Web Service Reliability Test Method Based on Log Analysis

Xuetao Tian[1], Honghui Li[2], Feng Liu[3]
School of Computer and Information Technology Beijing Jiaotong University
Beijing 100044, China
Engineering Research Center of Network Management Technology for High Speed Railway of MOE
Email: 1.xttian@bjtu.edu.cn 2.hhli@bjtu.edu.cn 3.fliu@bjtu.edu.cn

***Abstract*—A web service reliability test method for C/S architecture software based on log analysis is presented in this paper. In this method, the software usage model is constructed automatically to describe the real situation on the users' access to the web service by Markov chain. The test cases are generated according to Random Walk and applied to software reliability test. In the experiment process, MTBF (focusing on server crash) was chosen to be the software reliability evaluation index. Through the testing and analysis of a real web software, MTBF obtained by testing result is similar to that from the realistic log, and the web service reliability test method is validated.**

***Keywords*—Web Service; Reliability Test; Log Analysis; Markov Usage Model; Test Cases**

1. Introduction

With the popularity of Internet technology and smartphone, the utilization of mobile applications based on the C/S architecture has increased dramatically, and the enormous quantity of users' accesses puts forward higher requirements for the web service reliability. Web service failure will directly lead to client unavailability. Therefore, it is necessary to test and evaluate the reliability of the web application service, in order to determine whether it can be guaranteed faultless in the preset time and environment. On the basis of the test, the next software failure time may be predicted by software reliability evaluation to facilitate the maintenance job.

Software reliability test is a software fault-facing test method, which detects software defects and verifies whether the requirements of users are met or not under real or simulated environment [1]. Such a test method can effectively find software defects that have large impact on reliability so as to improve software reliability. The statistical test method based on usage model is commonly used for reliability test at present. The modeling process is the description of using software. Through it, reliability test cases are generated and applied to reliability test.

In this paper, an exploratory method of web service reliability test based on log analysis is presented. This method will construct software Markov usage model by using log analysis and generate test cases in accordance with realistic user behavior. According to the failure data from the testing experiment, the reliability evaluation results for web service are similar to the real situation.

The rest of this paper is organized as follows: Section 2 introduces the related work and achievement of the web service reliability test. Section 3 elaborates the process to construct a usage model based on log analysis and generate test cases. Section 4 experiments to validate that the test method in this paper and the realistic failure data have similar evaluation of the web service reliability. Conclusions and perspectives are put forward in Section 5.

2. Related Work

Previous researchers have done sufficient research work on collecting information from users to be used in software testing and model-driven web test method. For the former, Penelope A. Brooks and Atif M Memon applied usage profile to GUI testing [2]. Mette Arleth tested cartographic web-based applications by using log analysis [3]. Jingjun Zhu et al proposed a web load test method based on log analysis [4]. For the latter, Nuo Li et al summarized the key technologies to implement model-driven web application test [5]. It is mentionable that Zhao Li and Jeff Tian elaborated the suitability of Markov chain as web service usage models [6].

Similarly, the research on the web service reliability has increasingly become perfect. Software reliability [7] means: (1) under specified conditions and within the specified time, the probability that the software does not cause the system failure. The probability is a function of the system input and usage, as well as a function of the defects existing in the software. The system input will determine whether existing defects will be encountered. (2) Standards approved by the American Institute of Standardization as the American national standards and adopted by China national standard GB/T-11457 [8] in 1989. References [9, 10] studied on evaluating web service reliability. It is especially emphasized that failure data are used to evaluate the reliability.

On the basis of previous research, this paper tentatively applies log analysis to web service reliability test with the aid of Markov chain. In order to validate the practicability and effectiveness, MTBF (mean time between failures) will be used in the experiment.

978-1-5386-2072-4/17 $31.00 © 2017 IEEE
DOI 10.1109/QRS-C.2017.38

3. Test Cases Based on Log Analysis and Markov Chain

In existing software reliability test, that the usage model can describe the actual software usage scenarios depends on the analysis of the software model, functions, task requirements and related inputs by Reliability Engineers and their understanding about the operating frequency of these system patterns, functions, and tasks [11]. This section will dig out the usage model which reflects the users' habits accurately from the web access log based on Markov chain, and generate the test cases to be used in the reliability test according to this model.

3.1. Markov Model and Its Variant

Markov model describes the software using process by Markov chain. It shows a finite state machine with probability characteristics, which can not only describe the software status or the transfer of atomic operation in the software-using process, but also simulate users' habits through the probability distribution.

Fig.1 shows a simple directed graph to represent Markov model. It can be seen as a combination of an unweighted finite state machine and the probabilities added for each edge. Each vertex in the figure is regarded as the current state of the software. The edges with the same starting point is considered as a transfer mode for subsequent operations after an operation. It is observed that the sum of the corresponding probabilities of these edges with the same starting point is 1 [12].

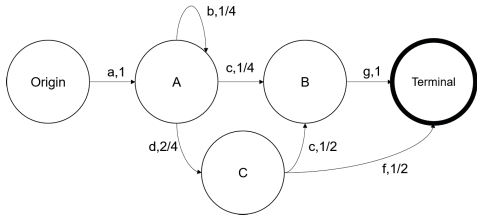


Figure 1. Directed graph representation of Markov model.

The main idea of the Markov model is that each operation of a user is only related to the current state of the software from the previous operation [13]. A sequence of user operations is a series of follow-up operations order based on the previous operation. When using an application, the status of the software after a user operation and the subsequent operations are definite, and the sum of the probabilities of different operations is 1.

The software test cases focus more on the one by one atomic operation, especially in the web service testing. In order to facilitate observation, Markov model can be deformed by using vertices to describe the users' operations. The model shown in Fig.2, through which we can obtain test cases, has the same probability distribution as the model in Fig.1, so these two can be considered equivalent.

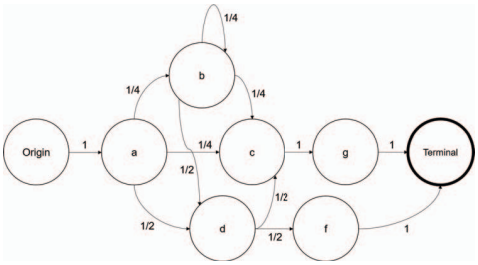


Figure 2. Directed graph representation of transformative Markov model.

3.2. Constructing Usage Model Based on Log Analysis

On the foregoing basis, a method that automatically constructs Markov usage model based on web service access log analysis is put forward. In order to facilitate the description of the log information, the data structure ***Record*** used to represent a access record is designed as below. Then access log is abstracted as a set of *Record* that is sequentially ordered.

Definition 1. We define a access record as ***Record (user, device, event, subevent, parameter, time, result, exception)***, where
(1) *user* and *device* respectively denote the user information and device information;
(2) *event* and *subevent* jointly determine a specific user operation;
(3) *parameter* shows parameter information attached to the current operation;
(4) *time* represent the operating time;
(5) *result* and *exception* represent operating results and the attached error message when an error operating result happens.

In the section, a small-scale case is analyzed to elaborate this automated method. The main process can be globally understood from Fig.3.

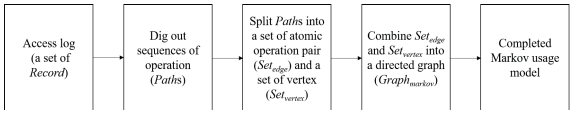


Figure 3. Overview of the automated method.

As shown in Fig.4, the access log consisting of *Record*s is grouped in view of *user* and *device*. Apparently, only the continuous operations with the same *user* and *device* can be an operation sequence when using software once. In the same way, a test case is an operation sequence of the software. In the following, *Path* is applied to express a continuous operation sequence for the sake of easy description.

Δ_{time} which is the maximum time interval threshold between using software twice is set empirically for a grouped *Record* collection, thus dividing each group of *Record* into a *Path* subset. Apparently, the users' habits of using software

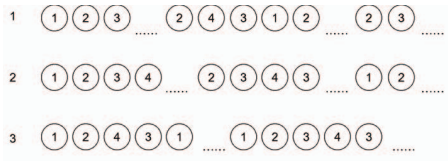


Figure 4. Groups of access log by user and device.

should be cyclical, and the operation sequences are similar at all time. If an interval between two atomic operations is relatively longer, then they should belong to two different operation sequences. As shown in Fig.5, according to the *time* attribute of *Record* and Δ_{time}, each group of user's operation is divided into several instances of *Path*.

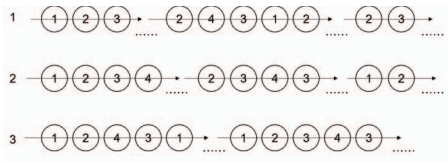


Figure 5. Set of Path.

After completing the above procedures, the original access log (that is, the seemingly out-of-order *Record* set), has been systemized to a set of *Path*. Each *Path* is made up of *N* vertices (user atomic operations) and (*N* - 1) directed edges (the transfer from a completed atomic operation to a next atomic operation).

The next procedure is to split *Path*. Each *Path* is split into separate (*N* - 1) directed edges. It can be considered that each edge is a basic element to describe the transfer of user's atomic operations. As shown in Fig.6, a set of directed edges, denoted as Set_{edge}, will result from the above procedures. These directed edges are an important part to construct the final Markov model.

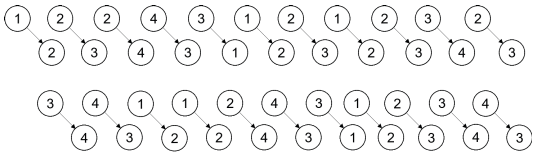


Figure 6. Set of directed edge.

At the same time of the above procedures, it is also necessary to group *Record* by *event* and *subevent* to extract some required information. The data structure Map (namely, key/value pairs) is used to store these information, denoted as Map_{event}. The key represents the users' atomic operation that is identified by *event* and *subevent*, and value represents the set of *parameter* in each *Record*, which is shown in Fig.7. This job is obviously easy to complete. To be sure, the value of Map_{event} has no relationship to the construction of usage model, but can be used to generate test cases.

Markov model is regarded as a finite state machine with probability characteristics. If the normalized probability distribution is ignored, the model becomes a directed graph,

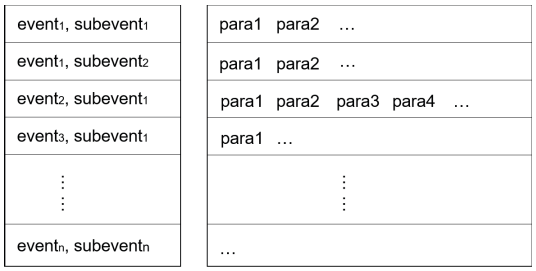
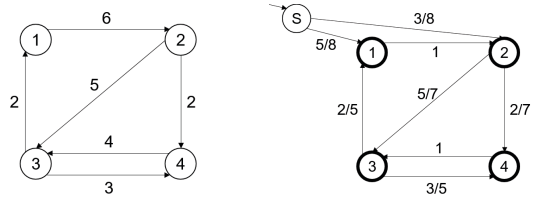


Figure 7. Groups of parameter.

denoted as $Graph_{markov}$. From the above description, the set of the key in Map_{event} is the vertex set of $Graph_{markov}$, denoted as Set_{vertex}. The edge set of $Graph_{markov}$ can be obtained from the Set_{edge}, and the weight of every edge is marked with a repeated number. As shown in Fig.8(a), with the edge weights ignored, $Graph_{markov}$ is constructed completely after this procedure.

Finally, the following operations are conducted for each vertex ($Vertex_i$): the directed edges with starting point $Vertex_i$ normalize their weights to reach the sum 1. That is, the weight of each edge is updated to the probability of its occurrence in the edges with starting points $Vertex_i$. In addition, the adjacent points of the finite state machine starting points and the ending points are collected from the access log by the following ways: to artificially give a starting point *S* and edges from *S* to each starting point in any *Path*. The weight of each edge is replaced by the probability of its ending point occurrence in the starting point set of all *Path*. Each ending point of all *Path* is an element of the finite state machine ending points. Then a Markov usage model with a variant form is completed, as shown in Fig.8(b).



(a) Markov Directed graph. (b) Markov usage model.

Figure 8. Finished Markov usage model.

3.3. Generating Reliability Test Cases by Using Usage Model

A test case is a continuous sequence of user operations. An operation should include *event*, *subevent* and *parameter*. Based on the above discussion, this section introduces the process of generating test cases by Random Walk. In applied Mathematics, Random Walk is the utilization of random factors to simulate the real system behavior. In this way, generating a test case can be seen as a multi-step Random Walk in Markov usage model.

Following the idea of Random Walk, Alg.1 describes a step walk process with the input of Markov chain usage model and the current vertex *u* and the output of a random adjacent vertex *v*. Alg.2 elaborates the process of generating multiple test cases. The Markov usage model and Map_{event} will be the inputs of Alg.2 and the number of test cases *N* and the max length of a test case *M* can be specified.

Algorithm 1 Single Step Random Walk

Required: *Markov chain usage model*,
Vertex *u* /* the current vertex */
assign $v \leftarrow$ a object of Vertex;
assign $number \leftarrow$ a random float number from 0 to 1;
for all $e \in$ edges with starting point u **do**
  assign $number = number-$ the weight of e;
  if $number <= 0$ **then**
    assign $v =$ the ending point of e;
    break;
  end if
end for
Output: Vertex *v* /* the next vertex through single-step Random Walk */

Algorithm 2 Generating Multiple Test Cases

Required: *Markov chain usage model*,
Map_{event},
N /* the number of test cases */,
M /* the max length of a test case */.
assign $cases \leftarrow$ an object of List<List<(event, subevent, parameter)>>; /* the result of test cases */
for all $i \in 1$ to *N* **do**
  assign $path \leftarrow$ an object of List<(event, subevent, parameter)>;
  assign $u \leftarrow$ an object of Vertex; /* record the current vertex*/
  assign $u \leftarrow S$ /* the origin of Markov model*/
  for all $j \in 1$ to *M* **do**
    Randomly choose parameters from according list of Map_{event} and attach them to u;
    Add u and the parameters to $path$;
    if u is in Set_{end} **and** not exists edge with starting point u **then**
      break;
    end if
    assign $u \leftarrow Single\_Step\_Random\_Walk($*Markov chain usage model*, u)
  end for
  Add $path$ to $cases$;
  Clear($path$)
end for
Output: *cases* /* *N* test cases with max length *M* */

4. Experiments and Verification

For the web service, an occasional failed request should be accepted, but a server crash which leads to client availability is intolerant. It is the main concern to avoid a server crash. Therefore, the failure data for a server crash has been selected to experiment. "Pandian Game" App supplies the failure data and access log. As shown in Tab.1, the time of realistic server crashes in the past three months is listed.

TABLE 1. THE TIME OF REALISTIC SERVER CRASHES

Crash No.	Real time	Interval time
1	2016/12/14 06:08:14	–
2	2016/12/23 05:22:27	167:14:13
3	2017/01/02 21:39:22	256:16:55
4	2017/01/13 05:22:27	247:43:05
5	2017/01/24 02:23:18	247:01:01
6	2017/01/26 02:28:34	048:05:16
7	2017/01/30 02:10:50	095:42:16
8	2017/02/03 11:17:42	105:06:52
9	2017/02/20 04:40:22	401:22:40
10	2017/02/22 05:35:50	049:15:28
11	2017/03/01 05:13:59	167:18:09
12	2017/03/08 16:30:34	179:16:35

In this experiment, to facilitate operation and observation, a simulation environment has been set up in the laboratory. To reduce experiment period, these requests in access log are re-executed at a speed of 50 times the realistic situation. The failure data in this experiment is shown in Tab.2. MTBF is about 150.5 minutes and the interval time variance is about 638.

TABLE 2. THE RESULT OF RE-EXECUTING THE REQUESTS IN ACCESS LOG

Crash No.	Real time	Interval time
0	2017/03/18 12:00:00	start time
1	2017/03/18 14:20:11	–
2	2017/03/18 17:01:24	02:41:13
3	2017/03/18 18:56:12	01:54:48
4	2017/03/18 21:48:44	02:52:32
5	2017/03/19 00:22:21	02:33:37
6	2017/03/19 03:07:55	02:45:34
7	2017/03/19 05:48:13	02:40:18
8	2017/03/19 07:42:02	01:53:49
9	2017/03/19 10:00:32	02:18:30
10	2017/03/19 13:17:31	03:16:59
11	2017/03/19 16:30:57	03:13:46
12	2017/03/19 18:50:13	02:19:16
13	2017/03/19 21:34:09	02:43:56
14	2017/03/19 23:29:44	01:55:35
15	2017/03/20 02:09:17	02:39:33
16	2017/03/20 04:57:28	02:48:11
17	2017/03/20 06:40:00	end time

The test cases is generated by the method in this paper with the same size as the above requests. Also in the laboratory environment, the test cases are executed to access the web service at the identical speed. The same steps have been operated three times. The failure data is shown in Tab.3.

From Tab.3, MTBF is respectively 146.8, 151.8, 152.6 minutes. And MTBF and variance are almost in a same order of magnitude. Thus, MTBF obtained by testing is similar to that from the realistic log. The web service reliability test method is validated.

TABLE 3. THE FAILURE DATA FROM EXECUTING TEST CASES

No.	Crash counts	MTBF	Variance
1	17	About 146.8 minutes	About 968
2	16	About 151.8 minutes	About 1058
3	16	About 152.6 minutes	About 672

With the purpose of further proof for the experiment results, two little experiments have been conducted as well. With the same speed and request size, the failure data have been obtained respectively when the web service has only heartbeat request or only a certain request (selecting image request) in the simulation environment. The result shows no crash for the former and only twice crashes for the latter.

5. Conclusions and Perspectives

A web service reliability test method based on log analysis is presented in this paper. The method can automatically construct Markov usage model and generate test cases for web service reliability test. The experiment contrasting MTBF proves that the test method in this paper and the realistic failure data have similar evaluations of the web service reliability. The method for web service reliability test is validated to be feasible and have some practical values.

However, the scenario considered in this paper is too simple. For complex data structures, Markov model should be replaced by a more appropriate model so as to make sufficient representation of complex algorithm model. In future work, the following researches will be discussed as well. Dig out the software state information from web service log information, thereby generating client application test cases. And Learn appropriate logic from the existing parameters to produce parameter attributes for each test case, instead of using log parameter information directly.

Acknowledgments

This work has been supported by Project No. 2015AA043701 of the National High-Technology Research and Development Program of China (863 Program) and Project No. 2015BAF08B02 of the National Key Technology Research and Development Program of China.

References

[1] Pandey, Ajeet Kumar, and N. K. Goyal. Early Software Reliability Prediction. *Springer India*, 2013.

[2] Penelope A. Brooks, Atif M Memon. Automated GUI Testing Guided By Usage Profiles. In *IEEE/ACM International Conference on Automated Software Engineering*, pages 333-342. ACM, 2007.

[3] Mette Arleth. Using Log-File Analysis for Testing Cartographic Web-Based Applications. In *4 of the International Cartographic Conference*. Scientific and Technical Program Committee, 2008.

[4] Jingjun Zhu, Haiyan Wu, Guozhu Gao, Zhirui Cheng. Web Load Test Method Based on Log Analysis. *Computer Engineering*, 36(23):25-27, 2010.

[5] Nuo Li, et al. A Framework of Model-Driven Web Application Testing. In *Computer Society Signature Conference on Computers, Software and Applications*, pages 157-162. IEEE, 2006.

[6] Zhao Li, Jeff Tian. Testing the Suitability of Markov Chains as Web Usage Models. In *Computer Society Signature Conference on Computers, Software and Applications*, pages 356-361. IEEE, 2003.

[7] American National Standards Institute IEEE Standards Board, "IEEE Standard Dictionary of Measures to Produce Reliable Software" IEEE Std 982.1-1988.

[8] GB/T 11457-95. The national standard of the People's Republic of China - software engineering terms.

[9] Jeff Tian, Sunita Rudraraju, Zhao Li. Evaluating Web Software Reliability Based on Workload and Failure Data Extracted from Server Logs. *IEEE Transactions on Software Engineering*, 30(11):754-769, 2004.

[10] Guangzheng Wang, Xifeng Wang, Min Xia. Ontology-based Dynamic Reliability Evaluation Model for Web Services. *Computer Science*, 39(11):98-101, 2012.

[11] Guofeng Bu, Xiaodong Zhu, Caihuo Wu, Yigang Wang. Study on the Generation of Software Reliability Test Case Based on Markov Chain. *Science Mosaic*, 2009.

[12] James A. Whittaker and Michael G. Thomason. A Markov Chain Model for Statistical Software Testing. *IEEE Transactions on Software Engineering*, 20(10):812-824, 1994.

[13] Yigang Wang, et al. A method for software reliability test case design based on Markov chain usage model. *International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering*, pages 1207-1210. IEEE, 2013.