Journal of Computers Vol. 30 No. 5, 2019, pp. 145-158
doi:10.3966/199115992019103005011

An Improved Hierarchical Clustering Algorithm for Performance Testing Based on User Sessions

Lu Lu[1,2], YiXin Tu[1*]

[1] School of Computer Science & Engineering, South China University of Technology, Guangzhou 510006, China
lul@scut.edu.cn, tuyixin1993@163.com

[2] Modern Industrial Technology Research Institute, South China University of Technology, Zhongshan 528400, China
lul@scut.edu.cn

Received 4 December 2017; Revised 30 March 2018; Accepted 2 June 2018

Abstract. Web applications are important parts of global information infrastructures. It attracts more and more researchers to study Web application performance testing. In this paper, a user-session-based approach combined with an improved agglomerative hierarchical clustering algorithm is proposed to generate performance test cases automatically. Such an approach generates test cases by exploiting user-sessions from server logs. It can not only reduce the manual effort when test engineers design the test cases, but guarantee the validity of the testing results. In our approach, this paper firstly give a definition of how to achieve the similarities between two URLs, then employ a dynamic programming algorithm to calculate the similarities between two user sessions. According to the similarity matrix, a bottom-up agglomerative hierarchical clustering is employed to cluster the user sessions and generate test cases. Finally, the experimental results of our approach shows its validity.

Keywords: agglomerative hierarchical clustering, performance testing, user sessions

1 Introduction

In recent years, Web applications became the important components of Global Information Infrastructure on the Internet. Web applications are growing fast which could support a wide range of activities such as online storage or network services, furthermore, it could also provide advertisements, marketing and management services in e-commerce and Business-to-Business areas. Many Web applications are gaining an increasing popularity, with millions of users online simultaneously. Therefore, the performance of a Web application becomes a crucial factor of its success. Many tools are widely used for automatic performance testing. The Record/Replay technique is applied by most of these automated testing tools. However, test engineers usually spend much time on recording or scripting the actions on Web applications and generate test data manually. Nowadays, Web applications have the characteristics of distributed, heterogeneous, and platform-independence, which make traditional Web applications performance testing techniques no more applicable [1].

To solve the problems mentioned above, this paper proposes a method leveraging user sessions data to improve automatic performance testing. User sessions provide valuable insight into the dynamic behavior of Web applications. Over the past decade, many researchers have made important contributions to test method based on user sessions [2-3], but seldom have concerned user-session-based techniques with performance testing. So it's meaningful for researchers to combine performance testing with user-sessions-based techniques: by using techniques to capture data in user sessions, the Web testing method based on user sessions can create test cases automatically and therefore reduces the artificial efforts.

* Corresponding Author

Moreover, test cases generated by user-session-based approach can reflect the behavior of real users, which is significant for Web applications performance testing; meanwhile, user sessions can be easily captured and available for test engineers; thus, it is feasible to use user sessions recorded in server logs to generate performance test cases.

However, the number of test cases generated by user sessions is usually enormous and possibly unfeasible. Also, a considerable numbers of test cases are redundant, that is, the redundant test cases exercise the same behaviors then output a similar set of test results. In addition, it is time-consuming to execute all test cases. For these reasons, reduction, prioritization and selection of user sessions should be applied for user-session-based performance testing. These techniques can effectively enhancing the efficiency and performance of test suites. Clustering, a widely used technology in data mining area is capable of organizing the objects into groups whose members are similar in some way. By using cluster technology, user sessions can be clustered into different classes which can be taken as different test cases. For each test suite, typical test cases are selected and the redundant ones are deleted. In this case, the numbers of test cases are decreased and remaining ones are typical and valid.

The approach proposed in this paper aims at reducing the manual effort when test engineers design the performance test cases, outputting the realistic, valid, optimized test suites to simulate system behaviors. One innovative key point is applying sequence alignment algorithm to calculate the similarity between each user session, which makes the generated test cases more meaningful for the reason that the order of user sessions is ensured (user session is sequence of event in a click of visitation, the use of sequence alignment algorithm maintains the order of URL in a user session). The other key point is that agglomerative hierarchical clustering algorithm is employed to cluster user sessions and then pick up the representative user sessions as test cases.

The remaining sections of this paper are organized as follows. In Section 2, a brief overview of the background and related work is provided. In Section 3, a modified hierarchy clustering algorithm for user-session-based performance testing is proposed and a detailed description is given. Section 4 contains a case study illustrating the process of the proposed algorithm and analysis of the performance. Section 5 is the final conclusion.

In summary, this paper is dedicated to providing automated performance testing methods, which has been applied in the performance testing of E-commerce and education platforms such as Guangdong cloud education. The methods proposed in this paper have helped them find out the potential problems in the web systems, which provide service for dozens of schools in Guangdong. In the future, this paper plan to deploy more testing services using this algorithm, and collect a variety of runtime information to perform corresponding analysis. With the development of mobile applications, this algorithm will be applied in mobile internet testing services and performance evaluation.

2 Background and Related Work

2.1 Performance Testing

Up to date, there are two kinds of methods which prevail in generating test cases in the performance testing area: hand-writing script and record/replay method.

In many cases, due to the complexity and the lack of effective automated test tools, performance testing is still done by hand-writing scripts that describe user behavior as a subprogram [4-5]. For each virtual user, the subprogram is called and each subprogram describes certain aspects of each virtual user's behavior. The hand-written scripts method is the most flexible, as it suits all kinds of Web applications. However, it needs a lot of manual work to construct the whole test script

The Record/replay technology is also widely employed by many automated test tools. For example, load testing industrial product LoadRunner [6] can record the actions performed by real users, and then simulate thousands of concurrent users to put the application through the rigors of real-life user loads, while collecting information from key infrastructure components (Web servers, database servers etc.) The results can then be analyzed in detail, to explore the reasons for particular behavior. It takes no great programming skills to conduct the test cases, but test engineers usually spend much time on recording or designing the test scripts manually. Moreover, it only represents the operations of test engineers and cannot stand for all the true users, which means the loads generated by test script may not be in accord with the loads generate by actual users, therefore, the validity of the testing results cannot be ensured.

Actually, there already exist some studies about user-session-based approach for assisting Web application performance testing. In [7], Diwakar et al. use request logs for system, present a technique to automatically create a synthetic workload that has specified characteristics and maintains the correct inter-request dependencies. In paper [8], a method leveraging user session data to assist Web application performance testing is proposed. The researchers discussed the prospect of a performance test area and indicated the insufficiency of traditional performance test approach. But they only present a process model and did not give a specific method to cluster the user sessions or to optimize the test cases. In [9], Quan and her colleagues presented a session-based user behavior Meta model (SUBM) to automatically generate test cases for user level QoS load testing. SUBM applied decision-tree-based algorithm to cluster name-value pairs in server logs for automatic test data generation and employed the Longest Common Subsequence (LCS) algorithm to seek common HTTP request sequences for test scripts generation. However, their approach did not apply a cluster algorithm but simply used the LCS algorithm to process the URL sequences, where time complexity is high and the test suit is incomplete.

2.2 User-session-based Technique and Web Sessions Clustering

Although most research on user-session-based technique was done for functional testing, it's still worth reviewing predecessors' work. In paper [10], Janhavi and Singh proposed a theory that the selection of test cases can improve test efficiency because considerable test cases can lead to similar test results. Nowadays, some researchers have applied cluster algorithm to assist test cases selection. For example, Felderer M regard user sessions as entities when apply it to cluster user sessions [11], on that basis, they achieve user-session clusters by using a clustering algorithm, and find the representative user session from each test suite.

However, most of the methods presented in the literature to deal with cluster user sessions treat sessions as sets of visited pages within a time period and do not consider the sequence of click-stream visitation. It is significant to maintain the order of user sessions because the order of URLs in user sessions reflects the order of the operations of real users. In order to generate the real loads, it's essential to consider the order of each user sessions. Most of the cluster algorithms, when used for clustering user sessions, have treated sessions as unordered sets of URLS. The measures used to compare the similarity between sessions are simply based on intersections of these sets, such as the cosine measure (exp.1 left) or Jaccard measure(exp. 1 right) [12] as follow, in which X and Y means the set of elements.

$$sim(X,Y)=\frac{X^{t}\cdot Y}{\|X\|\|Y\|}\quad sim(X,Y)=\left|\frac{X\cap Y}{X\cup Y}\right|. \tag{1}$$

For example, there is no difference between the session "abcd", "bcad" and "abdc" by using Jaccard Coefficient similarity measure [13]. However, when used for performance test cases, they are quite different: "abcd" is more similar to "abdc" than to "bcad". In this paper, it's believed that a URL sequence could better represent a user session than a set.

In section 3, details of the improved hierarchy clustering Algorithm for user-session-based performance testing is discussed. In addition, a novel method of calculating the similarity between each session and a method of selecting typical test cases are presented. Table 1 shows the comparison between our method with traditional performance testing method.

Table 1. Compare between traditional performance testing method with our method

Method	scalability	Analyzing ability	Degree of automation	scenarios
Traditional Performance Testing	low	weak	weak	narrow
User-session-based Method	high	strong	strong	wide

3 User-session-based Performance Testing Based on A Improved Hierarchy Clustering Algorithm

In this section, the process model of our method along with the algorithm is first presented in Fig. 1, and then this paper describes the detail of the algorithm.

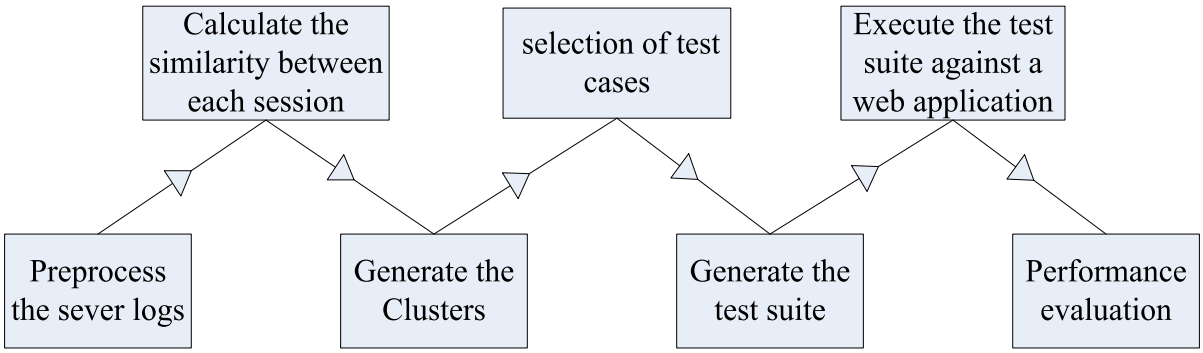

Fig. 1. Process model of the modified hierarchy clustering algorithm

Our method firstly preprocesses the server logs, and then uses a dynamic programming algorithm to calculate the similarities between two user sessions. Secondly, a bottom-up agglomerative hierarchical clustering is employed to cluster the test cases based on user sessions. Thirdly this paper propose a method to select the representative test cases. Finally the typical test suite is generated.

The following algorithm on the Table 2 is proposed to measures the distance between user sessions on the basis of the similarity between user sessions, and then clusters initial user-session-based test cases. After clustering test cases into test suites, relevant and typical test cases are selected to replace suites [14].

Table 2. Clustering algorithm

An improved hierarchy-based clustering algorithm for user-session-based performance testing

Input:

Initial User Sessions $U = \{US_1, US_2, US_3 ... US_n\}$

Each User session includes the url requests $US_i = \{url_1, url_2, url_3 ... url_k\}$

Output: optimized test cases R

Algorithm Procedure:

1. Initialization. Calculate distance matrix M (a matrix contains the distance between each user sessions) according to the following formula
   $d(US_i, US_j) = |1 - sim(US_i, US_j)|$, ($d(US_i, US_j)$ means the distance between sessions US_i and sessions US_j, the definition of $sim(US_i, US_j)$ will presented in the following chapter) $C = \{C_1, C_2 C_3 ... C_m\}$, C_i means the cluster of user sessions $\{US_1, US_2, US_3 ...\}$. Initial cluster set $C_i = US_i (1 \le i \le n)$
2. Static two user sessions with the shortest distance in matrix M, then gain the most similar user sessions i and j. Then $C_i = C_i \cup C_j, C = C - C_j$
3. Recalculate the distance between C_i and C_k, $k = 1, 2 ... n \quad k \ne i \quad n_i$, n_j means the number of sessions in C_i and C_k

$$d_{avg}(C_i, C_k) = \frac{1}{n_i \cdot n_k} \sum_{US \in C_i} \sum_{US' \in C_k} sim(US, US')$$

4. Repeat Step 2 and Step 3 until no more user sessions can be clustered
5. Check the redundancy
   for each C_i in C {
   for each US_k in C_i {
   If US_k is fully contained in C_i ($US_k \subseteq US_l, US_l \in C_i$,) then $C_i = C_i - \{US_k\}$;
   }}
6. output the optimized test cases R. $R = C_i$ Output R

3.1 Similarities between User Sessions

The first problem needed to be solved is how to measure the similarity between sessions on the premise of ensuring the URLs' order. Most of the previous related works apply either Euclidean distance for vector or set similarity measures, but the disadvantages are obvious:

(1) The transferred space could be of very high dimension;

(2) The original click stream is naturally a click sequence which cannot be fully represented by a vector or a set of URLs where the order of clicks is not considered;

(3) Euclidean distance has been proven not to be suitable for measuring similarity in categorical vector space [15].

There exist two steps in our definition of session similarity. Firstly, it is necessary to define similarity between two Web pages because each session includes several Web pages; the second step is to define session similarity using page similarity as an inner function.

3.2 Preprocess Server Logs Files and Calculate the Similarity between Web Pages

It's known that most of the Web applications use a directory tree [16] to organize Web pages or other kinds of recourses. Therefore, every URL's position is layered. We can make use of this characteristic to measure the difference between each URL.

One example is like the following three URL in Table 3.

Table 3. Web pages URLS.

URL#1	http://www.unl.edu/ucomm/research/index.html
URL#2	http://www.unl.edu/ucomm/research/current.html
URL#3	http://www.unl.edu/newsroom/index.html

It's not difficult to find out that *URL*#1 and *URL*#2 are very similar pages in the same subdirectory "research". There is some similarity between *URL*#1 and *URL*#3, but the similarity is obviously not as strong as the similarity between *URL*#1 and *URL*#2 in the previous example. A systematic method is needed to give a numerical measure for calculating the similarity between two URLs.

In order to measure the similarity between two Web pages, each level of a URL is presented by a token; thus the token string of the full path of a URL is the concatenation of all the representative tokens of each level. This process corresponds to marking the tree structure of a website as shown in Fig. 2.

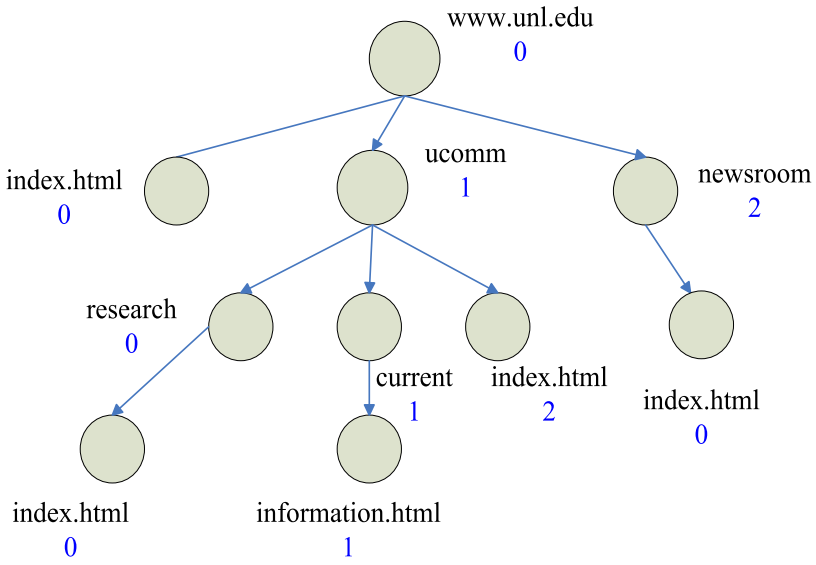

Fig. 2. A labeled tree structure of a Web application

For example, page “www.unl.edu/ucomm/current/information.html” in Fig. 2 is presented by the token string as “0111” and “www.unl.edu/newsroom/realeases/publications.html” is presented by token string as “0200”. Any URLs can be converted into a token string.

It should be emphasized that each URL contains at least two tokens (the first token must be root directory ‘0’). Our Web similarity computation is divided into two steps:

Step 1: Each corresponding token of the two token strings is compared one by one from the beginning, and this process stops at the first pair of tokens which are different, then records the number of the tokens which are the same as *N*.

Step 2: Suppose the length of the longest token string is *M*. The similarity of two Web pages is computed by first determining the length of the longest token string among the two; and then a weight is given to each level of tokens: the last level of the longest token string is given weight 1; the second to the last is given weight 2, etc. Next, the similarity between two token strings is defined as the sum of the weight of those matching tokens divided by the sum of the total weights. The formula is as follow (*M* means the length of the longest token string, and *N* means the number of the tokens which are the same):

$$sim(url_a, url_b) = \frac{\sum_{i=M-N+1}^{M} i}{\sum_{j=1}^{M} j}. \quad \textbf{(2)}$$

For example, the similarity between strings “010” and “011” is 0.833.

3.3 Calculate the Similarity between Two User Sessions

After defining the method of calculating the similarity between two urls, this paper can measure sessions’ similarity on the basis of it, and use a dynamic programming technique to find the best matching between two sequences. Most of the sequence alignment algorithms are on basis of dynamic programming, In fact, sequence alignment is a fundamental operation in bioinformatics [17]. Pairwise sequence alignment is used to determine homology in both DNA and protein sequences to gain insight into their purpose and function. In our paper, we borrow the basic ideas from sequence alignment to calculate the similarity between each session.

This paper use a scoring system which helps find the optimal matching between two session sequences. An optimal matching is an alignment with the highest score [18]. The score for the optimal matching is then used to calculate the similarity between two sessions. These are the principles in matching the sequences:

Definition 1: The session sequences can be shifted right or left to align as many pages as possible. For example, session# 1 includes a sequence of URLs 1, 12, 121, 122; here each Web page is represented by its token string as described previously. Suppose session#2 includes a sequence of visiting to URLs 12, 121, and 122. The best matching between the two User-Session (*US*) sequences can be achieved by shifting session#2 in the Table 4.

Table 4. User-Session#1 and User-Session#2

US#1	1	12	121	122
US#2	-	12	121	122

Definition 2: Gaps are allowed to be inserted into the middle, beginning or end of session sequences. This is helpful for achieving better matching.

Definition 3: In our program, each identical matching, i.e. a pair of pages with similarity 1.0, is given a positivescore 20; each mis-matching, i.e. a pair of pages with similarity 0.0 or match a page with a gap, is given a *penaltyscore* (penaltyscore should be negative) −10(The value of positivescore and penaltyscore depends on the length of the user sessions). For a pair of pages with similarity α, where $0.0 \le \alpha \le 1.0$, the score for their matching is between −10 and 20. However, the matching score corresponds with the length of the longest token string. The final matching similarity is:

$$sim(US_1, US_2) = \frac{totalscore}{Maxlength(US_1, US_2) \cdot postivescore}. \quad (3)$$

For example, Table 5 shows the matching score of *US*#1 and *US*#2.

Table 5. Scores of *US*#1 and *US*#2

US#1	1	12	121	122
US#2	-	12	121	122
Score	-10	20	20	20

According to the equation (3), the similarity between *US*#1 and *US*#2 is 0.625.

We do not simply count the number of identical Web pages when session sequences are aligned. Instead, a scoring function based on Web page similarity measure is employed. For each pair of Web pages, the scoring function gives a similarity score where a higher score indicates higher similarity between Web pages. A pair of identical Web pages is only a special case of matching, the scoring function returns 1.0 means the two sessions are exactly the same. Suppose there are two user sessions:*US*#3= {1, 123, 126, 1, 2}, *US*#4= {1, 12, 123, 124, 12, 22}.

The process can be described by using a matrix as shown in Fig. 3. One sequence is placed along the top of the matrix and the other sequence is placed along the left side. There is a gap added to the start of each sequence which indicates the starting point of matching. Any step in any path can only go right, down or diagonal. Every diagonal move corresponds to matching two Web pages. A right move corresponds to the insertion of a gap in the vertical sequence and matches a Web page in the horizontal sequence with a gap in the vertical sequence. A down move corresponds to the insertion of a gap in the horizontal sequence and matches a Web page in the vertical sequence with a gap in the horizontal sequence.

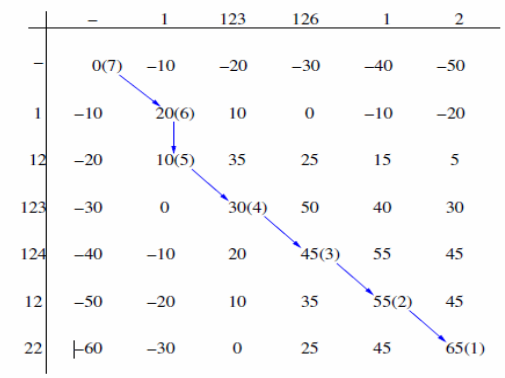

Fig. 3. Session matching matrix

In order to solving the optimal matching problem, the dynamic programming algorithm propagates scores from the matching start point, to the destination point of the matrix. The optimal path is then achieved through back propagating from destination point to starting point. In the given example, the optimal path found through back propagating is connected by arrows where the numbers in brackets indicate the step number in back propagating. The score of any element in the matrix is the maximum of the three scores that on the left, above or the above-left of itself.

According to the sequence alignment algorithm in [19], the score of x_4 is the maximum score among $x_1 + sim(c,d)$ (the corresponding two Web pages can match), $x_2 + penaltyscore$ (insert a gap in the vertical sequence) and $x_3 + penaltyscore$ (insert a gap in the horizontal sequence).

The score that ends up in the lower-right corner is the optimal sequence alignment score. After finding the final score for the optimal session alignment, the final similarity between the two sessions is computed by considering the final optimal score and the length of the two sessions.

Table 6 Shows the final matching score of *US*#3 and *US*#4.

Table 6. Scores of *US*#3 and *US*#4

US#3	1	-	123	126	1	2
US#4	1	12	123	124	12	22
Score	20	-10	20	15	10	10

According to the formula (3), the similarity between *US*#1 and *US*#2 is 0.5416.
Table 7 shows the algorithm of this part.

Table 7. Sequence algorithm

Sequence Algorithm

Input: Initial user sessions US_1 and US_2; $US_i = \{url_1, url_2, ...url_n\}$. The length of US_1 is M and the length of US_2 is N.

Output: The matching score between US_1 and US_2

Algorithm Procedure

1. Initial the matching matrix *Ma* , *penaltyscore*, *postivescore*.
   Ma[0][0]=0; for(i = 1;i<=*M*;i++){
   Ma[0][i]=*M*[0][i-1]+*penaltyscore*;
   }
   for(j = 1;i<=*N*;j++){
   Ma[j][0]=*M*[j-1][0]+*penaltyscore*;
   }
2. One sequence is placed along the top of the matrix and the other sequence is placed along the left side.
   for(i = 1; i<=*M*; i++){
   for(j=1; j<=*N*; j++){
   Ma[i][j]=Max(*Ma*[i-1][j-1]+ $sim(US_1[i], US_2[j])$,
   Ma[i-1][j]+*penaltyscore*, *Ma*[i][j-1]+*penaltyscore*)
   }}
3. Calculate the final matching score. *totalscore*=the biggest score in matrix *Ma*

$$sim(US_1, US_2) = \frac{totalscore}{Maxlength(US_1, US_2) \cdot postivescore}$$

From the above algorithm, we can analyze that both the time complexity and space complexity are $O(M \times N)$. Compared with DNA sequence (millions of bases), the URL sequence is relatively short (dozens to hundreds of clicks at most per session). Therefore, it is feasible to use above algorithm to calculate the similarity between each sessions.

3.4 Clustering User Sessions with Agglomerative Hierarchical Clustering

Based on the distance measure method in the above description, an agglomerative hierarchical clustering algorithm is employed to cluster the test cases generated from user sessions now. In data mining [20], an agglomerative hierarchical clustering algorithm is a method of cluster analysis which seeks to build a hierarchy of clusters. Each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy until the terminal condition is met. Compared to *K* partitioning clustering, the advantage of agglutinate hierarchical clustering is the input parameters are fewer. Also, an agglomerative hierarchical clustering can apply to numerical data as long as categorical data. Fig. 4 shows how the agglomerative hierarchical clustering algorithm works.

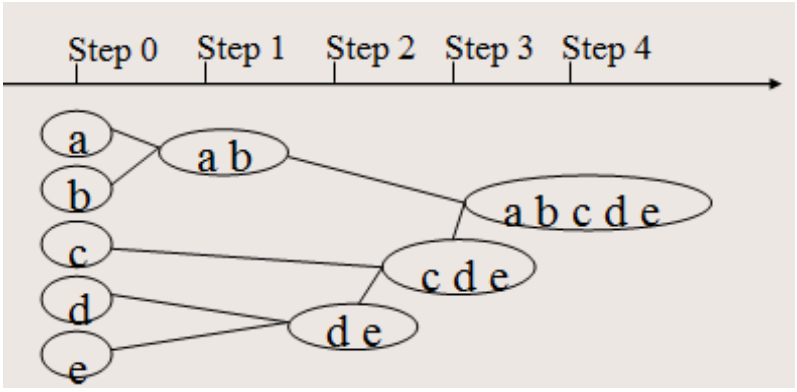

Fig. 4. Agglutinate hierarchical clustering

The modified clustering algorithm is described as follow.

Step 1: Initially, each test case is regarded as a test suite. The initial distance matrix is generated, and then two sessions which have the minimum distance into one test suite are clustered from the distance matrix.

Step 2: If the numbers of test suites meets demand, quit, otherwise continue clustering (a minimum distance $d_{\min}(i,j)$ is defined to stop the algorithm), then calculate the distance matrix. The distance between new clusters can be calculated by formula 4.

$$d_{avg}(C_i,C_j)=\frac{1}{n_i n_j}\sum_{\text{us}\in C_i}\sum_{us'\in C_j}\left|\text{US}-\text{US}'\right|. \tag{4}$$

Where C_i denotes the ith test suite; n_i and n_j are the size of each test suite; $d_{avg}(C_i,C_j)$ denotes the average distance between two clusters.

Step 3: Output the final clustering result of user sessions.

3.5 Generation of Test Cases

Performance test case is a sequence of URL request with time interval, it defines all the actions that a group of virtual users have to perform. Performance test case is the guidance of Web application performance testing, therefore the design of test cases is crucial.

Performance test cases mainly concern about the number of concurrent users [21], as well as the distribution characteristics of different types of users, in order to analyze the condition of Web application. Our method utilizes user sessions in server logs to help construct test cases. The utilization of user session can analyze how users access Web application. Each cluster of user sessions represents a kind of user access pattern and form a test case for a group of users who submitted the same transaction. Also, this paper can obtain the proportion of user access patterns as well as the proportion of different types of user access pattern in different time segment. According to the results of analysis provided by user sessions, the proportion of each type of test cases in a test suite is decided.

4 Experiments

In this section, this paper will show a running example, an e-business ordering Web application, to illustrate the test cases generation process.

4.1 E-Commerce Site

A prototype for online Electronic Device shopping Website is designed for the experiment by using JSP, Java Bean and Servlet. The running platform is a Tomcat server while Oracle is employed to manage the database. The main component modules include Home, Registration, Shopping Cart, Group Purchase, and Login. The experiment mainly concentrates on the functions such as searching, browsing, registering, shopping cart in the Web site, which are similar to other popular Websites. The database is composed of

twelve tables for recording Electronic Device information, transaction and other data objects. JavaScript and Cookies are used to offer identity authentication and personal service. 267 Electronic Devices of the Web site are divided into five categories, such as Computer Device, Internet Security Device, Office equipment and so on.

4.2 Collecting User Sessions

In order to collect user session data, our team recruited some college students as volunteers. The volunteers play the role as customers, then visit the e-commerce site with their own habits. Considering time and efficiency of the experiment, one week's Web log files are collected.

4.3 Preprocess the User Session

After collecting the user sessions from the server logs, it is necessary to preprocess the session data. Uncorrelated and invalid data should be eliminated from the user sessions. There are two types of invalid data should be eliminated:

Usually, the users of Web application do not explicitly request the graphics files and css files. Therefore, the URL which ends with jsp, jpeg, JSPG, GIF, css and js should be deleted.

The failed requests, whose return code are 4xx (request error) or 5xx (server error).

After cleaning the user sessions, each URL is represented by a token as illustrated in section 3.

4.4 Case Study

After data preprocessing, 120 initial user sessions are collected. In order to applying the clustering algorithm, the user sessions are expressed as the token.

For examples, in the Fig. 5 there are 10 user sessions in the shape of token array.

$$u_0[] = \{"00","01","01","021","0213"\}$$
$$u_1[] = \{"00","00","01","00","03"\}$$
$$u_2[] = \{"00","01","016","21"\}$$
$$u_3[] = \{"00","01","023","0231","0233"\}$$
$$u_4[] = \{"00","03","022","021","00","012"\}$$
$$u_5[] = \{"00","00","01","021","0213"\}$$
$$u_6[] = \{"00","02","01","021","022"\}$$
$$u_7[] = \{"00","02","01","012","021"\}$$
$$u_8[] = \{"00","01","013","022","031"\}$$
$$u_9[] = \{"00","00","01","0213","021"\}$$

Fig. 5. User sessions in the shape of token array

The character in the quotation mark denotes the URL which the user visits. According to the algorithm in Section 3, the distance matrix is calculated for the first time.

Fig. 6 shows the result. If distance is larger than 0.6, the clustering will be terminated.

	u0	u1	u2	u3	u4	u5	u6	u7	u8	u9
u0	0.00	0.37	0.13	0.55	0.80	0.36	0.24	0.58	0.20	0.61
u1	0.37	0.00	0.36	0.44	0.73	0.59	0.78	0.19	0.37	0.59
u2	0.13	0.36	0.00	0.60	0.80	0.44	0.38	0.56	0.34	0.70
u3	0.55	0.44	0.60	0.00	0.71	0.70	0.78	0.70	0.55	0.70
u4	0.80	0.73	0.80	0.71	0.00	1.37	1.13	0.77	0.58	1.37
u5	0.36	0.59	0.44	0.70	1.37	0.00	0.55	0.51	0.61	0.10
u6	0.24	0.78	0.38	0.78	1.13	0.55	0.00	0.78	0.45	0.78
u7	0.58	0.19	0.56	0.70	0.77	0.51	0.78	0.00	0.58	0.51
u8	0.20	0.37	0.34	0.55	0.58	0.61	0.45	0.58	0.00	0.61
u9	0.61	0.59	0.70	0.70	1.37	0.10	0.78	0.51	0.61	0.00

Fig. 6. Distance matrix for the first time

The minimum distance in the matrix is d($u5$, $u9$)=0.10, so user cases $u5$ and $u9$ are clustered into one test suite, then calculate new distances among new test suites, and Fig. 7 shows the first clustering result.

	u0	u1	u2	u3	u4	(u5,u9)	u6	u7	u8
u0	0.00	0.37	0.13	0.55	0.80	0.49	0.24	0.58	0.20
u1	0.37	0.00	0.36	0.44	0.73	0.59	0.78	0.19	0.37
u2	0.13	0.36	0.00	0.60	0.80	0.57	0.38	0.56	0.34
u3	0.55	0.44	0.60	0.00	0.71	0.70	0.78	0.70	0.55
u4	0.80	0.73	0.80	0.71	0.00	1.37	1.13	0.77	0.58
(u5,u9)	0.49	0.59	0.57	0.70	1.37	0.00	0.67	0.51	0.61
u6	0.24	0.78	0.38	0.78	1.13	0.67	0.00	0.78	0.45
u7	0.58	0.19	0.56	0.70	0.77	0.51	0.78	0.00	0.58
u8	0.20	0.37	0.34	0.55	0.58	0.61	0.45	0.58	0.00

Fig. 7. Distance matrix for the first clustering

In the same way, the minimum distance $d(u_0,u_2)=0.13$, then cluster u0 and u2 into one test suite, and then calculate the distances among new test suites.

Repeat the same calculation. And Fig. 8 shows the clustering result for the sixth time.

	(u0,u2,u8)	(u1,u5,u9)	(u6,u7)	u3	u4
(u0,u6,u8)	0.00	0.69	0.87	0.63	0.83
(u1,u7,u9)	0.69	0.00	0.93	0.64	1.06
(u6,u7)	0.87	0.93	0.00	0.77	0.85
u3	0.63	0.64	0.77	0.00	0.71
u4	0.83	1.06	0.85	0.71	0.00

Fig. 8. Distance matrix for the sixth time

The test suite set calculated from the cluster result is as below:

$$S_1 = \{u_0, u_2, u_8\},\ S_2 = \{u_1, u_5, u_9\},\ S_3 = \{u_6, u_8\}\ \ S_4 = \{u_3\},\ S_5 = \{u_4\}$$

At last, five test case: $TC_1 = \{u_0, u_2, u_8\}$, $TC_2 = \{u_1, u_5, u_9\}$, $TC_3 = \{u_6, u_8\}$, $TC_4 = \{u_3\}$, $TC_5 = \{u_4\}$ are produced. The proportions of each test case in a test suit are 30%, 30%, 20%, 10%, 10%.

4.5 Experiment Result and Verification

To validate the effectiveness of our method, this paper must insure the behavior of virtual users generated by test cases corresponds with realistic users' behavior (both behavior generate same loads against the application). In this experiment, this paper choose the LoadRunner as playback tool to replay the test cases.

The first test uses all the initial user sessions as test cases to test the application, and the second test uses the test cases generated by the method presented in this paper. The comparison results of two tests are shown in Fig. 9 and Fig. 10.

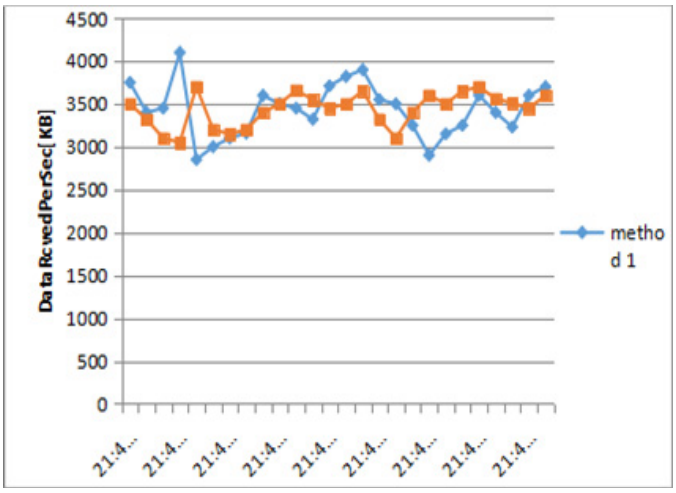

Fig. 9. Comparison on throughput between two methods

Fig. 10. Comparison on http-hit between two methods

From the comparisons above, it can be seen that, the trend curves of throughput and http-hits of two tests are relatively consistent. This result proves that it is valid to utilize user sessions to assist performance testing.

Compared with the traditional performance testing methods in section 1, the performance testing method in this paper can effectively optimize the system by analyzing Web logs, so as to achieve the goal of automatic performance testing.

5 Conclusions

Today, the performance testing against a Web application is more necessary. To guarantee its reliability and quality, authentic and effective test method is needed to be applied to Web applications. For the Web application which has run for a period of time (in order to collect enough log files), a novel method named user-session-based performance testing based on a modified agglomerative hierarchical clustering algorithm is proposed. This selects test cases produced from user sessions by employing a sequence alignment algorithm and a bottom-up hierarchical clustering algorithm. The key technique calculates the diverging distance between each user session on the premise of maintaining URL's order.

The testing method presented in this paper has several advantages over the other performance testing techniques. First, by utilizing user sessions in server logs, our method generates test cases automatically, potentially reducing the effort involved when test engineers are required in test case generation. Second, by leveraging user session data of real users, the virtual users can generate the same workloads as real users do, which can ensure test result's authenticity and validity.

However, the work of testing Web applications based on mining user sessions is a complex systematic project. It is not easy to get an effective and practical tool and is over-dependent on the session data.

In summary, this paper is dedicated to providing automated performance testing methods, which has been applied in the performance testing of E-commerce and education platforms such as Guangdong cloud education. The methods proposed in this paper have helped them find out the potential problems in the web systems, which provide service for 28 primary schools and tens of thousands of people in Zhuhai, Guangdong. In the future, this paper plan to deploy more testing services using this algorithm, and collect a variety of runtime information to perform corresponding analysis. With the development of mobile applications, this algorithm will be applied in mobile internet testing services and performance evaluation.

Acknowledgements

This work was supported in part by the National Nature Science Foundation of China (No. 61370103), Guangdong Province Application Major Fund (2015B010129007), Guangzhou Produce and Research Fund (201802020006) and Zhongshan Produce & Research Fund (2016A1038).

References

[1] D. Huang, B. He, C. Miao, A survey of resource management in multi-tier Web applications, IEEE Communications Surveys & Tutorials 16(3)(2014) 1574-1590.

[2] M. Shahbaz, P. Mcminn, M. Stevenson, Automatic generation of valid and invalid test data for string validation routines using web searches and regular expressions, Science of Computer Programming 97(2014) 405-425.

[3] P.E. Román, R.F. Dell, J.D. Velásquez, P.S. Loyola, Identifying user sessions from web server logs with integer programming, Intelligent Data Analysis 18(1)(2014) 43-61.

[4] D. Zhang, Web services composition for process management in E-business, Data Processor for Better Business Education 45(2)(2016) 83-91.

[5] A. Ali, N. Badr, Performance testing as a service for web applications, in: Proc. IEEE Seventh International Conference on Intelligent Computing and Information Systems, 2016.

[6] A. Kattepur, M. Nambiar, Service demand modeling and performance prediction with single-user tests, Performance Evaluation 110(8)(2017) 1-21.

[7] Z. C. Liu, D. E. Ren, P. P. Wang et al., Concurrent workload simulation for application performance testing, 2017.

[8] L. Lu, X. Quan, Session-based user behavior meta-model of Web applications for user-level QoS load testing, in: Proc. Supply Chain Management and Information Systems (SCMIS), 2010.

[9] M. Daoud, L. Tamine-Lechani, M. Boughanem, B. Chebaro, A session based personalized search using an ontological user profile, in: Proc. the 2009 ACM symposium on Applied Computing, 2016.

[10] Janhavi, A. Singh, Efficient regression test selection and recommendation approach for component based software, in: Proc. 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2014.

[11] M. Felderer, E. Fourneret, A systematic classification of security regression testing approaches, International Journal on Software Tools for Technology Transfer 17(3)(2015) 305-319.

[12] S. Deshpande, J. Rathi, S. Gandhi, M. Shinde, V. Deshmukh, Sentiment Analysis Tool using Cosine and Jaccard Implementation, International Journal of Computer Applications 115(12)(2015) 17-19.

[13] N. Kumar, A.R. Reddy, Frequent segment clustering of test cases for test suite reduction, WSEAS Transactions on Computers 13(13)(2014) 368-380.

[14] M. Sato-Ilic, P. Ilic, Visualization of fuzzy clustering result in metric space, Procedia Computer Science 96(2016) 1666-1675.

[15] P.M. Kruse, J. Nasarek, N.C. Fernandez, Systematic testing of Web applications with the classification tree method, in: Proc. the 36th International Conference on Software Engineering, 2014.

[16] D. Zhu, L. Wang, T. Wang, X. Wang, A simple linear space algorithm for computing a longest common increasing subsequence, IAENG International Journal of Computer Science 45(3)(2016) 11.

[17] I.B. Kuznetsov, M. Mcduffie, PR2ALIGN: a stand-alone software program and a web-server for protein sequence alignment using weighted biochemical properties of amino acids, BMC Research Notes 8(1)(2015) 1-8.

[18] R. Yang, Y. Zhao, Y. Su, C. Pan, A high-throughput gene sequence alignment strategy using parallel computing, in: Proc. 7th International Conference on Biomedical Engineering and Informatics IEEE, 2014.

[19] G. Kumar, A Survey on Clustering - A Data Mining Technique. <https://www.researchgate.net/publication/276922361_A_Survey_on_Clustering_-_A_Data_Mining_Technique>, 2015.

[20] A.S. Nagdive, R.M. Tugnayat, M.P. Tembhurkar, A.S. Nagdive, R.M. Tugnayat, Overview on performance testing approach in big data, International Journal of Advanced Research in Computer Science 5(8)(2014) 165-169.

[21] T. Mardiana, T.B. Adji, I. Hidayah, The comparation of distance-based similarity measure to detection of plagiarism in Indonesian text, in: R. Intan, CH. Chi, H. Palit, L. Santoso (Eds.), Intelligence in the Era of Big Data, Springer, Berlin, 2015, pp. 155-164.