Software Health Measurement Method Based on Aging-related Bugs

You-zhi Zhai¹, Qiu-ying Li²,* and Hang-chao You²

School of Reliability and Systems Engineering, Beihang University, China
Science & Technology on Reliability & Environmental Engineering Laboratory, Beijing 100191, China

*Corresponding author

Keywords: Software health measurement, Software health characterization elements, Gaussian mixture model, Aging-related bugs.

Abstract. Aging-related bugs can lead to software aging, whose performance is the slow response on software system though the software does not crash. This paper aims to propose the measurement method of software health to quantitatively show the impact of aging-related bugs on software. The effect of the bugs and the elements were analyzed. The Gaussian mixture model was developed to measure the degree of software health, which provides the basis for the software failure alarming and failure prediction. Finally, the details of the method were shown by a case study on Hadoop Distributed File System.

Introduction

Software defects are divided into three types by Trivedi and Vaidyanathan, including Bohrbugs, Heisenbugs and Aging-related bugs[1-2]. Bohrbugs is essentially a permanent (unchanging) design error that is easily reproduced and recognized in the software lifecycle. Heisenbugs is an instantaneous internal fault and it is not reproduced and identified easily. Aging-related bugs can’t cause failure immediately, but it causes the system to run out of resources, the fragmentation of storage space, data corruption or numerical error accumulation, which results in software aging. Software aging cause software degrade performance over a long period of continuous execution, and even leads to a sudden crash of software[3]. Software aging can cause system performance degradation, which often cause huge losses in the industrial, technological and military fields [4-6]. Due to the aging is inevitable, people can only minimize the number of software failures occurring. But how to alert software aging that help users identify the health of software before failure, it is a problem to be solved.

Software Health Measurement Method

Aging-Related Bugs and Software Health

The occurrence of software aging-related bugs is due to the accumulation of error conditions (such as rounding errors of program variables) or changes in operating environment during system interaction process. The wrong conditions do not immediately lead to failure, and there is a change process or even a delayed process. This defect causes software system to use the depletion of resources and leads to data corruption, error accumulation, which make software aged, it ultimately leads to paralysis of system services. It will bring software performance problems, making the system gradually collapse. In addition, the cumulative error caused by aging-related bugs could cause fatal effects in security tasks. The phenomenon of software aging has its inevitability [8].

We use software health to measure the functions of software in recession which is threatened by aging-related bugs. The occurrence of these bugs could be observed from the external behavior, as well as the performance of the operating environment. As a result, it is possible to predict and mitigate
failures before software fails. Thus software health refers to measure the ability of software to complete tasks and it refers to a software state affected by aging-related bugs not a failure.

**Elements of Software Health Characterization**

The elements of software health characterization refer to the expression of system resources occupied by software at runtime. When aging-related bugs are activated, software tends to have some external exception events, such as software response time and memory usage. Therefore, software health characterization elements can be selected from factors that affect system’s performance. These system resources can be selected as elements of software health characterization, including CPU, memory, cache and disk usage. Table I summarizes the elements of software health characterization.

<table>
<thead>
<tr>
<th>Elements of Software Health Characterization</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
</tr>
<tr>
<td>CPU utilization</td>
</tr>
<tr>
<td>Idle CPU load</td>
</tr>
<tr>
<td>Number of processes</td>
</tr>
<tr>
<td>CPU average load</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Memory usage</td>
</tr>
<tr>
<td>Memory avail capability</td>
</tr>
<tr>
<td>Exchange space usage</td>
</tr>
<tr>
<td>Cache and Disk</td>
</tr>
<tr>
<td>Disk usage</td>
</tr>
<tr>
<td>SWAP folder used</td>
</tr>
<tr>
<td>File I/O</td>
</tr>
</tbody>
</table>

**The Degree of Software Health**

We use software health degree to quantitatively describe the software health. It is a quantitative evaluation of software health which represents percentage of software health, expressed as $h$:

$\forall H_x(t) = (x_1(t), x_2(t), \ldots, x_n(t))$, $h = f(H_x(t))$  

(1)

Where $f(*)$ is a software health function, $H_x(t)$ is the set of software health characterization elements, the ranges of $h$ is $[0,1]$. As there is a delay process from an aging-related bug to a failure, the health level is not only normal or failed, but there is also a middle state, we call it sub-health, which is a state aging-related bugs effect. When degree $h=1$, it means software is running normally; when $h = 0$, it means software failed; when $h$ is between $(0,1)$, it means the stability and availability of software gradually decreased. Thus the curve[9] of software degeneration from a healthy state to a failure is shown in Figure 1.

![Figure 1. The health curve of software in running.](image-url)
Measurement Method

The Gaussian mixture model is an essentially multidimensional probability density function [10-11]. The specific method is as follows.

The elements are expressed as \( X = (X_1, X_2, \ldots, X_i, \ldots, X_n) \), \( n \) is number of elements selected, \( X_i \) represents the \( i \)-th health characterization element, \( 1 \leq i \leq n \); the vector \([x_{i1}, x_{i2}, \ldots, x_{ij}, \ldots, x_{im}]\) represents a set of observed values for the \( i \)-th element, where \( x_{ij} \) represents the \( j \)-th observed value for \( X_i \), \( 1 \leq j \leq m \), \( m \) indicates the total number of observation for the elements. Let the variable \( x_i \) denote the \( i \)-th element, \( p_i(x_i|\mu_i, \sigma_i^2) \) denotes the probability density of Gaussian distribution of the \( i \)-th element, \( \mu_i \) represents the mean value, and \( \sigma_i^2 \) represents the variance.

\[
p_i(x_i|\mu_i, \sigma_i^2) = \frac{\exp\left\{-\frac{1}{2}(x_i - \mu_i)^2/\sigma_i^2\right\}}{(2\pi)^{1/2}\sigma_i}
\]

(2)

The probability density is expressed as:

\[
G(x_1, x_2, \ldots, x_i, \ldots, x_n) = \sum_{i=1}^{n} w_i p_i(x_i, \mu_i, \sigma_i^2)
\]

(3)

Where \( w_i \) is the weight of the \( i \)-th element. We should collect data of two states, the expected normal state and the metric current operation state. The parameters \( \mu_i \), \( \sigma_i^2 \) and \( w_i \) are estimated by using the data of two states separately. The probability density of the Gaussian mixture models corresponding to the two states are obtained as \( G_1(x_1, x_2, \ldots, x_n) \) and \( G_2(x_1, x_2, \ldots, x_n) \), Let \( G_1(x_1, x_2, \ldots, x_n) \) express the model of an sub-health state and \( G_2(x_1, x_2, \ldots, x_n) \) is the model of a health state.

To estimate the value of \( \mu_i \) and \( \sigma_i^2 \) by MLE, a set of observed values of \( X_i \), as \([x_{i1}, x_{i2}, \ldots, x_{ij}, \ldots, x_{im}]\), is used as follows. The logarithmic likelihood function is shown in (4):

\[
L(\mu, \sigma^2 | x_i) = \ln \prod_{j=1}^{m} p_i(x_{ij}, \mu_i, \sigma_i^2) = -\frac{m}{2}\ln(2\pi) - m \ln \sigma_i - \frac{1}{2}\sum_{j=1}^{m} \frac{(x_{ij} - \mu_i)^2}{\sigma_i^2}
\]

(4)

To calculate partial derivative of \( \sigma_i^2 \) and \( \mu_i \) in function (5):

\[
\frac{\partial L(\mu, \sigma^2 | x_i)}{\partial \mu} = \sum_{j=1}^{m} \frac{x_{ij} - \mu_i}{\sigma_i^2} = 0
\]

(5)

\[
\frac{\partial L(\mu, \sigma^2 | x_i)}{\partial \sigma_i^2} = -\frac{m}{2\sigma_i^2} + \frac{1}{2}\sum_{j=1}^{m} \frac{(x_{ij} - \mu_i)^2}{2\sigma_i^4} = 0
\]

(6)

According to unbiasedness, the results of parameter estimation of \( \mu_i \) and \( \sigma_i^2 \) are as follows:

\[
\hat{\mu}_i = \frac{\sum_{j=1}^{m} x_{ij}}{m}, \quad \text{and} \quad \hat{\sigma}_i^2 = \frac{\sum_{j=1}^{m} (x_{ij} - \hat{\mu}_i)^2}{m - 1}.
\]

(7)

The standard data and the collected data are substituted into (7) and (8), we can estimate the value of \( \mu_i \) and \( \sigma_i^2 \) in both cases, as well as \( w_i \), thus we can get the Gaussian mixture model of two cases. Then the degree of software health \( h \) is obtained as follow.
\[
h = \frac{1}{\sqrt{\int_{0}^{C_1} \int_{0}^{C_2} G_1(x_1, x_2 \ldots x_n) \, dx_1 \, dx_2 \ldots \, dx_n} \sqrt{\int_{0}^{C_1} \int_{0}^{C_2} G_2(x_1, x_2 \ldots x_n) \, dx_1 \, dx_2 \ldots \, dx_n}}
\]

Let \( C = [C_1, C_2, \ldots, C_n] \) correspond to the maximum of elements \( x_1, x_2 \ldots x_n \). For example, \( C_i \) represents the maximum of the i-th health characterization element.

**Case Study**

The Hadoop Distributed File System (HDFS) [12-13] provides data storage for Hadoop. As the main effect of HDFS is data storage management between the client node and the system, thus we select memory of JVM as a health element of HDFS to conduct an experiment (the unit is MB).

In the experiment, we mainly compare the memory available two scenarios: 1) In scenario 1, we simulate normal accessing to HDFS, randomly generate load and collects data of the JVM memory available after running for some time. This scenario is a standard for software to be healthy; 2) In scenario 2, we increase the access to HDFS, simulating memory leakage, and accelerate HDFS software downturn by nbench[14], resulting in a rapid reduction of available memory in JVM. The interval is 10s, and 5500 data are collected. The data are shown in the following figures.

Using the data between [4500, 5000] of the two groups in Matlab, the parameters estimated are \( \theta_1 = \{ \mu, \sigma^2 \} = \{516.1781, 838.9070\} \) and \( \theta_2 = \{ \mu, \sigma^2 \} = \{488.8441, 779.4006\} \). The Single Gaussian model of Scenario 1 and Scenario 2 are:

\[
G_1(x) = \exp \left( \frac{(x - 516.1781)^2}{1677.814} \right) \frac{1}{72.5830}, \quad G_2(x) = \exp \left( \frac{(x - 488.8441)^2}{1558.8012} \right) \frac{1}{69.9617}
\]

(9)
The two models above are substituted into formula 8, and $h$ is 0.7936, which means HDFS is in sub-health state. To compare fig.2 to fig.3, there is a decline in the memory available of fig.3. We use the proximity of health degree in different moments to compare the change of HDFS health in fig 4.

![Figure 4. The health change curve of HDFS.](image)

By analysis of the above chart, we find in Scenario 2 that we increase memory leak speed through nnbench, the HDFS software health degree gradually decrease after the sample collection number of 4500 times, which indicates HDFS aging is serious. This is consistent with the experiment, at this time there is a clear response time longer, the operation of the system significantly slow down.

Conclusions

The Gaussian mixture model is suitable for training samples, but in the absence of failure data, the main idea of using Gaussian mixture model is to calculate the degree of "offset" of two Gaussian mixture models. If the health level obtained through the model fluctuates in a small range and the software is running without exception, we can update the training data set under normal circumstances or use the current health value as a health reference. For example, in this experiment, in a long period of time, the health value obtained is 0.95 up and down fluctuations, in practical applications, it can be considered as a healthy state if the value is more than 0.95. In addition, the results of the assessment can provide a reference for subsequent software health management. We can set a software health threshold as needed. If the software health degree is lower than the threshold, it shows the software is in an "unacceptable" state that we should take some measures to avoid further deterioration.

References


