A Quick Testing Model of Web Performance Based on Testing Flow and its Application

Guangzhu Jiang
School of Computer Science and Technology
China University of Mining and Technology
Xuzhou, China
jgzcumt@163.com

Shujuan Jiang
School of Computer Science and Technology
China University of Mining and Technology
Xuzhou, China
Shjjiang@cumt.edu.cn

Abstract—**To improve the quality and reliability of Web applications, the paper presents a quick testing model of Web performance based on testing flow. It includes successful requests rate, the user behavior modeling, the quick testing flow and treatment for errors in experimental data. It also contributes a testing method of the testing model combining with LoadRunner testing tool, this provides an effective solution to Web performance testing quickly. Finally, the experimental results between current test model and the improved test model are analyzed in detail, which proves that the improved model is more practical and effective.**

Keywords-Web Performance Testing; Testing Model; Behavior Modeling

I. INTRODUCTION

In the last decade, with the prevalence of the Internet, Web applications have grown quickly because of its universality, interaction and convenient. These Web applications are being used to support a wide range of important activities: business transactions such as E-Commerce and scientific activities such as information searching and sharing. However, since Web applications' carrier, the Internet, have characters such as distributed, dynamic, multi-platform and interactive, which make their running environments more heterogeneous and autonomous.

Given the above, it is more difficult to carry out the testing tasks. So in order to enhance the testing efficiency, appropriate testing methods and tools are needed [1].

At present, most of the traditional testing models are confined to theoretical research, it is a pity that they are far from practical use, and there are not corresponding testing tools introduced, which make the models lack universality and practicability [2]. So, this paper makes some improvements of current models, contributing a quick Web performance testing model based on testing flow. It adds a new performance index (successful requests rate), and a treatment for removing the effect in experimental data. It also applies LoadRunner performance testing tool [3]. The Experiments indicate that the method provides an effective solution to Web performance testing quickly and accurately.

The remainder of this paper is organized as follows. Section 2 describes our research in the area of improving Web testing model. Section 3 conducts an experiment for validating the improved model. The conclusion is given in section 4.

II. RELATED WORKS

In this section, we first apply a practical testing tool into Web application testing; then we add a new performance indicator and propose a quick testing flow combining with LoadRunner testing tool, so as to increase the practicability. Finally, we put forward a method for reducing the effect in experimental data.

A. Introducing LoadRunner Automated Performance Testing Tool

Web applications typically undergo maintenance at a faster rate than other software systems and this maintenance often consists of small incremental changes. To accommodate such changes, Web testing approaches must be automatable and test tools must be adaptable [4].

Automated Testing is the automating of test cases so that tests may be run multiple times with minimal efforts. To do this, the test cases themselves are scripts or other codes to run the program being tested. Automated Testing has advantages of convenience, high efficiency and good repeatability, and can be used to eliminate human error, automatic generate test reports. Thus, the technique of automated testing has become an increasingly important trend in software testing.

We introduces LoadRunner automated performance testing tool into our model, which has complete functions, friendly interfaces, and is easy to operate and runs stably. Nowadays LoadRunner occupies the largest market share in the performance testing market worldwide, it supports common standards and development techniques in the field of software. Thus, it makes our model more practical and comprehensive.

The main focus of the performance testing is to generate multiple virtual users and each virtual user performs a set of activities simultaneously. It is not possible to create a real life environment at the production centre but one can create a virtual environment by using LoadRunner and proper

* This work was supported in part by the Scientific Key Project of Ministry of Education of China (108063), Natural Science Foundation of Jiangsu Province (BK2008124), the Science Research Foundation of China University of Mining and Technology (0D080310).

hardware and software. Sometimes it is necessary to create cluster of clients as a representative of different Operating Systems, multiple browsers and architectures. It may be required to create different stress level associated with different clients [7]. The typical architecture for LoadRunner is shown in Figure 1.

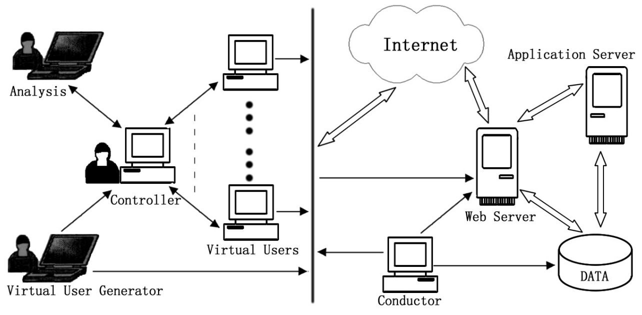

Figure 1. The Typical Architecture of LoadRunner.

A single virtual user generator can typically simulate the behavior of tens of hundreds of users through the use of very small Web clients known as virtual users.

Virtual users communicate directly with the Web server or through the conductor browser available with the client. These virtual users are grouped and mapped on to a specific client and such many clients may be created. However, all the clients can be controlled from a controller.

The delays for the users think time and network latency can also be analyzed by an analysis.

B. Quick Testing Process based on LoadRunner Testing tool

In general, the behavior of the web site is divided into set of sub behaviors represented by many integrated objects. The division of the behavior depends on the criticality of the complete or part of the web site. Sub-behaviors are independent on its own and may be clubbed together to represent the overall behavior [6, 7].

In the paper, we plan to generate test cases by modeling user behavior. As an example: after an online bookstore user logins the Web application, he usually surfs the webpage for relevant information in the first place, then selects the book he wants, finally purchases the book. From the view of the client, the user executes the above sub-behaviors for completing visiting the website. That's exactly our method, when we design test session scenarios using LoadRunner, the overall website behavior can be divided into a series of sub-behaviors or sub-transaction, then each sub-behavior testing is executed in a proper manner. From the above example, testers can get three divided sub-behaviors: scanning information, searching books and purchasing books, then testers use tools to test them in proper order.

To reduce the testing cost, the automatic level of the testing process should be improved, and the related tools must have a certain degree of intelligence [1]. The paper introduces LoadRunner performance testing tool, and contributes a quick testing process based on LoadRunner, thus forming the testing process of the proposed model. The whole process can be divided into five phases: planning the test, script creation, scenario definition, scenario execution and results analysis. Figure 2 shows the overall process.

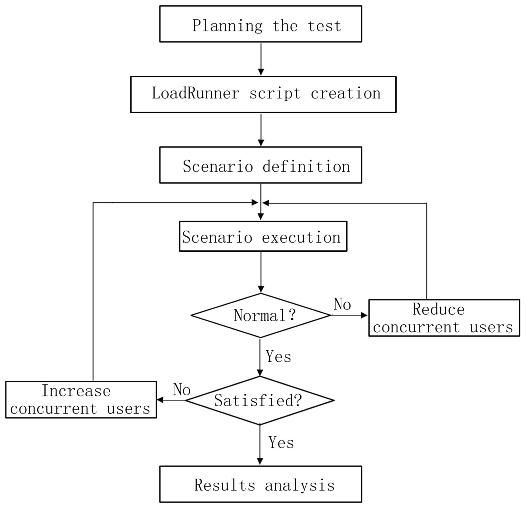

Figure 2. Quick Testing Process Based on LoadRunner.

C. Performance Indices of the proposed model

To ensure the stability of the Web site, performance model provides a standard to evaluate performance, and the standard is formed by a series of performance indices. Typical indices are response time, system throughput, concurrent users, http transactions per second and session per second, etc [5].

In this paper, in order to evaluate website performance and user's feeling more accurately and completely, we add a special performance index (successful requests rate).

$$\textit{the successful requests rate} = \frac{\textit{the number of success requests}}{\textit{the number of total requests}} \quad (1)$$

Here, what needs to be stressed is that criterion of “success request” depends on requirement analysis. Testers can reach a criterion after a deliberation: getting a server response in certain second or millisecond can be considered as a success HTTP request, or it is a failure request if it has overrun the predetermined time.

D. Treatment for errors in experimental data

In the experiment, how to reduce experiment errors and enhance the credibility of data becomes obviously important, this paper puts forward a method for reducing the effect in experimental data, which is the data processing component of the proposed model.

This is an undeniable fact that noise is always generated in communication system. The noise shows statistically Gaussian distribution. It is considered that the measured large sample data is generated by the additive Gaussian Noise affecting a specified value. We get the following relationship: $x_i=x+n_i$, the noise random variable n follow a Gaussian distribution, we write $n \sim N(\mu,\sigma^2)$.

As to errors in experimental data, a new method of error revising was described, our research tries to find out a point(x)，so as to minimize the sum of the squares of the distances of the measuring points(x_i) from the point(x), the formula is as follows:

$$MIN\{(x-x_1)^2+(x-x_2)^2+\cdots+(x-x_i)^2\} \tag{2}$$

The corresponding equation is given by:

$$f(x)=(x-x_1)^2+(x-x_2)^2+\cdots+(x-x_i)^2 \tag{3}$$

In order to get the minimum average noise, f(x) should reach a minimum value by means of Higher Mathematics methods. So, we can get reliable and acceptable processing data, which is actually a way to analyze population characteristics by inference from sampling.

III. Usage of the Model and Experimental Results

To evaluate our approach, we performed Experiments I and II. This section firstly presents the experiment environment, then presents the empirical study, and finally, discusses the limitations and threats to validity of the evaluation.

A. *Experiment Environment*

A suite performance testing experiment with the improving model is implemented in the providing access to a large stock of information resources "http://bbs.cumt.edu.cn/", a University Campus BBS, which is an ideal test environment with stable user groups, good network status and high-bandwidth communication. A typical architecture for such BBS [8] is shown below as Figure 3.

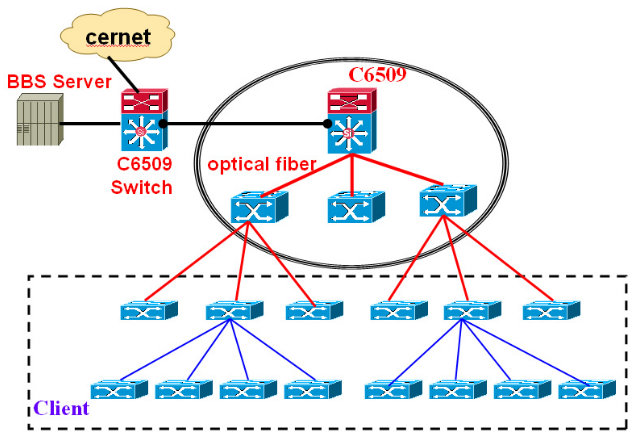

Figure 3. A Typical Architecture for BBS.

BBS is a platform on information resources sharing. It provides access to a large stock of information resources and bears enormous access load. Here, the concurrent operations and transaction response time are the key aspects of performance test.

The first experiment is done to study concurrent user logins. After users login to the BBS, they usually click the hot post on BBS page, which is the second experiment about concurrent clicks. We can get a general user behavior flow: Login → Click → Logout. We just implement the above first and second step to execute the testing.

The main issue in performance testing is to set the proper parameters and analyzing the results of the testing. The selection of performance parameters mainly depends on the type of web sites and performance requirements [1]. We select transaction response time, successful requests rate and the number of concurrent users in this experiment.

One of the best recent series of response time studies was conducted by Anna Bouch (University College - London), Allan Kuchinsky and Nina Bhatti (Hewlett Packard Labs - Palo Alto) [9]. They attempted to identify how long users would wait for pages to respond. They reported the following ratings:

- Very high (very good): Up to 2 seconds
- High (good): From 2 to 5 seconds
- Average: From 6 to 10 seconds
- Low (poor): Over 10 seconds

According to the characteristics of BBS, the experiment chooses 2 seconds as the maximum acceptable click response time and 5 seconds as the maximum acceptable login response time. And in accordance with the required standards of the testing industry, successful requests rate is on less than 90%.

B. *Experiment*

We implement the above First step to determine the relationship between transaction response time, successful requests rate and concurrent users of the login transaction. We create four different testing scenarios according to the quick web performance testing model. They base on the same test scripts and execute the same login transaction, but the number of concurrent users is different from each other. The testing results are showed as Table 1.

TABLE 1

EXPERIMENTAL RESPONSE TIMES AND SUCCESSFUL REQUESTS RATE FOR DIFFERENT CONFIGURATIONS

The testing times	The number of concurrent users							
	10 concurrent users		15 concurrent users		20 concurrent users		25 concurrent users	
	Response time (s)	Successful requests rate	Response time (s)	Successful requests rate	Response time (s)	Successful requests rate	Response time (s)	Successful requests rate
1st	1.523	100%	1.878	100%	3.640	100%	4.577	45%
2nd	1.496	100%	2.060	100%	3.290	100%	5.780	25%
3rd	1.613	100%	2.280	100%	2.913	100%	3.627	88%
4th	1.240	100%	1.843	100%	3.292	100%	3.309	72%
5th	1.664	100%	2.197	100%	2.838	100%	5.667	32%
6th	1.689	100%	2.470	100%	2.841	100%	4.231	88%
7th	1.615	100%	1.931	100%	3.321	100%	4.741	89%
8th	1.655	100%	2.081	100%	3.317	100%	2.879	88%
9th	1.512	100%	2.609	100%	2.431	100%	3.968	76%
10th	1.761	100%	2.007	100%	3.556	100%	5.267	28%

We implement the above treatment for errors to revise transaction response time from Table 1. Optimized response time, which is highly reliable, is shown in Table 2.

TABLE 2

MODELED RESPONSE TIMES FOR DIFFERENT CONFIGURATIONS

The number of concurrent users	Optimized response time
10 concurrent users	1.577 s
15 concurrent users	2.136 s
20 concurrent users	3.144 s
25 concurrent users	4.405 s

Figure 4 below is generated from Table 1, and Figure 5 from Table 2. Comparing Figure 4 with Figure 5, we can find that the Gaussian noise in response time of Figure 4 is filtered, and we get treated response time in Figure 5, which is a relatively steady value.

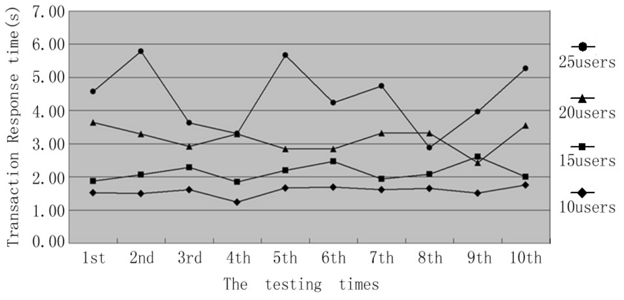

Figure 4. Experimental Response Time VS Number of Concurrent Users.

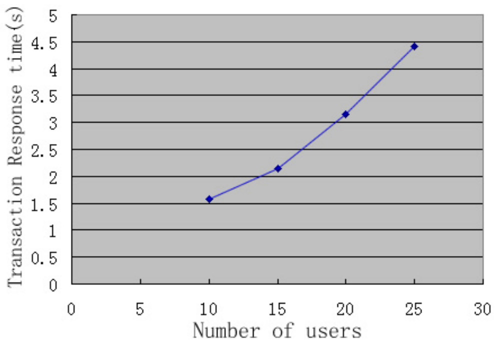

Figure 5. Modeled Response Time VS Number of Users.

As it can be noted from Figure 5, as the number of concurrent users increases, the response time shows a rising trend and the trend is more and more sharp. The response time for 10 users, 15 users, 20 users and 25 users in Figure 5 is under 5 seconds, in the eyes of the existing Web performance testing model, the Web application can meet the needs of 25 users logining the system simultaneously. But this is not the fact, we can see that successful requests rate is less than 90% from testing results listed in Table 1. This means that most of users receive the server response more than 5 seconds, so many users have a bad operation experience.

Therefore, according to our improved model and methods, the login transaction can meet the needs of at least 20 users requesting simultaneously, but cannot support 25 concurrent users. We could see that in Figure 4 the response time for 25 users fluctuates greatly due to login resources bottleneck and unstable resources use. The phenomenon also confirms that the improved model is more practical and effective, and can evaluate user's feeling more accurately and completely.

C. *Experiment II*

We implement the above Second step to determine the relationship between transaction response time, successful requests rate and concurrent users of the click transaction. We create different testing scenarios according to the quick web performance testing model. The testing results are showed as Table 3.

TABLE 3
EXPERIMENTAL RESPONSE TIMES AND SUCCESSFUL REQUESTS RATE FOR DIFFERENT CONFIGURATIONS

The testing times	The number of concurrent users			
	50 concurrent users		60 concurrent users	
	Response time (s)	Successful requests rate	Response time (s)	Successful requests rate
1st	0.651	100%	1.196	73%
2nd	0.666	100%	1.274	85%
3rd	0.551	100%	0.929	88%
4th	0.611	100%	1.614	45%
5th	0.827	100%	1.764	63%
6th	0.547	100%	1.052	85%
7th	0.654	100%	1.757	53%
8th	0.593	100%	0.898	90%
9th	0.898	100%	0.962	88%
10th	0.748	100%	1.499	75%

We apply the above treatment for errors of our model into revise transaction response time from Table 3. The Gaussian noise in response time is filtered, and we get treated response time in Table 4, which is highly reliable and optimized data. The comparison is shown in Figure 6.

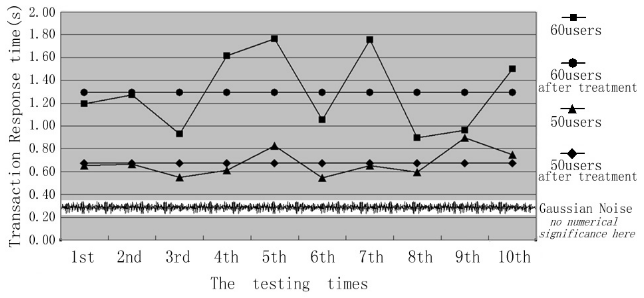

Figure 6. Experimental Response Time VS Number of Concurrent Users.

TABLE 4
MODELED RESPONSE TIMES FOR DIFFERENT CONFIGURATIONS

The number of concurrent users	Optimized response time
50 concurrent users	0.675 s
60 concurrent users	1.295 s

As we can see from Table 4, the response time for 50 users and the response time for 60 users are both under 2 seconds. But successful requests rate of 60 users is less than 90% from testing results listed in Table 3, this means that most of users receive the server response more than 2 seconds, so many users have a bad operation experience. Thus, we may safely draw the conclusion that the click transaction can meet the needs of at least 50 users requesting simultaneously, but cannot support 60 concurrent users. The conclusion depends on the improved web performance testing model, which is more close to actual feeling of users.

IV. CONCLUSIONS

Web applications develop very rapidly, but their properties make it much difficult to carry out the testing [10]. Furthermore, the testing process usually needs cost, and the testing efficiency is not high [11]. Based on the previous web performance testing model [12], this paper presents a quick Web performance testing model for distributed applications based on testing flow. This model has been developed particularly to allow for fast performance testing in order to provide a better estimation of applications usually met in Web applications. It adds the index of successful requests rate, introduces LoadRunner automated performance testing tool, models the user behavior, contributes a quick testing process and a treatment for errors in experimental data. Finally, our experiments show that this model is more practical, accurate and cost-effective.

REFERENCES

[1] L. Xu, and B. W. Xu, “Applying Agent into Intelligent Web Application Testing”, 2007 International Conference on Cyberworlds,2007.

[2] C. H. Liu, “A Formal Object-Oriented Test Model for Testing Web Applications”, Doctor Dissertation, 2002.

[3] Loadrunner, http://www.mercuryinteractive.com/

[4] Zhongsheng Qian, Huaikou Miao and Hongwei Zeng, “A Practical Web Testing Model for Web Application Testing”, Third International IEEE Conference on Signal-Image technologies and Internet-Based System, 2008.

[5] John W. Cane, “Performance Measurements of Web Applications”, SoutheastCon, 2003.

[6] F. A Torkey, Arabi Keshk, Taher Hamza and Amal Ibrahim, “A New Methodology for Web Testing”, Information and Communications Technology, 2007.

[7] B. M. Subraya and S. V. Subrahmanya, “Object driven Performance Testing of Web Applications”, The First Asia-Pacific Conference on Quality Software, 2000.

[8] CUMT Net Center, http://netcenter.cumt.edu.cn/

[9] Wait Time, http://webusability.com/article how long should users wait.htm

[10] Yu Qi, David Kung and Eric Wong, “An agent-based data-flow testing approach for Web applications”, The 29th Annual International Computer Software and Applications Conference, 2005.

[11] China Research and Development Environment Over Wide-area Network, http://www.crown.org.cn/

[12] Kurtovic Ezudin, Glavinic Vlado, “Modeling the Performance of a Distributed Application Based on Web Services”, Software in Telecommunications and Computer Networks, 2006.