Reliability Analysis for Software Cluster Systems based on Proportional Hazard Model

Chunyan Hou[1], Chen Chen[2,*], Jinsong Wang[1], Kai Shi[1]
[1]School of Computer and Communication Engineering, Tianjin University of Technology, Tianjin, China
[2]College of Computer and Control Engineering, Nankai University, Tianjin, China
chunyanhou@163.com, nkchenchen@nankai.edu.cn, {jswang, shikai}@tjut.edu.cn

***Abstract*—With the universal application of software cluster systems, their reliability is drawing more and more attention from academia to industry. A cluster system is a kind of software load-sharing system (LSS) whose reliability is significantly dependent on system software. Therefore, traditional reliability analysis methods for hardware LSSs are not applicable for cluster systems. In this paper, we develop a reliability analysis model for redundant cluster systems consisting of initial servers and cold standby servers used to replace failed ones. System reliability process is modeled with a state-based non-homogeneous Markov process (NHMH), where each state corresponds to a non-homogeneous Poisson processe (NHPP). NHPP arrival rate is expressed using Cox's proportional hazard model (PHM) in terms of cumulative and instantaneous workload of system software. In addition to redundant cluster systems without repair, the model also can be extended to analyze those with restart. The analysis results are meaningful to support cluster management and design decisions. Finally, the evaluation experiments show the potential of our model.**

Keywords—cluster system*; *load-sharing system*; *cumulative workload*; *software reliability*; *software aging

I. Introduction

Fast development of new technologies has led to a large number of critical commercial applications on the Internet. As the users become dependant on these services, service failure or interruption can cause great loss for service providers. Therefore high availability as well as high performance have become increasingly important to satisfy more demanding quality of service (QoS) requirements. A widely adopted technique to significantly improve system availability and performance is clustering [1]. A cluster is a set of servers and related resources that act like a single system and provide high availability, load balancing and parallel processing. These servers are usually identical. If one server fails, another can act as a backup. Compared to the expensive high availability systems with proprietary tightly coupled hardware and software, cluster systems use commercially available computers networked in a loosely-coupled fashion, and provide high availability and performance in a cost-effective way.

In the past few decades, computing capacity of cluster systems has increased dramatically. However, a linear increase of cluster size results in an exponential failure rate. System software and applications running on cluster systems is becoming more and more complex, which makes them prone to bugs and other software failures. It has been reported that software faults and failures result in more outages in larger computer systems than hardware faults [2] and they cause huge economic losses or risk to human lives. A large percentage of the software failures is due to software aging [3]. Fifty years ago, the notion of software aging was formally introduced in [4]. Since then, much theoretical and experimental research is conducted in order to characterize and understand this important phenomenon. Software aging can be understood as being a continued and growing degradation of the software internal state during its operational life. A general characteristic of this phenomenon is the gradual performance degradation and/or an increase in failure rate [5].

To counteract software aging, a proactive fault management technique called software rejuvenation was proposed. It involves occasionally stopping the running software, cleaning its internal state and/or its environment and restarting it. In order to determine the time epochs for triggering software rejuvenation, analytic models [6, 7], monitoring system resources followed by statistical analysis [8, 9], or a combination [10] have been proposed. However, most of these research works concentrated on the impact of rejuvenation on cluster availability, not system capacity and performability. It also did not consider the workload and failure rate variations caused by user behavior patterns. Many empirical studies of mechanical systems [11] and computer systems [12] have proved that workload strongly affects system failure rate. On the other hand, software rejuvenation is a kind of preventive maintenance technique so that it is not useful for system design decisions.

System design is the foundation of building a software system. Just as a good beginning is half done, good design helps to shorten the period of software development, reduce the cost to operate and maintain a system, and avoid costly rework. During design phase, a cluster system is designded to meet not only functional requirements of customers, but also non-functional ones. Nowadays system non-functional quality properties have drawn more and more attention from customers, which involve system lifetime, reliability, performance, failure rate and so on. As to software aging phenomenon, customers may want to know how long a cluster system would run before aging. Traditionally system designers only make fuzzy estimation about these quality properties based on their previous experience, which is subjective, inaccurate and misleading. Therefore, a reliability model is necessary for cluster system development to present system reliability-related properties under various workload and system desgin schemes precisely. The numerical quality indexes obtained from the model are not only deliverable for customers but also very meaningful for system designers and maintainers to make related decisions. In this paper, we aim to contribute such a reliability model of cluster systems.

0730-3157/16 $31.00 © 2016 IEEE
DOI 10.1109/COMPSAC.2016.177

The remainder of the paper is organized as follows. Section II surveys related work. Section III presents a reliability model of redundant cluster systems. Section IV analyzes the reliabilty model to estimate cluster system reliability. Section V documents the case study before section VI concludes the paper.

II. Related Work

A cluster system is a kind of *k*-out-of-*n* load-sharing system (LSS) in which at least *k*-out-of-*n* components must work for the triumphant operation of the system. The *k*-out-of-*n* configuration redundancy finds capacious purpose in both industrial and military systems. In the real-world, many systems are load-sharing, such as electric generators sharing an electrical load in a power plant, CPU in a multiprocessor computer system, cables in a suspension bridge, bolts used to hold a wheel assembly to a truck, and valves or pumps in a hydraulic system. For above examples, a common feature is that all components in the system share the workload, and comply with a certain set of load sharing rules until one component suddenly fails, after which the load is automatically re-distributed to the remaining components. In most circumstances, increased load induces higher failure rate to a component. A comprehensive discussion about sharing rules (e.g., equal load sharing, tapered load sharing, local load-sharing, etc.) is provided by[13]. The load sharing mechanism introduces dependency between the times to failure among the components, making modeling and inference of such systems different from simpler redundant systems[14].

Statistical methods are generally employed to analyze the relationship between LSS reliability and workload. The classification of statistics-based accelerated life testing (ALT) models are shown in Fig.1. Statistics-based models can be divided into parameter and semi-parameter models. Accelerated failure-time models (AFTMs) are the most popularly used parameter models [15].

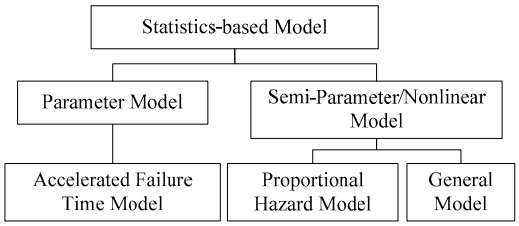

Fig. 1. Statistics-based ALT model classification.

AFTM is first proposed by Pike [16] and has been widely applied since its introduction [17]. AFTM is the technique that utilizes the failure time data of products under higher stresses to extrapolate the lifetime and reliability of the products under normal operating conditions[18-22]. This model specifies that the effect of load is multiplicative in time. AFTMs have significant effects on the accuracy of the estimations of lifetime and reliability of products. The main problem with the existing AFTMs is that most of them model hardware LSSs and not suitable for software systems. When a hardware LSS is put into operation, all components works all the time. A cluster system is a kind of software systems, whose reliability and lifetime is significantly dependent on system software. In contrast to hardware, software performs only when it is called. System software deployed in a cluster executes intermittently so that the regular chronological time scale is not applicable for modeling a cluster system. This condition implies that the existing AFTMs for hardware LSSs are not adequate for cluster systems.

On the other hand, component failure rates in AFTMs are usually modeled with exponential, Weibull, Gaussian, lognormal, and Gamma distributions. The potential assumption is that component failure rates at different application scenarios meet identical distribution. In practice, when failure process is very complicated or failure time distributions are inconsistent, AFTMs are unable to accurately process data and predict reliability. Under these circumstances, proportional hazard models (PHMs) are more suitable and more flexible to estimate LSS reliability due to their distribution-free property which makes them follow more closely and need relatively less observation data. The PHM is first proposed by Cox [23] and has been widely applied to relate the failure probability to both historical service lifetime and condition monitoring variables [24]. For this time-dependent model, failure prediction is treated as estimating the remaining lifetime for a system with regard to a specific hazard level under the current conditions. Some PHMs have been developed to analyze reliability of hardware LSSs [25], which are not applicable for modeling a cluster system.

III. System Reliability Model

In this section a reliability model is defined in every detail to formalize the factors related to the reliability of a cluster system for the purpose of reliability analysis. The definition of the reliability model meets the following assumptions.

- There are *n* i.i.d servers in a cluster system, where an application software is deployed and running. When a cluster is launched, *m* initial servers are started to process user requests while the rest (*n*-*m*) substitute servers are cold standby and ready to replace failed ones. The system functions successfully if and only if the number of active servers is not less than *k*.
- User requests arriving at a cluster is a stationary stochastic process with a constant arriving rate, which are distributed to all working servers equally.
- No repair is considered. When a server fails, it is replaced by a substitute server if there is anyone. The time taken for replacement is non-negligible, during which no another server crashes.
- The more time a server has worked for, the more prone it is to failures caused by software faults without condieration of hardware faults.

A. Software Behavior Model

We adopt the measure of failure rate to describe software behavior. Failure rate is defined as conditional probability that failures occur at a time interval $t \sim (t + dt)$ while software doesn't fail before time *t*. Failure rate during software lifetime experiences three phases. Early life failures occur at software testing stage. As testing progresses, faults are discovered and corrected, which makes failure rate decrease and reliability grow. In the last few decades, a great many software reliability growth models [26] have been proposed to depict software

testing process. These models are especially useful to estimate the time point to stop testing and distribute software. After that time point, more testing work would not decrease failure rate significantly, and software settles into a relatively stable state. After a period of stable operation, software steps into wear-out phase, during which failure rate gradually increases since software ages and degrades.

System software is deployed and runs in a cluster system after it is distributed. Thus, software application phase is really concerned in this paper, during which software operates stably for a period of time with low proximately constant failure rate, and then ages with increasing failure rate. The rate at which software wares out is significantly dependent on software workload. Cox's PHM is adopted to model the relationship between software failure rate and workload including cumulative and instantaneous load. We introduce cumulative execution time $X(t)$ up to time t to describe cumulative load which reflects software age. $X(t)$ is defined as how long software has taken to process user requests from time 0 to t. In contrast to hardware, software performs only when it is called so that $X(t)<t$. Instantaneous load denotes the rate at which user requests arrive at a server. In general, the faster user requests arrive, the more possibly a server crashes. The degree to which the two kinds of workload affect software failure rate is dependent on the technology and experience used to develop the software.

Based on above analysis, software failure rate is given by

$$h(t) = b\exp\left(\alpha \cdot X(t) + \beta \cdot Y(t)\right) \quad (1)$$

where $h(t)$ is software failure rate at time t; b is the constant baseline failure rate, the value of which depends on how well software is developed and tested; α and β are regression coefficients estimated with observed failure data; $X(t)$ is the cumulative execution time up to time t; and $Y(t)$ is the rate at which user requests arrive at a server.

Equation (1) indicates that software failure rate approximately meets extreme distribution. Extreme distribution matches the variation law of the failure rate at software application stage, which keeps approximately constant for a period of time and then gradually increases.

B. System Behavior Model

A cluster system is a load-sharing k-out-of-n redundant system, whose reliability model is defined as Definition 1.

Definition 1. (CSRM) A **CSRM** is a cluster system reliability model, and is defined by the tuple <int n, int k, int m, Software *soft*, Profile profile, Server[n] s, double[] R, State *state*>, where n is the total number of servers in a cluster; k is the minimal number of servers required for system successful operation; m is the number of initial servers; *soft* is the application software deployed in a cluster; *profile* describes how latent users visit a cluster; s is a set of servers constituting a cluster; R is time-varying system reliability; and *state* denotes system state.

Definition 2. (Software) A Software models an application software running on a cluster, and is defined as the tuple <double b, double α, double β>, where b, α and β denote the same as those in (1).

Definition 3. (Profile) A Profile models how a cluster system is visited, and is defined as the tuple <double μ, int ω>, where μ is the rate at which user requests arrive; and ω is the average amount of workload included in a request.

Definition 4. (Server) A Server models hardware aspect of a cluster system, and is defined by the tuple <int *no*, double γ, State *birth_state*, double σ, {*standby*, *working*, *failed*} *status*>, where *no* is the sequence number of a server; γ is the rate at which a server processes user requests; *birth_state* is the system state where a server is booted; σ is the response time taken to replace a failed server with a standby one; and *status* indicates server status composed of three kinds of status.

Definition 5. (State) A State represents cluster system state, and is defined as the tuple <int s, double τ>, where s is the number of failed servers, and τ is the expected time for which a state lasts.

We define system states as the number of failed servers, which ranges from 0 to (n-k+1). When a cluster system is launched at time 0, m initial servers are started with the rest (n-m) substitute servers standby. During system operation, substitute servers are continually consumed to replace failed ones. When the number of failures is less than (n-m), the system stays in standby phase with m working servers. After state (n-m), the system enters into degrading phase with gradually decreasing number of working servers. A cluster system eventually breaks down when the number of failures exceeds (n-k). Since the combination and workload of working servers at various states are heterogeneous, system failure rate is non-continuous but state-based.

From a whole point of view, system failure process can be represented by pure birth Markov process as shown in Fig.2 without consideration of repair or maintenance. It can be seen that a cluster experiences three phases during its lifetime, which is standby, degrading, and breakdown phase. Since system failure rate relies on software failure rate which is not constant but varies with time, system failure process is a state-based non-homogeneous Markov process (NHMP). The state-based NHMP can be further divided into (n-k+1) NHPPs each of which corresponds to a state. According to the definition of system state, only one server breaks down during each NHPP.

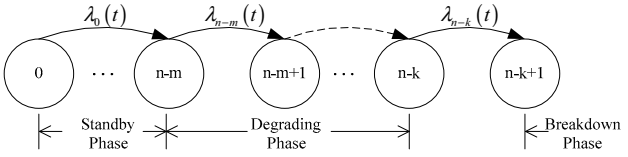

Fig. 2. Pure birth process for the failure process of a cluster system.

IV. Reliability Analysis

The object of reliability analysis is to obtain time-dependent reliability process of a cluster system. It has been analyzed in the last section that the reliability process of a cluster system is a state-based NHMP composed of several NHPPs. Before the NHMP can be solved to obtain cluster reliability process, the failure rate at each state should be analyzed at first. For convenience and without loss of

generality, we make the following assumptions for the analysis procedure.

- A sequence number is assigned to each server from 1 to n, among which the servers from 1 to m are initial servers and the rest ones from (m+1) to n are substitute servers.
- The server with smaller sequence number would fail earlier than that with larger number. In other words, the sequence by which all servers break down is from 1 to (n-k+1).
- If a server with number $l\left(l \le (n-m)\right)$ failed, a substitute server with number (m+l) would replace it.

There are two circumstances for failure rate analysis. One is shown in Fig.3, where the number of initial servers is more than substitute servers and it turns out to be $2m \ge n$, while the other is shown in Fig.4, where the relationship between the two numbers is opposite and it turns out to be $2m < n$. The combination or/and number of working servers are different at different system states, which can be grouped in terms of state m and (n-m). Both of states are refered to as critical states. All initial servers would fail at state m while all substitute servers would run out at state (n-m). Therefore, all system states in the two circumstances can be grouped into the following six cases for failure rate analysis, which are also marked in Fig.3 and 4.

- Case 1: $0 \le s \le (n-m)$ and $2m \ge n$;
- Case 2: $(n-m) < s \le m$ and $2m \ge n$;
- Case 3: $m < s \le (n-k)$ and $2m \ge n$;
- Case 4: $0 \le s \le m$ and $2m < n$;
- Case 5: $m < s \le (n-m)$ and $2m < n$;
- Case 6: $(n-m) < s \le (n-k)$ and $2m < n$.

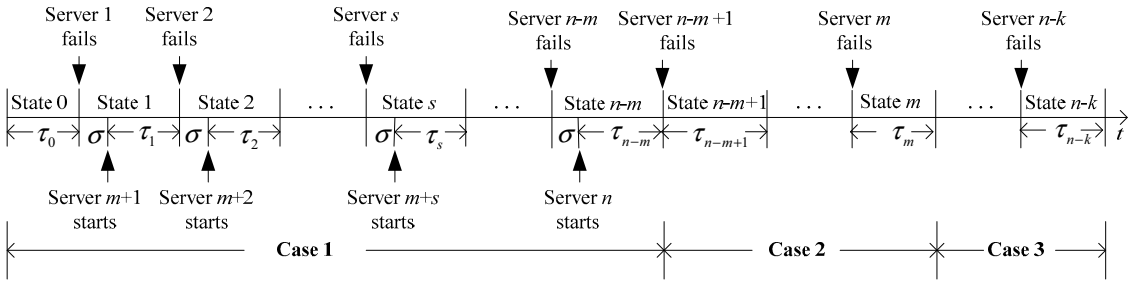

Fig. 3. State distribution under the condition of $(2m \ge n)$.

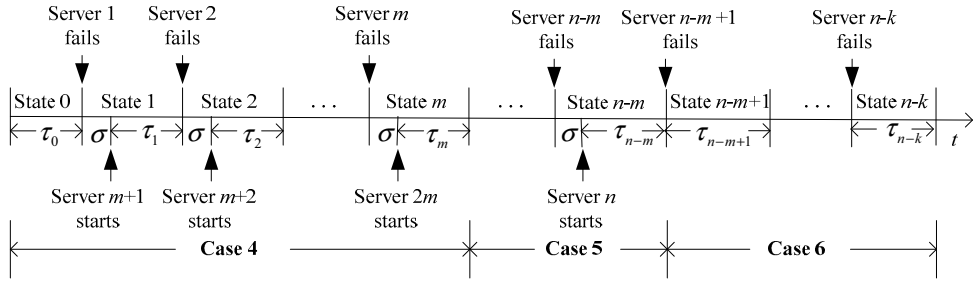

Fig. 4. State distribution under the condition of $(2m < n)$.

A. Failure Rate Analysis in Case 1 and 4

The failure rates in case 1 and 4 can be analyzed together. For all states in both cases, the number of active servers is equal to m, which are composed of initial and substitute servers except for state 0. The active servers at state 0 only consist of initial servers so that the failure rate at state 0 will be analyzed seperately. Before performing the analysis, we define an intermediate variable.

$$\phi_i = \frac{\alpha\mu\omega}{i\gamma},\ k \le i \le m \tag{2}$$

where i indicates the number of working servers; and other notations refer to Definition 2-4.

All working servers at state 0 are synchronous with the same birth time. The cumulative execution time of software on a server is dependent on usage profile and server performance. At time t of state 0, the amount of user requests arriving at a server is $\mu t/m$. The time a server takes to process a request is ω/γ. Thus, software cumulative execution time on the server i at time t is

$$X_{0,i}(t) = \frac{\mu t}{m} \cdot \frac{\omega}{r} = \frac{\mu\omega}{m\gamma} t \tag{3}$$

The failure rate of server i at state 0 is expressed as

$$\begin{aligned} h_{0,i}(t) &= b\exp(\alpha X_{0,i}(t) + \beta Y_0(t)) \\ &= b\exp(\phi_m t + \beta\frac{\mu}{m}) \end{aligned} \tag{4}$$

Given the failure rate of each server, system failure rate at state 0 that is also NHPP arrival rate is

$$\lambda_0(t) = mb\exp(\phi_m t + \beta\frac{\mu}{m}) \tag{5}$$

During the NHPP corresponding to state 0, server 1 breaks down. Thus, we have

$$\int_0^{\tau_0} \lambda_0(t)\,dt = 1 \tag{6}$$

where τ_0 is the expected time at state 0, which can be solved from above equation as

$$\tau_0 = \frac{1}{\phi_m}\ln\left[\frac{\phi_m}{mb\exp\left(\frac{\beta\mu}{m}\right)}+1\right] \tag{7}$$

Server replacement should be considered for the failure rate analysis at other states except for state 0. At the replacement interval of the state $s(s\neq 0)$, there are m working servers composed of (m-s) initial ones from (s+1) to m and (s-1) substitute ones from (m+1) to (m+s-1). So the cluster failure rate is expressed as

$$\begin{aligned}\lambda_s(\Delta t) &= \sum_{i=s+1}^{m} h_{s,i}(\Delta t) + \sum_{i=m+1}^{m+s-1} h_{s,i}(\Delta t)\\ &= (m-s)b\exp\left(\phi_{m-1}\Delta t + \sum_{i=0}^{s-1}\phi_m\tau_i + (s-1)\phi_{m-1}\sigma + \frac{\beta\mu}{m-1}\right)\\ &+ \sum_{i=m+1}^{m+s-1} b\exp\left(\phi_{m-1}\Delta t + \sum_{j=i-m}^{s-1}\phi_m\tau_j + \sum_{j=i-m}^{s-2}\phi_{m-1}\sigma + \frac{\beta\mu}{m-1}\right)\end{aligned} \tag{8}$$

where Δt is the absolute time at state s and meets $0\le\Delta t<\sigma$.

When server replacement completes, a new server with sequence number (m+s) starts so that the number of working servers resumes to be m. In this situation, system failure rate becomes

$$\begin{aligned}\lambda_s(\Delta t) &= \sum_{i=s+1}^{m} h_{s,i}(\Delta t) + \sum_{i=m+1}^{m+s-1} h_{s,i}(\Delta t) + h_{s,m+s}(\Delta t)\\ &= (m-s)b\exp\left(\phi_m\Delta t + \sum_{i=0}^{s-1}(\phi_m\tau_i + \phi_{m-1}\sigma) + \beta\frac{\mu}{m}\right)\\ &+ \sum_{i=m+1}^{m+s-1} b\exp\left(\phi_m\Delta t + \sum_{j=i-m}^{s-1}(\phi_m\tau_j + \phi_{m-1}\sigma) + \beta\frac{\mu}{m}\right)\\ &+ b\exp\left(\phi_m\Delta t + \beta\frac{\mu}{m}\right)\end{aligned} \tag{9}$$

where Δt denotes the absolute time after server replacement completes.

Similarly, only one failure occurs during the NHPP corresponding to state s. Therefore, we have

$$\int_0^{\sigma}\lambda_s(\Delta t)\,d\Delta t + \int_{\sigma}^{\sigma+\tau_s}\lambda_s(\Delta t+\sigma)\,d\Delta t = 1 \tag{10}$$

The expected execution time without failures at state s can be solved as

$$\tau_s = \frac{1}{\phi_m}\cdot \ln\left(\frac{\phi_m\left(\begin{array}{l}1-\frac{b}{\phi_{m-1}}\left(\exp(\phi_{m-1}\sigma)-1\right)\cdot\\ \exp(\frac{\beta\mu}{m-1})\left(\begin{array}{l}(m-s)\exp\left(\sum_{i=0}^{s-1}\phi_m\tau_i+(s-1)\phi_{m-1}\sigma\right)\\ +\sum_{i=m+1}^{m+s-1}\exp\left(\sum_{j=i-m}^{s-1}\phi_m\tau_j+\sum_{j=i-m}^{s-2}\phi_{m-1}\sigma\right)\end{array}\right)\end{array}\right)}{b\exp\left(\beta\frac{\mu}{m}\right)\left(\begin{array}{l}(m-s)\exp\left(\sum_{i=0}^{s-1}(\phi_m\tau_i+\phi_{m-1}\sigma)\right)\\ +\sum_{i=m+1}^{m+s-1}\exp\left(\sum_{j=i-m}^{s-1}(\phi_m\tau_j+\phi_{m-1}\sigma)\right)+1\end{array}\right)}+1\right) \tag{11}$$

B. Failure Rate Analysis in Case 2

Case 2 belonging to degrading phase is composed of the states from (n-m+1) to m. In constrast to standby phase with the constant number of working servers, the number of working servers would gradually decrease as failures occur at degrading phase. The working servers in case 2 consists of initial and substitute ones. Take the state s in case 2 for example, (m-s) initial servers and (n-m) substitute ones are processing user requests with the total number equal to be (n-s).

Since all working servers equally share user requests, the rate at which user requests arrive at a server is

$$Y_s(t) = \frac{\mu}{n-s} \tag{12}$$

The sequence numbers of surviving initial servers range from (s+1) to m while those of active substitute servers are from (m+1) to n. The system failure rate at state s is equal to the sum of those of all active servers.

$$\begin{aligned}\lambda_s(\Delta t) &= \sum_{i=s+1}^{m} h_{s,i}(\Delta t) + \sum_{i=m+1}^{n} h_{s,i}(\Delta t)\\ &= (m-s)b\exp\left(\begin{array}{l}\phi_{n-s}\Delta t + \sum_{i=n-m}^{s-1}(\phi_{n-i}\tau_i) +\\ \sum_{i=0}^{n-m-1}(\phi_m\tau_i+\phi_{m-1}\sigma) + \beta\frac{\mu}{n-s}\end{array}\right)\\ &+ \sum_{i=m+1}^{n} b\exp\left(\begin{array}{l}\phi_{n-s}\Delta t + \sum_{j=n-m}^{s-1}(\phi_{n-j}\tau_j) +\\ \sum_{j=i-m}^{n-m-1}(\phi_m\tau_j+\phi_{m-1}\sigma) + \beta\frac{\mu}{n-s}\end{array}\right)\end{aligned} \tag{13}$$

One server breaks down at state s. Thus, we have

$$\int_0^{\tau_s}\lambda_s(\Delta t)\,d\Delta t = 1 \tag{14}$$

The expected time at state s can be solved as

$$\tau_s = \frac{1}{\phi_{n-s}} \ln \left(\begin{array}{l} \dfrac{\phi_{n-s}}{b \exp\left(\sum_{i=n-m}^{s-1} \left(\phi_{n-i} \tau_i \right) + \beta \dfrac{\mu}{n-s} \right) \cdot} \\ \left((m-s) \exp\left(\sum_{i=0}^{n-m-1} \left(\phi_m \tau_i + \phi_{m-1} \sigma \right) \right) \right. \\ \left. + \sum_{i=m+1}^{n} \exp\left(\sum_{j=i-m}^{n-m-1} \left(\phi_m \tau_j + \phi_{m-1} \sigma \right) \right) \right) \\ +1 \end{array} \right) \quad (15)$$

C. Failure Rate Analysis in Case 3 and 6

Both case 3 and 6 belong to degrading phase with the decreasing number of active servers. In both cases, all initial servers have broken down while all substitute ones have been started. So the active servers are totally composed of substitute ones. At state *s*, the servers from (*s*+1) to *n* are running to process user requests. Thus, system failure rate is expressed as

$$\begin{aligned} \lambda_s(\Delta t) &= \sum_{i=s+1}^{n} h_{s,i}(\Delta t) \\ &= \sum_{i=s+1}^{n} b \exp\left(\begin{array}{l} \phi_{n-s} \Delta t + \sum_{j=n-m}^{s-1} \left(\phi_{n-j} \tau_j \right) + \\ \sum_{j=i-m}^{n-m-1} \left(\phi_m \tau_j + \phi_{m-1} \sigma \right) + \beta \dfrac{\mu}{n-s} \end{array} \right) \end{aligned} \quad (16)$$

Only one failure occurs at state *s*. Thus, we have

$$\int_0^{\tau_s} \lambda_s(\Delta t) \, d\Delta t = 1 \quad (17)$$

The expected time at state *s* can be solved as

$$\tau_s = \frac{1}{\phi_{n-s}} \ln \left(\begin{array}{l} \dfrac{\phi_{n-s}}{\sum_{i=s+1}^{n} b \exp\left(\begin{array}{l} \sum_{j=n-m}^{s-1} \left(\phi_{n-j} \tau_j \right) + \\ \sum_{j=i-m}^{n-m-1} \left(\phi_m \tau_j + \phi_{m-1} \sigma \right) + \beta \dfrac{\mu}{n-s} \end{array} \right)} \\ +1 \end{array} \right) \quad (18)$$

D. Failure Rate Analysis in Case 5

Case 5 involves the states from (*m*+1) to (*n*-*m*) at standby phase, where all initial servers have broken down but substitute ones have not ran out yet. The working servers are totally composed of *m* substitute ones. At initial replacement time interval of state *s*, the servers from (*s*+1) to (*m*+*s*-1) are running while a substitute one with number (*m*+*s*) is going to replace the just failed one. At this time, the system failure rate is given by

$$\begin{aligned} \lambda_s(\Delta t) &= \sum_{i=s+1}^{m+s-1} h_{s,i}(\Delta t) \\ &= \sum_{i=s+1}^{m+s-1} b \exp\left(\phi_{m-1} \Delta t + \sum_{j=i-m}^{s-1} \phi_m \tau_j + \sum_{j=i-m}^{s-2} \phi_{m-1} \sigma + \beta \frac{\mu}{m-1} \right) \end{aligned} \quad (19)$$

After the replacement completes, the server with number (*m*+*s*) starts, and then the system failure rate becomes

$$\begin{aligned} \lambda_s(\Delta t) &= h_{s,m+s}(\Delta t) + \sum_{i=s+1}^{m+s-1} h_{s,i}(\Delta t) \\ &= b \exp\left(\phi_m \Delta t + \beta \frac{\mu}{m} \right) + \\ &\sum_{i=s+1}^{m+s-1} b \exp\left(\phi_m \Delta t + \sum_{j=i-m}^{s-1} \left(\phi_m \tau_j + \phi_{m-1} \sigma \right) + \beta \frac{\mu}{m} \right) \end{aligned} \quad (20)$$

One server would crash during the NHPP corresponding to state *s*. Thus, we have

$$\int_0^{\sigma} \lambda_s(\Delta t) \, d\Delta t + \int_{\sigma}^{\sigma+\tau_s} \lambda_s(\Delta t + \sigma) \, d\Delta t = 1 \quad (21)$$

The expected time at state *s* can be solved as

$$\tau_s = \frac{1}{\phi_m} \cdot \ln \left[\frac{\phi_m \left(\begin{array}{l} 1 - \dfrac{\exp(\phi_{m-1}\sigma) - 1}{\phi_{m-1}} \cdot \\ \sum_{i=s+1}^{m+s-1} b \exp\left(\sum_{j=i-m}^{s-1} \phi_m \tau_j + \sum_{j=i-m}^{s-2} \phi_{m-1} \sigma + \beta \dfrac{\mu}{m-1} \right) \end{array} \right)}{b \exp\left(\beta \dfrac{\mu}{m} \right) \left(1 + \sum_{i=s+1}^{m+s-1} b \exp\left(\sum_{j=i-m}^{s-1} \left(\phi_m \tau_j + \phi_{m-1} \sigma \right) \right) \right)} + 1 \right] \quad (22)$$

E. Reliability Analysis

A cluster system is a load-sharing *k*-out-of-*n* redundant software system, where $n \geq k$. A cluster system would fail if and only if it is unable to respond to user requests promptly, which means that the number of working servers is less than *k*. The value of *k* is dependent on usage profile and the efficiency how fast servers process user requests, which is set as the lower limit value for the usability of a cluster system. Thus, the value of *k* is given by

$$k = \left\lceil \frac{\mu \omega}{\gamma} \right\rceil \quad (23)$$

As a cluster system operates continuously and ages, its failure rate gradually grows while the reliability decreases correspondingly. As the correlated quality properties, their correlation can be expressed as

$$\lambda(t) = \frac{f(t)}{R(t)} \quad (24)$$

where $R(t)$ is system reliability; and $f(t)$ is failure probability density function given by

$$f(t) = \frac{dF(t)}{dt} = \frac{d(1 - R(t))}{dt} = -\frac{dR(t)}{dt} \quad (25)$$

where $F(t)$ is cumulative failure distribution function.

With (24) and (25), the correlationship between failure rate and reliability can be solved as

$$\lambda(t)\,dt = -d\ln R(t) \tag{26}$$

System reliability process is a NHMP composed of (*n*-*k*+1) normal states and one failure state. Since system failure rate has been analyzed in the last several sub-sections as state-based time-dependent function, system reliability at normal states can be solved from that.

At first, the system reliability at state 0 can be solved as

$$R_0(\Delta t) = \exp\left(-\frac{\lambda_0(\Delta t)}{\phi_m}\right) \tag{27}$$

Server replacement should be taken account for the reliability analysis at other states except for state 0 in standby phase. At the replacement interval of state *s*, system reliability is given by

$$R_s(\Delta t) = \exp\left(-\frac{\lambda_s(\Delta t)}{\phi_{m-1}}\right), 0 < s \le (n-m) \tag{28}$$

After the replacement, the system reliability at standby phase becomes

$$R_s(\Delta t) = \exp\left(-\frac{\lambda_s(\Delta t)}{\phi_m}\right), 0 < s \le (n-m) \tag{29}$$

The system reliability at state *s* of degrading phase is

$$R_s(\Delta t) = \exp\left(-\frac{\lambda_s(\Delta t)}{\phi_{n-s}}\right), (n-m) < s \le (n-k) \tag{30}$$

State (*n*-*k*) is the last normal state, where one more failure would cause a cluster system crash. In that case the system reliability is

$$R_s(t) = 0, s > (n-k) \tag{31}$$

V. Illustrative Example

In this section, we present an example cluster system to illustrate the proposed cluster reliability analysis model. The model is applied for the reliability process and lifetime analysis of a redundant cluster system. Some experiments are also be constructed to show how the reliability model can be used to support system maintenance and design decisions. In addition to redundant cluster systems without repair, the model also can be extended to consider restart.

A. Experimental Setup

Figure 5 illustrates a high-level view on the Business Reporting System (BRS) [24], which generates management reports from business data collected in a database. The model is based on an industrial system. The BRS system consists of 12 software components deployed on different servers. The web server propagates user requests to a scheduler server, which hosts, among others, a scheduler and a user management component. From there, requests reach the main application server and are possibly dispatched to further application servers by a load balancer component. A database server hosts the database and a corresponding data access component. The system includes caches to reduce the need of database accesses.

The bottleneck to BRS reliability is up to a load-sharing cluster system named GCS. GCS is composed of several servers where software component named Core Graphic Engine is deployed and running.

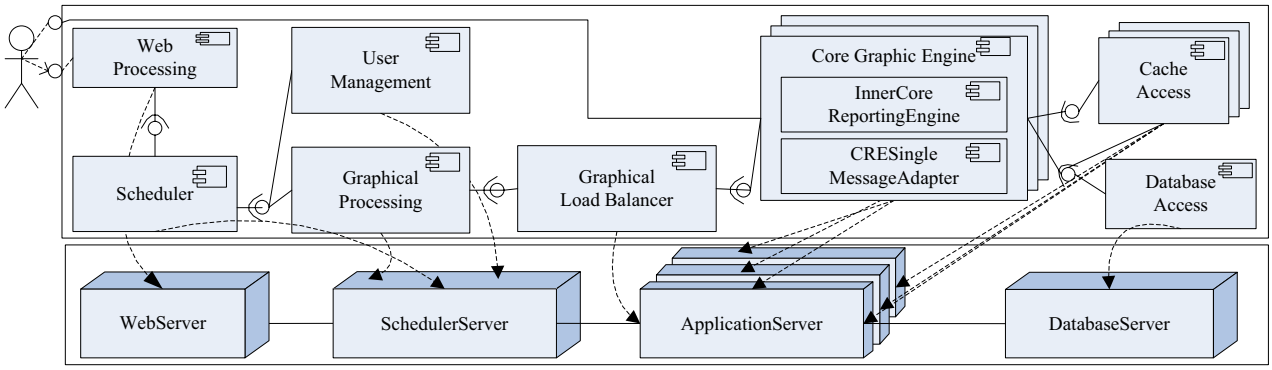

Fig. 5. An Overview of the Business Reporting System.

B. GCS Reliability Analysis

GCS uses a *k*-out-of-*n* structure, where *n*=10 and *m*=6. Assume that the initial failure rate of system software is 0.00001 failures per day. User requests arrive at GCS at an average rate of 100 requests per second. The time that a server takes to respond to a request is about 30ms. According to the usage profile and server performance, *k* is solved as 3. Thus, GCS reliability process consists of eight normal states ranging from 0 to 7, among which state 4 and 6 are critical ones. All substitute servers run out at state 4 while all initial servers break down at state 6.

At first, we calculate the expected time at each state where two coefficients α and β are assumed as $\alpha = 0.1$ and $\beta = 0.000001$. The results are shown in Table I.

It can be seen from Table 1 that the trend how the expected execution time varies with the states can be divided by both critical states. The expected time from state 0 to 4 gradually decreases, that from state 4 to 6 grows, and after state 6 that

resumes to decrease with states. The reason for this trend is discussed as follows. When a cluster system is launched, all initial servers are started to run. After the longest-time operation at the first state, all of them become old and failure-prone. Although young substitute servers continue to replace the failed old ones at standby phase, old ones still command the mojority of working servers utill state 4, which leads to high system failure rate and low state duration. After state 4, young substitute servers form the majority of activer servers while old initial ones breaks down and are removed continually, which makes the system revive. In this situation, system failure rate gradually decreases and state duration prolongs, which reaches its peak at state 6 with all old initial ones failed. After a farily long-time operation at state 6, substitute servers also become old without new ones for replacement. Thus, the state duration shortens gradually.

TABLE I. STATE DURATION WITH n=10 AND m=6

State	0	1	2	3
Time (hours)	196.871	15.443	10.047	8.255
State	4	5	6	7
Time (hours)	8.006	9.631	114.984	13.251

With state durations known, system failure rate and reliability can be solved as shown in Fig. 6 and 7.

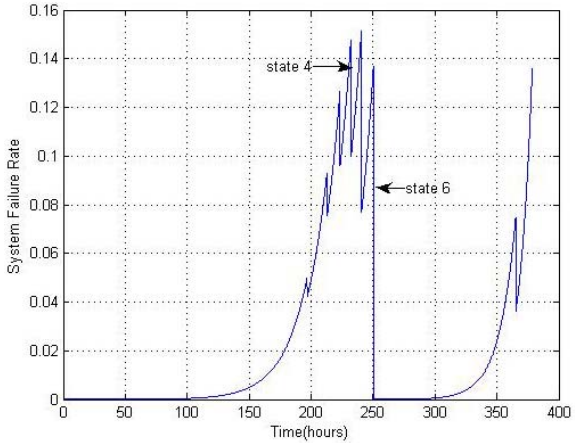

Fig. 6. System Failure Process with n=10 and m=6.

It can be seen from Fig 6 and 7 that GCS failure rate and reliability vary oppositely with time. Similarly, both critical states can be identified obviously as the turning points of the variation trends. The general trend of system reliability conforms to that of state duration while system failure rate follows a opposite trend. If we look closely at the two figures, it can be found that a jitter would occur when the process enters into a new state, during which system failure rate plummets and then bounces rapidly. This phenomenon explains why the reliability process of a cluster system is modeled as a non-countinous state-based NHMP. In standby phase before critical state 4, a jitter occurs due to server replacement. It can also be observed that the jitters become larger and larger from state 0 to 4, which reflects the replacement effect since the age difference between both sides of the replacement increases with the states. The jitters at the states in degrading phase occur due to server crash. An older server crash introduces more effect to system failure rate and reliability.

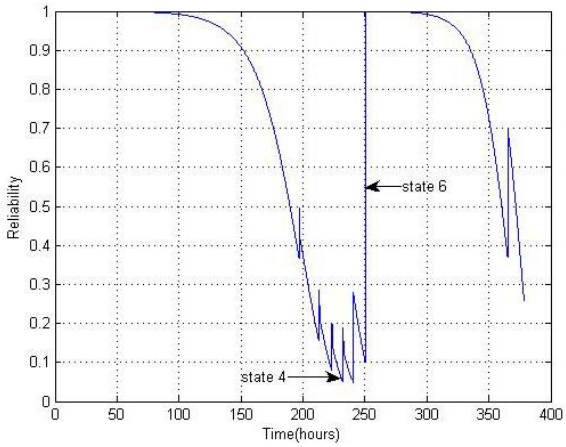

Fig. 7. System Reliability Process with n=10 and m=6.

The analysis of failure rate and reliability is useful for cluster system maintenance. Especially for some safety-critical systems, it is usually required that system failure rate couldn't exceed a threshold. Cluster maintainers take the analysis results as reference to replace or restart the old servers timely in case they introduce high failure rate. By this way, our cluster reliabilty analysis model contributes to a cost-effective system maintenance.

C. Reliability Analysis under Various Configurations

In this sub-section, some experiments will be conducted to show how our reliability analysis model is able to assist cluster system designers to make decisions. A cluster system is a load-sharing k-out-of-n redundant software system. System designers should determine the values of n, m, and k to ensure a cluster meet the practical requirements. K as the minimal number of active servers can be solved according to usage profile and server performance. N as the total number of servers is usually dependent on the tradeoff between cost and system lifetime. The larger the value of n is, the longer the system executes for with higher operation and maintenance cost. How system lifetime varies with n is shown in Fig. 8, where the value of m is set to be 3 for all cases. It is helpful for system designers to determine the scale of a cluster system according to the requirement about system operation lifetime.

Given cluster size, the designers further need to consider how to divide these servers into two groups will make the system perform best, which involves how much the value of m is set. How system failure rate varies with m is shown in Fig. 9, where n=10.

It can be seen that system durations in above various cases are approximately equal since the total numbers of servers are equal. We can't gain obvious enlightment from Fig.9 about which configuration is optimal. In this case, we further compare the performance of a cluster system configured with different values of m. The optimal system should achive highest reliability and meanwhile the least failure rate. From economical perspective, the less the number of initial servers is,

the less costly the system operates and maintains. Thus, we compare the differences between system reliability and failure rate, which are shown in Table 2. It can be concluded from Table 2 that m=11 is the optimal system configuration.

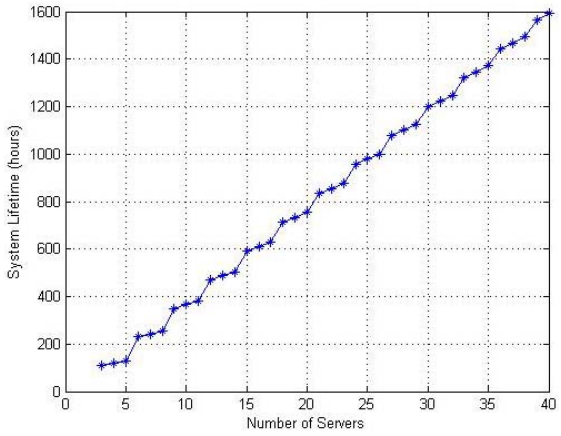

Fig. 8. System Lifetime with Various n and m=3.

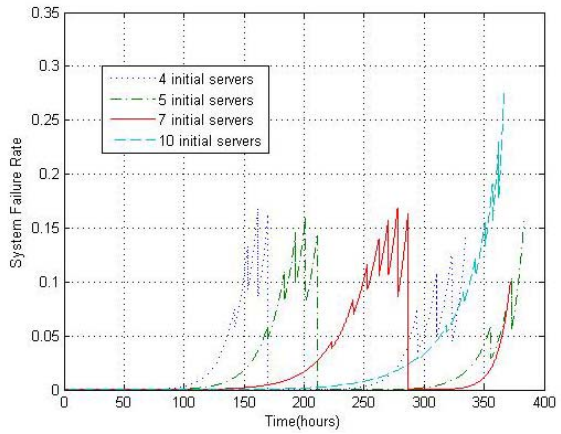

Fig. 9. System Failure Process with Various m and n =10.

TABLE II. DIFFERENCES BETWEEN AVERAGE RELIABILITY AND FAILURE RATE WITH n=30 AND VARIOUS m

m	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
D	0.16	0.23	0.27	0.32	0.36	0.47	0.45	0.46	0.60	0.59
m	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
D	0.58	0.57	0.60	0.58	0.58	0.59	0.59	0.59	0.59	0.59
m	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**		
D	0.59	0.59	0.58	0.58	0.58	0.57	0.58	0.58		

D. Reliability Analysis Considering Restart

In contrast to hardware systems, software systems are able to rejuvenate by restart. When a server breaks down, cluster maintainers probably restart it before replacing it with a new one. After restart, the system internal state and/or its environment are cleaned up so that the system will work anew. Our reliability analysis approach can be extended to estimate the reliability of a cluster system with restart. It is assumed that restart would always work without consideration of hardware faults. In this situation, a cluster system can be regarded as a combination of m initial servers and indefinite substitute servers. System reliability process is not finite with a termination state any more, but infinite with infinite states belonging to standby phase. Therefore, reliability analysis only involves case 4 and 5.

With consideration of restart, cluster designers only need to determine the value of m, which is how many servers they would start to process user requests. Taking cluster design assistance as the starting point, an experiment is constructed to analyze and compare cluster reliability processes under various initial configuration. The results are shown in Fig.10.

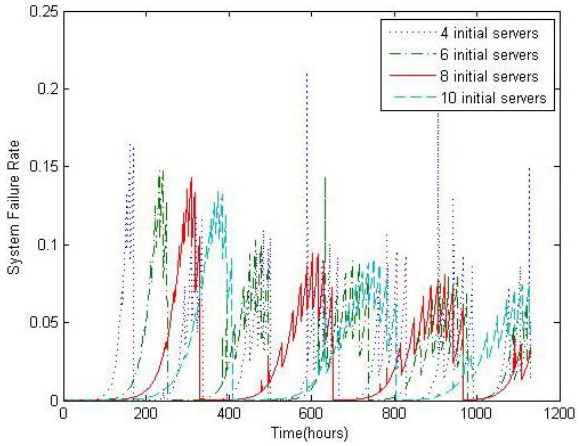

Fig. 10. System Failure Process Considering Restart.

It can be seen from Fig.10 that system failure rate presents cyclic fluctuation which gradually levels off with time. The peirod of fluctuation lasts longer with the larger number of initial servers. It visually seems that more initial servers introduce more instability into system operation. In order to approve this view, we further compare system failure rate and reliability numerically, which are shown in Table 3. It can be found out that the cluster system with less initial servers performs better than those with more ones. Of course, this conclusion is based on an assumption that restart would always work and server hardware would never break. Obviously, the assumption does not hold because the operation lifetime of hardware is limited in contrast to software. Just as a coin has two sides, less initial servers make a cluster perform better but meanwhile introduce higher hardware utilization, which is also shown in Table 3. Higher utilization would wear hardware out quickly to lead to shorter cluster system duration. In our future work, we will further consider hardware faults and lifetime in cluster reliability analysis.

TABLE III. VALUES OF RELIABILITY PROPERTIES WITH VARIOUS m.

	***m*=4**	***m*=6**	***m*=8**	***m*=10**
Failure Rate	0.0249	0.0257	0.0263	0.0268
Reliability	0.7281	0.6071	0.50276	0.4146
Utilization	7.5%	5%	3.75%	3%

VI. CONCLUSION

In this paper we present an approach to model and analyze cluster system reliability. Using the model and method, it is easy to analyze multi-state software LSSs that arise due to reliability degradation associated with software aging and the failures of load-sharing components. The analysis results are close-form solutions for system reliability and failure rate which are very meaningful to support cluster management and design decisions. The study on cluster reliability analysis has just started so that there are still many problems that need to be solved. In the future work, we will try to refine the reliability model further to make it closer to practical operation scenarios of a cluster system. For example, the usage profile in this model is simply described with a constant arrival rate. However, the way how latent users manipulate a cluster system in practice could be very complicated and present a sophisticated stochastic process. Moreover, hardware faults also fairly affect the reliability and lifetime of a cluster system so that they should be considered in the reliability analysis in addition to software faults. By this way, we will improve the accuracy of cluster reliability analysis, and at the same time expand the application of the cluster reliability model.

ACKNOWLEDGMENT

This research was supported by the National Natural Science Foundation of China under Grand No.61402333, No. 61402242, No. 61272450 and No.61202381, and Tianjin Municipal Science and Technology Commission under Grand No. 15JCQNJC00400. The authors would also like to thank Tianjin Key Lab of Intelligent Computing and Novel Software Technology and Key Laboratory of Computer Vision and System, Ministry of Education, for their support to the work.

REFERENCES

[1] Dazhi Wang, Wei Xie, Kishor S. Trivedi. Performability analysis of clustered systems with rejuvenation under varying workload. Performance Evaluation, 2007, 64: 247-265.

[2] A. Avritzer, E. Weyuker, Monitoring smoothly degrading systems for increased dependability, Empirical Software Engineering 2 (1997), 55–77.

[3] L. Li, K. Vaidyanathan, K.S. Trivedi, An approach to estimation of software aging in a web server. In: Proceedings of the International Symposium on Empirical Software Engineering, Nara, Japan,October 2002, pp.91–100.

[4] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton, "Software rejuvenation: analysis, module and applications," in Proc. 25th Intl Symposium on Fault-Tolerant Computing, 1995, pp. 381–390.

[5] M. Grottke, R. Matias, and K. Trivedi, "The fundamentals of software aging," In Proc of Workshop on Software Aging and Rejuvenation, in conjunction with IEEE International Symposium on Software Reliability Engineering. 2008.

[6] Z. Ye, M. Revie, and L. Walls, "A load sharing system reliability model with managed component degradation," IEEE Trans. Reliability , vol. 63, no. 3, pp. 721–730, September 2014.

[7] D. Wang, W. Xie, and K. S. Trivedi. "Performability analysis of clustered systems with rejuvenation under varying workload," Performance Evaluation, vol. 64, no. 3, pp. 247-265, 2007.

[8] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S. Trivedi. "A methodology for detection and estimation of software aging," In Proc. 9th International Symposium on Software Reliability Engineering, pp. 283–292, 1998.

[9] M. Grottke, L. Li, K. Vaidyanathan, and K. Trivedi. "Analysis of software aging in a web server," IEEE Transactions on Reliability, vol. 55, no. 3, pp. 411-420, 2006.

[10] 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.

[11] H. Liu, "Reliability of a load-sharing k-out-of- n:g system: Non-iid components with arbitrary distributions," IEEE Trans. Reliability , vol. 47, no. 3, pp. 279–184, September 1998.

[12] L. Huang, and Q. Xu, "Lifetime reliability for load-sharing redundant systems with arbitrary failure distribution," IEEE Trans. Reliability , vol. 59, no. 2, pp. 319–330, June 2010.

[13] Amin, A., Grunske, L., Colman, A., 2013. An approach to software reliability prediction based on time series modeling. Journal of Systems and Software, 86 (7): 1923-1932.

[14] Anwar, S., Ramzan, M., Rauf, A., Shahid, A.A., 2010. Software Maintenance Prediction Using Weighted Scenarios: An Architecture Perspective. In: 2010 International Conference on Information Science and Applications (ICISA), Seoul, South Korea, 1-9.

[15] Hamlet, D., 2009. Tools and experiments supporting a testing-based theory of component composition. ACM Transactions on Software Engineering and Methodology (TOSEM), 18 (3): 12-52.

[16] Hou, C.Y., Chen, C., Wang, J.S., Shi, K., 2015. A scenario-based reliability analysis approach for component-based software. IEICE Transactions on Information and System, E98-D(3): 617-626, 2015.

[17] Goseva-Popstojanova, K., Trivedi, K.S., 2001. Architecture-based approach to reliability assessment of software systems. Performance Evaluation, 45 (2): 179-204.

[18] M. Finkelstein, "On systems with shared resources and optimal switching strategies," Rel. Eng. Syst. Safety, vol. 94, no. 8, pp.1358–1362, 2009.

[19] H. Kim and P. Kvam, "Reliability estimation based on system data with an unknown load share rule," Lifetime Data Ana., vol. 10, no. 1, pp.83–94, 2004.

[20] C. Park, "Parameter estimation for the reliability of load-sharing systems," IIE Trans., vol. 42, no. 10, pp. 753–765, 2010.

[21] C. Park, "Parameter estimation from load-sharing system data using the expectationmaximization algorithm," IIE Trans., vol. 45, no. 2, pp. 147–163, 2013.

[22] B. Singh, K. K. Sharma, and A. Kumar, "A classical and bayesian estimation of a k-components load-sharing parallel system," Comput. Statist. Data Anal., vol. 52, no. 12, pp. 5175–5185, 2008.

[23] D. R. Cox, "Regression models and life tables (with discussion)," J. Royal Soc., Series B, vol. 34, no. 2, pp.187220, 1972.

[24] Q. Zhang, C. Hua, and G. Xu, "A mixture of Weibull proportional hazard model for mechanical system failure prediction utilising lifetime and monitoring data," Mechanical Systems and Signal Processing, vol. 43, pp. 103-112, 2014.

[25] R. Mohammad, A. Kalam, and S. V. Amari, "Reliability of load-sharing systems subject to proportional hazards model,", In: Proc. of Reliability and Maintainability Symposium (RAMS), Orlando, FL, pp. 1-5, 2013.

[26] P.K. Kapur, H. Pham, A.G. Aggarwal, and G. Kaur, "Two dimensional multi-release software reliability modeling and optimal release planning," IEEE Transactions on Reliability, vol. 61, no. 3, pp. 758-768, 2012.