Abstract

Fault tolerance by redundancy is a key technique for improving web services reliability. But, the reliability and fault tolerance are not well supported by Service-oriented architecture (SOA) conceptual model. Most of the existing analytical models to predict the reliability of web services assume their structures are static which do not conform to reality and the estimated reliability precision is lower. In this paper, first, an extended SOA model is proposed for improving the reliability of web services. Then, a reliability model is presented to evaluate the reliability of web services based on the birth-death process. Last, the reliability of web services is evaluated using simulation approach and a case study is designed, implemented and analyzed to support our model. The results of experiment show the feasibility, validity of our reliability model.

Keywords—Web Service Reliability; Birth-Death Process; SOA Conceptual Model

I. Introduction

Fault tolerance is considered to be an effective way to improve system reliability. In traditional software development, fault tolerance is expensive and only some crucial module can be designed by redundancy. However, at the present time, there are many function-equivalent web services in the Internet which are often provided by different service providers. So, it is practicable to upgrade the reliability of web services by redundancy.

Web services reliability is more hard to predict than general software system. There are several fault-tolerant reliability analysis approaches which have been proposed in [1][2][3][4] and etc, most of their reliability models are based on architecture, mainly considering the static structure and transition probability, without taking into account their dynamic change with time, so the precision is not higher. In fact, the reliabilities of services are influenced by many factors in the Internet environment and a failure can occur at any arbitrary time. The reliability of a service is not a constant but a function of time[5]. Fault tolerance is not well supported by SOA conceptual model, hence, in this paper, we extend the SOA conceptual model by redundancy, propose a fault-tolerant conceptual model and present a reliability model for estimating fault-tolerant web services based on birth-death process.

The rest sections of the paper are organized as follows: Section II describes a fault-tolerant reliability model. Section III describes the reliability prediction algorithm. Section IV does some experiments to verify our model. Section V discusses the conclusion and our future works.

II. Fault-Tolerant Reliability Model

A. Extended SOA Conceptual Model

The ESOACM (extended SOA conceptual model) is shown in Figure 1. The web services manager is responsible for all web services. Web services are defined as $WS = \{S, S_1, S_2, \ldots, S_n\}$. $S$ is the primary service and $S_1, \ldots, S_n$ is the backup of $S$. The web service manager will choose the best service as the primary service. When service requesters send a request for $WS$, the UDDI
III. Reliability Prediction Algorithm

In this section, we discuss how to design algorithm. The failure behavior of web services can be denoted by reliability, failure rate. First, we define some notations which are shown in Table I.

A. Reliability

Algorithm 1 simulates how to calculate the reliability if the failure behavior is denoted by reliability. It accepts as input the reliability of every service, the number of services including the primary and back-up services and returns the reliability of web services with fault tolerance. The $run\_count$ represents the total run times. For each run, if $r_i < x$ and $i < n$, the procedure will check next service. The service is failed if and only if $r_i < x$ and the state is $n$. The variable $fault\_count$ records the number of faults. This procedure repeats $run\_count$ times and the reliability can be computed just like in algorithm 1.

Algorithm 1 Getreliability1($r$, $n$)

1: Initialize some variables $run\_count$, $run$, $fault\_count$;
2: while $run \leq run\_count$ do
3: for $i = 1$ to $n$ do
4: Generate a random number $x$ in the range of [0, 1];
5: if $r_i < x$ then
6: if arrive the state $n$ then
7: the model failed and increase $fault\_count$ by 1;
8: else
9: transferred to next state;
10: end if
11: end if
12: increase $run$ by 1;
13: end for
14: $R = 1 - fault\_count/run\_count$;
15: end while
16: return $R$;

TABLE I. Notations of our algorithm

<table>
<thead>
<tr>
<th>Notations</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_i$</td>
<td>reliability of service $i$;</td>
</tr>
<tr>
<td>$R$</td>
<td>reliability of the Redundant web services;</td>
</tr>
<tr>
<td>$\lambda_i$</td>
<td>failure rate of web service $i$;</td>
</tr>
<tr>
<td>$\mu_i$</td>
<td>repair rate of web service $i$;</td>
</tr>
<tr>
<td>$time$</td>
<td>a variable of $[0, total_time]$;</td>
</tr>
<tr>
<td>$dt$</td>
<td>time step;</td>
</tr>
<tr>
<td>$flag$</td>
<td>failed flag of the process.</td>
</tr>
<tr>
<td>$fault_count$</td>
<td>total number of faults;</td>
</tr>
<tr>
<td>$run_count$</td>
<td>total number of runs;</td>
</tr>
<tr>
<td>$total_time$</td>
<td>the total execution time during a run;</td>
</tr>
</tbody>
</table>

Fig. 1. Extended SOA Conceptual Model

Fig. 2. State Transition Diagram
B. Constant Failure Rate

When failure behavior is denoted by constant failure rate. The stochastic failure procedure is described by a pure-birth process. Algorithm 2 simulates how to calculate the reliability of our model. For each time step $dt$, we compare $\lambda_i * dt$ with a floating random number $x$ between 0 and 1. If $\lambda_i * dt > x$, we say the $i$th service fails and turn to check the $(i+1)$th service by the same way until the $n$th service. Simulate the above procedure $run_count$ times to obtain the reliability of web services with fault tolerance which can be computed by the expression $R = 1 - fault_count/run_count$. The total time can be varied and the reliability with time can be computed just like in algorithm 2. From these, we can analysis the change trend of reliability with time.

Algorithm 2 GetReliability2($\lambda, n$)

1. initialize $run_count, dt, t, runflag, fault_count$
2. for all $total_time$ in a range do
3.   for $run = 1$ to $run_count$ do
4.     $time = 0, i = 1, flag = false$
5.     while $time < total_time$ and $flag$ is false do
6.       $time = time + dt$
7.       generate a random number $x$ in range of $[0, 1]$
8.       if $\lambda_i * dt > x$ then
9.         if the state is transferred to $i+1$
10.        $fault_count[total_time] ++$
11.       else
12.         transferred to next state;
13.       end if
14.     end if
15.   end while
16. end for
17. $R(total_time) = 1 - fault_count[total_time]/run_count$
18. return array $R$

C. Constant Failure Rate and Repair Rate

When failure behavior is denoted by failure rate $\lambda_i$ and repair rate $\mu_i$, the procedure is a birth-death process. We presume the repaired time follows an exponential distribution with a parameter $\mu_i$. The simulation procedure is shown in Algorithm 3. Firstly, sample a time $t$ from exponential distribution with parameter $(\lambda_i + \mu_i)$; Secondly, the transition probability from state $i$ to state $i+1$ and to $i-1$ is $p_i = \lambda_i / (\lambda_i + \mu_i)$ and $1 - p_i$. Generate a random number $x$ from $\{0, 1\}$ with probability $p_i$, the state is transferred to $i+1$ when $x$ is 1 and to state $i-1$ when $x$ is 0. Repeat the above procedure for $run_count$ times and reliability can be computed just like in algorithm 3.

Algorithm 3 GetReliability3($\lambda, \mu, n$)

1. initialize $run_count, run, flag$ and array $fault_count$
2. for all $total_time$ in a range do
3.   for $run = 1$ to $run_count$ do
4.     $time = 0, i = 1, flag = false$
5.     while $time < total_time$ and $flag$ is false do
6.       sample time $t$ from an exponential distribution with a parameter $(\lambda_i + \mu_{i-1})$
7.       generate a random number $x$ from $\{0, 1\}$ with probability $p_i$
8.       if $x$ is 1 then
9.         the state is transferred to $i+1$
10.    else
11.       the state is transferred to $i-1$
12.    end if
13.   end while
14. end for
15. $R(total_time) = 1 - fault_count[total_time]/run_count$
16. return array $R$

D. Parameter Estimation

We discuss how to estimate the parameters in our algorithm. The reliability, failure rate and repair rate presented by service provider are not enough to be convincing. So, we reform it by collecting failure and repair information to improve accuracy. In our ESOACM, the server manager use log file to record services behavior information. For parameter reliability $r_i$, we assume that:
1) an initial value $r_{i,1}$ presented by service provider;
2) service invoked times $n$ and failure occurred times $f$ in an interval $(0, t)$;
3) tested times $N$ provided by service provider;
4) reliability estimated from log files $r_{i,2}$.

then the revised reliability is calculated by

$$r_i = \frac{N}{N + n} r_{i,1} + \frac{n}{N + n} r_{i,2}$$

(1)

where $r_{i,2} = 1 - \frac{f}{n}$. When the service is never called, the invoked times $n = 0$, the reliability $r_i = r_{i,1}$. If the service provider doesn’t present the initial reliability $r_{i,1}$, the reliability $r_i = r_{i,2}$. So, the revised reliability represents the static reliability information and the dynamic reliability information. Similar methods can be used for the estimation of $\lambda_i$ and $\mu_i$ which can be expressed by

$$\lambda_i = \frac{N}{N + n} \lambda_{i,1} + \frac{n}{N + n} \lambda_{i,2}$$

(2)
where $\lambda_2 = \frac{t}{M}$ and
\[ \mu = \frac{M}{M+k} \mu_{1,1} + \frac{k}{M+k} \mu_{1,2} \] (3)
where $\mu_{1,2} = \frac{k}{M}$ and $k$ is the repaired times in an interval $(0, t)$, $M$ is total repaired times.

IV. Experiments

A. Experiment 1

For algorithm III-A, we set the reliability of services $r = [0.5, 0.5, 0.6, 0.3, 0.6], n = 5$. The run times $run\_count$ varies from 100 to 1000 step by 100. Here, we set the time spent on a run is one time unit. So, the number of total run is identify to the times of total run. The results of our experiment are shown in Table II. Column 'Exp1' is the reliability obtained from algorithm III-A and column 'Zheng' is reliability according to Zheng's [3] method. From the table, we can discover that the reliability is very close to Zheng's result.

<p>| TABLE II. Results of Experiment 1 |</p>
<table>
<thead>
<tr>
<th>Run times</th>
<th>Exp1</th>
<th>Zheng</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.970</td>
<td>0.972</td>
</tr>
<tr>
<td>200</td>
<td>0.980</td>
<td>0.972</td>
</tr>
<tr>
<td>300</td>
<td>0.973</td>
<td>0.972</td>
</tr>
<tr>
<td>400</td>
<td>0.967</td>
<td>0.972</td>
</tr>
<tr>
<td>500</td>
<td>0.970</td>
<td>0.972</td>
</tr>
</tbody>
</table>

B. Experiment 2

For algorithm III-B, we randomly set failure rate $\lambda = [0.0025, 0.003, 0.02, 0.02, 0.02], n = 5$. The run_count is set to 1000 at every specified total_time. We also present the reliability of web services without redundancy. The reliability of single service is calculated by expression $r_i = e^{-\lambda_i t_i}, i = 1, 2, \ldots, n \lambda_i$ is the failure rate of service $i$ and the $t_i$ is the execution time spent in service $i$ during a run[6]. The reliability with fault-tolerant and the max reliability of single services (MRSS) without fault-tolerant is shown in table III. From the table we can say that the reliability of services by redundancy is higher than a single service and the reliability will approach to zero with the time increased.

<p>| TABLE III. Results of Experiment 2 |</p>
<table>
<thead>
<tr>
<th>Run times</th>
<th>Exp2</th>
<th>MRSS</th>
<th>Run times</th>
<th>Exp2</th>
<th>MRSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.997</td>
<td>0.779</td>
<td>600</td>
<td>0.654</td>
<td>0.223</td>
</tr>
<tr>
<td>200</td>
<td>0.972</td>
<td>0.607</td>
<td>700</td>
<td>0.539</td>
<td>0.174</td>
</tr>
<tr>
<td>300</td>
<td>0.918</td>
<td>0.472</td>
<td>800</td>
<td>0.456</td>
<td>0.135</td>
</tr>
<tr>
<td>400</td>
<td>0.814</td>
<td>0.3678</td>
<td>900</td>
<td>0.402</td>
<td>0.105</td>
</tr>
<tr>
<td>500</td>
<td>0.743</td>
<td>0.287</td>
<td>1000</td>
<td>0.307</td>
<td>0.082</td>
</tr>
</tbody>
</table>

C. Experiment 3

To verify the validity of our model and to analyze simply in math, for algorithm III-C, we consider two services with same failure rate and repair rate, the parameter $\lambda = [0.003, 0.003], \mu = [0.5, 0.5]$. The results obtained from our experiment and from the system of differential equations (SDE) of stochastic process are shown in table IV. From the table, we can say that the results of two methods are very close.

<p>| TABLE IV. Results of Experiment 3 |</p>
<table>
<thead>
<tr>
<th>Run times</th>
<th>Exp3</th>
<th>SDE</th>
<th>Run times</th>
<th>Exp3</th>
<th>SDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.996</td>
<td>0.998</td>
<td>600</td>
<td>0.992</td>
<td>0.989</td>
</tr>
<tr>
<td>200</td>
<td>0.995</td>
<td>0.997</td>
<td>700</td>
<td>0.992</td>
<td>0.989</td>
</tr>
<tr>
<td>300</td>
<td>0.996</td>
<td>0.995</td>
<td>800</td>
<td>0.990</td>
<td>0.986</td>
</tr>
<tr>
<td>400</td>
<td>0.990</td>
<td>0.993</td>
<td>900</td>
<td>0.991</td>
<td>0.984</td>
</tr>
<tr>
<td>500</td>
<td>0.989</td>
<td>0.991</td>
<td>1000</td>
<td>0.986</td>
<td>0.982</td>
</tr>
</tbody>
</table>

V. Conclusions

An extended SOA conceptual model for reliability enhancement by redundancy is presented in this paper and a birth-death process to estimate the reliability of our model is proposed. The birth-death process model is more precision because the information from service provider and log files are both considered. But, there are some details that we have not taken into account. So, our future work will consider other factors that cause composite services failure based on other architecture.

References