Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Identifying worst-case user scenarios for performance testing of web applications using Markov-chain workload models

Tanwir Ahmad *, Dragos Truscan, Ivan Porres

Faculty of Science and Engineering, Åbo Akademi University, Vattenborgsvägen 5, 20500 ÅBO, Finland

ARTICLE INFO

Article history:
Received 2 August 2016
Received in revised form 1 January 2018
Accepted 20 January 2018
Available online xxxx

Keywords:
Performance testing
Markov chain
Genetic algorithms
Graph-search algorithms

ABSTRACT

The poor performance of web-based systems can negatively impact the profitability and reputation of the companies that rely on them. Finding those user scenarios which can significantly degrade the performance of a web application is very important in order to take necessary countermeasures, for instance, allocating additional resources. Furthermore, one would like to understand how the system under test performs under increased workload triggered by the worst-case user scenarios. In our previous work, we have formalized the expected behavior of the users of web applications by using probabilistic workload models and we have shown how to use such models to generate load against the system under test. As an extension, in this article, we suggest a performance space exploration approach for inferring the worst-case user scenario in a given workload model which has the potential to create the highest resource utilization on the system under test with respect to a given resource. We propose two alternative methods: one which identifies the exact worst-case user scenario of the given workload model, but it does not scale up for models with a large number of loops, and one which provides an approximate solution which, in turn, is more suitable for models with a large number of loops. We conduct several experiments to show that the identified user scenarios do provide in practice an increased resource utilization on the system under test when compared to the original models.

1. Introduction

A tremendous growth has been seen in the field of web technologies during the last two decades. The role of the web applications has changed from the traditional document presentation system to the feature-rich distributed application that is accessible worldwide. Web applications are increasingly being utilized by a large number of companies to run critical business tasks. Thus, ensuring the reliable and stable performance of web applications is imperative for these companies. Poor performance makes the end-users abandon the use of web applications and can cause reputational and financial damage to those companies which rely on web-based platforms [1].

Performance testing is the process of evaluating the responsiveness and scalability of a system under test (SUT) when it is under a certain *synthetic workload* [2] corresponding to a specified number of concurrent *virtual users*. During this process, different *key performance indicators* (KPIs) (e.g., CPU, memory utilization) are monitored in order to determine the performance level of the SUT.

In order to raise the level of abstraction and to promote the reuse and faster update of performance test specifications, in our previous work, we have investigated how the expected behavior of the users of web applications can be specified by using probabilistic models [3,4]. Such models capture the expected behavior of a set of users by encoding information about the order of their interactions with the SUT, the delay (*think time*) between these interactions, and the probability of a given sequence of interactions to occur. Each traversal of the model graph simulates a timed sequence of interactions between the virtual user and the web application. By simulating concurrently one model for each virtual user, we can generate the corresponding synthetic workload using our MBPeT model-based performance testing tool [3].

In many situations (e.g., stress testing) one would like to know, before the performance testing session begins, the worst-case user scenario in a given workload model that will potentially trigger the highest utilization of a given resource on the SUT. Such scenario can then be used to benchmark the SUT for possible performance bottlenecks. In practice, this implies finding the path in the workload model graph which will generate the sequence of interactions with the highest resource utilization on the SUT over a sustained period of time.

In this article, we attempt to answer two research questions:

* Corresponding author.
E-mail addresses: tanwir.ahmad@abo.fi (T. Ahmad), dragos.truscan@abo.fi (D. Truscan), ivan.porres@abo.fi (I. Porres).

https://doi.org/10.1016/j.future.2018.01.042
0167-739X/

1. RQ1: how can we identify the sequence of interactions in a given workload model which causes the highest utilization of a given resource of the SUT under a sustained period of time?
2. RQ2: what is the scalability of the proposed approach?

In order to answer RQ1, we propose two distinct methods for identifying the worst-case user scenario. The first method is based on graph-search algorithms and provides the exact solution, whereas the second method provides a near-optimal solution. For validation purposes, we run an example where the solutions resulting from applying the two proposed methods are used to generate synthetic workload against the SUT. We then compare the resource utilization they trigger on the SUT with the one triggered by the original workload model.

In order to answer RQ2, we analyze and compare the two methods with respect to their complexity and, respectively, to the precision of the solution, and discuss their benefits and drawbacks.

The rest of the paper is structured as follows: Section 2 provides background information on our proposed approach. We describe the process of load generation for performance testing in Section 3. Section 4 presents an overview of the related work. In Section 5, we describe in detail the steps of our approach. We empirically validate and evaluate our approach in Section 6, while we present conclusions in Section 7.

2. Using Markov chains for modeling the workload

A Discrete Time Markov Chain (DTMC) [5] is a discrete time stochastic process which has the property that given the current state, the future of the process is conditionally independent of the previous states. Let $X_n, n = 0, 1, 2, 3$ be a stochastic process which takes on a finite number of states or values which can be written as a set of non-negative integers $\{0, 1, 2, \ldots\}$. If the process is currently in state s_n at time n, we denote it as $X_n = s_n$. If we suppose that whenever the process is in state s_n, the process will change its state to s_{n+1} with a fixed probability $P_{s_n s_{n+1}}$, then we can state the property of Markov chains as

$$P\{X_{n+1} = s_{n+1} | X_n = s_n, X_{n-1} = s_{n-1}, \ldots X_1 = s_1, X_0 = s_0\} = P_{s_n s_{n+1}}$$

where $P_{s_n s_{n+1}}$ denotes the probability of transitioning from one state to another state in a single step (or one unit of time), and it is known as the *one-step transition probability*.

We model the expected behavior of the users using a slightly modified version of DTMC model, which we formally define as a tuple $M = (S, T, U^r, Util^r, s_I)$ where:

1. $S = \{s_0, s_1, \ldots, s_n\}$ is a finite set of states;
2. $T = \{t_0, t_1, \ldots, t_n\}$ is a finite set of transitions, such that $t_i = \{\langle s_i, s_j \rangle | s_i, s_j \in S\}$ for all $i, j\ 0 \leq i, j \leq n$;
3. $U^r = \{u_0^r, u_1^r, \ldots, u_n^r\}$ is a finite set of resource utilizations for a given resource r, and u_i^r is in correspondence with s_i. $Util^r$ is a mapping function from s_i to u_i^r so that $Util^r(s_i) = u_i^r$;
4. $s_I \in S$ is the start state.

Informally, we extend DTMC with two additional labels on the edges: probability value and think time. The *probability value* specifies the chances of that particular edge being chosen according to a probability mass function, whereas the *think time* represents the amount of time that a user waits between two consecutive interactions. In addition, each state in the DTMC model is tagged with an *action* specifying the interaction between the user and the SUT. An action specifies either an HTTP request or a set of HTTP requests that the user sends to the SUT whenever the corresponding state is visited.

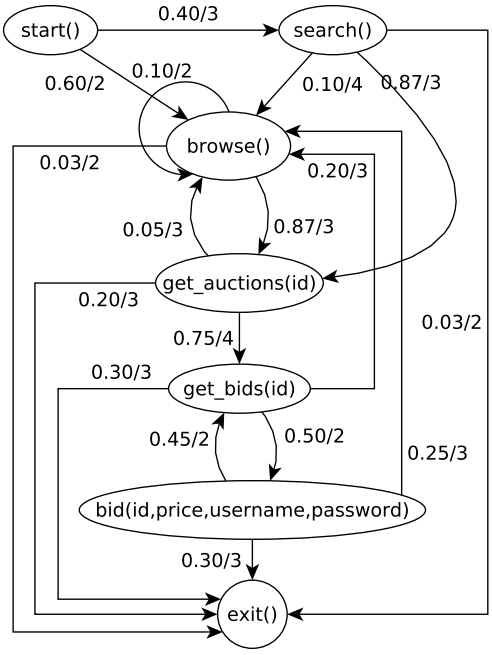

Fig. 1. Markov Chain model of a user.

The DTMC model in Fig. 1 shows a workload model of an auctioning web application, which allows registered users to search, browse, and bid on auctions that other users have created. For instance, after performing a *browse()*, the user can execute either *get_auctions()* action with a probability of 0.87 (after waiting for 3 s) or *exit()* with a probability of 0.03 after waiting for 2 s. In the model, *start()* and *exit()* are pseudo-states which are only used to indicate the initial and the optional final state of the model, respectively, and they cause no interaction with the SUT.

Different works suggested that such workload models can be obtained from either the requirements of the SUT or Service Level Agreements (SLAs) [4], or by analyzing the historical usage of the system [6–8]. The workload model in Fig. 1 is built using the latter approach following the method described in [6].

3. Workload generation

In this paper, we use our MBPeT (Model-based Performance Testing) [3] tool for load generation. MBPeT is an online performance testing tool, which generates a synthetic semi-random workload against the SUT by simulating a workload model, such as the one in Fig. 1 for each concurrent virtual user. The simulation of a workload model for a virtual user begins from the *start()* state. On each state, the tool chooses the next state according to the probability mass function of the current state, while observing the think time values on the visited edges. The simulation ends when the *exit()* state is reached. In short, a sequence of states is generated and executed during the simulation. Whenever a state is visited, the corresponding action is executed against the SUT via a test adapter.

There are different parameters of the testing process that can be provided as command line parameters to the tool such as a ramp function (specifying the amount of concurrent virtual users during a test session), duration of the test session, etc. The workload is generated in a distributed fashion, using multiple load generating nodes, and applied in real-time to the SUT, while measuring several key performance indicators, such as response time, throughput,

Please cite this article in press as: T. Ahmad, et al., Identifying worst-case user scenarios for performance testing of web applications using Markov-chain workload models, Future Generation Computer Systems (2018), https://doi.org/10.1016/j.future.2018.01.042.