A Simulation-Based Reliability Analysis Approach of the Fault-Tolerant Web Services

Yanjun Shu, Zhibo Wu, Hongwei Liu and Yixiang Gao
School of Computer Science and Technology
Harbin Institute of Technology
Harbin, China
{yjshu, wzb, liuhw}@hit.edu.cn, fanofowen@gmail.com

Abstract — Service-oriented computing is emerging as a new way to developing the extensible computing system which evolves from the component-based software engineering. Reliability is an important factor for choosing, ranking and compositing web services. As there are many web services provide the same function on the internet, applying the fault-tolerant strategies is an effective way to improve the reliability. However, the existing reliability analysis approaches for web services are seldom considering the fault-tolerant strategies. Simulation-based approaches have been used to the component-based software reliability analysis which can study the different architecture and the impact of various failure behaviors. In this paper, the algorithms of discrete-event simulation for common used fault tolerant strategies of web services are proposed. Based on these algorithms, the simulation-based reliability analysis approach is implemented. The ability of the simulation approach in complex failure scenarios is exhibited by several experiments.

Keywords - Web Service; Fault Tolerant; Reliability Model; Simulation

I. INTRODUCTION

Service-Orientation Architecture (SOA) has played an extremely important role in the evolution of Information Technology in the last decade [1]. It provides the foundations for some of today’s most significant advancements, such as Web 2.0, Cloud computing, and the Internet of Things. Reliability is a key issue of SOA which has been applied in many critical domains such as military or financial web services systems. The dynamic nature of internet environment and unclear remote web services make the reliability improvement more challengeable.

Fault-tolerant is an effective way to achieve high reliability in traditional software engineering. A few researchers have proposed some meaningful ways to apply the fault tolerant mechanism in the composite web services. Some fault tolerant strategies of web services have been proposed recently, such as FT-SOAP [2], FT-CORBA [3], FTWeb [4], Thema [5] and WS-ReliableMessage [6] etc. Contrast to the traditional software system, it is easier to apply fault tolerant strategies since there are a lot of similar or identical web services as redundant replications. Moreover, the web services composition is an efficient and cost-effective way to develop fault tolerant web services which can be combines simpler, loosely coupled, reusable web services by using service composition via the Business Process Execution Language (BPEL) [7].

Analysis the reliability of web services is very useful to evaluate design feasibility, compare design alternatives, and track reliability improvement. A lot of reliability prediction models of web services have been proposed. Most of these models are evolved from the traditional architecture-based software reliability analysis, such as SAMM [8] and SCA-ASM [9]. They always formulate many analytic equations to integrate the reliabilities of component services with the high level architecture of the component services. These reliability analytic formulations are very cumbersome and unsolvable sometimes. The discrete-event simulation offers an attractive alternative to analytical models as it can represent the impact of several strategies that may be employed during testing and different deployment configurations during operation. Although there are some simulation-based approaches of component software systems have been proposed, they are not suitable for fault tolerant web service applications.

The fault tolerant always uses redundancy to achieve the highly reliable web services which costs more computational resources or execution time. It is hard for developers to determine the fault tolerant strategy based on the service-oriented application. In this paper, the simulation algorithms for different fault tolerant strategy are proposed. Accounting for the unreliable internet environment, the response time is another important factor in developing reliable web services. Therefore, the simulation algorithms also considered the response time in analysis. Then, the simulation-based reliability analysis approach of the fault-tolerant web services is demonstrated through two case studies. The rest of paper is organized as follows: Section II introduces the fault tolerant strategies of web services. Section III describes the simulation algorithms. Section IV illustrates the experiments in real world. Section V contains some conclusions plus some ideas for further work.

II. THE FAULT TOLERANT STRATEGIES OF WEB SERVICES

The web services are independent and loose-coupled which implement diverse functions. A web service system may operate a very long period (several hours, days or months). Considering the distributed operation environment, the reliability of web services is vulnerable for the unstable
internet. Fault tolerance is a technology to guarantee the system still work regularly when meets errors. The fault-tolerant mechanism is always implemented by redundancy which is difficult and expensive. But for the web services, there are a lot of web services provide the similar functions which are implemented by different companies and organizations. The fault tolerant web service can be developed by combining simpler, loosely coupled, reusable web services via the Business Process Execution Language or middleware.

According to the character of web services, there are two types of fault tolerant mechanisms: time redundancy and space redundancy [10]. Time redundancy uses extra communication or computation time to tolerant the faults. Retry is the main strategy in time redundancy mechanism. Space redundancy uses extra hardware or software resource to tolerant fault. Space redundancy includes active replication strategy and passive replication strategy. The three strategies of two types fault tolerant mechanisms are detailedly explained in the following.

1) Retry. The most common used software fault tolerance strategy, such as WS-ReliableMessage [5]. When the service fails, it will run repeatedly until success or reach the max specified times. This strategy is simple to implement and usually effective for the transient errors. Taking overheads into consideration, retries should be the premise of ensuring reliability of the system, as small as possible.

2) Active replication. This strategy is derived from N version programming which is a transition software fault tolerant strategy. Active replication invokes all web services at the same time and the first proper returned result is used as the final outcome. Active replication strategy has a very high reliability and needs a lot of redundant web services as replicas. The FTweb [4] and Thema [5] have employed it to improve the reliability.

3) Passive replication. This strategy is derived from Recovery block which is a well-known software fault tolerant strategy. Passive replication employs the primary web service to process the service request first. If the primary web service fails, the back up replication is invoked. Therefore, there are a primary web service and a sequence of back up replicas for employing this strategy.

Compare to the traditional fault tolerant software system, the web services have less correlation to each other and there are a lot of available redundant replicas on the internet. The strategies introduced above can be combined in the application to achieve a higher reliability

III. THE SIMULATION-BASED RELIABILITY ANALYSIS FOR FAULT TOLERANT STRATEGIES

It is not feasible and costly to explore every fault tolerant strategy via test in real internet environment. The simulation technique can help the developers in determining how the fault tolerant web services will perform when they are employed. It plays an important role by exploring the ‘what-if’ questions for determination of appropriate fault tolerant strategies [11, 12]. By applying simulation, the developers can obtain the feedback on the process and modify strategies by replacing services that do not provide required service time averages or modifying the process structure based on the simulation runs.

The discrete-event simulation technique has been used to obtain the failure behavior of web services which is described by a pure-birth non-homogeneous continuous time Markov chains {NHCTMC} process [13]. The main idea of this technique is compare a random number \( x \) with the probability of an event occurs in the infinitesimal interval \( (t, t+dt) \) which is given by \( \lambda dt \). \( \lambda dt \) is the failure rate function denotes the state of system which is provided by the developers. If \( x > \lambda dt \) then a failure happened and \( \text{occurs}() \) returns 1, otherwise the web service executes success and \( \text{occurs}() \) returns 0.

As the web services are distributed on the network, the response time is another important factor when applying fault tolerant strategies. However, the execution time is seldom considered in the traditional software reliability simulation approaches. Therefore, the Round-Trip Time (RTT) of web services is also calculated in our simulation algorithms.

In Section II, three kinds of fault tolerant strategies are introduced which are retry, active replication and passive replication. The simulation algorithms of these three strategies are shown in Fig. 1, Fig. 2 and Fig. 3. In the retry simulation algorithm, the retryTimes denotes the max specified retry times. When a failure is happened, the web service will be restarted. It executes success or reaches the retryTimes, the simulation algorithm returns FTstatus and curr_time which mean the status of execution and the execution time.

```c
FTretry(dt, curr_time, retryTimes, lambda(), RTT)
{
    for(i=0; i<retryTimes; i++)
    {
        local_time=0;
        while(local_time<RTT)
        {
            if(occurs(dt, lambda(local_time)))
            {
                local_time+=dt;
                else curr_time+=local_time; break; //A failure is happened.
            }
            if(local_time>RTT)
            {
                FT_status=true; curr_time+=local_time; break;
                else FT_status=false;
            }
        }
        return FT_status, curr_time;
    }
}
```

Figure 1. Simulation algorithm of the retry strategy.

Fig. 2 is the simulation algorithm of the active replication strategy. In this strategy, all replicas of the web service are executed in parallelism. Therefore, the local_time.i denotes the execution time of every replica which is 0 at the beginning of the algorithm. Then, if there are replicas execute successfully, the FTstatus is true and the minimum local_time.i is returned. Otherwise, the FTstatus is false and the max fail_time.i value of all replicas is returned since the
active replication strategy cannot determine whether it is failure until all the replicas execute failed.

```pseudocode
FTactive(dt, curr_time, (*lambda)[n][i], RTT[n])
{  for(i=0;i<n;i++)
    {  local_time=i=0; // All replicas execute parallel.
        while(local_time.i<RTT.i)
          {  if(!occurred(dt, lambda[i][local_time.i]))
             {  local_time.i++;
                else fail_time.i=local_time.i; break;
              }
            if(local_time.i>RTT.i)
              FTreplica_status[i]=true; break;
            else FTreplica_status[i]=false;
            }
        if(exist FTreplica_status[true])
          {  FTTstatus=true:
              curr_time=local_time.i && (FTreplica_status[i]=true);
              else FTTstatus=false; curr_time=max(fail_time.i);
            return FTTstatus, curr_time;
          }
  }
}
```

Figure 2. Simulation algorithm of the active replication strategy.

The simulation algorithm of the passive replication strategy. First, all the replicas are sequenced by some kind of attribute, such as reliability, RTT, or cost. Then, the first order replica executes. When it fails, the second order replica will be tried sequentially. If there is replica executes successfully, the FTTstatus is true and the curr_time is the execution time of replicas. Otherwise, the FTTstatus is false and the curr_time is the sum of all replicas' failure execution time.

```pseudocode
FTpassive(dt, curr_time, (*lambda)[n][i], RTT[n])
{  Sequence(replicas[n][j]); //All replicas are sequenced in a priority order.
  for(i=0;i<n;i++)
    {  local_time=i=0;
      while(local_time.i<RTT.i)
        {  if(!occurred(dt, lambda[i][local_time.i]))
           local_time.i++;
          else curr_time+=local_time.i; break;
          }
        if(local_time.i>RTT.i)
          FTTstatus=true; curr_time+=local_time.i; break;
        else FTTstatus=false;
        }
  return FTTstatus, curr_time;
}
```

Figure 3. Simulation algorithm of the passive replication strategy

As the web services have less correlation and there are a lot of available redundant replicas, the three kinds of fault tolerant strategies can be combined to achieve higher reliability or satisfy some constraint [10]. Here, the simulation algorithms of the combination strategies are not presented because they can be implemented easily by integrating the above algorithms.

IV. EXPERIMENTS

A composite web service of the bank transaction has been proposed in our previous work [14]. In the component services of the bank transaction, the loan service needs very high reliability to ensure users can get the money on time. Therefore, there are three versions of loan service which are named as loanservice1, loanservice2 and loanservice3. By applying SoapUI which is a web service test tool, the three loan services are evaluated. Table I shows the evaluation results of three loan services.

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Cases</th>
<th>RTT(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoanService1</td>
<td>1079</td>
<td>0.804</td>
</tr>
<tr>
<td>LoanService2</td>
<td>1235</td>
<td>0.793</td>
</tr>
<tr>
<td>LoanService3</td>
<td>1104</td>
<td>0.788</td>
</tr>
</tbody>
</table>

In this section, the loan service is used to demonstrate the ability of the proposed simulation algorithms. Since the developers just provide the average reliability of loan services, the failure rate function lambda() is a constant which is represented in Equation (1).

\[
\text{lambda}(\cdot) = \frac{(1-R)}{\text{RTTavg}}
\]  

The failure behavior of the loan service applied three fault-tolerant strategies can be simulated by our algorithms. Table II shows the results of 1 million simulation times. The three loan services are sequenced by their reliability. Thus, the loanservice1 is the primary service, and the loanservice2 and the loanservice3 are the second replica and the third replica separately. Here, the retryTimes in the retry strategy is set to be 2. From Table II, The retry strategies have the highest reliability and the active replication strategy has lowest RTTavg.

<table>
<thead>
<tr>
<th>Fault Tolerant Strategy</th>
<th>Fail cases</th>
<th>R</th>
<th>RTTavg(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retry</td>
<td>5709</td>
<td>0.9943</td>
<td>96.84</td>
</tr>
<tr>
<td>Active replication</td>
<td>6578</td>
<td>0.9935</td>
<td>88.86</td>
</tr>
<tr>
<td>Passive replication</td>
<td>6532</td>
<td>0.9935</td>
<td>96.37</td>
</tr>
</tbody>
</table>

As the loan service is a commercial application, its reliability should be higher than 0.999. However, the three fault tolerant strategies cannot satisfy the reliability requirement. The developers want to employ the combinations of three fault tolerant strategies to achieve a higher reliability. The three fault tolerant strategies can implement 6 kinds of fault tolerant strategy combinations which are Retry+Active, Active+Retry, Retry+Passive, Passive+Retry, Active+Passive, and Passive+Active. The Active+Passive, and Passive+Active strategies need at least 4 services to implement. Then, the first 4 kinds of combinations are considered in our simulation-based reliability analysis. Table III shows the results for 1 million simulation times of the 4 combinations.

TABLE I. EVALUATION RESULTS OF THREE LOAN WEB SERVICES

TABLE II. RELIABILITY AND RTT RESULTS OF THREE FAULT TOLERANT STRATEGIES FOR 1 MILLION SIMULATION TIMES
Thus, the Active+Retry strategy is the best of 4 combinations, which means it has fewest failures during the execution. The red solid line represents the lowest failure increase rate combination strategies during 400,000 execution times. The failure details should be considered for choosing the best 4 combination strategies have very close reliability, their have very high reliability which is more than 0.9999. As the weight to the services.

Developers can calculate the cost by setting the different loan service needs different computation resource, the times, it is an appropriate strategy for considering the cost of Retry+Passive strategy has the minimum total execution details of 4 combination fault tolerant strategies. As the strategy is the best combination in failure situation analysis, for 1 million simulation times. Although the Active+Retry execution of 4 combination fault tolerant strategies easily. A loan service of the bank transaction is used to demonstrate the simulation-based reliability analysis approach. To achieve a high reliability, the combination fault tolerant strategy is employed in the application. However, it is very complex for the developers to analysis. The simulation-based approach can analysis the execution details of different fault tolerant strategy. Developers can find an appropriate strategy for different constraints by applying our approach. In future, the operation profile of web services will be considered in the simulation for making a more accurate analysis.

V. CONCLUSIONS

In this paper, the discrete-event simulation technique is explored to analysis the reliability of fault tolerant web services. Three common used fault tolerant strategies which are Retry, Active replication and Passive replication are studied. Then, the simulation algorithms of these fault tolerant strategies are proposed. These algorithms can be integrated to implement the combination fault tolerant strategies easily. A loan service of the bank transaction is used to demonstrate the simulation-based reliability analysis approach. To achieve a high reliability, the combination fault tolerant strategy is employed in the application. However, it is very complex for the developers to analysis. The simulation-based approach can analysis the execution details of different fault tolerant strategy. Developers can find an appropriate strategy for different constraints by applying our approach. In future, the operation profile of web services will be considered in the simulation for making a more accurate analysis.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (Grant No.61202091) and the Fundamental Research Funds for Central Universities (Grant No.NSRIF.2016050).

REFERENCES


From Table III, the 4 kinds of combination strategies have very high reliability which is more than 0.9999. As the 4 combination strategies have very close reliability, their failure details should be considered for choosing the best strategy. Fig. 4 shows the expected failure numbers of 4 combination strategies during 400,000 execution times. The red solid line represents the lowest failure increase rate which means it has fewest failures during the execution. Thus, the Active+Retry strategy is the best of 4 combinations.

TABLE III. RELIABILITY AND RTT RESULTS OF 4 COMBINATION FAULT TOLERANT STRATEGIES FOR 1 MILLION SIMULATION TIMES

<table>
<thead>
<tr>
<th>Fault Tolerant Strategy</th>
<th>Fail cases</th>
<th>R</th>
<th>RTTavg(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retry+Active</td>
<td>40</td>
<td>0.999960</td>
<td>89.02</td>
</tr>
<tr>
<td>Active+Retry</td>
<td>38</td>
<td>0.999962</td>
<td>89.30</td>
</tr>
<tr>
<td>Retry+Passive</td>
<td>45</td>
<td>0.999955</td>
<td>97.94</td>
</tr>
<tr>
<td>Passive+Retry</td>
<td>40</td>
<td>0.999960</td>
<td>99.23</td>
</tr>
</tbody>
</table>

From Table III, the 4 kinds of combination strategies have very high reliability which is more than 0.9999. As the 4 combination strategies have very close reliability, their failure details should be considered for choosing the best strategy. Fig. 4 shows the expected failure numbers of 4 combination strategies during 400,000 execution times. The red solid line represents the lowest failure increase rate which means it has fewest failures during the execution. Thus, the Active+Retry strategy is the best of 4 combinations.

TABLE IV. EXECUTION DETAILS OF 4 COMBINATION FAULT TOLERANT STRATEGIES FOR 1 MILLION SIMULATION TIMES

<table>
<thead>
<tr>
<th>Fault Tolerant Strategy</th>
<th>Loan Service1</th>
<th>Loan Service2</th>
<th>Loan Service3</th>
<th>Total execution times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retry+Active</td>
<td>1006451</td>
<td>1006451</td>
<td>1006451</td>
<td>3019353</td>
</tr>
<tr>
<td>Active+Retry</td>
<td>1179994</td>
<td>1188342</td>
<td>1192445</td>
<td>3560781</td>
</tr>
<tr>
<td>Retry+Passive</td>
<td>1179536</td>
<td>38083</td>
<td>1358</td>
<td>1218977</td>
</tr>
<tr>
<td>Passive+Retry</td>
<td>1006561</td>
<td>180025</td>
<td>33991</td>
<td>1220577</td>
</tr>
</tbody>
</table>

From Table III, the 4 kinds of combination strategies have very high reliability which is more than 0.9999. As the 4 combination strategies have very close reliability, their failure details should be considered for choosing the best strategy. Fig. 4 shows the expected failure numbers of 4 combination strategies during 400,000 execution times. The red solid line represents the lowest failure increase rate which means it has fewest failures during the execution. Thus, the Active+Retry strategy is the best of 4 combinations.

From Table III, the 4 kinds of combination strategies have very high reliability which is more than 0.9999. As the 4 combination strategies have very close reliability, their failure details should be considered for choosing the best strategy. Fig. 4 shows the expected failure numbers of 4 combination strategies during 400,000 execution times. The red solid line represents the lowest failure increase rate which means it has fewest failures during the execution. Thus, the Active+Retry strategy is the best of 4 combinations.

From Table III, the 4 kinds of combination strategies have very high reliability which is more than 0.9999. As the 4 combination strategies have very close reliability, their failure details should be considered for choosing the best strategy. Fig. 4 shows the expected failure numbers of 4 combination strategies during 400,000 execution times. The red solid line represents the lowest failure increase rate which means it has fewest failures during the execution. Thus, the Active+Retry strategy is the best of 4 combinations.

From Table III, the 4 kinds of combination strategies have very high reliability which is more than 0.9999. As the 4 combination strategies have very close reliability, their failure details should be considered for choosing the best strategy. Fig. 4 shows the expected failure numbers of 4 combination strategies during 400,000 execution times. The red solid line represents the lowest failure increase rate which means it has fewest failures during the execution. Thus, the Active+Retry strategy is the best of 4 combinations.

From Table III, the 4 kinds of combination strategies have very high reliability which is more than 0.9999. As the 4 combination strategies have very close reliability, their failure details should be considered for choosing the best strategy. Fig. 4 shows the expected failure numbers of 4 combination strategies during 400,000 execution times. The red solid line represents the lowest failure increase rate which means it has fewest failures during the execution. Thus, the Active+Retry strategy is the best of 4 combinations.

From Table III, the 4 kinds of combination strategies have very high reliability which is more than 0.9999. As the 4 combination strategies have very close reliability, their failure details should be considered for choosing the best strategy. Fig. 4 shows the expected failure numbers of 4 combination strategies during 400,000 execution times. The red solid line represents the lowest failure increase rate which means it has fewest failures during the execution. Thus, the Active+Retry strategy is the best of 4 combinations.

From Table III, the 4 kinds of combination strategies have very high reliability which is more than 0.9999. As the 4 combination strategies have very close reliability, their failure details should be considered for choosing the best strategy. Fig. 4 shows the expected failure numbers of 4 combination strategies during 400,000 execution times. The red solid line represents the lowest failure increase rate which means it has fewest failures during the execution. Thus, the Active+Retry strategy is the best of 4 combinations.

From Table III, the 4 kinds of combination strategies have very high reliability which is more than 0.9999. As the 4 combination strategies have very close reliability, their failure details should be considered for choosing the best strategy. Fig. 4 shows the expected failure numbers of 4 combination strategies during 400,000 execution times. The red solid line represents the lowest failure increase rate which means it has fewest failures during the execution. Thus, the Active+Retry strategy is the best of 4 combinations.

From Table III, the 4 kinds of combination strategies have very high reliability which is more than 0.9999. As the 4 combination strategies have very close reliability, their failure details should be considered for choosing the best strategy. Fig. 4 shows the expected failure numbers of 4 combination strategies during 400,000 execution times. The red solid line represents the lowest failure increase rate which means it has fewest failures during the execution. Thus, the Active+Retry strategy is the best of 4 combinations.

From Table III, the 4 kinds of combination strategies have very high reliability which is more than 0.9999. As the 4 combination strategies have very close reliability, their failure details should be considered for choosing the best strategy. Fig. 4 shows the expected failure numbers of 4 combination strategies during 400,000 execution times. The red solid line represents the lowest failure increase rate which means it has fewest failures during the execution. Thus, the Active+Retry strategy is the best of 4 combinations.


