Multi-Scale Entropy: One Metric of Software Aging

Pengfei Chen[+] Yong Qi[+] Pengfei Zheng[+] Jianfeng Zhan[*] Yihan Wu[‡]
[+]Computer Science of Xi'an Jiaotong univercity Xi'an, China
[*]Advanced Computer Systems Laboratory Institute of Computing Technology, Chinese Academy of Sciences, China
[‡]School of Electronic Engineering & Computer Science, Peking University, China
{fly.bird.sk ,qiy, p.f.zheng }@stu.xjtu.edu.cn, jfzhan@ncic.ac.cn, wuyh10@sei.pku.edu.cn

***Abstract*—The phenomena of service performance or availability degradation have been widely observed in the long running software systems, which is called 'software aging'. It's hard to measure software aging due to the inherent complexity and dynamic of software systems. In our conjecture, the metrics of software aging should have three properties: *stability, monotonicity, integrity*, but as far as we know some pervious metrics (e.g. the original continuously-observed performance data, hölder exponent) could not simultaneously satisfy all the properties. We extend classical multi-scale entropy analysis, propose a novel aging metric called multidimensional multi-scale entropy and prove that it have all the three properties through theory and experiment analysis. Comparing with other metrics, our metric has several advantages: (i) it's a holistic measurement integrating multiple properties of software system (ii) it can largely reduce the false alarms of pseudo software aging and further increase the availability of software and decrease the cost brought by unnecessary software rejuvenations (iii) it can be used in an online aging detection method and an offline state classification method.**

Keywords-software aging; measurement; multi-scale entropy; software rejuvenation; availability

I. INTRODUCTION

As the increasing of software requirements, the development phase of software is becoming short resulting in incomplete software verification and test. Some bugs which are hard to be detected without thorough test such as *Heisen* bugs and Bohr bugs are released to the running period. Moreover, the extraordinary complexity of modern software system makes this situation even worse [4]. During system running, certain of these bugs may be triggered occasionally by the variable workloads or the software itself leading to performance degradation, available reduction and even system crash. This phenomenon which is most like 'aging' in organism is reported as software aging [3][5][6]. In literature [6], Grottke attributed bugs causing system gradual degradation at runtime to aging related bugs. Actually, software aging is a complex process affected by external environment (e.g workloads, virus) [8] and internal environment (e.g system management policies, available resources) [2][9]. Software aging has been discussed for more than a decade. Although the complete fundamental mechanism of this phenomenon is still argued so far, some partial reasons are accepted publicly including memory leak, data corruption, unreleased file locks and round-off error mainly concentrating on resources exhaustion [3]. Similar to the universality of biological aging, software aging exists in many software systems. Up to now, people have found software aging in operating system [10], java virtual machine [11], online transaction processing server [3], web server [7], and telecom accounting system [5]. It can cause great loss in different systems. In an optimistic situation, software aging only results in performance degradation affecting QoS. Consequently, the experience of customers degraded and the total number of trades or transactions decreases, further leading to revenue decrease. This situation is common in web service providers like Google, Baidu and Amazon. In a pessimistic situation, software aging can bring serious consequence. Several studies have reported that one of the causes of unplanned outage is software aging. Unplanned outage not only interrupts service currently running, but may pollute the data stored in database due to unfinished transactions and further cause errors in the subsequent transaction processing. This situation will get even worse in modern supper commercial data centers. The loss is beyond calculation. In some military systems, software aging not only brings performance loss but also human lives. During Gulf War, an error caused by software aging happened in American 'patriot' missile system. Because of this error, 28 American soldiers lost their lives.

So it's very important to study how software aging happens, how to detect software aging and how to counteract software aging. This paper is concerned with how to detect software aging. Our motivations are as follows.

1) Motivation 1

Software rejuvenation proposed by Huang [5] is an effective and efficient method to counteract software aging. It heals systems from aging states to health states through proactively cleaning the internal states occasionally. But unnecessary rejuvenation may bring extra cost. So the optimized scheduling of software rejuvenation is still worthy studied until now. Recently, two rejuvenation strategies are developed including model-based methods and measurement-based methods. The aim of model-based methods is to figure out the optimized rejuvenation cycle through solving mathematical models like Markov model or differential equation. In the Markov model [14][15] (e.g. Markov chain, Markov decision process), the assumption assuming that software aging process goes through several states is given more often than not. Few of them validate the aging states in real experiment. Another shortcoming of

978-0-7695-4944-6/12 $26.00 © 2012 IEEE
DOI 10.1109/SOSE.2013.23

Markov model is that it's not appropriate to be used in reality due to the inherent dynamic property of software systems. To resolve this problem, measurement-based rejuvenation method emerges. This kind of method implements time series prediction methods such as AR (Auto Regression) or else to detect software aging proactively before system crash. As far as we know, previous measurement-based rejuvenation methods mostly take continuously-observed performance parameters (e.g cpu utilization, memory exhaustion or disk queue) as the metrics of software aging [2][3][7][9]. When the thresholds preset on these metrics are reached, rejuvenation is conducted. However in reality, much noise due to the dynamic of software will be mixed in the original performance observation which makes precise prediction nearly impossible and leads to false alarms.

2) Motivation 2

Another reason driving us to explore new aging metrics is the belief that some fundamental metrics are hidden underlying the superficial observations.

What properties should the metrics have? In our conjecture, the metrics may have the following properties.

- ***Stability***. The variation of these metrics will not be affected by the dynamic of software systems so much. They keep relatively stable whatever the workload of software system changes.
- ***Monotonicity***. This property is obvious as software aging is a gradually decreasing process with time. But note that it is not strictly non-decreasing or non-increasing because of external or internal covariates effect.
- ***Integrity***. Software aging is the exhibition of the holistic system health state, hence these metrics should integrate all the properties of the system, and namely they are the measurements of multi-dimensional observations.

This property set may be not complete, but we strongly believe that the metrics satisfying these properties can measure the degree of software aging.

Vaidyanathan and Trivedi [15] proposed a semi-Markov reward model based on system workload and resource usage to estimate the time of failure of a system. However, the data they collected tend to fluctuate a great deal from the supposed linear trends, resulting in prohibitively wide confidence intervals. Their software aging metric is weak in *stability*. Shereshevsky [1] introduced another aging metric named Hölder exponent. But his work only concentrated on memory data without considering other system properties. And it is also weak in counteracting the dynamic of software aging process due to the detection at every time point.

Motivated by these two motivations, we propose a novel metric called multi-dimensional multi-scale entropy (abbreviated as MMSE). This metric is similar to Hölder exponent aiming to measure the irregular variations [19-23] of performance observations, but it is stronger in *stability*.

Multi-scale entropy has been widely used to measure the irregularity variation of pathological data such as electrocardiogram data. This paper leverages this metric to measure software aging degree. But the original metric doesn't meet all the properties proposed above (lack of *integrity*), so we extend the original method to multi-dimensional method. It can be calculated on line to detect software aging degree in real time, when the degree exceeds a threshold proper rejuvenation is conducted. Meanwhile it can also be calculated off line to classify the aging states during the whole aging process, which is a complementary to markov decision process. In this paper, we will not prove the three properties of MMSE theoretically rather than validate them experimentally.

The rest of this paper is organized as follows. In Section Ⅱ we give several real cases showing multi-scale entropy increases with software aging. In Section Ⅲ, we describe the classical multi-scale entropy and our extended multi-dimensional multi-scale entropy. Section Ⅳ gives the detailed design of our experiment. Then in Section Ⅴ, we analyze the multi-scale entropy of several observations collected in our test bed and compare the aging detection based on our metric with two prediction method based on auto regression and Hölder exponent. Section Ⅵ concludes this paper.

II. ENTROPY AND SOFTWARE AGING

In the introduction section, we point out software aging metrics should have three properties. In this section, we will give several real cases showing the ***monotonicity*** of multi-scale entropy.

In reality, not only our work is concerned with entropy increasing phenomenon in software aging, but also other related works have reported similar phenomenon. In literature [1], the author took Hölder exponent as a metric of software aging which is the basic numerical expression in the multifractal analysis. Hölder exponent manifests the degree of irregularity and chaos of a function. While entropy can also denotes the irregular and chaotic properties of a function. Therefore these two measurements have similar variation process. As reported by the author of literature [1], the Hölder exponent becomes lower when system gets aging, that is to say the irregularity and chaos degree increases, which is the same as our conclusion. Another paper [18] presented the number of log message per hour in a real production system, Red Storm DDN system, evolved over the course of system's life. From the data implemented in this paper, intuitively we can see the number of log becomes irregular with the passage of time.

In a real campus VoD (Video on Demand) system which is used to share movies among students, the entropy increasing phenomenon is also observed over a long time running. We collected the CPU utilization data generated over 50 days and at the end of the collection the system crashed (proved to be caused by aging). The data is divided every ten of thousands records. The first 40000 records are denoted by D_1, D_2, D_3, D_4 and the last 40000 records are denoted by D_5, D_6, D_7, D_8 (See Figure 1). The multi-scale entropies of D_1, D_2, D_3, D_4 and D_5, D_6, D_7, D_8 for 30 scales are calculated implementing method proposed in this paper (See Figure 2). It's apparently to see the entropy of D_5, D_6, D_7, D_8 is much larger than the one of D_1, D_2, D_3, D_4 for all scales.

VoD will be implemented as our test bed in the following part. But because of security and privacy of the real VoD system, we can't get enough data. So, a new test bed is constructed in our experiment.

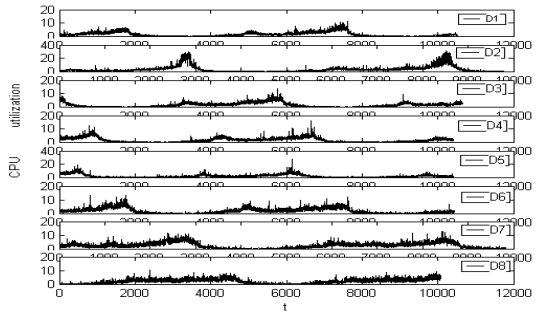


Figure 1 CPU utilization of a real VoD system

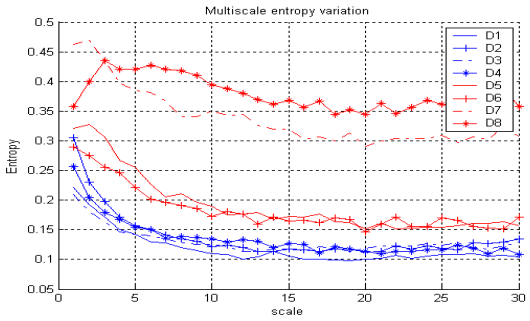


Figure 2 The entrop variation for all scales

From upper cases, we discovery entropy increasing exists in software aging process and it shows the ***monotonicity*** property. Although this property is not proved by rigorous theory, it is still an important indicator to measure software aging.

III. MULTI-SCALE ENTROPY METRIC

A. Classical multi-scale entropy

In order to capture the temporal structure and self similarity embedded in time series, sample entropy is proposed by Richman and Moorman [24]. Sample entropy has been widely used in real time series as it is less dependent on time series length and yields more consistent estimators. But dynamic system may exhibit different structure characters at different scales. Single scale analysis is not enough to capture these hidden characters, so multi-scale analysis provides a potential solution. Multi-scale entropy proposed by Costa [25] extends sample entropy from one scale to multi-scale. The coarse-graining step is added into the calculation process of sample entropy. As limited by the space of this paper, the principle and calculation of multi-scale entropy are not described here. We strongly recommend the readers to refer the paper [25].

In literature [26], Lake discussed the problem of determining the embedded dimension m and distance threshold r. He suggested $m=2, r=0.15\sigma$, σ is the standard deviation of the original time series. $m=2$ is implemented in our paper but the value of r is modified to suit with our metric.

Classical multi-scale entropy has been widely used to measure single dimensional time series, but it can't be implemented as our metric yet as it lacks property *integrity*. Some expansions are made in the following part.

B. Multi-dimensional multi-scale entropy

In order to satisfy the property *integrity* and to be analyzed in real time, the classical multi-scale entropy is extended in several aspects.

- Our metric will be used to detect and measure software aging in real time, so we need to set a data window which will be fed into multi-scale entropy calculation process, namely to determine N. N should not be set too large as that would exhaust too much CPU cycles once calculation or too small as that would miss some temporal structures in a long range, leading to false alarms. In this paper we find out a tradeoff among these factors. The detail will be illustrated in our experiment result analysis section.
- Different observations collected at software running time such as CPU utilization or disk queue have different units. Before fed into the entropy calculation process, these data should be normalized. Suppose that $(A_1, A_2, \cdots, A_i, \cdots A_m)$ is a group of system observations, the data window is N, A_i is normalized as follows:

$$A_{ij}^{'} = \frac{A_{ij} - \min(A_i)}{\max(A_i) - \min(A_i)} \times 100, \quad 1 \le \mathrm{i} \le \mathrm{m}, 1 \le \mathrm{j} \le \mathrm{N}$$

Where $A_{ij}^{'}$ is the normalized data, $A_{ij}^{'} \in [0,100]$, A_{ij} is the original data of i th observation, $\min(A_i)$ denotes the minimum value of i th observation in data window N, $\max(A_i)$ denotes the maximum value.

- In classical multi-scale entropy calculation process, in order to get $C^m(r)$, we need to count the number of vectors that satisfy $d[u_m(i), u_m(j)] \le r$. $u_m(i)$ is a one dimensional vector. Considering multi dimensional properties, every element of $u_m(i)$ is extended to a vector and $u_m(i)$ becomes a matrix. So the Euclidean distance in one dimensional space is extended to the one in multi-dimensional space.
- The standard deviation used to determine the threshold r is modified. Instead of calculating the standard deviation of the original time series, we implement the standard deviation of the data transformed as the following steps. Suppose that $(A_1, A_2, \cdots, A_i, \cdots A_m)$ denotes a group of original observations, $(A_1^{'}, A_2^{'}, \cdots, A_i^{'}, \cdots A_m^{'})$ denotes the normalized observations, and N is the data window.
  1. Transform $(A_1^{'}, A_2^{'}, \cdots, A_i^{'}, \cdots A_m^{'})$ into one dimensional Euclidean distance vector E with length N.

$$E_i = \sqrt[2]{\sum_{j}^{m} (A_{ji}^{'})^2}, \qquad 1 \le i \le N \tag{1}$$

2. Calculate the standard deviation of E, denoted by $sd(E)$.
3. Let $r = sd(E)$ instead of $r = 0.15 \times sd(E)$.

- Considering the entropy for all scales, we propose a holistic aging metric. Let T denote the number of scales and $(e_1, e_2, \cdots, e_T)$ denote the entropy for each scale. The aging metric $Aging_e$ is defined as:

$$Aging_e = \frac{\sqrt[2]{\sum_{i=1}^{T} e_i^2}}{T} \tag{2}$$

Through the several modifications to the original multi-scale entropy analysis, we can see the aging metric $Aging_e$ has the properties *monotonicity* and *integrity*. This metric is more stable than other metrics based on single point threshold as it is a data window based method and integrates the entropy values at all scales. The *stability* comparison (false alarm comparison) will be illustrated in our experiment result analysis section. So far, $Aging_e$ satisfies all the three properties proposed in the introduction section and it could be used as an aging metric. The complete process of calculating this aging metric is demonstrated in Figure 3.

Algorithm: Multi-dimensional multi-scale entropy

Symbol notes:

m : the embedded dimension
T : the number of scales
N :the length of data window
τ : the scale factor
E :an Euclidean distance vector
A_i :the original observation
y_i^{τ} :the coarse-grained data

Set: m , T , N

Input: $(A_1, A_2, \cdots, A_l)$

For each scale $\tau = 1 \sim T$
  Normalize A_i, $1 \le i \le l$
  Transform $(A_1, A_2, \cdots, A_l)$ into E
  Calculate the standard deviation of E
  For each $i = 1 \sim l$
    Calculate $\{y_i^{\tau}\}$
  End
  Calculate the sample entropy of $\{y_i^{\tau}\}$
End
Calculate the aging metric $Aging_e$

Output: $Aging_e$

Figure 3 Algorithm of extended multi-scale entropy

IV. EXPERIMENT DESIGN

A. Experiment framework

We implement a stream media software system, Helix Server, as a test bed to validate the multi-scale entropy metric proposed above. HelixServer as a mainstream stream media software system transforms video and audio between clients and servers using RTSP/ RTP protocol. At present, there is no standard stream media benchmark. So test tools are few, this paper develops a client emulator called '*HelixClientEmulator*'employing RTSP and RTP protocols. '*HelixClientEmulator*' includes three threads, one for audio processing, one for video processing and the other for session management. It can generate multiple concurrent clients to access media files on HelixServer. Our test bed includes one server computer, three client computers and one Gbps exchanger.

B. Experiment methodology

100 rmvb media files with different rates are deployed on the HelixServer. This paper implements *HelixClient Emulator* to generate different workloads based on DOE (Design of Experiment) methodology. Here workload is represented by a tuple (*client_count, file_ploy, file_object, file_max_object, sleep_time, file_difference*). The detailed description of these attributes is listed in Table 1. In every experiment, *client_count* is fixed. If a client finishes file transportation, a new client will be created. In order to avoid the impact from former experiment, HelixServer is restarted before each experiment.

To make sure the workloads are within the capacity of software system, capacity test is performed. When request number arrives 900, HelixServer is restarted automatically and the transportation speed drops immediately. So 900 is the limited capacity of our test bed, the following experiment will not exceed this limit. The bandwidth of every simulated client doesn't exceed 140kbps, so the total bandwidth is no more than 126Mbps within the service capacity of our exchanger. Therefore the transportation ability is not a bottleneck in our test bed. During this software system running, thousands of performance counters and events are generated. In order to trade off between collection effort and information loss, this paper selects some of the parameters at four different levels: *HelixClientEmulator, operatingSystem, HelixServer, and rmserver process* to monitor.

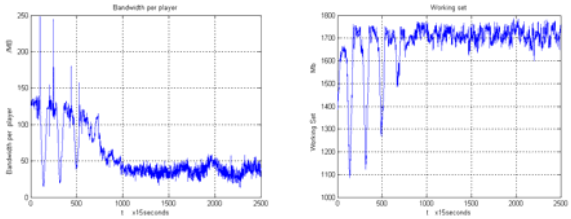
Table 1 workload attributes

attribute	description
client_count	total emulated Helix clients
file_ploy	Access distribution: 0 denotes random access, 1 denotes sequential access, 2 denotes Poisson access, 3 denotes single file access
file_object	Files allowed accessed, users have no authorization to access files beyond these files.
file_max_object	Maximal files allowed accessed within 0 to file_object concurrently
sleep_time	Interval between two continuous requests
file_difference	Whether accessed files are the same: 0 denotes different, 1 denotes same

V. EXPERIMENT RESULT ANALYSIS

A. Software aging in the test bed

We conducted 20 experiments and selected three representative ones using different workloads. They are (600,0,100,20,1000,0), (700,0,100,20,1000,0), (600,0,100,15,1000,0), represented by w_1, w_2, w_3, respectively. We detect the aging trend using two common observations: memory utilization and bandwidth throughput.



a. Bandwidth per player b. Memory utilization

Figure 4 Bandwidth and memory

Even a cursory glance at these figures, a decreasing trend in bandwidth and an increasing trend in memory utilization are observed during software running. From Figure 4, we can get this ***conclusion*****:** at the initial phase of the software system, the aging speed is slow; after a gradually increasing phase, the aging speed tends to be zero, in this situation the system can't provide normal service, as the bandwidth is less than 30 /Kbps causing serious video and audio frame loss. Therefore our test bed experiences software aging during a long time running under these workloads. The multi-scale entropy will be analyzed during the aging process in the following parts.

B. Length of data window

In our method, the length of data window N should be determined first. Let the embedded dimension $m = 2$, and the number of scales $T = 10$. We select four observations including '\Process(rmserver)\Working Set', '\Physical Disk (_Total)\Avg.DiskQueueLength', '\Processor(_Total)\% Processor Time', and 'Average Bandwidth Output Per Player(bps)' at different levels under workload w_2. The computation time with respect to different length of data window is exhibited in Figure 5. The computation time ranges from less than 1s to more than 70s. But it doesn't mean the smaller the length of data window, the better the aging detection. We also need to consider the number of false alarms with respect to different length of data window. Our test bed could not provide acceptable service at around 1200x15 second time point and if the multi-scale entropy before this time point is greater than the multi-scale entropy at this time point (If there is no entropy at this time point, we use the one after this time point) , a false alarm rises. Taking the length 50 as an example, before 1200x15 second time point, there are five entropy values greater than 2.5 which is the entropy at that time point, therefore five false alarms are generated (See Figure 6). False alarms may cause unnecessary software rejuvenation and should be avoided. The number of false alarms with respect to length of data window is demonstrated in Figure 7. From Figure 7, we can see when the length of data window is greater than 100; the number of false alarms becomes zero. As the above considerations, we choose 150 as our length of data window.

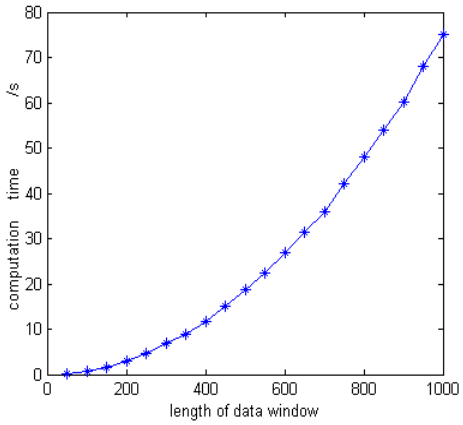


Figure 5 Computation time *VS* length of data window

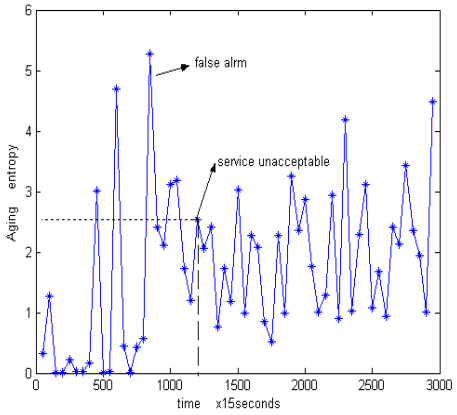


Figure 6 The multi-scale entropy when *N=50*

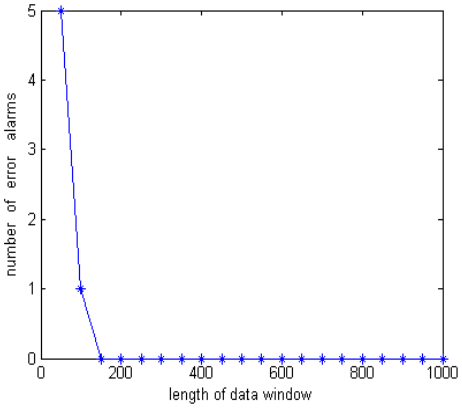


Figure 7 False alarms of different length of data window

C. Multi-scale entropy analysis

We set the embedded dimension $m = 2$, the number of scales $T = 10$ and the length of data window $N = 150$ in our multi-scale analysis. During software aging process, the entropy of certain observation may not change significantly (See Figure 8). But considering multidimensional

observations, the variation of entropy is significant (See Figure 9). That's the benefit of the extension from classical multi-scale entropy to multidimensional multi-scale entropy.

Now we use our extended multi-scale entropy to analyze the observations collected at different level. At 'Rmserver process' level, we select four observations including '\Process(rmserver)\% Privileged Time', '\Process (rmserver)\% User Time', '\Process(rmserver)\IO Data Bytes/sec', '\Process(rmserver)\Working Set'. The multi-scale entropy of workload w_2 is shown in Figure 10. From this figure, we can see before service unacceptable time point, the multi-scale entropy value is increasing monotonously and after this time point the multi-scale entropy changes slightly. Therefore, we can use the entropy value at service unacceptable time point as our referential threshold. If the entropy of software system reaches this threshold, software rejuvenation or other actions are conducted. Note that there are no false alarms here.

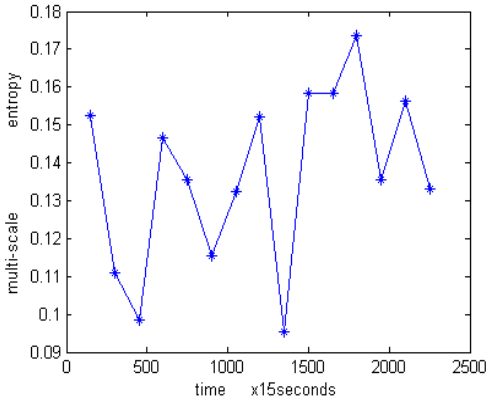


Figure 8 Multi-scale entropy of '\Processor (_Total) \% Interrupt Time' under workload w_1

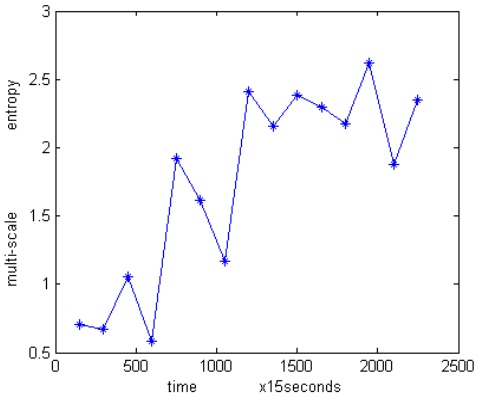


Figure 9 Multi-scale entropy considering four observations under workload w_1

At 'client' level, four observations including 'Jitter in last 10 seconds', 'Total jitter', 'Average bandwidth', 'Average response time' are selected. The multi-scale entropy under workload w_3 is exhibited in Figure 11. Apparently, we can detect and measure software aging easily shown in Figure 11.

Due to the limited space, the multi-scale entropy analysis of other observations and workloads are not presented here.

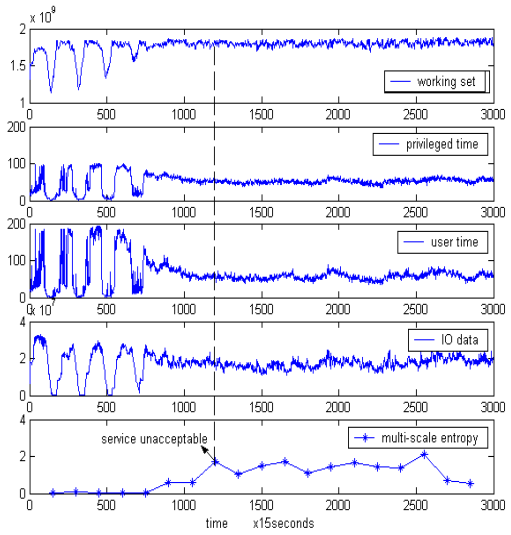


Figure 10 Multi-scale entropy at 'process' level under w_2

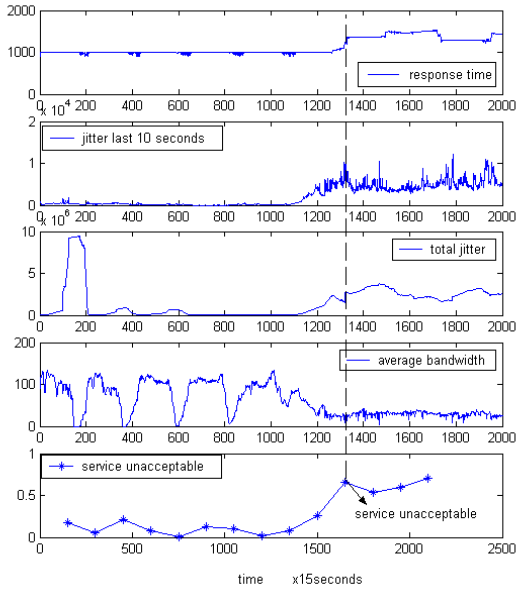


Figure 11 Multi-scale entropy at 'client' level under w_3

The analysis above has shown that the multi-scale entropy has been increasing before service unacceptable time point and after this time point the entropy tends to be stable, satisfying the property ***monotonicity***. Through the multi-scale entropy analysis in real time, we can detect the software aging degree on line. If the entropy value exceeds the one at service unacceptable time point, appropriate rejuvenations are conducted. The advantage of nearly no false alarms largely reduces the cost of unnecessary rejuvenations. And it could be used as a covariate in Condition Based Maintenance for software systems.

D. Comparison

Comparing with other software aging metrics, our extended multi-scale entropy metric can extract the character of each segment of original observations and counteract the noise brought by the inherent dynamic of software system. Hence it largely reduces false alarms and could be used in an offline state classification method.

1) Comparison with AR model

Lots of software aging detection methods based on the original continuously-observed performance time series prediction are proposed such as AR. But the noise in the original data due to the dynamic of software system makes the prediction result imprecise. Here we implement AR model to predict the observation 'Average Bandwidth Output Per Player' under workload w_2. If the prediction result before service unacceptable time point is lower than the valve at this time point, an alarm rises (See Figure 12). The upper part of Figure 12 demonstrates the original observation and AR prediction result and the lower part is the false alarm. Suppose 1200x15 second time point is the service unacceptable time point, 69 false alarms are reported before this time point. However there are no false alarms implementing our multi-scale entropy metric mentioned above. Therefore our method outperforms the previous prediction method based on auto regression.

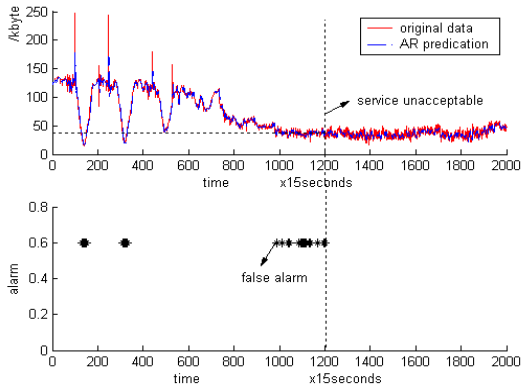


Figure 12 AR prediction and false alarms

2) Comparison with Hölder exponent

Hölder exponent as a software aging metric was proposed by Shereshevsky in literature [1]. He pointed out the Hölder exponent of the memory resources would decrease with system running. In order to compare with his conclusion, we use this metric to measure the original observations including '\Process (rmserver)\Working Set', '\Process(rmserver)\% User Time', 'Average Bandwidth Output Per Player(bps)', '\PhysicalDisk (_Total) \Avg.Disk QueueLength', under workload w_2. The upper part of Figure 13 is the Hölder exponent of the original observations. Apparently, the Hölder exponent decreases significantly after the service unacceptable time point, but it is still full of dynamic. In Shereshevsky's paper, he implemented Shewhart algorithm to detect the breakdown of time series. But this needs a long section of historical data to train and detect. Taking the Hölder exponent at service unacceptable time point as a referential threshold, 99 false alarms are reported through AR prediction before that time point, much larger than our metric. So our metric is more stable than Hölder exponent.

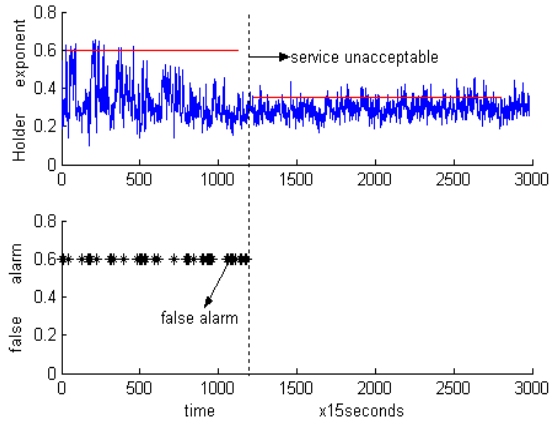


Figure 13 Hölder exponent metric and false alarms

3) Offline state classification

Another advantage of our metric is that it could be used as a state classification method offline. Suppose that if the multi-scale entropy value increases more than 0.3, software system steps into a new aging state otherwise it stays at previous state. Therefore the aging process is divided into several aging states (See Figure 14). At different states, appropriate software rejuvenation methods are conducted. For example when system is at state 2, software rejuvenation based on reconfiguration is conducted; when system steps into state 3, the rejuvenation based on hard reboot is conducted. So our metric is complementary with software rejuvenation based on Markov decision process.

Although our metric is effective to reduce false alarms of pseudo software aging, it may delay aging reports. Suppose that the software system can't provide acceptable service at time t, but the number of observation data collected from the system hasn't reached the length of data window at this time, our aging detection will not conducted and the aging report is delayed at most a length of data window. This problem doesn't exist in previous prediction methods based on AR or Hölder exponent. However, we say, the delay of aging reports in a non-real time system is tolerant.

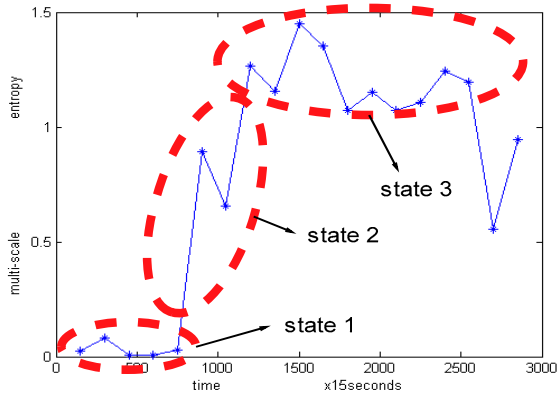


Figure 14 State classification based on multi-scale entropy

VI. CONCLUSION AND FUTURE WORK

To explore the fundamental measurement hidden in the dynamic of software aging process, this paper proposed a novel software aging metric named multidimensional multi-scale entropy which is an extension of classical multi-scale entropy. Through experiment analysis, we validate that MMSE as a software aging metric satisfies all the three properties: *stability*, *monotonicity* and *integrity* discussed in previous section. The aging detection based on our metric can largely reduce false alarms caused by pseudo aging compared with other aging prediction methods based on the continuously-observed performance data (e.g. AR model) or Hölder exponent. Another benefit of this metric is that it could be implemented in aging state classification, which is complementary with software rejuvenation based on Markov decision process.

The *monotonicity* of MMSE is not a sufficient property of software aging. In our future work, we will concentrate on the mechanism of software aging and try to find out the sufficient and necessary properties. And to implement this metric in different systems, we will add an adaptive data window regulation mechanism in the aging detection process.

ACKNOWLEDGMENT

This work is supported by the Fund: Research on Networked Software Aging Mode and Rejuvenation Approach (Grant No. 60933003) which is sponsored by the Key Project of National Natural Science Foundation of China.

REFERENCE

[1] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota and Y. Liu , "Software Aging and Multifractality of Memory Resources," in DSN'03, pp. 721 – 730, 22-25 June 2003.

[2] J. Alonso.; J. Torres.; J.L, Berral.; R, Gavalda, "Adaptive on-line software aging prediction based on Machine Learning", in DSN'10, pp. 507 – 516, June 28-July 1 2010

[3] K.J. Cassidy, K.C. Gross, A. Malekpour, "Advanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers," in DSN'02, pp. 478 – 482, December 10, 2002.

[4] C. Stewart and K. Shen, "Performance Modeling and System Management for Multi-component Online Services," in NSDI'05, pp. 71-84, Berkeley, CA, USA, 2005.

[5] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," inDigest of the 25th Annual International Symposium on Fault-Tolerant Computing, (Pasadena, CA), pp. 381−390, June 1995.

[6] M. Grottke, R, Matias, K.S. Trivedi, "The fundamentals of software aging", in ISSRE Wksp 2008, pp. 1-6, 11-14 Nov. 2008

[7] M, Grottke, L. Li, K. Vaidyanathan, and K.S. Trivedi, "Analysis of Software Aging in a Web Server", IEEE TRANSACTIONS ON RELIABILITY, VOL. 55, NO. 3, SEPTEMBER 2006, pp. 411 – 420.

[8] Y.J. Bao, X.B. Sun, and K.S. Trivedi, "A Workload-Based Analysis of Software Aging, and Rejuvenation," IEEE TRANSACTIONS ON RELIABILITY, VOL. 54, NO. 3, pp 541 – 548, SEPTEMBER 2005

[9] K.C. Gross ; V. Bhardwaj ; R. Bickford, "Proactive Detection of Software Aging Mechanisms in Performance Critical Computers," Proceedings. 27th Annual NASA Goddard/IEEE on Software Engineering Workshop, 2002, pp. 17 – 23, 5-6 Dec. 2002

[10] S. Garg, A. Puliafito, M. Telek, K S. Trivedi, "A Methodology for Detection and Estimation of Software Aging", Intl Symp. On Software Reliability Engineering, ISSRE 1998, Nov. 1998.

[11] D. Cotroneo, S. Orlando, S. Russo. "Characterizing Aging Phenomena of the Java Virtual Machine", in SRDS'07.pp:127-136,2007.

[12] Sm@rtPartner magazine September 18, 2000.

[13] H . Okamura, T . Dohi, "Application of Reinforcement Learning to Software Rejuvenation," Autonomous Decentralized Systems (ISADS), pp. 647 – 652, 23-27 March 2011.

[14] K. Vaidyanathan and K S. Trivedi, "A Comprehensive Model for Software Rejuvenation," IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 2, NO. 2, APRIL-JUNE 2005, pp. 124 – 137.

[15] K.S. Trivedi K. Vaidyanathan. "A measurement-based model for estimation of resource exhaustion in operational software systems". Proceedings of the 19th International Symposium on Software Reliabiltiy Engineering, 1998. pp. 84 – 93.

[16] R. Hasan, R. Burns, "The Life and Death of Unwanted Bits: Towards Proactive Waste Data Management in Digital Ecosystems." *CoRR,*2011, abs/1106.6062.

[17] P. Rocchi, "Calculations of System Aging through the Stochastic Entropy," Entropy 2006, 8, pp. 134-142.

[18] A. Oliner, J. Stearley, "What Supercomputers Say: A Study of Five System Logs," in DSN'07, pp. 575 – 584, 25-28 June 2007.

[19] V. Latora and M. Baranger, "Kolmogorov-Sinai Entropy Rate versus Physical Entropy," PHYSICAL REVIEW LETTERS, VOLUME 82, NUMBER 3, pp. 520.-523, 18 JANUARY 1999.

[20] P. Grassberger and I. Procaccia, "Estimation of the Kolmogorov entropy from a chaotic signal," Phys. Rev. A 28, 2591,1983.

[21] J.P. Eckmann and D. Ruelle, "Ergodic theory of chaos and strange attractors," Rev. Mod. Phys. 57, 617,1985.

[22] S. M. Pincus, I. M. Gladstone, and R. A. Ehrenkranz, J. Clin. "A regularity statistic for medical data analysis," Monit. 7, 335, pp. 335-345, 1991.

[23] S. Pincus, "Approximate entropy as a measure of system complexity," Proceedings of the National Academy of Sciences of the United States of America, vol. 88, no. 6, pp. 2297–2301, Mar. 1991.

[24] J. Richman and J. Moorman, "Physiological time-series analysis using approximate entropy and sample entropy," American Journal of Physiology- Heart and Circulatory Physiology, vol. 278, no. 6, pp. 2039–2049, 2000.

[25] Costa M,Goldberger A.L,Peng C.K. "Multi-scale entropy analysis of Complex Physiologic Time Series". PHYSICAL REVIEW LETTERS, 89(6):068102-1-4.,2002

[26] D E. Lake, J S. Richman, M P. Griffin, et al. "Sample entropy analysis of neonatal heart rate variability," Am J Physiol, 283(3) pp.789-797, 2002.