Method for Evaluation of Quality Properties in SaaS Rejuvenation using Markov Model

Huynh Quyet-Thanh, Nguyen Ngoc-Dung, Nguyen Hung-Cuong
School of Information and Communication Technology
Hanoi University of Science and Technology
Hanoi, Vietnam

Abstract - Software fault-tolerance techniques have been widely used in computing systems to achieve high level of quality. Rejuvenation, a modern software fault-tolerance technique, has attracted a large number of researchers in software engineering area. Evaluating the effectiveness and feasibility of this technique becomes extremely important in selecting, comparing and applying it in actual software systems. The study of important-quality-attributes is the scientific basis for assessing the performance of software fault-tolerance techniques. This paper presents availability, reliability, safety evaluation of rejuvenation systems. Derived mathematical relations between failure probabilities and modeling parameters enable us to gain a great deal of quantitative results.

Keywords—software reliability, software availability, software safety, Markov chain

I. INTRODUCTION

Nowadays, computer science has more and more application in human life, from economic to society, from education to medicine. So there is a requirement that developer has to build a high quality system to support user work. Most regular properties of software quality are reliability, availability and safety, that are studied in many fields: in component based by Larsson [1], consider maintenance and security issues by Xiong [2], base on properties and architecture of system by Roshandel [3], etc… The reliability relates with the correctness of result of work, whereas the availability ensures that system is ready to serve and the safety minimize the probability that a serious accident occurs in running time. There are two main approaches to analysis those properties of fault-tolerant software: practical testing and modeling. Results of practical testing are more believable than those from modeling, which is more well-known. However, testing can only establish the presence of errors but cannot assure their absence. Also, for highly dependable systems, the testing method is not always feasible and tends to be expensive to implement and then, to obtain statistically significant results.

Since the concept of fault-tolerant software was presented so far, many techniques has been proposed and applied successfully in practice. Rejuvenation (preventive maintenance - PM) is a new software fault tolerance technique, which Y.Huang was proposed in 1995 [2] and now it has attracted the interest and the research of many scientists [2], [3], [4], [5]. Assessing effectiveness and feasibility of this technology becomes extremely important in choosing, comparing and applying it to practical software systems.

Markov chain (more specific: discrete-time Markov chain) is a stochastic math system, containing a set of finite (or countable) number of possible states and a set of transitions between two of them. Given the past and present circumstance of Markov chain system, future behavior only depends on the present state and not on the past one. This model has large number of applications in natural science.

There are several techniques to evaluating quality properties of computer system with different approaches [6,7,8]. Thus, based on advantages of Markov chain model, this study introduces a model and applies it to evaluate the quality attributes of rejuvenation-software systems.

This paper is organized as follow: after this introduction section, section 2 explains definition of three aspects of software quality: reliability, availability and safety. Next, section 3 introduces rejuvenation - a software fault-tolerance technique. Section 4 proposes a method for evaluation those quality properties in rejuvenation systems using Markov model. Then section 5 shows experimental result of proposed method in simulation experiments. In this section we present also the experiment results in real system – BKOJ software, run as SaaS in the BKCloud system. Section 6 discusses some related and future problem to extend current work.

II. BASIC ASPECTS OF SOFTWARE SYSTEM QUALITY

Larsson [1] introduce dependability is the main quality attribute of safety-critical system development. Although software quality has complex meaning and depends on many problems, there are three aspects that are discussed following.

A. Reliability

Definition of reliability is based on probability that a system will fail in a specific period of time in given context and can be reflected by mean time to failure (MTTF) equation:

\[
\text{Reliability}(A) = \frac{1}{P_f(A)}
\]  

(1)

While A is a module and \( P_f(A) \) is a probability that this module fails per time unit. Being an important property of system, reliability is focused widely by researchers: Roshandel [3], Hoang P. [4], etc… It often used as an indicator for software release policy and can be got by using practical analysis of math models. Roshandel [3] introduce technique to calculate system reliability from this property of element. Hoang P. [4] summaries some statistical models and focuses on NHPP models.
B. Availability

Although problems of availability is larger than reliability, Xiong [2] notes that availability is a probability that system is ready-for-work in given time and given environment. Relate with reliability attribute, Larsson [1] introduce formal calculation:

\[ \text{Availability}(A) = \frac{\text{MTTF}(A)}{\text{MTTF}(A) + \text{MTTR}(A)} \]  

(2)

While MTTR is mean time to repair. This attribute of system has high commercial contribution: users will satisfy if they can use product service at every time. The difference between reliability and availability is that availability dependson the dynamic state of the system.

C. Safety

Larsson [1] consider software safety as an attribute that relates with the interaction between the system and the environment. It is a full-system property, either a component or an assembly property. Safety depends on where and how the system is deployed, in other way is dependent on the environment of system, so a top-down approach should be used in analyzing process. Safety of system is more important in the safe-critical systems, which will cause heavy damage to people or environment if they encounter a failure.

D. General method to evaluate fault-tolerant systems

Authors K. S. Trivedi and Goseva-Popstojanova [5, 6] have proposal to use Markov model in evaluating fault-tolerant system:

Step 1. Markov model implementation

In the first step, the Markov state map is being developed by identifying the status of the system and the transition between states.

Step 2. Building Chapman-Kolmogorov equations

In the second step, the Markov state chart, which has been developed, is being converted to a collection of the Chapman-Kolmogorov equations to find the matrix of transition state probability of the system.

Step 3. Solving Chapman-Kolmogorov equations

Solving Chapman-Kolmogorov equations is relatively complex. Some current resolutions such as analytics analysis, Laplace-Stieltjes transform or use ODEs in Matlab can simplify this task.

Step 4. Calculating and assessing the attributes of the fault-tolerant software

With each specific system, the software attributes will be evaluated according to specific parameters. This is general mechanism when using Markov chain in modeling. Real application depends on properties of environment, context and meaning.

III. SOFTWARE REJUVENATION

When software applications execute continuously for long periods of time (scientific and analytical applications run for days or weeks, servers in client-server systems are expected to run forever), the processes corresponding to the software in execution age or slowly degrade with respect to effective use of their system resources. The causes of process aging are: memory leaking, unreleased file locks, file descriptor leaking, data corruption in the operating environment of system resources, etc. Process aging will affect the performance of the application and eventually cause the application to fail [2].

If an application is developed in a perfect development environment and it operates correctly in the scenario work, the implementation process associated with this application will not be aging. However, practical software systems rarely are perfect. Therefore, their processes will be aging in the operating environment. The process aging and the software aging are fairly different. Software aging is related to source program, which will be inappropriate when requirements and maintenance are changing after many years. On the contrary, process aging is related to the decrease of application functions after several working days or weeks.

![Figure 1. Status model of Rejuvenation system](image-url)
interested in the other errors, which are considered as they occur and are repaired immediately, do not decrease the reliability of the system. When the system encounters serious error, all requests will be canceled and the system will become unsafe (state $B$), then evoke the error-recovery process.

![Error Recovering Operating Preventive Maintaining](image)

**Figure 2. Status and behavior of rejuvenation system**

We consider two different policies, which determine the time to perform preventive maintenance:

- **Policy I.** Purely time based: Preventive maintenance is initiated after a constant time $\delta$ has elapsed since it was started (or restarted).
- **Policy II.** Instantaneous load and time based: The actual preventive maintenance interval is determined by the sum of preventive maintenance wait and the time it takes for the queue to get empty from the point onward.

Let $\pi_i$ be the steady state probability that software is in state $i (i \in \{A, B, C\})$. From the well-known relation $\pi = \pi P$, we have:

$$\pi = [\pi_A, \pi_B, \pi_C] = \frac{1}{2} \lambda [2 \pi_A - 2 \pi_B + 2 \pi_C]$$

(3)

Let $U$ be a random variable denoting the sojourn time in state $A$ with its expectation $E[U]$. The steady state availability can be given as:

$$A_{SS} = \Pr\{\text{System is in state } A\} = \frac{\pi_A E[U]}{\pi_A E[U] + \pi_B Y_f + \pi_C Y_r}$$

(4)

Substituting the values of $\pi_A, \pi_B, \pi_C$:

$$A_{SS} = \frac{P_{AB} Y_f + P_{AC} Y_r}{E[U]}$$

(5)

The steady state safety can then be obtained as followed:

$$S = 1 - \Pr\{\text{System is in state } B\}$$

(6)

And:

$$S = 1 - \frac{\pi_B Y_f}{\pi_A E[U] + \pi_B Y_f + \pi_C Y_r}$$

(7)

Substituting the values of $\pi_A, \pi_B, \pi_C$:

$$S = 1 - \frac{E[U]}{E[U] + P_{AB} Y_f + P_{AC} Y_r}$$

(8)

In policy I, system is surveyed in the period $(0, \delta]$, so average reliability can be obtained as:

$$R_{m1} = \frac{1}{\delta} \int_0^\delta \sum_i p_i(t) dt$$

(9)

In policy II, system is surveyed in the period $(0, \infty]$, so average reliability can be obtained as:

$$R_{m1l} = \lim_{T \to \infty} \frac{1}{T} \int_0^T \sum_i p_i(t) dt$$

(10)

**Figure 3. Markov process with policy I**

**B. Policy I.**

$$\frac{dp_0(t)}{dt} = \mu(t) - [\lambda + \rho(t)] p_0(t)$$

(11)

$$\frac{dp_i(t)}{dt} = \mu(t) p_{i+1}(t) + \lambda p_{i-1}(t) - [\mu(t) + \lambda + \rho(t)] p_i(t)$$

(12)

$$\frac{dp_k(t)}{dt} = \lambda p_{k-1}(t) - [\mu(t) + \rho(t)] p_k(t)$$

(13)

$$\frac{dp_{k'}(t)}{dt} = \rho(t) p_{k'}(t)$$

(14)

For $\mu(t) = \mu(L(t))$ and $\rho(t) = \rho(L(t))$, where $L(t)$ is defined by:

$$L(t) = \int_0^t \sum_i c_i p_i(\tau) d\tau$$

(15)

The set of ODEs is first augmented by the following differential equation:

$$\frac{dL(t)}{dt} = \sum_i c_i p_i(t)$$

(16)

**Figure 4. Markov process with policy II**
The initial conditions: \( p_i(0) = 1, p_j(0) = 0 \) for \( 1 \leq i \leq L \) and \( p_i(0) = 0 \) for \( i' \leq k' \). Then

\[
P_{AB} = \sum_{i=0}^{K} p_i(\delta)
\]

(17)

And

\[
P_{AC} = 1 - P_{AB}
\]

(18)

The expected sojourn time in state \( A \) is given by:

\[
E[U] = \int_{t=0}^{\delta} \left[ \sum_{i=0}^{K} p_i(t) \right] dt
\]

(19)

C. Policy II

In this case, we need to distinguish between \( \leq \delta \) and \( t > \delta \), as policy II assumes that preventive maintenance will be initiated if and only if the buffer is empty after time \( \delta \) has elapsed. Similar to policy I, on step transition probability \( P_{AB} \) is computed by solving the system of ODEs at \( t = \infty \) and is given as:

\[
P_{AB} = \sum_{i=0}^{K} p_i(\infty)
\]

(20)

Then

\[
P_{AC} = 1 - P_{AB} = p_0(\infty)
\]

(21)

The mean sojourn time in state \( A \) is now given by:

\[
E[U] = \int_{t=0}^{\delta} \left[ \sum_{i=0}^{K} p_i(t) \right] dt + \int_{t=\delta}^{\infty} \left[ \sum_{i=0}^{K} p_i(t) \right] dt
\]

(22)

\[
E[U] = \int_{t=0}^{\delta} p_0(t) dt + \int_{t=\delta}^{\infty} \left[ \sum_{i=0}^{K} p_i(t) \right] dt
\]

(23)

V. RELIABILITY, AVAILABILITY AND SAFETY EVALUATION BY SIMULATION EXPERIMENTATION

The models are solved for multiple values of \( \delta \) and optimum value is determined. Using programming solution tool in MATLAB, we can estimate Chapman-Kolmogrov equations, thereby simulating the variability of \( \text{MTTF} \), \( \text{MTBF} \) and the upper bound of response time \( T_{res} \) with system parameters.

Model parameters: \( \gamma_f = 0.85 \text{(h)}; \lambda = 6.0 \text{(h}^{-1}); k = 50; \text{MTTF} = 240 \text{(h)} \text{Where h = hours.}

A. Simulation experiment I

In this experiment, \( \gamma_f \) is varied to ascertain the effect on the measures and on optimal \( \delta \). Service rate and failure rate are assumed to be functions of real time, i.e., \( \mu(t) = \mu (t) \text{ and } \rho(t) = \rho(t) \), where \( \rho(t) = \beta \alpha \gamma \), which is the hazard function of Weibull distribution. \( \alpha \) is fixed at 1.5 and \( \beta \) is calculated from \( \alpha \) and the MTTF as:

\[
\beta = \left[ \frac{\Gamma (1 + \frac{1}{\alpha})}{\text{MTTF}} \right]^{\frac{1}{\alpha}}
\]

(24)

And \( \mu(t) \) is defined as:

\[
\mu(t) = \frac{\mu_{\text{max}}}{\mu_{\text{min}}} \left( 1 - \frac{1}{\text{MTTF}} \right) \text{if } t \leq a
\]

(25)

\[
\mu(t) = \mu_{\text{min}} \text{if } t > a
\]

Where

\[
\beta = \frac{\mu_{\text{max}} - \mu_{\text{min}}}{\mu_{\text{max}} \text{MTTF}}
\]

(26)

\( \mu_{\text{max}} = 15 \text{h}^{-1}; \mu_{\text{min}} = 5 \text{h}^{-1} \). Under both polices, it can be seen that the higher the value of \( \gamma_f \), the lower is the availability for any particular value of \( \delta \).

![Figure 5. Availability under policy I](image)

![Figure 6. Availability under policy II](image)

Figure 7 and Figure 8 show that safety will decrease when increasing the value of parameter \( \delta \), while safety increases when raising the value of parameter \( \gamma_f \). Since then, we can comment that under the policy I, the sooner the preventive maintenance will be conducted, the safer the system will be.

![Figure 7. Safety under policy I where \( \gamma_f = 0.35 \)](image)
The safety of system under policy II will increase with the decrease of $\gamma_r$ (Figure 9). However, the dependency is relatively small. In addition, the safety will reduce rapidly along with the increase of $\delta$ to a threshold (marked on the drawings) and then will be almost unchanged. From theoretical calculation, we can see the average reliability of the system does not depend on $\gamma_r$. Therefore, we fix the value $\gamma_r = 0.55$ and survey the influence of the reliability on the time to wait to perform preventive maintenance $\delta$. The average reliability of system under policy I rises with the decrease of parameter $\delta$ (Figure 10). Clearly, under policies I, the sooner the preventive maintenance is conducted, the higher the level of reliability of system is kept. Under policy II, the reliability is calculated throughout the time domain. It can see that the reliability will increase along with the increase of $\delta$ to a threshold and then become almost unchanged.

B. Simulation experiment II.

In this experiment, $\gamma_r$ is fixed at 0.15; $\alpha$ is an assigned value of 1.0, 1.5 and 2.0, respectively.

For $\alpha = 1$, the time to failure has an exponential distribution, which, because of its no-memory property, contradicts aging. It is better not to perform Rejuvenation in this case if the objective is to maximize availability. For other two values of $\alpha$, however, rejuvenation maximizes availability at certain $\delta$. For a specific policy, the bigger the failure density, i.e., higher the value of $\alpha$, the higher is the maximum steady state availability. Also, with higher values $\alpha$, this maxima occurs at lower values of $\delta$.

C. Experimental results

In this section we will show some experimental result of the proposed method in a real application such as Online Judge system.
Online Judge

An online judge is an automated judge which checks a submitted solution for an existed problem and generates the output. It checks if the generated output was correct with respect to the output set that is saved as a full proof judge output set for that particular problem thus generate the result for the user such as Accepted, Wrong Answer, Runtime Error, etc.

An online judge is in general a server, which contains descriptions of problems from different contests, as well as data sets to judge whether a particular solution solves any of these problems. A user from anywhere in the world can register himself (or herself) with an online judge for free and solve as many problems as he likes. He can send as many solutions as he want till receiving satisfactory information, not only about the verdict, but also about the time that the code takes to run after improving the program and/or the algorithm used to solve the selected challenge. One of the main distinctive trait of the online judges is that they allow the users this self-competitive behavior to learn informatics, not only algorithms but also programming.

There are several existing popular online judges all over the World Wide Web. Here mentioned some of them: UVA Online Judge, Sphere Online Judge, and BKOJ Online Judge.

BKOJ Online Judge

BKOJ is an online judge of Ha Noi university of Science and Technology. It was built with the primary purpose of being used as a training tool for ACM /ICPC teams of the university.

BKOJ system consists of two major parts: Web (as Frontend) and Core (as Backend). The Web part plays as distributed information management system, managing information such as user registration, problem submission, solution submission, problem modification, etc.

The Core part as kernel of BKOJ system provides a method to judge all solutions, which were submitted by any users. In this paper, we only focus on the functions of the Core part.

BKOJ system has been installed in BKCloud platform as a software service running from 2012. Figure 15 shows its deployment in the platform.

The Core part as kernel of the system provides a method to judge all solutions, which were submitted by any users. In
this discussion, we only focus on the functions of the Core part.

Figure 15. BKOJ deployment in the BKCloud system

3) BKOJ - Core working process

Figure 16. The judgment process of BKOJ - Core

4) The experimental results in detail

In this experiment, we only consider applying the Policy I to BKOJ. We created a virtual contest with the simultaneous participation of 500 virtual contestants during 8 hours. The contest using the data collected from many private contests of our university – HUST - from 2012 to 2013.

Some basic information about the contest is shown in the following table:

<table>
<thead>
<tr>
<th>Server Software</th>
<th>Apache/2.2.14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document Path</td>
<td>/JudgeOnline/status.php</td>
</tr>
<tr>
<td>Concurrency Level</td>
<td>500</td>
</tr>
<tr>
<td>Time taken for tests</td>
<td>32.239 seconds</td>
</tr>
<tr>
<td>Complete requests</td>
<td>1000</td>
</tr>
<tr>
<td>Failed requests</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td>(Connect: 0, Receive: 0, Length: 207, Exceptions: 0)</td>
</tr>
<tr>
<td>Total transferred</td>
<td>7262294 bytes</td>
</tr>
<tr>
<td>HTML transferred</td>
<td>6956186 bytes</td>
</tr>
<tr>
<td>Requests per second</td>
<td>31.02<a href="mean">#/sec</a></td>
</tr>
<tr>
<td>Time per request</td>
<td>16119.251<a href="mean">ms</a></td>
</tr>
<tr>
<td>Time per request</td>
<td>32.239[ms]</td>
</tr>
<tr>
<td></td>
<td>(mean, across all concurrent requests)</td>
</tr>
</tbody>
</table>

Based on the information mentioned in the above table, we drew the continuous theoretical curves of the Availability, the Safety and the Reliability of BKOJ as in Figures 17 and 18.
respectively. In contrast, the black discrete points represent the actual values of the relating properties of BKIOJ. Model parameters: \( y_r = 0.5 \) (h); \( y_f = 0.5 \) (h); \( \lambda = 6.0 \text{(h}^{-1}) \); \( k = 30 \); \( MTTF = 240 \text{(h)} \); \( \mu(t) = \gamma r(t) \) and \( \rho(t) = \gamma f(t) \) are the same as (24)-(25)-(26); \( \alpha \) is fixed at 1.0; \( \delta \) is assigned values of 96(h), 144(h), 192(h), 384(h) respectively (where h = hours).

Figure 188. Availability of BKIOJ under policy I

Figure 199. Safety of BKIOJ under policy I

Figure 20. Reliability of BKIOJ under policy II

The experimental results show that the theoretical curves fit quite well with limited practical value, which confirmed the practical value of the method for evaluating the quality properties in a rejuvenation system using Markov model.

VI. CONCLUSIONS

Applying the theory of mathematical Markov model, the theory of rejuvenation, we have built a model to evaluate the software attributes of rejuvenation systems. The proposed approach used two Markov chain models with corresponding policy I and II. We showed expanding math calculation of model of this method by using the Matlab. The experiments with BKIOJ SaaS on BkCloud system are confirmed its worth.

In the future, based on the relationship between these software attributes and fault tolerance techniques on cloud environment, the research will be further developed. From the evaluation of the software attributes of fault-tolerant software in cloud environments, we can deliver the construction cost in rejuvenation-applied software. In addition, we can use the obtained results in the evaluation of the software attributes of the rejuvenation systems to study about client-server systems with \( K \) queues \((K > 1)\) and distributed fault-tolerant software.

ACKNOWLEDGMENT

This research is sponsored by the research Grant KC.01.01/10-15 by Ministry of Science and Technology Vietnam

VII. REFERENCES