International Journal of Innovative
Computing, Information and Control
Volume 11, Number 5, October 2015

ICIC International ©2015 ISSN 1349-4198
pp. 1725–1738

DYNAMIC PERFORMANCE EVALUATION OF COMPUTING SYSTEM BASED ON GENE EXPRESSION PROGRAMMING THEORY

Jing You, Wei Qin, Hui Feng, Rong Huang and Jinglun ShangGuan

Faculty of Information Science and Engineering
Changzhou University
Gehu Road, Wujin District, Changzhou 213164, P. R. China
{youjing; qinwei; fenghui; huangrong; shangguan}@cczu.edu.cn

Received March 2015; revised July 2015

Abstract. *Software aging, with the growing software complexity, which can cause the degradation of performance and even software failure, has never been able to be avoided. In order to monitor the software aging of server, some researchers proposed different modeling methods, such as linear regression and auto-regressive and moving average (ARMA) model, to analyze and predict the performance. However, these models cannot express the combined effects of a variety of resources on system performance. In this paper, a dynamic modeling algorithm based on gene expression programming (GEP) theory is put forward. The GEP model can monitor, evaluate and predict the system performance by building the direct relationship between some performance index and multiple computing resources including the utilization of CPU and memory, I/O performance, network transmission, etc. In addition, the new GEP model can be adjusted to trace the performance trend and the running state of system by moving window strategy. Finally, three experiments are designed and the results demonstrate that the GEP model can express the overall trend with essential details, which will have higher precision and practicability than those of traditional models when the trend of data has complex changes, and the multi-stage GEP model can be established to analyze and predict the healthy state of system during the process of aging.*

Keywords: Performance evaluation, Gene expression programming, Dynamic monitoring, Software aging, Performance model

1. **Introduction.** Recently, software aging has been usually used to describe a common phenomenon, in which the computing system undergoes a performance degradation or even a failure with time [1,2]. Rebooting the failure system was simple but it might be expensive in terms of downtime, so a proactive technology called software rejuvenation was studied by many researchers to ensure the high availability of the system [3-5]. In order to choose the right rejuvenation strategy, different Markov models [2-5] were established to determine an appropriate restart interval to improve the system availability. However, for the unstable system, it is difficult to divide the system into different states, and other methods to evaluate the performance of computing system should be developed.

Studies had shown that performance degradation often accompanies a loss of system resources such as memory leaks and overflows, unreleased file locks, storage space fragmentation, the error accumulation, lack of swap space and network bandwidth [2,6]. Consequently, the performance of computing system can be evaluated by modeling the wastage of all kinds of system resources during aging process. The dynamic performance evaluation method proposed in the paper is used to solve the problem. Compared with the common modeling methods, such as time series analysis methods [1,6], which were proposed to analyze the changing trend of each kind of resource and predict the system performance,

the new method can establish the direct relationship between system performance and multiple resources utilization simultaneously.

If we trace the recession degree of the system performance, it always takes a long time, several months or even years, to collect the effective data of the system resource wastage [6,7]. At the same time, some accidental interference may appear during the process. In addition, different server and complex running circumstances may cause imprecise data. Hence, aging simulation is an effective way to get data and evaluate performance. In the paper, a general simulation environment is built to collect the performance and resources data.

The outline of our paper is as follows. Section 2 provides a survey of related work. In Section 3, the corresponding relations between performance evaluation of computing system and gene expression programming (GEP) algorithm are analyzed. In Section 4, a new modeling method, dynamic gene expression programming method, is proposed to determine the combined effects of a variety of resources and evaluate the performance trend during aging process. Section 5 introduces the distributed simulation environment. In Section 6, three experiments are designed to compare the effects of different modeling methods and establish multi-stage GEP model of aging process. Finally, the conclusion is given in Section 7.

2. **Related Work.**

Software rejuvenation is a proactive technique [3,8] based on reboot systematically which can prevent unexpected or unplanned outages due to software aging. This method has also been applied to software system in a wireless sensor network [9] and a server virtualized system [5,10] besides traditional service systems [2,6]. How to determine an appropriate restart time depends on the running state of system. For a stable system, many models such as Petri nets and Markov were proposed to model the states of system and find the restart interval and threshold. Huang et al. [3] proposed a continuous-time Markov chain model and Dohi et al. [11,12] proposed a semi-Markov model to calculate the optimal software rejuvenation schedule. Zhao et al. [13] also applied semi-Markov model and non-parametric method to optimize the trigger interval. However, for unstable systems, we need to monitor and release system resources in real time so as to keep system performance. Based on the previous methods, the latest research by Santos et al. [14] explored the best restart time through searching the code autonomously via multi-agent.

Before software rejuvenation, the system resource data should be collected and analyzed whether the system is stable or unstable. Garg et al. [1] proposed a methodology for detection and estimation of aging in the UNIX operating system. They developed a distributed monitoring tool based on SNMP to collect operating system resource usage and system activity data at regular intervals from networked UNIX workstations. Sysmon [15] was built to monitor the web server's environment including the resources of operating system and Apache's daemon processes, such as physical and shared memory usage. All the data were collected from several files in the Linux/proc directory. Supermon [16] used remote state agreement of SunRPC to collect data from different computing nodes. The protocol was based on the symbols expression so that the nodes could communicate with each other in a heterogeneous environment, but the data collected could only stay in the central database, which did not meet the scalability. Dproc [17] was a system monitor tool based on Linux kernel. It supported a custom monitoring and the information could be filtered for reducing communication load. With reference to the above methods, an adaptive resource monitor was designed by our team based on Linux platform [18]. It could collect the specified resources data and adjust automatically the sampling interval. And then these data could be stored and analyzed using a suitable modeling method.

The difficulty of data collection is the reproduction of aging process, for which the system has to run for a long time and try to avoid non-aging failures. Garg et al. [1] collected the data every fifteen minutes and the data acquisition work lasted approximately 53 days. In their further studies, the same experimental setup and monitoring tools were used [8,19]. Others reported experiments utilized in software aging studies covered 600 hours [20], 2160 hours [21], and 6480 hours [22]. During the period, the system often suffered unexpected outage rather than software aging. Many software aging experiments found in the literature were terminated before the observation of system failures for aging. In Garg's experiment, only three machines did not experience any outage, and the other four machines suffered unexpected outage. In order to gather the effective resources data that can reflect the recession rule of specific systems as soon as possible, it is necessary to design a reasonable simulation system to simulate the loads and the running circumstance of servers. In our simulation system, "aging factor" put forward by Matias et al. [23] can be used together with the degradation test method to accelerate software aging and control the aging effects at the experimental level.

After data collection, we need to analyze and model the resources data for determining the running state of system and restart time. Generally, the unary linear regression model was used to estimate the entirety trend of single resource. Moreover, the multiple-linear regression model was intended to estimate the different resources on system performances changes. However, Shereshevsky et al. [24] criticized the linear models and they thought that the approach was not adequate if a large increase in the confidence intervals was observed in periods or the studied variables showed a non-linear behavior. Grottke et al. [25] proposed ARMA model to portray software recession and the consumption of resources because it could describe the details of resources' changes. Another study of software aging in an Apache web server was presented and it also used time series ARMA models to identify, characterize, and estimate the aging effects [26]. State space model was also used to model the change rule of system resources [27]. All these methods mentioned above can model the resource separately to analyze the influence factors, but it was difficult to predict the system performance that was affected by a variety of resources simultaneously, which is the reason that the dynamic GEP model is proposed in the paper. GEP method has been used to analyze the complicated relationship among multiple factors in different fields such as stock investment [28] and electricity demand [29].

3. **Performance Evaluation of Computing System and GEP Algorithm.** The majority of computing system may be affected by a variety of resources, so a new method is needed to make an effective evaluation of computing system performance. Its modeling results can be used to predict the performance recession of computing system and determine the restart interval or threshold value. The method should take various influence factors into account simultaneously and then build a direct relationship between system performance and influence factors. GEP model just meets the demands. GEP uses isometric linear symbol as genetic coding to analyze the combined effect of multiple factors on a particular element. The corresponding relations between performance evaluation of computing system and GEP algorithm are shown in Table 1.

Table 1 shows that the influence factors of system performance are expressed as the end symbol set in GEP theory, and the fitting operations applied to these factors are expressed as the function symbol set. A series of end symbols and function symbols constitutes one action rule of computing system, which is expressed as a gene in biological system. Each chromosome consists of one or more genes, that is, various rules can exert a comprehensive influence on computing system. A chromosome can be converted to an expression tree, and the mathematical model representing the performance of computing system can be

TABLE 1. Correspondence between performance evaluation of computing systems and gene expression programming algorithm

Performance Evaluation of Computing System	Gene Expression Programming Algorithm
Computing System	Biological System
Computing System Performance	Biological System Health Degree
Rules Combination	Population
Rules Search	Genetic Operators
One/Multiple Rules	Gene/Chromosome
Mathematical Model	Expression Tree
Influence Factors	End Symbol Set
Fitting Operations	Function Symbol Set
Model Accuracy	Fitness

obtained by inorder traversing the expression tree. Multiple genetic operators are designed in GEP algorithm to search the rules of computing system according to the chromosome's fitness. The higher fitness a chromosome has, the higher accuracy its mathematical model has. In order to ensure the legitimacy of expression trees, namely to ensure the availability of model, each gene must be composed of head and tail, and they have to meet some formation rules. Not all parts of the gene will be manifested in the expression tree. However, the part of gene which is not manifested equally involved in the subsequent operation of genetic operator, and it may eventually produce the highest fitness. This kind of evolution mode also can ensure global rules search.

4. Dynamic GEP Model of Aging Process.

4.1. Traditional models.

4.1.1. *Unary linear regression model.* In order to estimate the general trend of single resource, we can use the unary linear regression model which is simple and intuitive, and also has been used in lots of fields. If the system runtime is as variable x and the usage of resource is as variable Y, the model is $Y = a + bx + \varepsilon$, $\varepsilon \sim \mathrm{N}(0, \sigma^2)$. The variables a, b and σ^2 are not dependent on the unknown parameters of x. The variable b is called regression coefficient. The expression shows that the variable Y consists of two parts: one part $a + bx$ is the linear function of x, and the other part is random error ε which is uncontrolled.

4.1.2. *ARMA model.* ARMA (Auto Regressive and Moving Average, ARMA) model can be shown by the formula $X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdots - \theta_q \varepsilon_{t-q}$. In the formula, p and q are respectively the number of auto regressive ranks and moving average ranks. θ and ϕ are non-zero pending coefficients. ε_t is an independent error item. X_t is a stable, normal and zero-mean time sequences. ARMA model is the integration formula of AR (p) and MA (q). In the process of running the data model, we need to calculate the coefficients of ACF and PACF, and then identify the model according to the characteristics in Table 2, and finally establish the model configuration about the gathered data.

TABLE 2. Identification of model characteristics

Model	AR(p)	MA(q)	ARMA(p, q)
ACF	trailing	truncation	trailing
PACF	truncation	trailing	trailing

4.2. **GEP model based on user feeling.** The quality of service refers to many aspects including reliability, safety and continuity. However, users do not want to know the exact value of all these indicators, their direct service experience for the system performance is the response time of the requests which is affected by many factors. Therefore, a mathematical model based on GEP theory [30], which is called GEP model, is proposed to establish the relationship between response time and a variety of system resources and evaluate the computing system performance.

4.2.1. *Chromosome.* Each chromosome contains one or more genes, and each gene has two parts: the head and the tail. The head is composed of function symbols or end symbols. The first position of a gene must be a function symbol, and the tail must be made of end symbols. The length of head (h) must be first determined, the length of tail is defined as t, and the relationship of them is as follows:

$$t = h(n-1) + 1 \tag{1}$$

The n in the formula is defined as the number of the function parameters with the most variables. For example, we use the function symbols $\{+, -, *, /\}$, and the end symbols $\{a, b, c, d\}$. Now we have a chromosome A, $+/-abcd|-+-aaba|-+-bcaa$, which contains three genes. The length of each gene's head is 3 and the length of its tail is 4. Supposed the connector between genes is symbol $+$, the corresponding expression tree of chromosome A is shown in Figure 1. Then we perform inorder traversal for the tree and obtain the arithmetic expression $((a/b)+(c-d))+((a+a)-(b-a))+((b+c)-(a-a))$, and the result is $a/b+3a+2c-d$.

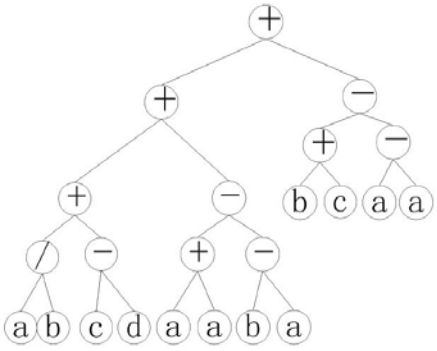

FIGURE 1. Expression tree of chromosome A

In this paper, the adaptive performance monitoring tool developed in the previous research is used to collect various parameters from the running computing system. According to preliminary analysis and empirical data, we determine the performance-related parameters, which are defined as the end symbols. Thus, the identified chromosome will reflect their combined effects on system performance.

4.2.2. *Evolution operator and evaluation function.* In the process of evolution of chromosomes, as long as the conditions that are mentioned above have been met, the offspring chromosomes are still valid. GEP algorithm has the same general choice, mutation and recombination as the genetic algorithm, but it also has some special operation such as interpolation operation, root interpolation operation, and gene recombination operation. We can check Ferreira's book [30] for these operations.

Before evolution, the fitness value of each chromosome should be calculated, and the formula is

$$fv_i = \sum_{j=0}^{l} (M - |C_{ij} - T_j|) \tag{2}$$

fv_i is the fitness value of the chromosome, M is a constant variable, C_{ij} is the calculated value of the i-th chromosome gene expression with the j sample value, T_j is the actual value of the j-th sample, and l is the number of samples.

4.2.3. *Algorithm description.* Algorithm is described as follows.

(1) Set the end symbols, function symbols, the fitness value of chromosomes, and the maximum number of iterations.

(2) According to the set of function symbols and the end symbols, generate 50 chromosomes by random and then initialize them. Calculate the fitness value of each chromosome, and stop the operation and output the chromosome as a result if the chromosome reaches the required fitness.

(3) Implement the single-point recombination, two-point recombination, gene recombination operation, interpolation operation, root interpolation operation and gene interpolation operation according to predefined rates.

(4) Calculate the fitness value of each chromosome, and stop operation and output the chromosome as a result if the chromosome reaches the required fitness.

(5) Sort all chromosomes by their fitness, and then choose the top 50 chromosomes as a new population to produce the next generation.

(6) Stop and output the chromosome with maximum fitness as a result if it reaches the maximum number of iterations; otherwise jump to step (3) and continue running.

4.3. **Multi-stage dynamic GEP model.** Due to software aging, the performance variation rule is different in different running stages. Therefore, a dynamic model with moving window is put forward to make continuous monitoring and dynamic evaluation of the performance. The method sets the single window of time domain as ΔT, and then builds the GEP model for all kinds of resources of three windows. The last model is applied to predict the data of next window, and then the forecast accuracy is analyzed to decide whether the model is corrected for the next window.

In the algorithm, P_0 is the initial population, and P represents the last population at the end of the model search. k is the number of iterations and the initial value is 0, and $k\max$ is the maximum value of iterations. w is the counter of windows and its initial value is 0, and W is the maximum value of counter. The chromosome fitness is calculated according to Formula (2).

Predefine Fv as the threshold of fitness and $Fv = \eta Ml$, η is model accuracy and generally greater than or equal to 80%. Algorithm is described as follows.

(1) Generate n chromosomes by random and then initialize the population if $k = 0$ and $w = 0$.

(2) Decode chromosome and calculate the fitness value for the window of $w+1 \sim w+3$.

(3) If $fv_i \geq Fv$ or $k \geq k\max$, the current round of model search and the iteration is ended. Set $w = w + 3$, record current chromosomes, output the value of w, $\max(fv_i)$, chromosomes and mathematical model, and then jump to step (7).

(4) Implement the single-point recombination, two-point recombination, gene recombination operation, interpolation operation, root interpolation operation, gene interpolation operation.

(5) Set $k = k + 1$, decode chromosome, and calculate the fitness value for the window of $w + 1 \sim w + 3$.

(6) Sort chromosomes by their fitness and set top n better chromosome as the new population, which are used to produce the next generation, and jump to step (3).

(7) Calculate the fitness value of the forecast window $(w+1)$ according to the mathematical model if $w \leq W$.

(8) If $fv \geq Fv/3$, set $w = w+1$, continue the cycle of the model to predict the value, and then jump to step (7).

(9) Set current population as initial population, that is $P_0 = P$, $k = 0$ and $w = w-2$, then jump to step (2) to start a new round of model search and correct the original model.

5. **Simulation Environment Based on Linux Platform.** The experimental simulation environment is shown in Figure 2. The whole environment is divided into three parts: server simulation environment, load simulation environment and system resources monitoring environment.

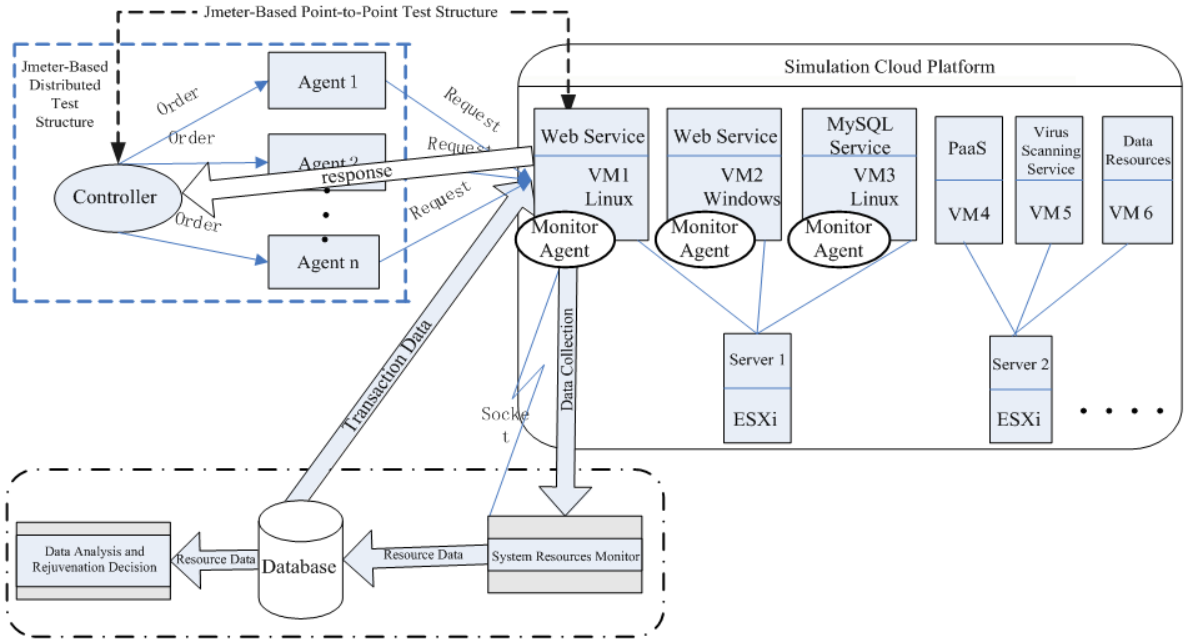

FIGURE 2. Aging simulation environment

The right part with the solid box is simulation environment of server based on cloud platform. Some common cloud computing applications are built based on VMware ESXi, such as web services, database services, and virus scanning service. Among them, two virtual machines provide web services, and they will be tested and monitored as an enterprise application system under cloud mode. One virtual machine provides web service by configuring Apache and Tomcat based on Linux platform. Apache takes charge of the static web-page of web services and only deals with simple browsing. Tomcat takes charge of the dynamic web-page and deals with interactive and database operation tasks. The other virtual machine can provide different web services by configuring IIS based on Windows platform.

The left part with the dashed box is JMeter load simulation environment. This environment adopts two kinds of test structure. One is a distributed structure and the other is a point-to-point structure. The distributed structure needs to configure several PC machines in LAN. One of them is used as controller, and the others are used as load generators. The controller controls the load generators to send HTTP requests to the server, and then the response results are sent back to the controller. The controller can

perceive the performance changes of servers, modify the parameter setting of the load generators and regulate the system load. The point-to-point structure is simple, as shown in black dotted line. It only needs a machine as a load generator to send a large number of requests to the server. Take the Web server based Linux as example. If we only visit static web-pages, the workload of server is light, so the distributed test structure is demanded to generate heavy loads for changing the performance of server and the quality of service in an appropriate range. If we visit dynamic web-pages, the workload of server is heavy, so point-to-point test structure is enough to generate heavy loads and ensure that the server is able to run smoothly without downtime.

The performance of computing system is often affected by many factors during running time. These factors fall into three categories: one is the usage condition of operation system resources, such as CPU utilization rate, memory utilization rate, and the size of usable exchange space; another is the usage condition of network resource, such as throughput, bandwidth, and frame loss rate; the third one is the usage condition of storage resource, such as utilization rate of the queue, and input and output. This experiment is based on plenty networks and storage resources. Therefore, we only need to monitor the operation system resources.

The marquee part in the below is the system resources monitoring environment. In order to minimize the resource consumption of the monitoring system itself, we configure another machine as the resource monitor. The resource monitor is designed by our team based on Linux [18]. It can achieve collection, storage, analysis and modeling of the resources data. The communication between resources monitor and web server is set up by socket protocol. During the running process, the resources monitor picks up periodically the resource usage information from the /proc virtual document system after configuring collection time intervals and resource parameters, and then saves the collected data to the document or database for following modeling. Furthermore, the monitor can adjust the sampling intervals automatically based on the actual change of collected data.

In addition, the performance data also need to be collected. What the end users want most is a safe service and timely response, so the response time of server can be used to reflect software performance in this experiment. The response time covers the time that starts from the client sending the request to the server and ends to the server returning the response to the client. In general, the response time includes front-end and back-end. The front-end is network latency, and back-end time is processing time of servers (for instance, Web Server, Application Server, and Database Server dealing with the requests). Because our experiment is carried out in local area network, the network latency can be ignored and the response time in our experiment equals the back-end time. And then it can be used to model the relationship between system performance and resources. In the experiment, we record the current system time when we send the requests to the server as T1 and the time when the server has finished processing the requests as T2, and then we can get the system response time T = T2 – T1.

6. **Experiment.**

In this section, three experiments are designed to test and verify the effects of GEP model and dynamic modeling method.

6.1. **Setting of experimental parameters.**

The environmental parameters of simulation cloud platform are shown in Table 3.

The experiment parameters of load simulation environment are shown in Table 4. *MaxClients* and *MaxRequestPerChild* are two significant parameters on the server side: the former sets the number of max clients that can connect to the server simultaneously and the latter sets the maximum number of requests for each sub-process can respond. If

TABLE 3. Experimental environmental parameters

	Test System		Web Server	Resource Monitor System
	Control	Agent1/2/3		
Operating System	*windows* 7	*windows* XP	*CentOS* 5.5	*windows* XP
Memory	*DDR*2 4G	*DDR*2 1G	*DDR*2 1G	*DDR*2 1G
Processor	*Pentium*(R) *Dual-Core CPU* E5300@2.60GHz	*Intel* (R) *Celeron* (R) *CPU* E3200@2.4GHz	*Intel*(R) *Xeon*(R) *CPU* E5405@2.00GHz	*Intel Core*2 *Duo* E4500@2.20GHz

TABLE 4. Load simulation parameters

MaxClients	*MaxRequestPerChild*	*Collection Resource Data*	*Threads*	*Ramp-Up Period*	*Cycle Times*
1024	0	900	3000	0.2	*Forever*
1024	0	3519	5000	0.1	*Forever*

TABLE 5. Algorithm parameters

n	Gene Number	h	W	T	M	η	l	$k_{\max}$	Variation Rates	Interpolation /Root Interpolation Rates	Single /Two-Point Recombination Rates	Gene Recombination /Translocation Rates
50	4	5	35	25 min	10	0.94/0.96	900/300	100	0.1	0.1/0.2	0.3	0.02

MaxRequestsPerChild is "0", the child process will never end. The other parameters in Table 4 are set at the client side, where the parameter *Ramp-Up Period* is set for avoiding too heavy loads due to the simultaneous requests from lots of threads. All the threads are not sent continually until the completion of data collection.

In resource monitor, we can customize the relevant parameters by an inspecting controller, such as monitored nodes, collected resource types, and sample interval. In our experiment, the collected resources data include CPU and memory information, based on which we will calculate CPU and memory utilization of system.

The most parameters of GEP algorithm are shown in Table 5. In addition, GEP model uses function symbol set $\sum = \{+, -, *, s, e\}$, in which s is sin function, and e is exponential function. The termination symbol is $\{0, 1, 2\}$, in which 0 indicates CPU utilization rate, 1 indicates memory utilization rate, 2 indicates the sampling time, and the response time is our target value.

6.2. **Experiment one: comparison of GEP model and traditional models.** GEP algorithm is proposed to model comprehensive relation between response time and multiple factors. However, like traditional modeling methods, GEP method can also be used for single resource. In this experiment the effect of GEP model will be compared with those of traditional models.

This experiment uses the distributed test structure, and the parameters of experimental environment are shown in Table 3. The load simulation parameters are shown in the first row of Table 4. The algorithm parameters are shown in Table 5, and in this experiment η is 0.94 and the number of samples l is 900, so Fv equals 8460.

We gather the CPU and memory information and calculate their utilization, and then we establish respectively linear regression model, ARMA model and GEP model for two data sets. The modeling results are shown in Figure 3 and Figure 4. We can see that

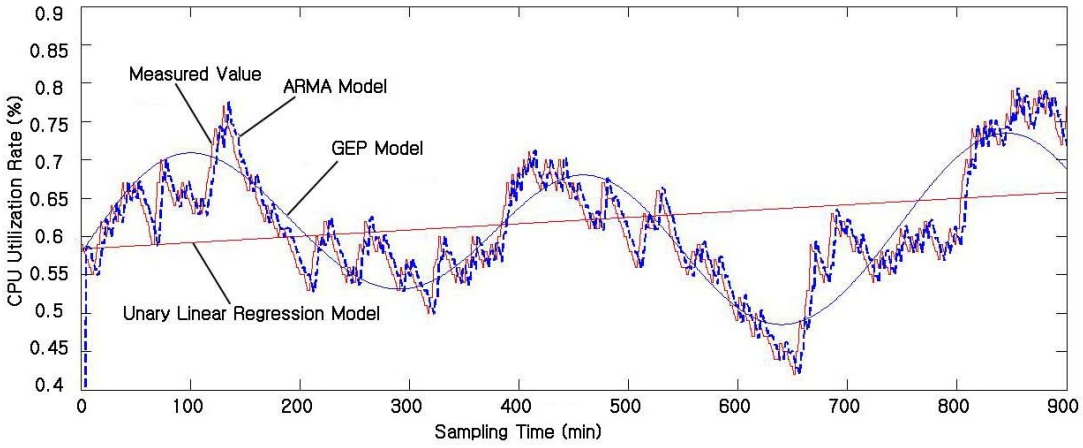

FIGURE 3. Comparison chart of three models on CPU utilization rate

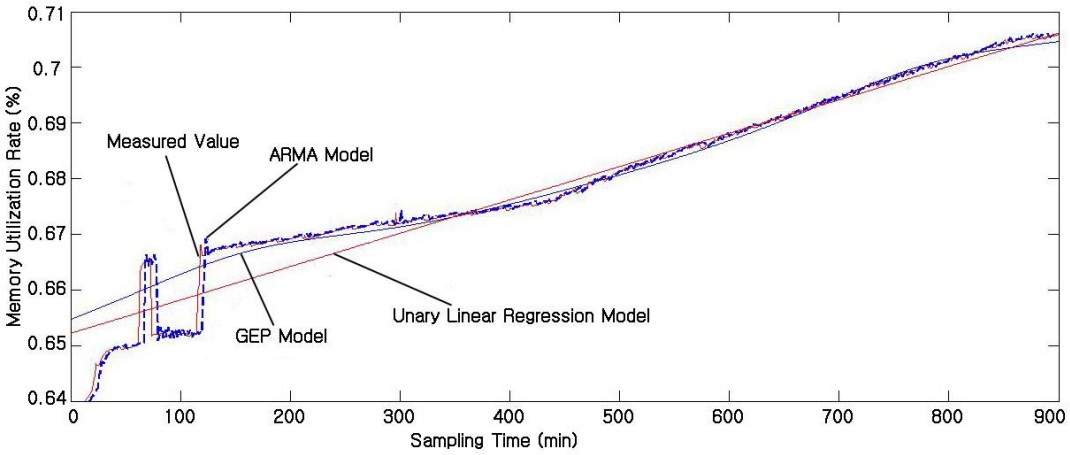

FIGURE 4. Comparison chart of three models on memory utilization rate

from the diagram, the linear regression model simulates the general trend of CPU and memory utilization rate. This modeling method is quite simple but it does not express the changing trend in detail, so the difference between the actual value and function value is too large. The ARMA model and GEP model are concerned about the overall trend with essential details, which will be very helpful for us to design reasonable control strategy.

6.3. **Experiment two: comparison of two GEP models.** In this experiment, we will compare two GEP models, $R(t)$ and $R(Cpu, Mem, t)$. Model I ($R(t)$) shows the direct relationship between response time (R) and sampling time (t). And Model II ($R(Cpu, Mem, t)$) takes the influence of CPU utilization (Cpu), memory utilization (Mem) and sampling time (t) to response time (R) into account simultaneously.

For Model I, the average evolution generation number of algorithm is 235.8 generations. And then we select a good chromosome B which can get a high fitness value, $-ss1s102002*ss0*212222-22-2222212**/002e1212$, whose expression tree is shown in Figure 5(a). According to the expression tree, we set the utilization of CPU and memory as a constant 1, and then we can obtain the mathematical model $R(t) = 0.0171478+0.0174524\sin t+t/e$. The fitness of model I is 8546.911.

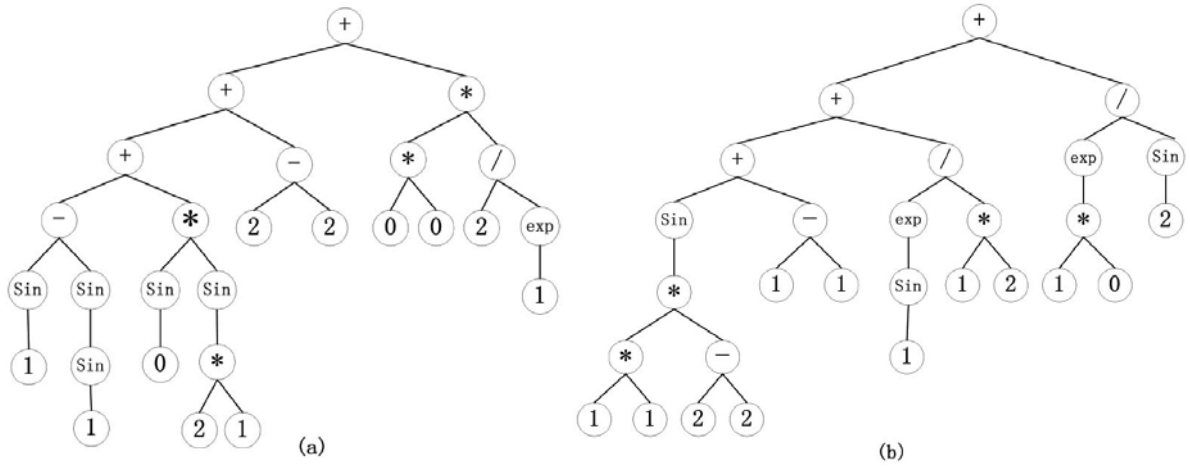

FIGURE 5. Expression tree of chromosome

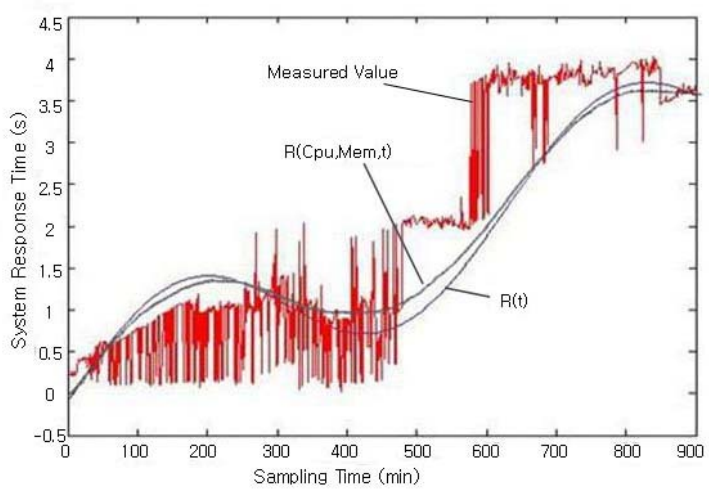

FIGURE 6. Comparison chart of two GEP models

For Model II, the average evolution number of algorithm is 18.2 generations. Likewise, we select a good chromosome C, $s*-*2211010-11*2020101/*e21s12222/se2*011112$, whose expression tree is shown in Figure 5(b). According to the expression tree, we can obtain the mathematical model $R(Cpu, Mem, t) = \frac{Mem*t}{e^{\sin(Mem)}} + \frac{\sin t}{e^{Cpu*Mem}}$. The fitness of Model II is 8551.21.

Figure 6 shows us the comparison chart of two models for all samples. From the above results and function curves, it can be seen that the results of two models are very similar but the evolution speed of Model II is much faster than that of Model I. Furthermore, the accuracy of Model I is a bit less than that of Model II. The main reason is that the response time may be affected by many factors, such as the system's resources, and network resources. CPU, Memory, and the sampling time are used simultaneously to model $R(Cpu, Mem, t)$, so the modeling speed will be improved, and the accuracy of the model will also be enhanced.

6.4. **Experiment three: multi-stage dynamic GEP model.** To study the changing trend of resource depletion and performance degradation during the process of software aging, the virtual servers need to deal with continuously heavy load for a long time. In

this experiment, the point-to-point test structure and the test of dynamic JSP pages are selected, so only one client called controller is needed to send the simulation load to the server. The load parameters are shown in the second row of Table 4. The parameters of algorithm are shown in Table 5, where the precision of algorithm η is 0.96, the number of samples l is 300 and $Fv = \eta Ml = 2877$.

The experiment collects 3519 group data, which takes 14.7 hours. Firstly, we select the data in first three windows to model and obtain the chromosome $/*s12-20122s*es2202122*s*22e21021/se1*012201$ after 18 generations. After that through expression tree the chromosome is transferred to the GEP model, $m_1 = \frac{b*c}{\sin(c-a)} + \sin(e^c * \sin(c)) + \sin(c)*c*e^c + \frac{\sin(b)}{e^{a*b}}$, whose fitness is 2920.6472. And then the model is tested and verified by the data in the next window to determine whether the results meet the fitness. According to the algorithm we build and check the model using moving window. This experiment has 35 windows, and finally we obtain four different chromosomes, which express four mathematical models. The results are shown in Table 6.

TABLE 6. Modeling results of moving window during the process of software aging

Window Numbers	Chromosome	Model	Fitness Value
1-3	$/*s12-20122s*es2202122$ $*s*22e21021/se1*012201$	$m_1 = \frac{b*c}{\sin(c-a)} + \sin(e^c * \sin(c))$ $+\sin(c) * c * e^c + \frac{\sin(b)}{e^{a*b}}$	2920.6472
4-11	$sss*1222222-se2-212202$ $/ss2s221220-++222s1000$	$m_2 = \sin(\sin(\sin(b*c))) + \sin(c) - e^{c-b}$ $+\frac{\sin(c)}{\sin(\sin c)} + c - \sin(b)$	2923.733
12-20	$+**212*0220see-1222222$ $*s-10s02012*-*012e2112$	$m_3 = (c*b) + (c^2*a) + \sin\left(e^{e^{b-c}}\right)$ $+\sin(b) * (a - \sin(a)) + (a-b) * (c * e^c)$	2885.8176
21-35	$/e/20-12220e/-e2001120$ $-e*2s122210+*-s1222110$	$m_4 = \frac{e^c*(b-c)}{a} + e^{\frac{c-a}{e^a}} + e^c$	2908.9302

Figure 7 shows the comparison of actual response time and predictive response time. The solid line in the figure is the predictive value of the system response time.

In the figure, it has been shown that the process of software aging can be divided into multiple stages, which means that the dynamic GEP algorithm with moving window can establish more reasonable multi-stage model with the optimal fitness value. The system in a strong condition (1-11 windows) in the beginning enters into possible failure state (12-20 windows). When the curve of the response time of the system becomes concave function (21-35 windows), the system can be regarded as a quasi-failure state. At this time, we can choose a suitable chance to perform the restart operation for recovering the system's performance and ensuring its availability before the system shuts down.

7. **Conclusion.** The software aging is a sustaining and pervasive phenomenon along with the increase of running time of service system. This article researches a universal method to simulate aging process of the server, gathers the resource data and models the changing trend of performance. This new GEP algorithm can be used to model not only simple relation between response time and single resource but complicated relation between response time and multiple factors as well. Specially, it shows us higher precision and practicability than traditional model because it can express the overall trend with essential details. In order to model the aging process of service system, the dynamic GEP algorithm with moving window is designed and multi-stage GEP model is established to predict the change of performance and further the failure of system.

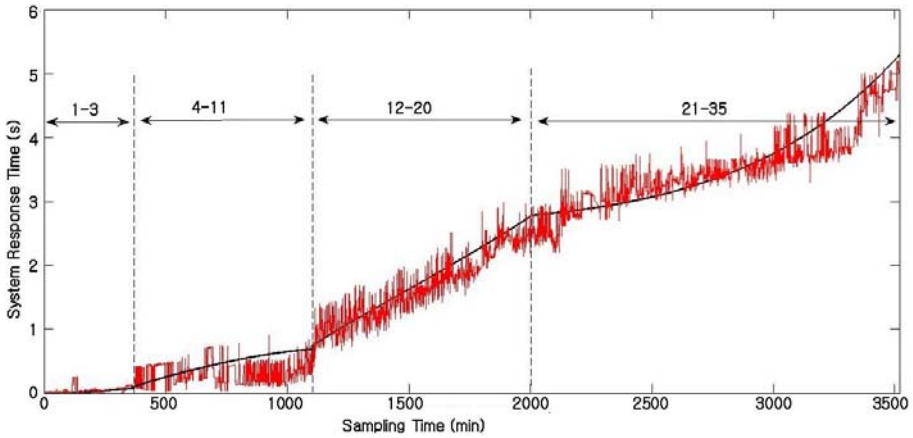

FIGURE 7. Comparisons chart of actual response time and predictive response time

There are some points needing to explain. First, the experiment is completed in LAN environment, so the impact of network transmission may be ignored. If the servers and clients run in WAN, the response time should include both the processing time of server and the transmission delay of network. Second, the response time as the only performance data in this paper is monitored and evaluated. If necessary, the system throughput which is another important indicator can also be measured to evaluate the system performance. Third, adjusting the value of η can help us to capture the state change of system. A real computing system will show the decline of performance only when it runs continuously for a long time. At this time the GEP model will have a greater change than before for the change of running state of system. Therefore, if we want to capture the state change quickly, we only need to reduce the value of η, which means that the subtle changes of performance are ignored. Finally, the resources monitor designed by the task group can achieve customized and self-adapting collection of the resource data, so we can reduce the amount of data collection and ensure the validity of data by opening the adaptive acquisition function.

Acknowledgments. This work is partially supported by Natural Science Foundation of Jiangsu Province (BK2009535) and Changzhou City Youth Talent Fund (CQ20100007). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] S. Garg, A. van Moorsel, K. Vaidyanathan and K. S. Trivedi, A methodology for detection and estimation of software aging, *Proc. of the 9th International Symposium on Software Reliability Engineering*, Paderborn, Germany, pp.283-292, 1998.

[2] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan and W. P. Zeggert, Proactive management of software aging, *IBM Journal of Research and Development*, vol.45, no.2, pp.311-332, 2001.

[3] Y. Huang, C. Kintala, N. Kolettis and N. D. Fulton, Software rejuvenation: Analysis, module and applications, *Proc. of the 25th Symposium on Fault Tolerant Computer System*, Pasadena, CA, pp.381-390, 1995.

[4] W. Xie, Y. Hong and K. S. Trivedi, Software rejuvenation policies for cluster systems under varying workload, *Proc. of the 10th International Pacific Rim Dependable Computing Symposium*, Papeete, Tahiti, pp.122-129, 2004.

[5] F. Machida, D. S. Kim and K. S. Trivedi, Modeling and analysis of software rejuvenation in a server virtualized system with live VM migration, *Performance Evaluation*, vol.70, no.3, pp.212-230, 2013.

[6] L. Li, K. Vaidyanathan and K. S. Trivedi, An approach for estimation of software aging in a web server, *Intl. Symposium on Empirical Software Engineering*, Nara, Japan, pp.91-100, 2002.
[7] K. Vaidyanathan and K. S. Trivedi, A comprehensive model for software rejuvenation, *IEEE Transactions on Dependable and Secure Computing*, vol.2, no.2, pp.124-137, 2005.
[8] K. S. Trivedi, K. Vaidyanathan and K. Goseva-Popstojanova, Modeling and analysis of software aging and rejuvenation, *Proc. of the IEEE Annual Simulation Symposium*, Washington, DC, pp.270-279, 2000.
[9] M. Woehrle, A. Meier and K. Langendoen, On the potential of software rejuvenation for long-running sensor network deployments, *Proc. of the 2010 ICSE Workshop on Software Engineering for Sensor Network Applications*, New York, NY, pp.44-48, 2010.
[10] L. Moura Silva, J. Alonso, P. Silva and J. Torres, Using virtualization to improve software rejuvenation, *IEEE Transactions on Computers*, vol.58, no.11, pp.1525-1538, 2009.
[11] T. Dohi, K. Goseva-Popstojanova and K. S. Trivedi, Estimating software rejuvenation schedule in high assurance systems, *The Computer Journal*, vol.44, no.6, pp.473-485, 2001.
[12] T. Dohi, H. Suzuki and K. S. Trivedi, Comparing software rejuvenation policies under different dependability measures, *IEICE Transactions on Information and Systems*, pp.2078-2085, 2004.
[13] J. Zhao, Y. B. Wang, G. R. Ning and K. S. Trivedi, A comprehensive approach to optimal software rejuvenation, *Performance Evaluation*, vol.70, no.11, pp.917-933, 2013.
[14] H. Santos, J. F. Pimentel, V. T. D. Silva and L. Murta, Software rejuvenation via a multi-agent approach, *The Journal of Systems and Software*, vol.104, no.6, pp.41-59, 2015.
[15] R. Matias, P. A. Barbetta, K. S. Trivedi and P. J. F. Filho, Accelerated degradation tests applied to software aging experiments, *IEEE Transactions on Reliability*, vol.59, no.1, pp.102-114, 2010.
[16] M. G. Sottile and R. G. Minnich, Supermon: A high speed clus termonitoring system, *Proc. of IEEE Int'l Conference on Cluster Computing*, Chicago, pp.39-46, 2001.
[17] S. Agarwala, C. Poellabauer, K. Jiantao and K. Schwan, Resource aware stream management with the customizable Dproc distributed monitoring mechanisms, *Proc. of the 12th IEEE International Symposium on High Performance Distributed Computing*, Washington, DC, 2003.
[18] J. You, K. N. Xu and H. Y. Wang, Design of distributed and adaptive performance monitoring system based on software rejuvenation, *Journal of Computer Application*, vol.30, no.6, pp.1642-1654, 2010.
[19] K. Vaidyanathan and K. S. Trivedi, A measurement-based model for estimation of resource exhaustion in operational software systems, *Proc. of ISSRE 1999*, Boca Raton, FL, pp.84-93, 1999.
[20] 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, pp.411-420, 2006.
[21] W. B. Nelson, *Accelerated Testing: Statistical Method, Test Plans, and Data Analysis*, Wiley, New Jersey, 2004.
[22] A. Avritzer and E. J. Weyuker, Monitoring smoothly degrading systems for increased dependability, *Empirical Software Engineering*, vol.2, no.1, pp.59-77, 1997.
[23] R. Matias, K. S. Trivedi and P. R. M. Maciel, Using accelerated life tests to estimate time to software aging failure, *IEEE the 21st International Symposium on Software Reliability Engineering*, San Jose, CA, pp.211-219, 2010.
[24] M. Shereshevsky, B. Cukic, J. Crowel, V. Gandikota and Y. Liu, Software aging and multifractality of memory resources, *Proc. of International Conference on Dependable Systems and Networks*, San Francisco, CA, pp.721-730, 2003.
[25] 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, pp.411-420, 2006.
[26] M. Grottke and K. S. Trivedi, Fighting bugs: Remove, retry, replicate, and rejuvenate, *Computer*, vol.40, no.2, pp.107-109, 2007.
[27] J. You, R. Shi, Y. Q. Sun and H. Y. Wang, The relationship research between usage of resource and performance of computer system, *WRI World Congress on Software Engineering*, Xiamen, China, pp.451-455, 2009.
[28] C. H. Lee, C. B. Yang and H. H. Chen, Taiwan stock investment with gene expression programming, *Procedia Computer Science*, vol.35, pp.137-146, 2014.
[29] S. M. Mousavi, E. S. Mostafavi and F. Hosseinpour, Gene expression programming as a basis for new generation of electricity demand prediction models, *Computers & Industrial Engineering*, vol.74, pp.120-128, 2014.
[30] C. Ferreira, *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*, Berlin, Springer-Verlag, 2006.