 20$, leading to a huge data window, say 10,000, especially when $\tau = 20$. A large sliding window not only increases the computational overhead but also makes prediction approaches insensitive to failures. Thus we constrain the sliding window in an appropriate range, say no more than 1,000, by limiting the range of τ. In our experiment we set τ in the range $1 \sim 10$. So a moderate data window $N = 1,000$ can cater for the basic requirement.

Threshold Based Approach. As a simple and straightforward approach, the threshold based approach is widely used in aging failure prediction [46], [47]. If the aging level exceeds a preset threshold, a failure occurs. However, an essential challenge is how to identify an appropriate threshold. Determining a threshold from empirical observations is a feasible approach. This approach learns a normal pattern when the system runs in the normal state. If the normal pattern is violated, a failure occurs. We call this approach *FailureThreshold* (*FT*). Assume that $CE = \{CE(1), CE(2), CE(3), \ldots, CE(n)\}$ represents a series of normal data where each element $CE(t)$ denotes a CE value at time t. The failure threshold ft is defined as $ft = \beta * max(CE)$, where β is a tunable fluctuation factor which is used to cover the unobserved value escaped from the training data. If there are a bunch of aging failure records, β could be set optimally according to a heuristic method. First initialize $\beta = \beta_0$ (e.g., $\beta_0 = 1$) and set the search space, namely $\beta \in [\beta_0, \beta_1]$; then iteratively calculate the prediction result using *FT* approach with $\beta = \beta + \Delta$ in each iteration; choose the β value which has the best prediction result as the optimal β. In this paper, $\beta \in [1, 2]$ and $\Delta = 0.1$. If there are not enough aging failure records, β is set to 1.5 by default. The search space and default value only work in the ARF prediction approaches based on MMSE. Different systems and different approaches based on other aging indicators should be considered separately. As mentioned above, MMSE increases with the degree of software aging. Thus a failure occurs only when the new observed CE exceeds ft, something like upper boundary test. For the aging indicators which have a downtrend such as *AverageBandwidth*, the max function in (9) will be replaced by min, something like lower boundary test. A failure occurs if the newly observed CE is lower than ft.

FT can be further extended to be an incremental version named *FT-X* in order to adapt to the ever changing running environment. *FT-X* learns ft incrementally from historical data. Once a new $CE(t+1)$ is obtained and the system is assured to stay in the normal state, then we compare $CE(t+1)$ with previously trained $max(\mathbf{C}E(t))$. If $CE(t+1) < max(\mathbf{C}E(t))$ then $ft = \beta * max(\mathbf{C}E)$; else $ft = \beta * CE(t+1)$. Besides the real-time advantage, *FT-X* needs very little memory space to store the new CE and previously trained maximum of $\mathbf{C}E$.

Time Series Approach. Although the threshold based approach is simple and straightforward, identifying the threshold is still a thorny problem. Thus, to bypass the threshold setting dilemma, we need a time series approach which requires no threshold or adjusts a threshold dynamically. To compare with existing approaches, we leverage the extended version of *Shewhart control charts* algorithm proposed in [31] to predict ARF. But one difference exists. In [31], the authors adopt the deviation d_n between the local average a_n and the global mean μ_n to predict aging failures. d_n is defined as

$$d_n = \frac{\sqrt[2]{N'}}{\sigma_n}(\mu_n - a_n), \tag{6}$$

where N' is used to represent the sliding window on entropy data calculated by MMSE algorithm in order to distinguish it from the sliding window N in MMSE algorithm, and the meaning of other relevant parameters can be found in [31]. The authors pointed out that *Hölder* exponent decreased with the degree of software aging. Therefore, they only took into account the scenario of $\mu_n > a_n$. In this

2. The neighborhood of a subset *S* is defined as a group of *k*-variable subsets which differ from *S* by only a single variable.

paper, we empirically show that MMSE increases with the degree of software aging. Thus, we only take into account the scenario of $\mu_n < a_n$. d_n is redefined by substituting $\mu_n - a_n$ with $a_n - \mu_n$. To make the prediction more resilient to system noises, we empirically determine that a change occurs when $d_n > \epsilon$ holds for p consecutive points where ϵ and p can be tuned in different experiments. In our experiments, we observe that a change is assured when $p = 4$. So $N^{'}$ and ϵ are the primary factors affecting the prediction results. In [31], the second change in *Hölder* exponent implies a system failure. By observing the MMSE variation curves obtained from Helix Server test platform and real-world AntVision system shown in Section 5, we find out that these curves can be roughly divided into three phases: slowly rising phase, fast rising phase, and failure-prone phase. Moreover, when the system steps into the failure-prone phase, a failure will come soon. Therefore, we also assume that the second change in MMSE data implies a system failure. Here we provide a heuristic method to set $N^{'}$ and ϵ, which is similar to the β setting method in threshold based approach. First set a search space gridded by $N^{'} \in [N_0^{'}, N_1^{'}]$ and $\epsilon \in [\epsilon_0, \epsilon_1]$; then find $N^{'}$ and ϵ which produce the best prediction result from the search space. In this paper, the search space is a 20×7 grid comprised by $N^{'} \in [1, 20]$ and $\epsilon \in [1, 7]$. $N^{'}$ and ϵ are set to 10 and 6 respectively by default.

5 Experimental Evaluation

We have designed and implemented a proof-of-concept prototype named ARF-Predictor and deployed it a controlled environment. To monitor the common process and operating system related performance metrics such as CPU utilization and context switch, we leverage some off-the-shelf tools such as Windows Performance Monitor shipped with Window OS or Hyperic [48]. To monitor other application related metrics such as response time and throughput, we develop several probes from scratch and deploy them in the test environment. The sampling interval in all the monitoring tools is 1 minute. All the collected data are stored the *InfuxDB* [40] in order to obtain the metric data in a specific time interval by a simple query language.

In the following experiments, we will address five questions: Does MMSE satisfy the property of *monotonicity* in practice? (Answer in Section 5.2); Which performance metrics are selected by PCA variable selection algorithm? (Answer in Section 5.3); Are the proposed ARF prediction approaches based on MMSE effective in the controlled test system and in the real-world system? (Answer in Section 5.4); Does MMSE outperform other explicit and implicit aging indicators? (Answer in Section 5.5); What is the overhead of ARF-predictor? (Answer in Section 5.6). Next, we will demonstrate the details of our experimental methodology and evaluation results in a VoD system, namely Helix Server, and in a real-world production system, namely AntVision.

5.1 Evaluation Methodology

To make comprehensive evaluations and comparisons from multiple angels, we deploy ARF-Predictor in a controlled VoD test environment. To evaluate the effectiveness of ARF-Predictor in real-world systems, we use ARF-Predictor to predict ARFs in AntVision system. We believe ARF-Predictor shows similar ARF prediction results in other systems (e.g., web servers [4], [14], JVMs [5] and Windows Operating Systems [31]), where aging phenomena have been observed.

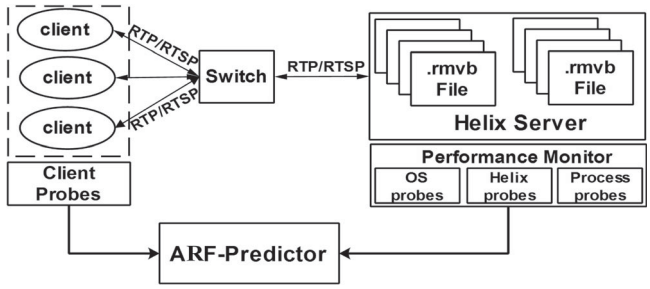


Fig. 6. The framework of Helix Server test system.

VoD System. We choose a VoD system as our test platform because more and more services involve video and audio data transmission. What is more, the "aging" phenomenon has been observed in such kinds of applications in our previous work [49], [50]. We leverage Helix Server 10.0 [51] as a test platform to evaluate our system due to its open source and wide usage. Helix Server as a mainstream VoD software system is adopted to transmit video and audio data via RTSP/RTP protocol. At present, there are very few VoD benchmarks. Hence, we develop a client emulator named $HelixClientEmulator$ employing RTSP and RTP protocols from scratch. $HelixClientEmulator$ involves three threads, one for audio processing (e.g., coding and decoding), one for video processing, and the third for session management (e.g., start and stop a file request). It can generate multiple concurrent clients to access media files on a Helix Server. Our test platform consists of one server hosting Helix Server, three clients hosting $HelixClientEmulator$ and one Gigabit switch connecting the clients and the server together. 100 rmvb media files with different bit rates are deployed on the Helix Server machine. The topology of our test system is demonstrated in Fig. 6. Each client machine is configured with one Intel dual core 2.66 Ghz CPU and 2 GB memory and one Gigabit NIC, and runs 64-bit Windows 7 operating system. The server machine is configured with two 4-core Xeon 2.1 GHZ CPU processors, 16 GB memory, a 1 TB hard disk and a Gigabit NIC, and runs 64-bit Windows server 2003 operating system.

We have deployed 100 rmvb media files with different bit rates on the Helix Server machine. $HelixClientEmulator$ is adopted to generate different types of workloads. Each workload is represented by a tuple $(client\_count, access\_distribution\_type, sleep\_time)$ where $client\_count$ denotes the concurrent number of emulated clients, $access\_distribution\_type$ denotes the request distribution on the 100 media files in order to distinguish their popularity and $sleep\_time$ denotes the time interval in second between two continuous requests generated by the same client. $access\_distribution\_type$ has 3 choices: 0, 1, 2 where "0" represents random access, meaning that the 100 files are accessed randomly, "1" represents Poisson access, meaning that the 100 files are accessed according to Poisson distribution, and "2" represents single file access, meaning that only

one file in the 100 files can be accessed. In this paper, the $access\_distribution\_type$ is set to 0 by default. Through 20 experiments, we observe that the Helix Server always restarts itself automatically and the transmission speed drops immediately when the concurrent number of clients reaches 900. Thus, 900 is the capacity of our test platform and the number of emulated clients will not exceed this limit in the following experiments. But to accelerate the aging process, we make the system run under a high pressure workload (i.e., $client\_count \geq 500$), which is also adopted in [20].

During system running, thousands of performance counters can be monitored. In order to trade off between monitoring effort and information completeness, we only monitor part of the runtime information at four different levels: Helix Client, OS, Helix Server, and server process with corresponding probes. At Helix Client level, we record the performance metrics such as *Jitter*, *Average Response Time*, etc., with the probes embedded in $HelixClientEmulator$. At OS level, we monitor *Network Transmission Rate*, *Total CPU Utilization*, etc., via Windows Performance Monitor. At Helix Server level, we monitor the application relevant metrics such as *Average Bandwidth Output Per Player(bps)*, *Players Connected*, etc., from the log produced by Helix Server. Finally, at process level, we monitor some of metrics related to the Helix Server process like *Process Working Set*. Due to the limited space, we will not show the 76 performance metrics.

AntVision System. Besides the evaluations in a controlled environment, we further apply ARF-Predictor to predict failures in AntVision system. AntVision is a complex system which is used to monitor and analyze public opinions and information from social networks like Sina Weibo. The whole system consists of hundreds of machines in charge of crawling information, filtering data, storing data and so on. More information about this system can be found in www.antvision.net. With the help of system administrators, we have obtained 7-*day* logs from AntVision. The log data not only contain performance data but also failure reports. Although the performance trace only involve two metrics i.e., CPU and memory utilization, it is enough to evaluate the failure prediction power of ARF-Predictor. According to the failure reports, we observe that one machine crashed on the 6th day without throwing exceptions in the console log and application log. After manual investigation, we conclude that the outage is likely caused by software aging. Because the query response time on this machine decreases gradually, and the CPU utilization exhibits an increasing trend.

In the controlled environment, we conducted 50 experiments by varying the workload parameters. In each experiment, the $client\_count$ parameter is randomly selected from $\{500, 550, 600, 650, 700, 750, 800\}$, the $access\_distribution\_type$ parameter is randomly selected from $\{0, 1, 2\}$ and the $sleep\_time$ is fixed as 1. In this paper, we aim to evaluate the effectiveness of ARF prediction approaches using the aging traces generated by different workloads rather than to discuss the impact of workload parameters on software aging. Therefore, the workload parameters are chosen randomly and 50 experiments are enough. Moreover, we guarantee the system runs to "failure" in each experiment. Here "failure" not only refers to system crashes but also resource exhaustion and severe performance degradation. As pointed in [20], when the degradation path $D(t)$ exceeds a critical level D_f, a failure occurs. This kind of failure is also called "pseudo-failure" [20]. D_f is always defined as the maximum degradation level that the system can tolerate. For example, [20] defines a pseudo-failure when the total memory used by Apache web server exceeds 400 MB ($D_f = 400$ MB). In our VoD system, if the *AverageBandwidth* is lower than 30 bps, Helix server cannot provide normal services any more because video and audio frames are severely lost at that moment. Hence, we set $D_f = 30bps$. To get the ground truth, we manually label the pseudo-failure point in each experiment. We search the failure point on the *AverageBandwidth* subject to the following constraints: i), lower than D_f for several consecutive times; ii), happen for the first time; iii), no increasing trends after the failure point. However, due to the ambiguity of manual labellings and the insensitivity of the sliding window, the failure prediction approaches may not predict the failure points precisely. Thus, we stipulate the prediction is correct if the predicted failure falls in a "decision window". The decision window is defined as a data window with a specific length whose right boundary is the "failure" point. [5] adopts a similar skill named "security margin" to judge the prediction result. A large decision window will enlarge $ATTF$. While a small one will result in inaccurate prediction results. Therefore, we set the length of decision window as 10 percent of the MMSE sliding window (i.e., 100 in this paper) as the decision window will not significantly affect the calculation of MMSE in this situation.

We leverage $Recall$, $Precision$, $F1\text{-}measure$ and $ATTF$ to quantitatively evaluate the effectiveness of ARF-Predictor. The former two metrics are defined as

$$Recall = \frac{N_{tp}}{N_{tp} + N_{fn}}, Precision = \frac{N_{tp}}{N_{tp} + N_{fp}},$$

where N_{tp}, N_{fn}, and N_{fp} denote the number of true positives, false negatives, and false positives respectively. It is worth noting that N_{tp}, N_{fn}, N_{fp} are the aggregated numbers over 50 experiments respectively. To represent the accuracy in a single value, $F1\text{-}measure$ is leveraged and defined as

$$F1 - measure = \frac{2 * Recall * Precision}{Recall + Precision}.$$

$ATTF$ is defined as the time span between the first failure report and the real failure. In a real-world system, once a failure is detected, the system may be rebooted or offloaded for maintenance. Thus, we choose the first failure report as a reference point. If the first failure report falls in the decision window, $ATTF = 0$. A large $ATTF$ may cause excessive system maintenance, leading to availability decrease and operation cost increase. Therefore, a lower $ATTF$ is preferred.

5.2 Monotonicity of MMSE

In this section, we show that MMSE satisfies the *monotonicity* property with the experimental data sets obtained from the Helix Sever system and the real-world trace data obtained from the AntVision system. In the Helix Server system, the phenomenon of "software aging" is observed in

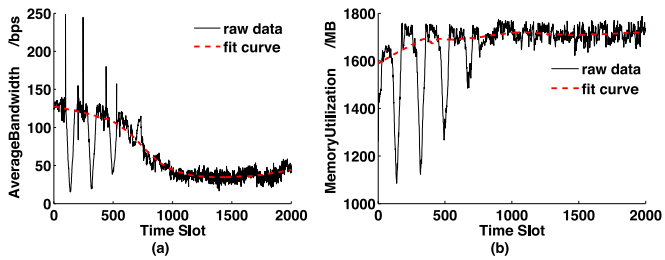


Fig. 7. Performance degradation in the Helix Server system under workload $(700, 0, 1)$. (a) denotes the degradation of Average Bandwidth and (b) denotes the increase of memory utilization. The red dash line denotes the smoothed curve using $Lowess$.

each experimental data set. For example Fig. 7 demonstrates the *AverageBandwidth* and *MemoryUtilization* variation curves along with time under workload $(700, 0, 1)$ in the Helix Server system. Even a cursory glance at these curves, a decreasing trend in *AverageBandwidth* and an increasing trend in *MemoryUtilization* can be observed. To exhibit these trends more clearly, we leverage $Lowess$ [52] method to smooth these two time series. From the smoothed curves, we observe that software aging indeed occurs in VoD system and the system goes to "failure" when serious video and audio frame losses occur.

In Appendix A, the $monotonicity$ of MMSE has been empirically verified. The observations in our experiments support our conclusion. Fig. 8 demonstrates the MMSE variation along with time in the the Helix Server system under different kinds of workloads. Fig. 9 demonstrates the CPU utilization and the corresponding MMSE values in the AntVision system. From these figures, we observe that the MMSE indicator shows a better monotonicity than some conventional aging indicators such as *AverageBandwidth* and *MemoryUtilization*. Especially before the system is close to a failure, it is nearly strictly monotone. But from Fig. 7, we observe that the Helix Server system is suffering from low *AverageBandwidth* even at a normal state. In the AntVision system, a failure occurs while the *CPU utilization* is lower than the preset threshold 0.9 and even lower than the *CPU utilization* at a normal state. Therefore, it is difficult to draw a line between normal and failure state based upon *AverageBandwidth* or *CPU utilization*. However, it becomes easy for MMSE. The high accuracy of the failure prediction approaches proposed in this paper is mainly attributed to the $monotonicity$ property of MMSE.

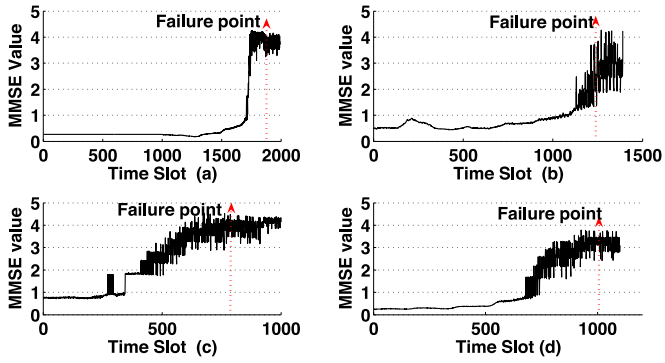


Fig. 8. The MMSE variations along with time in Helix Server system under under different workloads, namely $(700, 0, 1)$ (a), $(600, 0, 1)$ (b), $(650, 0, 1)$ (c),$(750, 0, 1)$ (d) respectively.

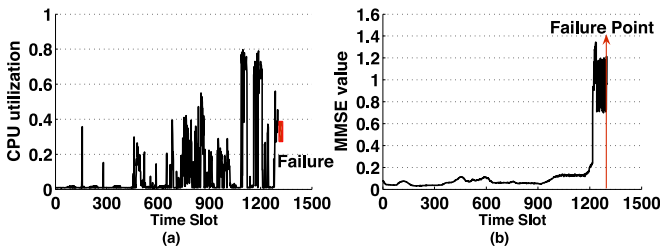


Fig. 9. The CPU utilization and its MMSE values in AntVision system.

5.3 Performance Metric Selection

In this section, we select a subset of performance metrics from the high-dimensional performance data collected from the controlled test system using PCA variable selection algorithm. By investigating all the performance metrics, we find that many metrics are highly correlated. Therefore, a small subset of metrics can be used as a surrogate of the full data set without significant information loss via PCA variable selection method presented in Section 4.1. We calculate the best GCD scores of different variable sets with specific cardinalities (e.g., $k = 3$) with the simulated annealing algorithm. The number of iteration is set to 10. Fig. 10 shows the variation of the best GCD sore along with the number of variables. From this figure, we observe that the GCD score does not increase significantly any more when the number of variables reaches 5. Hence, these five variables are already capable of representing the full data set. They are *Total CPU Utilization*, *AverageBadwidth*, *Process IO Operations Per Second* , *Process Virtual Bytes Peak*, and *Jitter* respectively. In the following experiments, we will use them to evaluate ARF-Predictor.

5.4 ARF Prediction

In this section, we will demonstrate the effectiveness of ARF prediction of ARF-Predictor using both of the experimental data sets and the real-world trace data. In the MMSE algorithm, we set the embedded dimension $m = 2$, the sliding window $N = 1{,}000$, and the number of scales $T = 10$. For approach FT, we need to prepare the training data and determine the fluctuation factor β first. Due to the lack of prior knowledge, the training data selection is full of randomness. To unify the way of training data selection, we leverage the slice of MMSE data ranging from the system starting point to the point where 200 time slots away from the right boundary of the decision window as the training data. We leave the remaining 200 time slots to conduct and

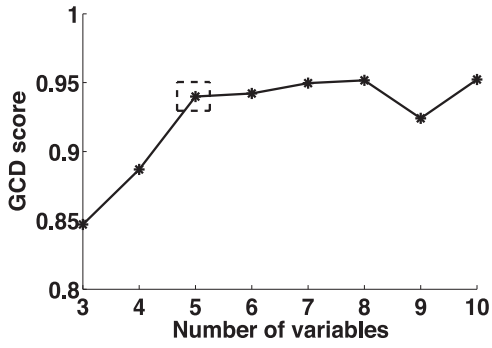


Fig. 10. The variation of GCD score along with the number of variables.

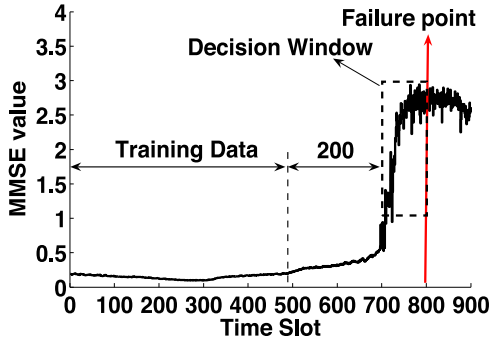


Fig. 11. Training data selection in FT approach.

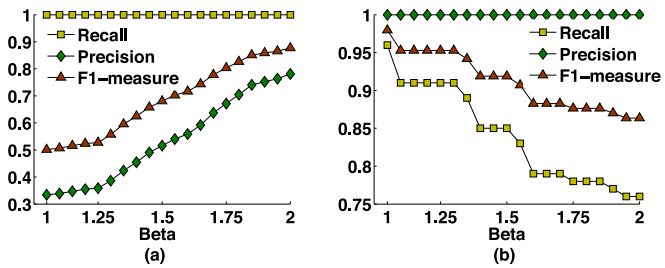


Fig. 12. The variations of $Recall$, $Precision$ and $F1$-$measure$ along with β values using the experimental data sets. (a) and (b) demonstrate the variations in FT approach and FT-X approach respectively.

compare to FT-X approach. Fig. 11 shows an example of training data selection in one experimental data set. In this figure, we set the 800th time slot as the pseudo-failure point. The decision window spans across the range $700 \sim 800$. Thus, the data slice in the range $0 \sim 500$ is selected as the training data.

Another problem is how to determine β. With the historical performance metrics and failure records, it is possible to achieve an optimal β according to the β setting method in Section 4.2. Fig. 12a demonstrates the failure prediction results of FT with different β values using the experimental data sets. From this figure, we observe that $Recall$ keeps a perfect value 1 when β varies in the range $1 \sim 2$, i.e., $N_{fn} = 0$ and the other two metrics: $Precision$ and $F1$-$measure$ increase with β. From Fig. 11, we can find some clues to explain these observations. In Fig. 11, the training data in the range $0 \sim 500$ are much smaller than the data in the decision window. Hence, no matter how β varies in the range $1 \sim 2$, the failure threshold ft is lower than the data in the decision window. The advantage is that all of the failures can be pinpointed (i.e., $N_{fn} = 0$). While the disadvantage is that many normal data are mistaken as failures (i.e., N_{fp} is large). Moreover, $Precision$ has an increasing trend due to the decreasing of N_{fp} with β. Similarly, the prediction results FT-X with different β values are shown in Fig. 12b. But quite different from the observations in Fig. 12a, $Precision$ keeps a perfect value 1 (i.e., $N_{fp} = 0$) while the other two metrics $Recall$ and $F1$-$measure$ decrease with β in Fig. 12b. Fig. 11 is also capable of explaining these observations. The failure threshold ft is updated by FT-X incrementally according to the system state. As the system runs normally in the range $500 \sim 700$, these data are also used to train ft. Hence, $max(\mathbf{C}E)$ calculated by FT-X is much bigger than the one calculated by FT. A bigger β can guarantee the detected failures are the real failures (i.e., $N_{fp} = 0$) but may result in a large failure

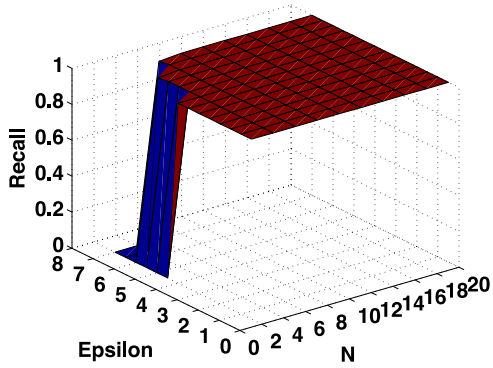


Fig. 13. $Recall$ variations.

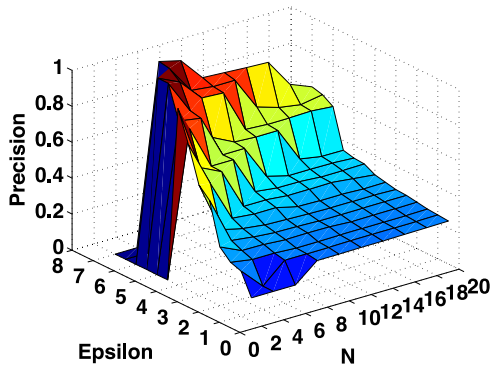


Fig. 14. $Precision$ variations.

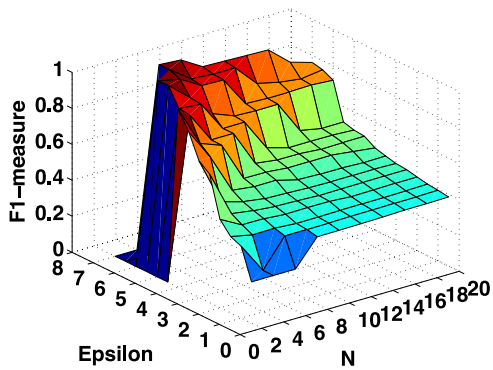


Fig. 15. $F1$-$measure$ variations.

missing rate (i.e., N_{fn} is large). From these two figures, we observe that FT achieves an optimal result when β is large, say $\beta = 2$, but FT-X achieves an optimal result when β is small, say $\beta = 1.1$. To carry out fair comparisons, we set $\beta = 2$ for FT and $\beta = 1.1$ for FT-X, namely their optimal results. However, in real-world systems, the optimal β is considerably difficult to attain when failure records are scarce. In that case, β is set to 1.5 by default.

Although the extended version of *Shewhart control charts* is capable of identifying failures adaptively, it is still necessary to determine two parameters, namely the sliding window $N^{'}$ and ϵ, in order to obtain an optimal prediction result. We leverage the method proposed in Section 4.2 to obtain the optimal $N^{'}$ and ϵ with the experimental data sets. Fig. 13, 14, 15, and 16 demonstrate the $Recall$, $Precision$, $F1$-$measure$, and $ATTF$ variations along with ϵ and $N^{'}$ respectively. The variation zone is organized as 20X7 mesh grid. From Fig. 13, we observe that in the area where

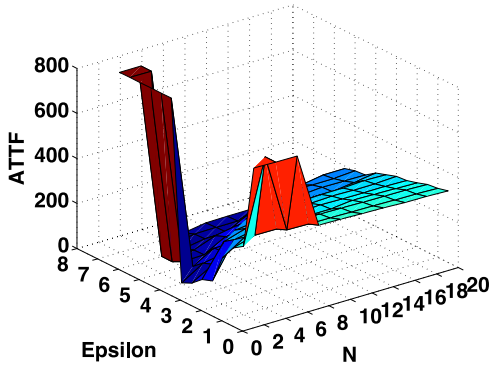


Fig. 16. *ATTF* variations.

$2 \leq N^{'} \leq 6$ and $4 \leq \epsilon \leq 7$, some values are 0 (i.e., $N_{tp} = 0$) as there are no deviations exceeding the threshold ϵ. Accordingly, *Precision* and *F1-measure* are 0 too. But in other areas, all the failure points are detected (i.e., *Recall* = 1). Thus *F1-measure* changes consistently with *Precision*. Here we choose the optimal result when $N = 6$ and $\epsilon = 6.5$ according to *F1-measure*. At this point, *Recall* = 1, *Precision* = 0.99, *F1-measure* = 0.995 and *ATTF* = 6. In the following experiments, we will compare the prediction results of *FT*, *FT-X* and the extended version of *Shewhart control charts* when they achieve the optimal results in the Helix Server system and in the real-world AntVision system. In different systems, we will determine the optimal results for different approaches separately.

Fig. 17 depicts the failure prediction results obtained by *FT*, *FT-X* and the extended *Shewhart control charts* in the Helix Server system. From Fig. 17a, we observe that the extended version of *Shewhart control chart* achieves the best result, *F1-measure* = 0.995; *FT-X* achieves the second best result, *F1-measure* = 0.9795; *FT* achieves the worst result, and *F1-measure* = 0.8899. The prediction results of the extended *Shewhart control chart* and *FT-X* have about 0.1 improvement compared to the one of *FT*. Meanwhile, a lower *ATTF* is obtained by the adaptive approaches such as *FT-X*, shown in Fig. 17b. Via these comprehensive comparisons, we find that the adaptive approaches outperform the statical approaches due to their adaptation to the ever changing runtime environment.

Fig. 18 shows one slice of MMSE time series in the range $1{,}100 \sim 1{,}320$ in the AntVision system. The failure reports generated by *FT*, *FT-X* and *Shewhart control chart* fall in the range $1{,}213 \sim 1{,}320$, $1{,}217 \sim 1{,}320$ and $1{,}219 \sim 1{,}320$ respectively. It is intuitively observed that the *Shewhart control chart* approach achieves the best prediction result as almost all of its failure reports fall in the decision window.

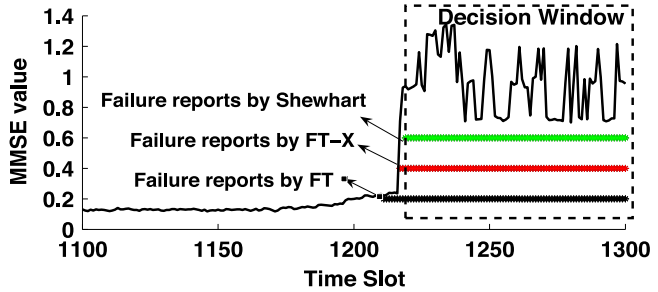


Fig. 18. One slice of MMSE data and the failure reports generated by *FT*, *FT-X* and *Shewhart control chart* in AntVision system.

The prediction results achieved by *FT* and *FT-X* are very similar. This is because there are no significant changes for MMSE in the range $1{,}000 \sim 1{,}220$, which results in the optimal threshold determined by *FT* and *FT-X* are very similar, namely 0.233 and 0.4 respectively. Fig. 19 demonstrates the failure prediction results in terms of *Recall*, *Precision*, *F1-measure* and *ATTF*. The results also show that the adaptive approach tends to be capable of achieving a better prediction accuracy and a lower *ATTF*. To make a broad comparison with the approaches based upon other aging indicators, we conduct the following experiments.

5.5 Comparison

In this section, we will compare the failure prediction results obtained by the approaches based upon MMSE and the approaches based upon other explicit or implicit aging indicators in different systems. In previous studies, QoS metrics (e.g., response time) or resource related metrics (e.g., CPU utilization) are more often than not adopted as explicit aging indicators. Accordingly, we employ *AverageBandwidth* as an explicit aging indicator in the Helix Server system and *CPU utilization* as an explicit aging indicator in the AntVision system. *Hölder* exponent mentioned in [31] is adopted as an implicit aging indicator in these two systems. For different aging indicators, the failure prediction approaches vary a little. For *AverageBandwidth* and *Hölder* exponent indicators, we employ a lower boundary test in the threshold based approach and the extended version of *Shewhart control chart* proposed in [31] in the time-series based approach both of which are depicted in Section 4.3, due to their downtrend characteristics. It is worth noting that β should vary in the same range (e.g., 1∼20 in this paper) for *FT* and *FT-X* in order to conduct fair comparisons. All of comparisons are conducted in the situations when these failure prediction approaches achieve optimal results.

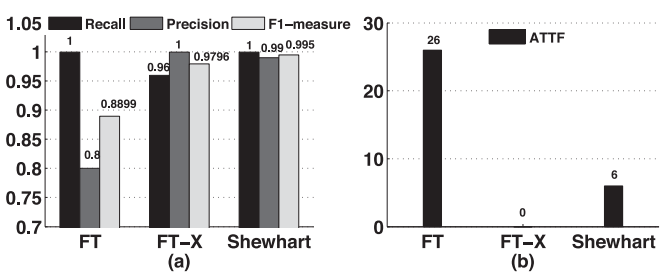


Fig. 17. The failure prediction results obtained by *FT*, *FT-X* and *Shewhart control charts* in Helix Sever system.

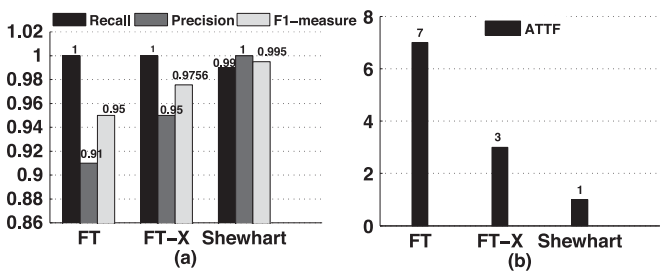


Fig. 19. The comparisons of the failure prediction results obtained by *FT*, *FT-X* and *Shewhart control charts* in AntVision system.

TABLE 1
The Optimal Conditions for Different Approaches Based upon Different Aging Indicators in Helix Server System

	FT	FT-X	Shewhart control chart
AverageBandwidth	$\beta = 1.8$	$\beta = 1.8$	$N^{'} = 440, \epsilon = 8$
MMSE	$\beta = 2$	$\beta = 1.1$	$N^{'} = 4, \epsilon = 6$
Hölder	$\beta = 5.3$	$\beta = 5.3$	$N^{'} = 40, \epsilon = 5$

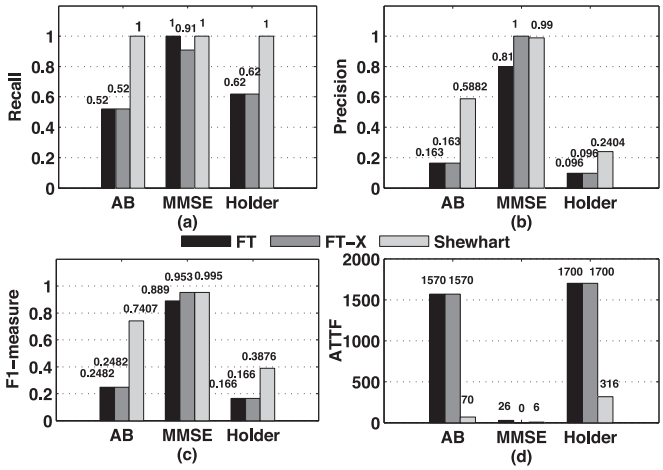


Fig. 20. The comparison results of the prediction approaches based upon different aging indicators in the Helix Server system. Here "AB" is short for *AverageBandwidth*.

We first determine the optimal conditions when these approaches achieve their optimal results in the Helix Server system. Table 1 demonstrates these optimal conditions. Fig. 20 shows the comparison results for different indicators in terms of *Recall*, *Precision*, *F1-measure* and *ATTF* respectively.

From Fig. 20a, we observe that the extended version of *Shewhart control chart* approach achieves an ideal recall (i.e., $Recall = 1$) no matter which indicator is chosen. However, for *FT* and *FT-X* approaches, the prediction result heavily depends on aging indicators. The *Recall* of *FT* and *FT-X* based upon MMSE are 1 and 0.91 respectively, much higher than the results obtained by the approaches based upon *AverageBandwidth*, namely 0.52 and *Hölder*, namely 0.62. The effectiveness of MMSE is even more significant than the other two indicators in terms of *Precision*. We observe that *Precision* of failure prediction approaches based upon MMSE is up to 9 times higher than the one of *FT* or *FT-X* based upon *Hölder*, and 5 times higher than the one of *FT* or *FT-X* based upon *AverageBandwidth*, shown in Fig. 20b. Accordingly, the MMSE is much more powerful to predict ARF than *Hölder* and *AverageBandwidth* in *F1-measure* demonstrated in Fig. 20c. From the point of view of *ATTF*, the approaches based upon MMSE obtain up to 3 orders of magnitude improvement than the ones based upon the other two indicators. For example in Fig. 20d, for *FT-X* approach, *ATTF* based upon *AverageBandwidth* and *Hölder* are 1,570 and 1,700 respectively, but the *ATTF* based upon MMSE is 0. The extraordinary effectiveness of MMSE is attributed to its three properties: *monotonicity*, *stability* and *integration*. However, the single runtime parameter cannot comprehensively reveal the aging state of the whole system

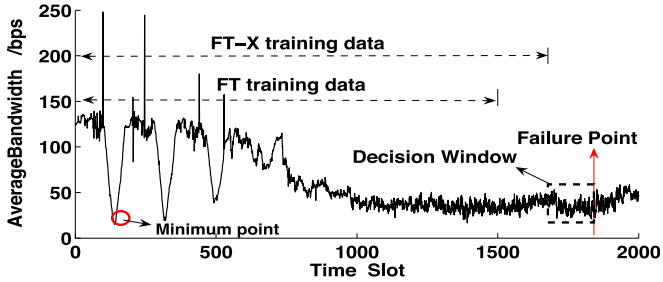


Fig. 21. The *AverageBandwidth* data from system start to "failure".

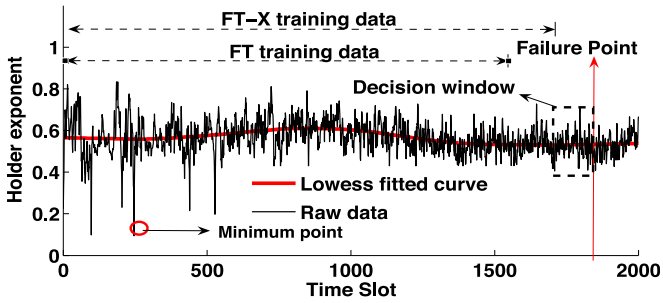


Fig. 22. The *Hölder* data from system start to "failure". The curve fitted by *Lowess* [49] is used to present the downtrend.

TABLE 2
The Optimal Conditions for Different Approaches Based upon Different Aging Indicators in the AntVision System

	FT	FT-X	Shewhart control chart
CPU Utilization	$\beta = 1$	$\beta = -$	$N^{'} = 75, \epsilon = 17$
MMSE	$\beta = 2$	$\beta = 1.3$	$N^{'} = 8, \epsilon = 7$
Hölder	$\beta = 4.0$	$\beta = 19$	$N^{'} = 165, \epsilon = 8$

and the fluctuations involved in this indicator result in much prediction bias.

Fig. 21 shows a representative *AverageBandwidth* variations from system start to pseudo-failure in an experimental data set. We observe that the *AverageBandwidth* is low even at a normal state. The *Hölder* exponent indicator also suffers from this problem. Although a downtrend indeed exists in *Hölder* exponent indicator indicating that the complexity is increasing, which is compliant with the result in [31] (shown in Fig. 22), its instability hinders to achieve a highly accurate failure prediction result. From above comparisons, we find out the prediction results obtained by *FT* and *FT-X* based upon *AverageBandwidth* or *Hölder* are the same. That is because the minimum points of the aging indicators are involved simultaneously in the training data of *FT* and *FT-X* demonstrated in Figs. 21 and 22 respectively. Therefore, the optimal threshold values calculated by *FT* and *FT-X* are the same.

The optimal conditions for these failure prediction approaches based upon *CPU Utilization*, *MMSE* and *Hölder* exponent in the AntVision system are listed in Table 2.

An interesting finding is that the optimal condition of *FT-X* based upon *CPU Utilization* indicator is "$\beta = -$", which means we cannot find an optimal β in the range $1 \sim 20$. By investigating the prediction results, we observe that *Recall*, *Precision* and *F1-measure* are all 0 no matter

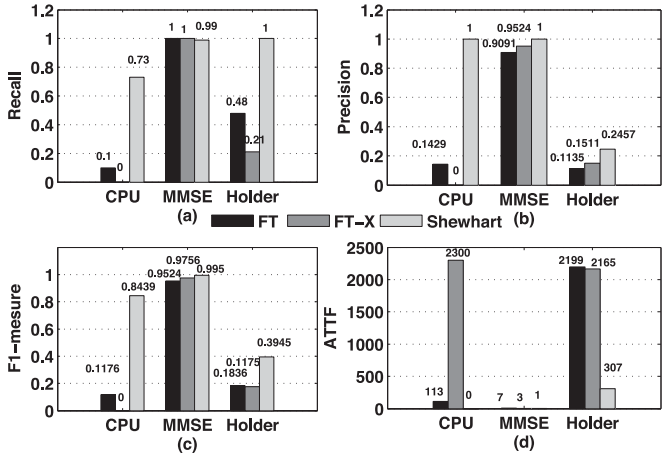


Fig. 23. The comparison results of the prediction approaches based upon different aging indicators in AntVision system. Here "CPU" means CPU utilization.

which value β is chosen in the range $1 \sim 20$. Fig. 9 provides the reason why we get this observation. The maximal CPU utilization involved in the training data in *FT-X* falling in the range $1 \sim 1{,}200$ exceeds all the CPU Utilization in the decision window. Therefore, according to the threshold calculated by *FT-X*, we cannot predict any failures (i.e., $N_{tp} = 0$). While for *FT* approach, the maximal CPU utilization in the training data is lower than the maximal CPU utilization in the decision window. Hence, some failure points can be detected by *FT*. This is the reason why *FT* outperforms *FT-X* based upon *CPU Utilization* in the AntVision system. This could also be regarded as a drawback of non-monotonicity of the *CPU Utilization* indicator.

Fig. 23 demonstrates the prediction results of the failure prediction approaches based upon different aging indicators using the real-world trace data. From this figure, we observe that $F1\text{-}measure$ achieved by MMSE-based approaches are higher than 0.95, which is much better than the one achieved by *CPU Utilization*-based and $H\ddot{o}lder$ exponent-based approaches. Meanwhile, *ATTF* is significantly reduced from a large number (e.g., 2,300) to a very small number (e.g., 1) by MMSE-based approaches. We also observe that the extended version of *Shewhart control chart* approach performs better than the other two approaches no matter which indicator is chosen.

Trend analysis is a common approach to predicting ARFs in previous work [1], [3]. This approach first leverages Man-Kendall test to check whether there is a trend in the time series. If there is a trend, Sen's slope [1], [3] is adopted to estimate the resource consumption speed. Then, the resource exhaustion time (i.e., the occurrence of ARF) is estimated by a linear formula [1], [3]. But the traditional Mann-Kendall-Sen approach is designed to work offline. To conduct real-time prediction, we leverage this approach to estimate the degradation speed in a sliding window which has the same length (i.e., 1,000) as the MMSE calculation data window. In the Helix Server system, we use this approach to estimate the failure time when $AverageBandwidth$ degrades to 30 bps. If the estimated failure time falls in the decision window, the prediction is correct. However, if it exceeds the length of the data, we say the prediction is still correct since no failure alarms or rejuvenation actions will

TABLE 3
The Comparison Between Trend Analysis and Our Approaches

Method	System	Indicator	Recall	Precision	F1	ATTF
FT	VoD	MMSE	1	0.51	0.675	50
FT-X	VoD	MMSE	0.825	1	0.904	5
Shewhart	VoD	MMSE	0.86	0.95	0.903	10
Sen	VoD	Band	0.35	0.6	0.442	80
HP	VoD	Band	0.51	0.82	0.629	60
FT	Ant	MMSE	1	0.88	0.936	15
FT-X	Ant	MMSE	1	0.94	0.961	6
Shewhart	Ant	MMSE	0.98	0.97	0.975	3
Sen	Ant	CPU	0	0	0	2,300
HP	Ant	CPU	0.04	0.18	0.065	80

be triggered. In the AntVision system, the CPU utilization threshold is set to 90 percent by the operation team. But we observe that even the maximum CPU utilization in the trace data does not exceed the threshold, which means we may not predict any ARFs using trend analysis. Therefore, we set an aggressive threshold to compare trend analysis with our approach. The CPU utilization threshold is set as the value with which *FT* obtains the optimal prediction result shown in Table 2, namely 55 percent. Another more advanced trend estimation approach, namely Hodrick-Prescott filter (HP filter), is adopted in [25]. To compare with this approach, we leverage HP filter to estimate the trend value. But different from Sen's slope method, HP filter extracts the trend value at each time point. These trend values form a non-linear curve which is hardly described by an explicit model (e.g., polynomial function) [25]. Therefore, we calculate the slope between the last two trend values, and use this slope to predict the occurrence of an ARF. To stress the advantages of our approaches, we predict ARFs with the default configurations (i.e., $\beta = 1.5$ in *FT* and $FT - X$, $\epsilon = 6$ and $N' = 10$ in *shewhart control chart* approach) rather than the optimal configurations. The comparison is shown in Table 3, where $F1$ denotes $F1 - measure$, Ant denotes AntVision, Sen and HP denote the trend analysis based on Sen's Slope and HP filter respectively, and Band denotes AverageBandwidth. From this table, we observe that our ARF prediction approaches outperform the trend based approaches. Trend analysis has a strong power to detect the onset of trend and characterize the degradation [25]. But it is weak in predicting ARFs due to its insensibility to the short-term system changes. Moreover, trend analysis is a univariate approach, which is not sufficient to determine the occurrence of an ARF [17]. Through comprehensive comparisons above, we conclude that MMSE-based approaches are superior to several state-of-the-art approaches. The high effectiveness of our approaches results from MMSE's three properties, namely *Monotonicity*, *Stability*, and *Integration*.

5.6 Overhead

The whole analysis procedure of ARF-Predictor except data collection is conducted on a separate machine. Hence, it results in a very little resource footprint in the production system. To evaluate whether ARF-Predictor satisfies the real-time requirement, we calculate the execution time of the whole procedure using the experimental data sets. The average execution times of different modules of

ARF-Predictor in the Helix Server system are shown in Table 3, where MS stands for Metric selection, and MMSE-C means MMSE calculation. Even the most computation-intensive module, namely Metric selection module, only consumes 0.875 second, and the whole procedure consumes a little more than 1 second. Therefore, ARF-Predictor is light-weight enough to satisfy the real-time requirement.

Discussion. First of all, ARF-Predictor is a pervasive tool. It can work in an extensive range of systems. However, to conduct effective prediction in real systems, several parameters should be taken into account carefully. The first parameter is the length of the sliding window N in MMSE calculation. A small sliding window can bias the sample entropy [34], but a large one increases the computational overhead and makes prediction approaches insensitive to failures. As mentioned in Section 4.2, we set $N = 1{,}000$. The embedding dimension m is set $m = 2$ which is validated in [35], [37] As recommended in [34], $\lfloor \frac{N}{\tau} \rfloor$ should stay in the range 10^m to 30^m. Hence, τ varies in the range $1 \sim 10$ if $N = 1{,}000$. The parameters of prediction approaches can also impact the prediction results. If there are enough failure traces, the optimal parameters, namely β, $N^{'}$ and ϵ can be obtained via heuristic methods proposed in Section 4.2. But if the failure traces are scarce, $\beta = 1.5$ and $N^{'} = 10$ and $\epsilon = 6$ by default. Amongst the proposed approaches, we observe that "*Shewhart chart control* + MMSE" seems a best choice. It is worth noting that the parameters of prediction approaches depend on the MMSE parameters. If N or the number of metrics change, β, $N^{'}$ and ϵ need to be recreated. Generally speaking, the recommended parameter configurations need to be regulated silently in order to work well in different systems. In the future work, we will design adaptive machine learning based approaches to bypass parameter setting.

6 Related Work

As the first line of defending software aging, an effective prediction of ARF is essential. A large quantity of work has been done in this area. Here we briefly discuss related work that has inspired and informed our design, especially work not previously discussed. The ARF prediction approach is based on two points, namely an aging indicator and a concrete prediction method. Most of previous work focus on the design and implementation of a concrete prediction method. While, the aging indicators are overlooked. According to the type of aging indicators, we roughly classify the related work into two categories: explicit indicator based method and implicit indicator based method.

*Explicit Indicator Based Method:*The explicit indicator based method usually uses the directly observed performance metrics as the aging indicators and develops aging detection or prediction approaches based upon these indicators. Actually according to our review, most of prior studies (e.g., [1], [2], [25], [29]) fall to this class. In [1], the authors leverage two memory resource related metrics, *usedSwapSpace* and *realMemoryFree*, to model the degradation of operation software systems. If the memory resource is exhausted, the system is rejuvenated. Grottke et al. [4] also adopt *usedSwapSpace* and *realMemoryFree* as an aging indicator of HTTP web servers and proposed a time series method which allowed seasonal pattern to predict aging. To detect and predict the aging phenomenon in JVMs, Alonso et al. [5] leverage *totalConsumedMmeory* to indicate the aging degree of JVMs. Moreover, a machine learning method is also introduced to predict aging. Similarly, the authors of [12], [26], [29], [53] also leverage memory resource related metrics as aging indicators. Different from these explicit system-specific aging indicators, other work leverage the application-specific metrics (e.g., response time) as aging indicators. Jia et al. [14] use *ResponseTime* as the aging indicator of HTTP web servers and propose an non-linear method to describe the aging process of web servers. In [13], [54], the authors leverage the number of execution event as the aging indicator of JVMs. Moreover, Silva et al. [55] adopt *requestPerSecond* as the aging indicator of SOAP servers. Recently, Matias Jr. et al. [17] propose a fast and robust software aging prediction approach towards the software systems which are suffering memory leak. They claim Resident Set Size (RSS) and Heap Usage (HUS) are the important indicators of software aging. Whether the system-specific aging indicators or the application-specific aging indicators, they can be observed directly.

Based on these indicators, ARFs can be detected or predicted via time-series analysis such as *trend* analysis [9], [12], [26], machine learning [5], [55], [56] or threshold-based approach [46], [47]. A common drawback of these approaches is embodied in the aging indicators' insufficiency due to their weak correlation with software aging. For instance, *trend* analysis requires a large data window as the fluctuation may bias the prediction result with a small data window, which hinders the wide usage of *trend* in real time prediction. What is worse, there may be no significant degradation or rising trend in performance metrics when software goes to failure. Hence, the prediction results cannot reach a satisfactory level, no matter what approaches are employed. Against this drawback, we propose a new aging indicator, namely MMSE, which is extracted from the directly observed performance metrics.

*Implicit Indicator Based Method:*Contrary to the explicit indicator based method, the implicit indicator based method leverage aging indicators embedded in the directly observed performance metrics. These aging indicators are declared to be more sufficient to indicate software aging. Our method falls into this class. Cassidy et al. [57] and Gross et al. [58] leverage "residual" between the actual performance data (e.g., queue length) and the estimated performance data obtained by a multivariate analysis method (e.g., Multivariate State Estimation Technique) as the aging indicator. Then the fault prediction procedure uses a Sequential Probability Ratio Test (SPRT) technique to determine whether the residual value behaved abnormally. Shereshevsky et al. [31] propose another implicit aging indicator: *Hölder* exponent. They show that the *Hölder* exponent of memory utilization decreased with the degree of software aging. By identifying the second breakdown of *Hölder* exponent data series through an online Shewhart algorithm, the ARF was detected. Although Jia et al. [14] do not introduce any implicit aging indicator, they show that software aging process is nonlinear and chaotic. Hence, some complexity-related metrics such as entropy, Lyapunov exponent, etc, are possible to be aging indicators. Our work is inspired by Shereshevsky et al. [31] and

```
errval_t connect(callback cb)
{
(1)  ...
     /* Connect */
(2)  err = coreset_iterate(set, NULL, iter_connect);
     if (err_is_fail(err)) {
(3)    USER_PANIC_ERR(err, "coreset_add failed");
     }

(4)  return SYS_ERR_OK;
}
```

Fig. 24. A code snippet taken from *Connection.c* of BarrelFish [61] source code. Some code above line (1) is truncated. The code at line (3) located in the gray region is triggered by abnormalities.

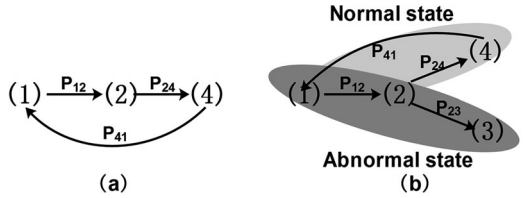


Fig. 25. Different state transitions under normal state and abnormal state. (a) represents the state transition without any exceptions and (b) represents the state transition with exceptions.

Jia et al. [14]. However, the prior studies give no empirical verifications of their implicit aging indicators, no abstraction of the properties that an ideal aging indicator should have, and no multi-scale extension. Moreover, the effectiveness of *Hölder* exponent is only evaluated under emulated increasing workload and a thorough evaluations under real workloads are absent in the their paper. These defects will result in bias in prediction results, which is shown in the real experiments in Section 5.

Another implicit indicator is MSE, which has been widely used to measure the irregularity variation of pathological data such as electrocardiogram data [33], [35], [59]. Motivated by these studies, we first introduce MSE to software aging area. However, we argue that software aging is a complex procedure affected by many factors. Hence, to effectively measure software aging, a multi-dimensional approach is necessary. We extend the conventional MSE to MMSE via several modifications. Wang et al. [60] also adopt entropy as an indicator of performance anomaly. But they measure the entropy using the traditional Shannon entropy rather than MSE.

7 Conclusion

In this paper, we propose a novel implicit aging indicator, namely MMSE, which leverages the complexity embedded in runtime performance metrics to indicate software aging. Through empirical verifications, we demonstrate that MMSE satisfies the properties of *Monotonicity*, *Stability*, and *Integration*. Based upon MMSE, we design and develop a proof-of-concept prototype named ARF-Predictor, which contains three failure prediction approaches, namely *FT* and *FT-X* and the extended version of *Shewhart control chart*. The experimental evaluation results in a VoD system and in a real-world production system, AntVision, show that ARF-Predictor can achieve extraordinarily high accuracy and a very small *ATTF*. Compared to previous approaches, the accuracy of failure prediction approaches based upon MMSE is increased by up to 5 times, and *ATTF* is even reduced by 3 orders of magnitude. In addition, ARF-Predictor is light-weight enough to satisfy the real-time requirement. We believe that ARF-Predictor is an important complement to conventional failure prediction approaches.

Appendix A
Empirical Verification of Entropy Increase in Software Aging

Before introducing the empirical verification of entropy increase with the degree of software aging, we first give a qualitative explanation from the point of view of software execution paths. Fig. 24 demonstrates a code snippet taken from *Connection.c* of BarrelFish [61] source code. If BarrelFish runs without any exceptions, the state transition is: $(1) \rightarrow (2) \rightarrow (4) \rightarrow (1)$ shown in Fig. 25a. As the system runs, exceptions may be thrown out continuously. So a branch stems from (2): one path goes to (3) and the other path goes to (4). In Fig. 25a the transition probabilities: P_{12}, P_{24} and P_{41} are all 1. Hence the system entropy is

$$-(P_{12} * ln(P_{12}) + P_{24} * ln(P_{24}) + P_{41} * ln(P_{41})) = 0.$$

While in Fig. 25b, the transition probabilities: P_{23} and P_{24} are not 1. Thus the system entropy is greater than 0. From this angle, we say the system entropy increases indeed when the system runs from a normal state to an abnormal state gradually. Next, we will show the empirical verification.

The empirical verification is based on three basic assumptions:

Assumption 1. *The software system only exhibits binary states during the whole running procedure: normal working state s_w and failure state s_f.*

Assumption 2. *The probability of s_f increases monotonously with the degree of software aging.*

Assumption 3. *If the probability of s_w is less than the probability of s_f , the system goes to a failure or be rejuvenated very soon.*

A software system may exhibit more than two states during running, but here we only take into account two states, normal working state and failure state, as we only take recovery actions when the system steps into a failure-prone state. This is compliant with the classical two-state aging model, i.e., up, down mentioned in [7], [19], [62] without considering the rejuvenation state. According to the description of software aging stated in the introduction section, the failure rate increases with the degree of software aging. Thus Assumption 2 is reasonable. Actually increasing failure probability is also a common assumption in previous studies [18], [19], [62], [63], [64], [65] in order to obtain an optimal rejuvenation scheduling. For a software system, it is unacceptable if only a half or even less of the total requests are processed successfully especially in modern service oriented systems. A software system is forced to restart before it enters into a non-service state. Therefore, Assumption 3 is proposed.

If the software system is represented as a single component, the system entropy at time t is defined as

$$E(t) = -(p_w(t) * ln(p_w(t)) + p_f(t) * ln(p_f(t))), \quad (7)$$

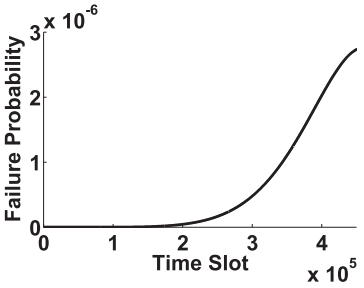


Fig. 26. $p_f(t)$ variation curve.

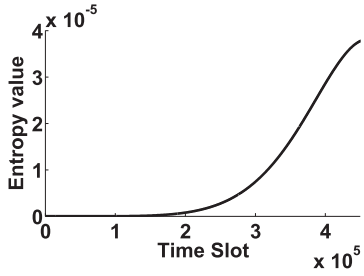


Fig. 27. $E(t)$ variation curve.

where $p_w(t)$ and $p_f(t)$ represent the probability of normal state s_w and failure state s_f at time t respectively, and $p_w(t) + p_f(t) = 1$. At the initial stage, namely $t = 0, p_w(0) = 1$, we say the system is completely new. At this moment, the entropy $E(t)$ equals 0. As software performance degrades, $p_w(t)$ decreases from 1 to 0 while $p_f(t)$ increases from 0 to 1. We assume the failure rate $h(t)$ conforms to a Weibull distribution with two parameters which is commonly used in previous studies [18], [19], [62]. The distribution is described as

$$h(t) = \frac{\beta}{\alpha}\left(\frac{t}{\alpha}\right)^{\beta-1} e^{-(\frac{t}{\alpha})^{\beta}}, \tag{8}$$

where β denotes the shape parameter and α denotes the scale parameter. Because

$$h(t) = \frac{dF(t)/dt}{1 - F(t)} = \frac{p_f(t)}{1 - F(t)}, \tag{9}$$

where $F(t)$ denotes the cumulative distribution function (CDF) of $p_f(t)$. Moreover,

$$F(t) = 1 - e^{\int_0^t h(t)dt} = 1 - e^{-(\frac{t}{\alpha})^{\beta}}. \tag{10}$$

Therefore, $p_f(t)$ could be expressed as

$$p_f(t) = \frac{\beta}{\alpha}\left(\frac{t}{\alpha}\right)^{\beta-1} e^{-2(\frac{t}{\alpha})^{\beta}}. \tag{11}$$

In [62], the authors determined α and β via parameter estimation and gave a confidence interval for α and β respectively. Based upon their result, we set $\alpha = 5.4E5$ and $\beta = 11$ in this paper. The failure probability, $p_f(t)$, from time 0 to time $4.5E5$ (system crash assumed) is depicted in Fig. 26. Accordingly the entropy, $E(t)$, is demonstrated in Fig. 27. From these two figures, we observe that entropy increases monotonously during the life time of the running system. In this case, the failure probability curve is truncated at the system crash, far from the point where $p_f(t) = p_w(t)$. In some corner cases, $p_f(t)$ can reach the point where $p_f(t) = p_w(t)$. However, the system suffers from SLA violations and restarts very soon when $p_f(t) > p_w(t)$. Thus we only take into account the scenario when $p_f(t) < p_w(t)$. In this scenario, the system entropy tends to increase monotonously. Therefore, Theorem 1 is true as long as $p_f(t)$ or $p_w(t)$ changes monotonously.

Theorem 1. *If $p_f(t)$ increases monotonously, the system entropy $E(t)$ monotonously increases with the degree of software aging when $p_f(t) < p_w(t)$ or $p_f(t) < \frac{1}{2}$*

Proof. When $p_f(t) = 0$ or $p_f(t) = 1$, $ln(1 - p_f(t))$ or $ln(p_f(t))$ is not defined. Hence, we assume $p_f(t) \in (0, 1)$. Substitute $p_w(t)$ with $1 - p_f(t)$ in Equation (7), we get

$$\begin{aligned} E(t) &= -((1 - p_f(t)) * ln(1 - p_f(t)) + p_f(t) * ln(p_f(t))) \\ &= -ln(1 - p_f(t)) + p_f(t) * (ln(1 - p_f(t)) - ln(p_f(t))). \end{aligned}$$

Regard $p_f(t)$ as an variable, the first order derivative and second order derivative of $E(t)$ are

$$E(t)' = ln(1 - p_f(t)) - ln(p_f(t)) \tag{12}$$

$$E(t)'' = \frac{-1}{(1 - p_f(t)) * p_f(t)}. \tag{13}$$

As $p_f(t) \in (0, 1)$, $E(t)'' < 0$. Therefore, $E(t)$ achieves the maximum value when $E(t)' = 0$, namely $ln(p_f(t)) = ln(1 - p_f(t))$. Finally, we get $p_f(t) = \frac{1}{2}$. As $p_f(t)$ increases monotonously, $E(t)$ increases monotonously when $p_f(t) < \frac{1}{2}$. Hence, Theorem 1 is proved. □

ACKNOWLEDGMENTS

The work is supported by the National Natural Science Foundation of China under grant No. 61672421 and No. 60933003, the National Key Research and Development Program of China under grant No. 2016YFB1000604, and the Research Grants Council of the Hong Kong Special Administrative Region of China under grant No. CUHK 415212. Moreover, the authors thank the anonymous reviewers. Yong Qi is the corresponding author.

REFERENCES

[1] K. Vaidyanathan and K. S. Trivedi, "A measurement-based model for estimation of resource exhaustion in operational software systems," in *Proc. 10th Int. Symp. Softw. Rel. Eng.*, 1999, pp. 84–93.

[2] K. Vaidyanathan, R. E. Harper, S. W. Hunter, and K. S. Trivedi, "Analysis and implementation of software rejuvenation in cluster systems," in *Proc. of ACM SIGMETRICS Performance Eval. Review*, 2001, vol. 29, no. 1, pp. 62–71.

[3] K. Vaidyanathan and K. S. Trivedi, "A comprehensive model for software rejuvenation," *IEEE Trans. Dependable Secure Comput.*, vol. 2, no. 2, pp. 124–137, 2005.

[4] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, "Analysis of software aging in a web server," *IEEE Trans. Reliability*, vol. 55, no. 3, pp. 411–420, Apr.–Jun. 2006.

[5] J. Alonso, J. Torres, J. L. Berral, and R. Gavalda, "Adaptive on-line software aging prediction based on machine learning," in *Proc. 40th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2010, pp. 507–516.

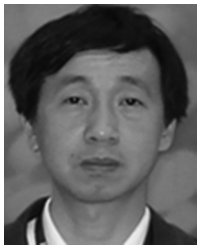
[6] S. P. Kavulya, S. Daniels, K. Joshi, M. Hiltunen, R. Gandhi, and P. Narasimhan, "Draco: Statistical diagnosis of chronic problems in large distributed systems," in *Proc. 42nd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2012, pp. 1–12.
[7] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *Proc. 25th Int. Symp. Fault-Tolerant Comput.*, 1995, pp. 381–390.
[8] J. Araujo, et al., "Software aging in the eucalyptus cloud computing infrastructure: Characterization and rejuvenation," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 10, no. 1, 2014, Art. no. 11.
[9] J. Araujo, R. Matos, P. Maciel, R. Matias, and I. Beicker, "Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure," in *Proc. 12th Int. Middleware Conf. Ind. Track Workshop*, 2011, pp. 1103–1108.
[10] K. Kourai and S. Chiba, "Fast software rejuvenation of virtual machine monitors," *IEEE Trans. Dependable Secure Comput.*, vol. 8, no. 6, pp. 839–851, Nov./Dec. 2011.
[11] K. Kourai and S. Chiba, "A fast rejuvenation technique for server consolidation with virtual machines," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2007, pp. 245–255.
[12] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "Software aging analysis of the linux operating system," in *Proc. 21st Int. Symp. Softw. Rel. Eng.*, 2010, pp. 71–80.
[13] D. Cotroneo, S. Orlando, and S. Russo, "Characterizing aging phenomena of the java virtual machine," in *Proc. 26th IEEE Int. Symp. Reliable Distrib. Syst.*, 2007, pp. 127–136.
[14] Y.-F. Jia, L. Zhao, and K.-Y. Cai, "A nonlinear approach to modeling of software aging in a web server," in *Proc. 15th Asia-Pacific Softw. Eng. Conf.*, 2008, pp. 77–84.
[15] M. Grottke, R. Matias, and K. S. Trivedi, "The fundamentals of software aging," in *Proc. IEEE Int. Conf. Softw. Rel. Eng. Workshops*, 2008, pp. 1–6.
[16] A. Bovenzi, D. Cotroneo, R. Pietrantuono, and S. Russo, "On the aging effects due to concurrency bugs: A case study on MySQL," in *Proc. 23rd IEEE Int. Symp. Softw. Rel. Eng.*, 2012, pp. 211–220.
[17] R. Matias, A. Andrzejak, F. Machida, D. Elias, and K. Trivedi, "A systematic differential analysis for fast and robust detection of software aging," in *Proc. 33rd IEEE Int. Symp. Reliable Distrib. Syst.*, 2014, pp. 311–320.
[18] Y. Bao, X. Sun, and K. S. Trivedi, "A workload-based analysis of software aging, and rejuvenation," *IEEE Trans. Rel.*, vol. 54, no. 3, pp. 541–548, Sep. 2005.
[19] K. S. Trivedi, K. Vaidyanathan, and K. Goseva-Popstojanova, "Modeling and analysis of software aging and rejuvenation," in *Proc. 33rd Annu. Simulation Symp.*, 2000, pp. 270–279.
[20] R. Matias, P. A. Barbetta, K. S. Trivedi, and P. J. F. Filho, "Accelerated degradation tests applied to software aging experiments," *IEEE Trans. Rel.*, vol. 59, no. 1, pp. 102–114, Mar. 2010.
[21] A. Bovenzi, D. Cotroneo, R. Pietrantuono, and S. Russo, "Workload characterization for software aging analysis," in *Proc. 22nd IEEE Int. Symp. Softw. Rel. Eng.*, 2011, pp. 240–249.
[22] B. Sharma, P. Jayachandran, A. Verma, and C. R. Das, "Cloudpd: Problem determination and diagnosis in shared dynamic clouds," in *Proc. 43rd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2013, pp. 1–12.
[23] V. Castelli, et al., "Proactive management of software aging," *IBM J. Res. Develop.*, vol. 45, no. 2, pp. 311–332, Mar. 2001.
[24] M. Grottke and B. Schleich, "How does testing affect the availability of aging software systems?" *Performance Eval.*, vol. 70, no. 3, pp. 179–196, Mar. 2013.
[25] P. Zheng, Y. Qi, Y. Zhou, P. Chen, J. Zhan, and M. Lyu, "An automatic framework for detecting and characterizing performance degradation of software systems," *IEEE Trans. Rel.*, vol. 63, no. 4, pp. 927–943, Dec. 2014.
[26] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S. Trivedi, "A methodology for detection and estimation of software aging," in *Proc. 9th Int. Symp. Softw. Rel. Eng.*, 1998, pp. 283–292.
[27] G. Jin, W. Zhang, D. Deng, B. Liblit, and S. Lu, "Automated concurrency-bug fixing," in *Proc. 11th USENIX Symp. Operating Syst. Des. Implementation*, 2012, pp. 221–236.
[28] J. Zhao, Y. B. Wang, G. R. Ning, K. S. Trivedi, R. Matias, and K. Y. Cai, "A comprehensive approach to optimal software rejuvenation," *Performance Eval.*, vol. 70, no. 11, pp. 917–933, Nov. 2013.
[29] J. Zhao, K. S. Trivedi, M. Grottke, J. Alonso, and Y. Wang, "Ensuring the performance of apache HTTP server affected by aging," *IEEE Trans. Dependable Secure Comput.*, vol. 11, no. 2, pp. 130–141, Mar. 2014.
[30] M. Grottke, R. Matias, and K. S. Trivedi, "The fundamentals of software aging," in *Proc. IEEE Workshop Softw. Aging Rejuvenation Conjunction ISSRE*, 2008, pp. 1–6.
[31] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu, "Software aging and multifractality of memory resources," in *Proc. 33rd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2003, pp. 721–730.
[32] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, 1948.
[33] M. Costa, A. L. Goldberger, and C.-K. Peng, "Multiscale entropy analysis of biological signals," *Physical Rev. E*, vol. 71, no. 2, 2005, Art. no. 021906.
[34] S. M. Pincus and A. L. Goldberger, "Physiological time-series analysis: what does regularity quantify?" *Amer. J. Physiology*, vol. 266, pp. H1643–H1643, 1994.
[35] M. U. Ahmed and D. P. Mandic, "Multivariate multiscale entropy: A tool for complexity analysis of multichannel data," *Physical Rev. E*, vol. 84, no. 6, 2011, Art. no. 061918.
[36] D. E. Lake, J. S. Richman, M. P. Griffin, and J. R. Moorman, "Sample entropy analysis of neonatal heart rate variability," *Amer. J. Physiology-Regulatory Integrative Comparative Physiology*, vol. 283, no. 3, pp. R789–R797, 2002.
[37] M. Costa, A. L. Goldberger, and C.-K. Peng, "Multiscale entropy analysis of complex physiologic time series," *Physical Rev. Lett.*, vol. 89, no. 6, 2002, Art. no. 068102.
[38] L. Cao, A. Mees, and K. Judd, "Dynamics from multivariate time series," *Physica D: Nonlinear Phenomena*, vol. 121, no. 1, pp. 75–88, 1998.
[39] L. Li, K. Vaidyanathan, and K. S. Trivedi, "An approach for estimation of software aging in a web server," in *Proc. IEEE Int. Symp. Empirical Softw. Eng.*, 2002, pp. 91–100.
[40] (2013). [Online]. Available: https://influxdb.com/
[41] I. Jolliffe, *Principal Component Analysis*. Hoboken, NY, USA: Wiley, 2005.
[42] J. Cadima, J. O. Cerdeira, and M. Minhoto, "Computational aspects of algorithms for variable selection in the context of principal components," *Comput. Statist. Data Anal.*, vol. 47, no. 2, pp. 225–236, 2004.
[43] J. F. Cadima and I. T. Jolliffe, "Variable selection and the interpretation of principal subspaces," *J. Agricultural Biol. Environ. Statist.*, vol. 6, no. 1, pp. 62–79, 2001.
[44] J. Ramsay, J. ten Berge, and G. Styan, "Matrix correlation," *Psychometrika*, vol. 49, no. 3, pp. 403–423, 1984.
[45] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "A survey of software aging and rejuvenation studies," *ACM J. Emerging Technol. Comput. Syst.*, vol. 10, no. 1, 2014, Art. no. 8.
[46] L. M. Silva, J. Alonso, and J. Torres, "Using virtualization to improve software rejuvenation," *IEEE Trans. Comput.*, vol. 58, no. 11, pp. 1525–1538, Nov. 2009.
[47] J. Alonso, Í. Goiri, J. Guitart, R. Gavalda, and J. Torres, "Optimal resource allocation in a virtualized software aging platform with software rejuvenation," in *Proc. 22nd IEEE Int. Symp. Softw. Rel. Eng.*, 2011, pp. 250–259.
[48] (2008). [Online]. Available: http://www.sourceforge.net/projects/hyperic-hq
[49] P. Chen, Y. Qi, P. Zheng, J. Zhan, and Y. Wu, "Multi-scale entropy: One metric of software aging," in *Proc. 7th IEEE Int. Symp. Service Oriented Syst. Eng.*, 2013, pp. 162–169.
[50] P. Zheng, Y. Zhou, M. R. Lyu, and Y. Qi, "Granger causality-aware prediction and diagnosis of software degradation," in *Proc. 11th IEEE Int. Conf. Services Comput.*, 2014, pp. 528–535.
[51] (2002). [Online]. Available: http://www.helix-server.helixcommunity.org/
[52] L. Jiang and G. Xu, "Modeling and analysis of software aging and software failure," *J. Syst. Softw.*, vol. 80, no. 4, pp. 590–595, 2007.
[53] J. Zhao, Y. Jin, K. S. Trivedi, R. Matias, and Y. Wang, "Software rejuvenation scheduling using accelerated life testing," *ACM J. Emerging Technol. Comput. Syst.*, vol. 10, no. 1, pp. 1–23, 2014.
[54] D. Cotroneo, S. Orlando, R. Pietrantuono, and S. Russo, "A measurement-based ageing analysis of the JVM," *Softw. Testing Verification Rel.*, vol. 23, no. 3, pp. 199–239, 2013.
[55] A. Andrzejak and L. Silva, "Using machine learning for non-intrusive modeling and prediction of software aging," in *Proc. IEEE Netw. Operations Manage. Symp.*, 2008, pp. 25–32.
[56] J. P. Magalhaes and L. Moura Silva, "Prediction of performance anomalies in web-applications based-on software aging scenarios," in *Proc. 2nd Int. Workshop Softw. Aging Rejuvenation*, 2010, pp. 1–7.

[57] K. J. Cassidy, K. C. Gross, and A. Malekpour, "Advanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers," in *Proc. 32nd IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2002, pp. 478–482.
[58] K. C. Gross, V. Bhardwaj, and R. Bickford, "Proactive detection of software aging mechanisms in performance critical computers," in *Proc. 27th Annu. NASA Goddard/IEEE Softw. Eng. Workshop*, 2002, pp. 17–23.
[59] M. U. Ahmed and D. P. Mandic, "Multivariate multiscale entropy analysis," *IEEE Signal Process. Lett.*, vol. 19, no. 2, pp. 91–94, Feb. 2012.
[60] C. Wang, V. Talwar, K. Schwan, and P. Ranganathan, "Online detection of utility cloud anomalies using metric distributions," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, 2010, pp. 96–103.
[61] (2010). [Online]. Available: http://www.barrelfish.org/
[62] J. Zhao, Y. Jin, K. S. Trivedi, and R. Matias, "Injecting memory leaks to accelerate software failures," in *Proc. 22nd IEEE Int. Symp. Softw. Rel. Eng.*, 2011, pp. 260–269.
[63] A. Bobbio, M. Sereno, and C. Anglano, "Fine grained software degradation models for optimal rejuvenation policies," *Performance Eval.*, vol. 46, no. 1, pp. 45–62, 2001.
[64] T. Dohi, K. Goseva-Popstojanova, and K. S. Trivedi, "Analysis of software cost models with rejuvenation," in *Proc. 5th IEEE Int. Symp. High Assurance Syst. Eng.*, 2000, pp. 25–34.
[65] R. E. Barlow and R. A. Campo, "Total time on test processes and applications to failure data analysis," *Rel. Fault Tree Anal.*, SIAM, Philadelphia, pp. 451–481, 1975.

Pengfei Chen received the PhD degree from Xi'an Jiaotong University, China, in 2016. His research interests include dependable computing, cloud computing, and distributed computing.

Yong Qi received the PhD degree from Xi'an Jiaotong University, China. He is currently a full professor with Xi'an Jiaotong University. His research interests include operating systems, distributed systems, and cloud computing. He is a member of the IEEE.

Xinyi Li is working toward the PhD degree at Xi'an Jiaotong University, China. Her research interests include cloud computing, distributed computing, and Internet of things.



Di Hou is currently an associate processor with Xi'an Jiaotong University. His research interests include big data, cloud computing, and database.

Michael Rung-Tsong Lyu received the PhD degree in computer science from the University of California, Los Angeles, in 1988. He is now a professor in the Department of Computer Science & Engineering, The Chinese University of Hong Kong. He initiated the First International Symposium on Software Reliability Engineering, in 1990. His research interests include software reliability engineering, distributed systems, fault-tolerant computing, data mining, and machine learning.He received the IEEE Reliability Society 2010 Engineer of the Year Award. He is a fellow of the IEEE and AAAS.