Automatic Performance Space Exploration of Web Applications Using Genetic Algorithms

Tanwir Ahmad and Dragos Truscan
Faculty of Science and Engineering
Åbo Akademi University
Vattenborgsvägen 5, 20500 ÅBO, Finland
{tanwir.ahmad, dragos.truscan}@abo.fi

ABSTRACT

We describe a tool-supported performance exploration approach in which we use genetic algorithms to find a potential user behavioural pattern that maximizes the resource utilization of the system under test. This work is built upon our previous work in which we generate load from workload models that describe the expected behaviour of the users. In this paper, we evolve a given probabilistic workload model (specified as a Markov Chain Model) by optimizing the probability distribution of the edges in the model and generating different solutions. During the evolution, the solutions are ranked according to their fitness values. The solutions with the highest fitness are chosen as parent solutions for generating offsprings. At the end of an experiment, we select the best solution among all the generations. We validate our approach by generating load from both the original and the best solution model, and by comparing the resource utilization they create on the system under test.

CCS Concepts

•Mathematics of computing → Evolutionary algorithms; •Information systems → Web applications; •Software and its engineering → Software testing and debugging; •General and reference → *Performance;* **•Computing methodologies →** *Modeling methodologies;*

Keywords

Performance exploration; performance testing; Markov Chain model; genetic algorithms

1. INTRODUCTION

Owing to the rapid development of the Internet, exponential growth has been seen in Web application systems. There are many factors which contribute to the popularity of web applications, for example, convenience, availability, universality and 700% increase in the number of Internet users since 2000 [9]. Web applications are being utilized in various domains such as e-commerce, social and governmental. Every day in US, 69% of the online user population uses web applications to buy different types of products [6]. Consequently, companies strive to offer high-quality services to retain Internet user base. For example, Amazon, a leading online retailer, has 270 million active customers worldwide [16].

A study reports that there are higher chances of system crashing due to performance issues (e.g., unanticipated workload) than system failures [17]. Additionally, it has been observed that 40% of the customers will abandon a web application if the response time is greater than 3 seconds [10]. Therefore, performance testing plays an important role during the development of web applications.

Performance testing is used to benchmark the system with respect to responsiveness and scalability and to detect potential performance bottlenecks under a particular synthetic workload [14]. The workload is usually generated by running pre-recorded user actions in a sequential order stored in a script. The main disadvantage of using static scripts to generate workload is that real users behave more dynamically than performing the actions serially [5]. In order to overcome this problem, in our previous work, we have proposed a model-based performance testing tool, called MBPeT [1], where we use probabilistic timed automata to specify user behaviour which later on is used for workload generation using a distributed cloud-architecture against the system under test (SUT).

In addition to probabilistic timed automata, MBPeT allows one to model the user behaviour as Markov Chain (MC) models [8]. A MC model consists of a finite set of states and edges with probabilities. The labels on the edges represent two values. Firstly, the *probability value* defines the likelihood of that particular edge being taken based on a probability mass function. Secondly, the *think time* expresses the time period that a user thinks or waits between two consecutive actions. Each state is labelled with an action which is executed whenever a state is visited. An action can be a single request or a set of requests that are sent to the system. For example, the MC model in Figure 1 represents a workload model of an auctioning web application, contain-

SAC 2016,April 04-08, 2016, Pisa, Italy
©2016 ACM. ISBN 978-1-4503-3739-7/16/04...$15.00
DOI: http://dx.doi.org/10.1145/2851613.2851864

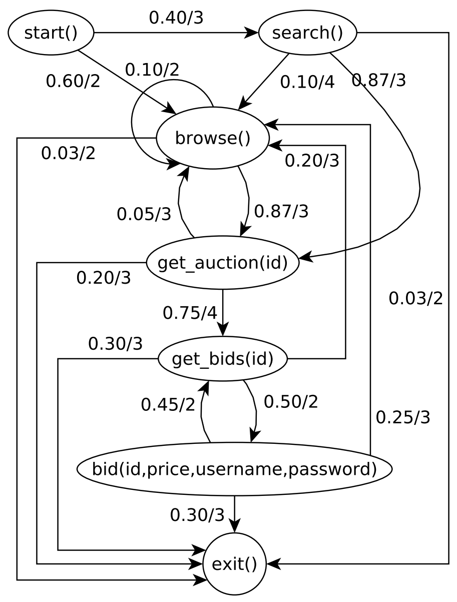


Figure 1: Markov Chain model of a user

ing different actions a user can execute against the system. For instance, after performing a *search()*, the user will execute with a probability of 0.87 (after thinking for 3 seconds) a *get_auctions()* action. In that model, the *start()* and *exit()* are dummy actions which only indicate the initial and the final state of the model, respectively. A user will always begin the execution the model at the *start()* state and finish at the *exit()* regardless of eventual loops in the model.

A workload model can be created in two ways: a tester can create the model manually based on his/her experience with the SUT [1] or one can extract automatically common behavioural patterns from web access log files using pattern mining [2]. In our previous work [1], we used workload models for performance testing, in this paper, we extend our approach to utilize the workload models for performance exploration.

A workload model built using any of the above mentioned approaches, will only represent the user behavioural patterns which are most commonly executed by the users against the SUT. As a result, there are always some rare user behavioural patterns left untested. These infrequent user behavioural patterns could degrade the performance of the SUT or even crash the system if they occur. In order to identify these eventual corner cases, in this paper we propose a method to determine the rare cases using genetic algorithms to optimize the probability distributions of a given workload model.

The rest of the paper is structured as follows: Section 2, provides an overview of the related work. In Section 3, we elaborate our evolutionary approach. Section 4 presents tool support for our approach, whereas, in Section 5, we demonstrate the applicability of our approach to an experiment. Finally, Section 6 discusses conclusions and future work.

2. RELATED WORK

Several approaches in which genetic algorithms are used for link prediction, data mining, etc. have been proposed. However, to the best of our knowledge, there is no approach that uses genetic algorithms for performance exploration.

Asllani and Lari [3] used genetic algorithms to provide a model-driven decision support system for multiple-criteria website optimization. The approach generates a solution for the best-possible arrangements of a given set of web-objects according to three multiple criteria: download time, visualization, and product association level. These criteria are related to both aesthetic design principles and the quality and relevance of the content offered. The approach differs from ours because it focuses on the usability rather than the performance of the system.

In [13], Sarukkai proposes a probabilistic link prediction and path analysis method where Markov Chains are used to dynamically model the URL access patterns that are observed in navigation logs based on the previous state. The Markov chain model can be modified on-the-fly with additional user navigation information. The models are used to predict the probability of traversing a link in the future provided a history of the visited links. This research is different from ours in the sense that we optimize the probability distribution of Markov Chains using genetic algorithms and use them for performance exploration.

3. APPROACH

We propose an approach to optimize the probability distribution of all the edges in a given workload model in order to find the worst workload model which can maximize the resource utilization of the SUT. The think time of an edge represents the duration a user waits between two consecutive actions. On average, the think time between two actions remains constant if the user-interface of a web application has not been redesigned or changed. Therefore, we keep the think time of all the edges constant during the optimization process. We use genetic algorithms (GA) [15] to explore the space of possible behavioral patterns of real users.

The GA is an optimization technique that follows the evolution paradigm. A population of solutions is maintained and an evolution process enables parent solutions to be selected from the population based on their fitness. The algorithm applies different genetic operators to the selected parents in order to create offsprings. The fitness of each solution can be related to the objective function value, in our case the expected level of resource utilization caused to SUT. Similar to biological evolution processes, the offspring solutions with good fitness levels, are more probable to survive and reproduce as compared to low fitness level solutions. After running the GA for a certain number of generations, we select the best solution among all the generations.

The following steps are performed in our GA approach (as shown in Figure 2):

3.1 Set GA parameters

Before starting a GA, we need to define the total number of generations which will give the stopping criteria and the

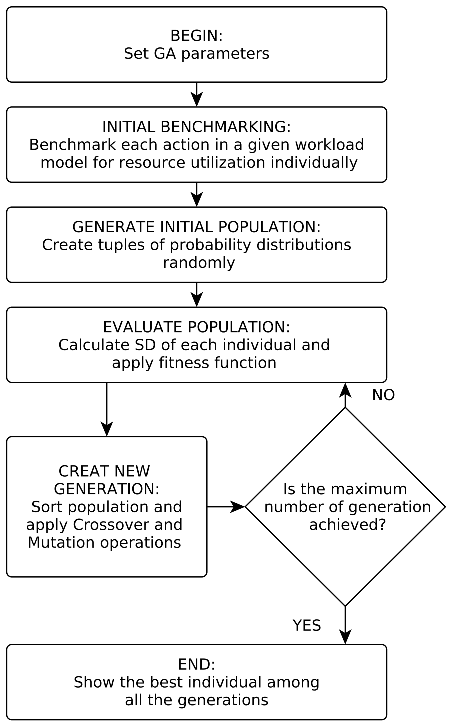

BEGIN:
Set GA parameters
INITIAL BENCHMARKING:
Benchmark each action in a given workload model for resource utilization individually
GENERATE INITIAL POPULATION:
Create tuples of probability distributions randomly
EVALUATE POPULATION:
Calculate SD of each individual and apply fitness function
NO
CREAT NEW GENERATION:
Sort population and apply Crossover and Mutation operations
Is the maximum number of generation achieved?
YES
END:
Show the best individual among all the generations

Figure 2: Applying GAs to performance space exploration

population size for each generation. Further, the crossover operator probability, mutation rate and mutation operator probability are set to control the usage of crossover and mutation operators during offsprings generation.

3.2 Initial benchmarking

We benchmark all the actions defined in a given base model individually for different resources (e.g., CPU, memory) utilization of the SUT.

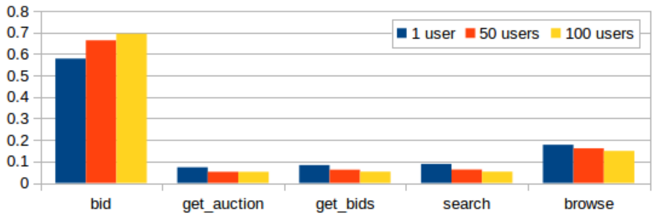
We ran several benchmarking sessions with different number of concurrent users and record the resource utilization of each action at the implementation level. Later on, we average the resource utilization of each action independently. This information is used to calculate the fitness of a solution (or model) during the evolution stage in our GA. For example, Figure 3 shows the comparison of normalized average values of resource utilization of all the actions in the model (shown in Figure 1) with 1, 50, and 100 concurrent users. We can choose the benchmark values of any session. In this paper, we have used the normalized values of the resource utilization of all the actions with one concurrent user, as listed in Table 1.

3.3 Chromosome coding

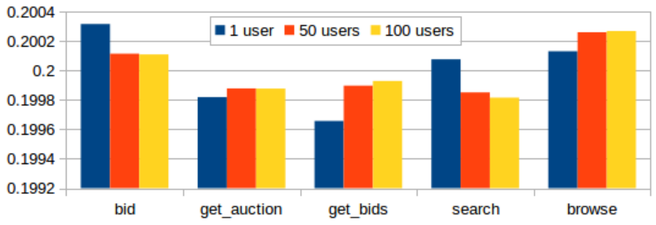
The search space of all possible solutions is mapped onto a set of chromosomes. Individual positions within a chromosome are known as genes. In our case, we are optimizing the probability distribution of the edges in a given MC model,

Table 1: CPU and Memory utilization with one concurrent user for Figure 1

Action	CPU	Memory
start()	0.0	0.0
get_bids(id)	0.083	0.198
search(string)	0.088	0.200
browse()	0.178	0.201
get_auction(id)	0.072	0.199
bid(id,price,username,password)	0.578	0.202
exit()	0.0	0.0



(a) Average CPU utilization



(b) Average Memory utilization

Figure 3: Comparison of resource utilization benchmarks with different number of users

therefore a chromosome is represented as a tuple of probability distributions of all the outgoing edges of each state in the model. Thus, a gene corresponds to the probability distribution function of a single state. For example, the model depicted in Figure 1 will be encoded into the following chromosome:

$$\langle\langle 0.60, 0.40\rangle, \langle 0.87, 0.10, 0.03\rangle, \langle 0.87, 0.10, 0.03\rangle,$$
$$\langle 0.20, 0.75, 0.05\rangle, \langle 0.30, 0.50, 0.20\rangle, \langle 0.30, 0.25, 0.45\rangle\rangle$$

where gene $\langle 0.60, 0.40\rangle$ describes the probability distribution of the *start()* state.

3.4 Initial population

A set of chromosomes is generated randomly under one constrain that the sum of the probabilities in a gene must be equal to 1. This establishes the initial population (i.e., the first generation).

3.5 Fitness evaluation

A *fitness value* is computed for each solution in the population. This value articulates the expected level of resource utilization of the SUT for the solution (i.e., MC model) each chromosome represents. Since we are searching for solutions with higher level of resource utilization, the genetic evolution will prefer chromosomes with a higher fitness value.

The fitness of a solution is computed in two steps. First, we build a transition matrix from the probability values represented in a given chromosome. The transition matrix allows us to calculate the *stationary distribution* (SD) of the solution. The stationary distribution of a Markov Chain with transition matrix P is some probability vector w, such that,

$$\lim_{n\to\infty} vP^n = w$$

where v is any probability vector and n represents the power of P matrix. This implies that, over the long run, no matter what the starting state was, the proportion of time that Markov Chain spends in state i is approximately w_i for all i. In summary, we can deduce which states in the model will be visited more frequently than the others based on their probability distributions. Next, we combine the SD with the benchmark results of resource utilization to calculate the fitness of a model. Let S be a set of all the states in the model M and vector g the CPU utilization of the action of each state s in the set of states S, then we can define the fitness of model M as follows:

$$f_M = \sum_{s\in S} g_s h_s$$

where h depicts the stationary distribution vector of model M.

For instance, consider the model in Figure 1. The stationary distribution vector x of the model and CPU utilization vector y from Table 1 can be defined as follows:

$$x = \langle 0.135, 0.195, 0.054, 0.178, 0.202, 0.097, 0.135\rangle$$
$$y = \langle 0.0, 0.083, 0.088, 0.178, 0.072, 0.578, 0.0\rangle$$

The fitness of model would be the by-product of the two vectors:

$$\begin{aligned} f &= 0.135 \times 0.0 + 0.195 \times 0.083 + 0.054 \times 0.088 \\ &+ 0.178 \times 0.178 + 0.097 \times 0.578 + 0.135 \times 0.0 = 0.11 \end{aligned}$$

3.6 Selection

A binary tournament selection method is used where two solutions are randomly chosen and between those a member with the larger fitness is chosen as parent. This procedure is repeated as long as parents have to be chosen. Two chosen parents generate two offsprings.

3.7 Offspring generation

We employ the *two-point crossover* operator to generate offsprings. The chromosomes of two parents are cut at two random points and the genes between the first and second cut points are interchanged to generate two offsprings. The application of the crossover operator is controlled by the *crossover operator probability*, which defines the probability of applying the operator to parents. Further, a mutation operator is applied to the two newly generated offsprings independently, in order to introduce gene diversity in the population. The usage of the mutation operator is regulated by two probability parameters: the *mutation operator rate* indicates the probability of using the mutation operator on a current individual; the *mutation rate* represents the probability of changing the probability distribution of a state in a selected chromosome. The newly generated offsprings replace the parents in the population.

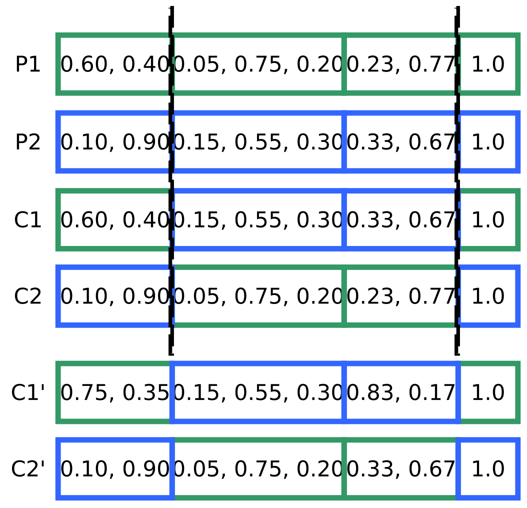


Figure 4: Two-point crossover

For example, Figure 4 demonstrates two-point crossover operator applicability where *P1* and *P2* represent two chromosomes of parents and after applying the operator we get two chromosomes of children (i.e., *C1* and *C2*). Then we apply mutation to the newborn children and acquire final set of children (i.e., *C1'* and *C2'*) which will replace the parents (P1 and P2) in the population.

3.8 Stop criteria

If the fixed number of generations is reached, the algorithm ends and we choose the best individual among all the generations.

4. TOOL SUPPORT

The proposed approach has been implemented as a tool in the Python [11] programming language. We have used DEAP [7], an evolutionary computation python framework, to set up our genetic algorithm.

The tool parses the structure of the given workload model into an internal representation. The resource utilization for each action in the model is computed manually and provided as an input to the tool. In addition to resource utilization, we also need to convey the following parameters to the tool: crossover operator probability, mutation operator rate, mutation rate, population size, and maximum number of generations. Furthermore, the tool analyzes the structure of the given base model and extracts certain pieces of information about the model, for example, number of states in the model, number of outgoing edges from each state, etc. This information is used to construct the skeleton of chromosome. Based on these parameters, the tool initiates the population and begins the evolution process. The tool records the statistics (e.g., minimum, average and maximum fitness) for each generation. After evaluating the last generation, the tool presents a workload model with the best fitness value among all the generations.

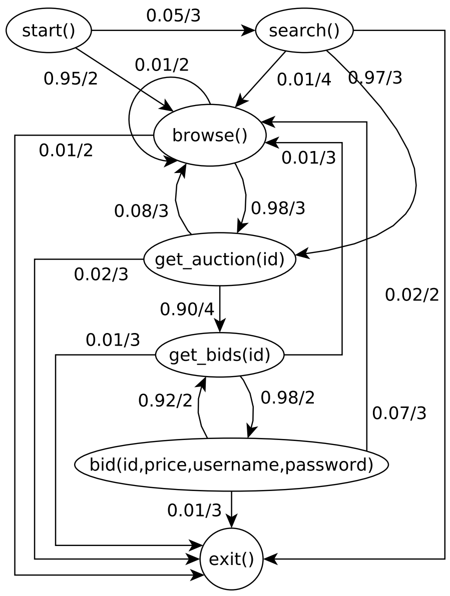


Figure 5: CPU Worst workload model

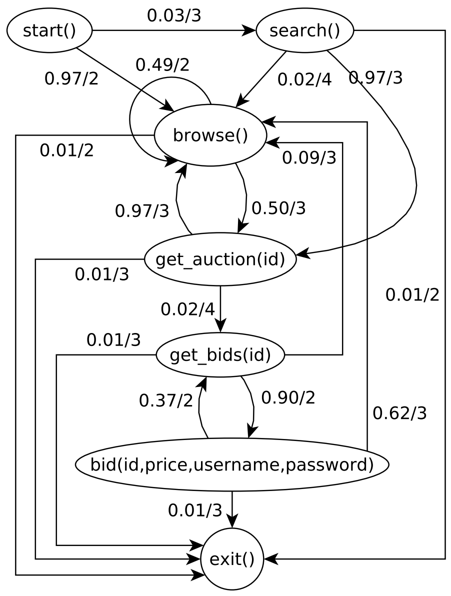


Figure 6: Memory Worst workload model

5. EXPERIMENTS

We ran two experiments to find two worst workload models for an auction web service; one model is optimized to stress the CPU and other is optimized to saturate the memory of the SUT. The web auction service has a *RESTful* [12] interface based on the HTTP protocol. The web application is implemented using Python [11] and the Django [4] framework. It allows registered users to search, browse, and bid on auctions that other users have created. We have manually created the base/initial workload model (see Figure 1) by analyzing the web application traffic.

We have benchmarked the actions in the model for the CPU and memory utilization, as listed in Table 1. The *bid* action has higher CPU and memory utilization than the other actions because it is the only action which involves data-write operation.

5.1 Test architecture

We ran our GA for 50 generations with the population size of 300. We have tried different combinations of the number of generations and population sizes. However, we noticed that there was no significant improvement in the average fitness of the population by increasing the number of generations or the population size more than 50 and 300 respectively. The crossover operator probability, mutation rate and mutation operator probability were set to 0.3, 0.01 and 0.5 respectively. We have used the same configuration for both experiments. The experiments ran for less than one minute on an Intel-i7 machine with 32 GB of memory.

5.2 Results

Figure 5 exhibits the worst workload model regarding CPU utilization. The model has achieved the maximum fitness value of 0.30 among 50 generations which is 2.5 times higher than the fitness value of the base model (i.e., 0.12). The probability distribution of the model shows that during load generation 95% of the virtual users will choose *browse* over *search* because the *browse* has higher CPU utilization. Further, the probability values of those transitions which lead from the *start* state to *bid* state have been increased significantly as compared to the base model. The reason is that the *bid* action has higher CPU utilization than the other actions (listed in Table 1). As a result, a virtual user will be able to reach to the *bid* state as quickly as possible and perform it more frequently compared to the base model.

Figure 6 illustrates the worst workload model with respect to memory utilization. The model has scored maximum fitness value of 0.19 among 50 generations, higher than the base model (i.e., 0.14). The probability distribution of the Memory worst workload model demonstrates that the probability of the *browse* action has been increased reasonably as compared to the base workload model. There are two possible reasons: *browse* is one of the actions which have relatively higher memory consumption than the other actions, secondly the *browse* state can be reached from almost every state in the model. Therefore, GA preferred those workload models which visit *browse* state more often.

5.3 Validation

In order to validate our approach, we used the MBPeT tool to generate load from the base and both worst workload models independently and monitor the resource utilization of the SUT. Figure 7 shows the comparison of resource utilization of the SUT caused by the base workload model, the CPU worst workload model and the Memory worst workload model. The average CPU utilization with the CPU worst workload model is 58% which is 2 times higher than the CPU utilization with the base workload model (i.e., 31%).

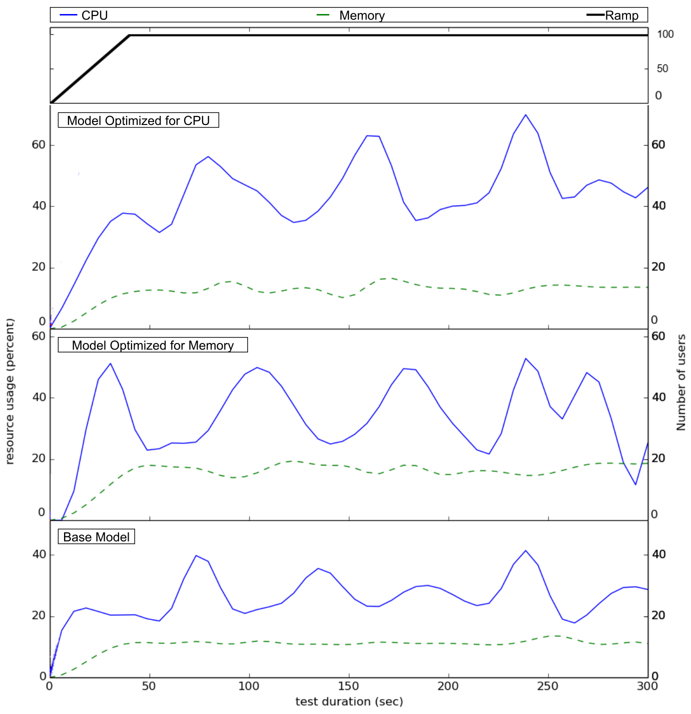


Figure 7: Resource utilization caused by the models

Similarly, the average memory utilization with the Memory worst workload model is close to 20% that reflects in an increase of 100% as compared to the memory utilization using the base workload model.

6. CONCLUSIONS

In this paper we have presented an evolutionary performance exploration approach that optimizes the probability distribution of a workload model in order to generate workload models which maximize the utilization of a given resource.

The experiments presented in the paper validates that the proposed approach can be used to discover potential vulnerabilities of the SUT by automatically exploring the user behavioral pattern space. We were able to increase the resource utilization of SUT two times by only optimizing the probability distributions of the model. The worst workload models allow developers to optimize the SUT against the worst access patterns represented by the model.

Currently, we are only optimizing the workload model for one resource at a time. For future development, we are planning to use multi-objective genetic algorithm (e.g., NSGA-II) and Swarm intelligence algorithms to optimize the workload model for more than one resource, and include network and disk utilization as well for evolution. Furthermore, we are interested to investigate how we can apply our approach on cloud-based scalable architectures.

7. REFERENCES

[1] F. Abbors, T. Ahmad, D. Truscan, and I. Porres. MBPeT: A Model-Based Performance Testing Tool. *2012 Fourth International Conference on Advances in System Testing and Validation Lifecycle*, 2012.

[2] F. Abbors, D. Truscan, and A. Tanwir. An automated approach for creating workload models from server log data. In H. Andreas, L. Therese, M. Leszek, and M. Stephen, editors, *Proceedings of the 9th International Conference on Software Engineering and Applications*, pages 14–25. Scitepress, 2014.

[3] A. Asllani and A. Lari. Using genetic algorithm for dynamic and multiple criteria web-site optimizations. *European journal of operational research*, 176(3):1767–1777, 2007.

[4] Django. Online at https://www.djangoproject.com/, September 2012.

[5] D. Draheim, J. Grundy, J. Hosking, C. Lutteroth, and G. Weber. Realistic load testing of web applications. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, pages 11–pp. IEEE, 2006.

[6] Forrester. Online Retail Sales To Top $480 Billion By 2019. https://www.forrester.com/Online+Retail+Sales+To+Top+480+Billion+By+2019/-/E-PRE7825, April 2015. Retrieved: September, 2015.

[7] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[8] C. M. Grinstead and J. L. Snell. *Introduction to probability*. American Mathematical Soc., 2012.

[9] ITU. ICT Facts and Figures - The world in 2015. Online at http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2015.pdf, September 2015.

[10] N. Patel. Great Expectations: 47% of Consumers Want a Web Page to Load in Two Seconds or Less. http://insights.wired.com/profiles/blogs/47-of-consumers-expect-a-web-page-to-load-in-2-seconds-or-less, 2009. Retrieved: September, 2015.

[11] Python. Python programming language. Online at http://www.python.org/, October 2012.

[12] L. Richardson and S. Ruby. *Restful web services*. O'Reilly, first edition, 2007.

[13] R. R. Sarukkai. Link prediction and path analysis using markov chains. *Computer Networks*, 33(1):377–386, 2000.

[14] J. Shaw. Web Application Performance Testing - a Case Study of an On-line Learning Application. *BT Technology Journal*, 18(2):79–86, 2000.

[15] M. Srinivas and L. M. Patnaik. Genetic algorithms: A survey. *Computer*, 27(6):17–26, 1994.

[16] Statista. Number of Amazon accounts worldwde 2014. http://www.statista.com/statistics/237810/, 2014. Retrieved: September, 2015.

[17] E. Weyuker and F. Vokolos. Experience with performance testing of software systems: issues, an approach, and case study. *Software Engineering, IEEE Transactions on*, 26(12):1147–1156, 2000.