# An Adaptive Bayesian Approach for URL Selection to Test Performance of Large Scale Web-Based Systems

Alim Ul Gias
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
alimulgias@gmail.com

Kazi Sakib
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
sakib@univdhaka.edu

## ABSTRACT

In case of large scale web-based systems, scripts for performance testing are updated iteratively. In each script, multiple URLs of the system are considered depending on intuitions that those URLs will expose the performance bugs. This paper proposes a Bayesian approach for including a URL to a test script based on its probability of being time intensive. As the testing goes on the scheme adaptively updates its knowledge regarding a URL. The comparison with existing methods shows that the proposed technique performs similar in guiding applications towards intensive tasks, which helps to expose performance bugs.

## Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging; H.3.4 [**Information Systems**]: Systems and Software—*Performance evaluation*

## General Terms

Algorithms, Design, Performance

## Keywords

Bayesian Learning, Performance Testing, Web-based Systems

## 1. INTRODUCTION

Software performance testing is highly essential for responsive systems in order to fix several performance issues so that services can be provided seamlessly [5]. It is important to automate the whole performance testing procedure for reducing the time required to complete the tests. However, testing large scale systems exhaustively to find all performance issues is infeasible hence the goal is to identify as many as performance bottlenecks within a limited period of time.

One approach of exposing bottlenecks could be steering the system execution towards the time intensive tasks. It is more likely that bugs will reside in the time intensive region of the application as that part usually involves huge computations. With a view of finding an appropriate steering scheme, this paper proposes a Bayesian approach for large scale web systems. The approach is adaptive which guides an application towards time intensiveness by selecting URLs based on its probability of being computationally intensive.

## 2. RELATED WORK

Static approaches have been taken for designing test scripts using schemes like combinatorial design interactions [4]. However, it is arguable that these scripts can cover all or at least a major part of the performance issues within the system. Work has also been done utilizing techniques like clustering analysis of execution profiles [2]. Although this work is used in functional testing, the scheme can also be used for performance analysis.

To the best of authors' knowledge, the most recent scheme for finding performance issues in large web-based systems is FOREPOST [3]. FOREPOST is adaptive in nature and it starts testing with $m$ out of $n$ number of URLs. After collecting and analyzing execution profiles, it generates rules saying which URLs ($g$) must be considered in the next iteration and which must not ($b$). In the next iteration, those $g$ URLs are kept and the rest $(m-g)$ positions are filled up from the $(n-b)$ URLs. If no rule is generated, $m$ URLs are again selected randomly.

Problems with FOREPOST are that more rules will eventually make the system complex and conflicting rules may arise. Moreover, if a URL is labeled as bad, it will be permanently discarded whereas it could turn out to be a good one if it were picked with another set of URLs. These issues are related to uncertainties which should be resolved to expose more accurate performance bugs.

## 3. THE BAYESIAN SCHEME

The proposed scheme uses similar execution profiles analysis approach like FOREPOST. Any element from the set $\mathcal{L} = \{good, bad, unassigned\}$ is used to label each out of $w$ profiles after completing the analysis. However, instead of producing rules the scheme assigns a probability $p_i \in \mathcal{P}$ to a URL $u_i \in \mathcal{U}$ using Bayes rule. This probability gives an intuition that how good this URL will be in guiding application's execution towards its time intensive region. Initially equal probabilities are assigned to all URLs. The URLs are selected using the roulette wheel selection algorithm similar to [1]. As the testing goes on, the probabilities are updated. A very small value $\epsilon$ is added with all these updated probabilities to avoid having zero values. This will also prevent a URL from being permanently discarded as it will still have a very small chance of getting selected from roulette wheel selection algorithm. The whole procedure is illustrated in Algorithm 1 and it can be called iteratively till a desirable time limit to steer the application's execution.

**Algorithm 1** Knowledge Update

```
Input: $w$, $\epsilon$, $\mathcal{U}$, $\mathcal{P}$, $\mathcal{L}$
Output: Updated $\mathcal{P}$
 1: $\mathcal{S} \leftarrow \emptyset$
 2: while $w$ profiles are not created do
 3:     Select $m$ URLs $\{u_1, u_2, ...., u_m\}$ using roulette wheel selection algorithm like [1]
 4:     $\mathcal{S} \cup \{u_1, u_2, ...., u_m\}$
 5:     Run test script with these $m$ URLs to create profile
 6: end while
 7: Label each profiles as any of $l_j \in \mathcal{L}$ (some may be good, some may be bad or unassigned)
 8: $\mathcal{T} \leftarrow \emptyset$
 9: $total = 0$
10: $tmpTotal = 0$
11: for each $u_i \in \mathcal{S}$ do
12:     $P(u_i|good) = \frac{\text{frequency of } u_i \text{ in good profiles}}{\text{frequency of profiles labeled as good}}$
13:     $P(good) = \frac{\text{frequency of good profiles}}{w}$
14:     for each $l_j \in \mathcal{L}$ do
15:         $P(u_i|l_j) = \frac{\text{frequency of } u_i \text{ in } l_j \text{ profiles}}{\text{frequency of profiles labeled as } l_j}$
16:     end for
17:     $P(good|u_i) = \frac{P(u_i|good)P(good)}{\sum_{l_j \in \mathcal{L}} P(u_i|l_j)P(l_j)}$
18:     $t_i = P(good|u_i) + \epsilon$
19:     $\mathcal{T} \cup \{t_i\}$
20:     $total+ = p_i$
21:     $tmpTotal+ = t_i$
22: end for
23: for each $u_i \in \mathcal{S}$ do
24:     $p_i = total \times \frac{t_i}{tmpTotal}$
25: end for
```

## 4. RESULTS

The proposed scheme is being implemented and tested on certain parts of two applications - JPetStore and MVC Music Store. As the goal is to guide the application towards time intensive region, performance of the proposed scheme is interpreted in terms of time taken for a certain number of transactions. The results are compared with two different approaches - Random Testing and FOREPOST. From Figure 1 and 2 it is seen that with the increase of transactions, better results can be obtained using Bayesian and FOREPOST scheme. Moreover, the Bayesian Scheme performs similar to FOREPOST making it an alternative for guiding application's execution towards time intensiveness.

## 5. CONCLUSION

This paper proposes a Bayesian technique for steering any web based systems towards its time intensive region. Experimental results show that the scheme provides similar results like FOREPOST. One of the key benefits of this scheme is that it will not discard any URL by labeling it as bad like FOREPOST. Instead, the proposed approach assigns a very small probability to that URL giving it a minor chance to get selected from the roulette wheel selection algorithm. This is important as this URL may turn out to be a good one in any other combination of URL selection.

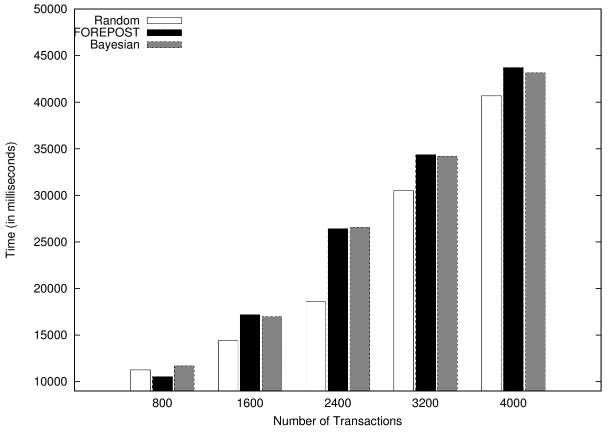

**Figure 1: Time comparison of three different approaches for the application JPetStore.**

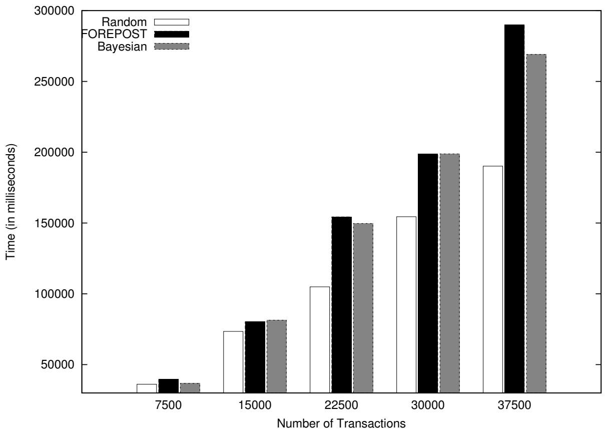

**Figure 2: Time comparison of three different approaches for the application MVC Music Store.**

## 6. REFERENCES

[1] A. Aleti and I. Meedeniya. Component deployment optimisation with bayesian learning. In *Proceedings of the 14th international ACM Sigsoft symposium on Component based Software Engineering (CBSE)*, pages 11–20. ACM, 2011.

[2] W. Dickinson, D. Leon, and A. Podgurski. Finding failures by cluster analysis of execution profiles. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE)*, pages 339–348. IEEE Computer Society, 2001.

[3] M. Grechanik, C. Fu, and Q. Xie. Automatically finding performance problems with feedback-directed learning software testing. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, pages 156–166. IEEE Computer Society, 2012.

[4] M. Grindal, J. Offutt, and S. F. Andler. Combination testing strategies: a survey. *Software Testing, Verification and Reliability*, 15(3):167–199, 2005.

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