

# Issues in Bottleneck Detection in Multi-Tier Enterprise Applications

Jason Parekh, *Member, IEEE*, Gueyoung Jung, Galen Swint, *Member, IEEE*,  
Calton Pu, *Senior Member, IEEE*, and Akhil Sahai, *Member, IEEE*

**Index Terms**—Bottleneck detection, machine learning, multi-tier enterprise systems, performance analysis

## I. INTRODUCTION

QUALITY of service (QoS) during operation is one of the key areas of systems research for large scale mission-critical applications. However, production is not the only phase during an application's life cycle during which QoS should apply; it must also be met during pre-production testing, or staging. Due to the complexity of today's enterprise class applications, system administrators monitor and analyze a massive number of application-specific metrics such as the number of active threads and the number of EJB entity bean instances, along with system metrics like CPU usage and disk I/O rate. This same complexity also demands automation of staging, and furthermore as the complexity of the applications increase, the importance of efficient analysis of testing results also increases. If staging can successfully identify metrics associated with performance limitation and subsequently correlate the metrics with performance goals, then the results can also be used in the production phase as valuable guidelines for system administrators.

The Elba project [1] addresses the automation of enterprise and tiered application staging. Automated staging in the Elba project is an iterative process whereby an application is refined and reconfigured at each iteration. Automating the process involves the creation of deployment plans, instrumentation, analysis tools, and recommendation engines from the policy level documents that govern both the staging and production policies of the application. Staging inherently demands an iterative approach to test an application adequately before placing it in a production environment. During each staging iteration, the application is subject to multiple trials of variable workload. These trials provide data

used to identify bottlenecks in the hardware/software configuration. After identifying bottlenecks, the application can be reconfigured and tested, again going through a series of trials, in the next iteration.

Machine learning classifiers constitute an important part of the metrics analysis in Elba. We have chosen machine learning techniques because they allow us to analyze many more metrics simultaneously than manual or ad-hoc approaches. As a result, the classifiers allow us to sort through these metrics to identify particular "bottleneck metrics" that indicate application mis-configuration correspondent with failed QoS. Our ongoing work demonstrated that a machine learning approach aids application tuning but avoided the questions related to the suitability of various types of classifiers to tuning multi-tier applications.

## II. BOTTLENECK DETECTION PROCESS

The first step of the bottleneck detection process is to stage the enterprise system with varying workload in order to collect metric data which is analyzed and delivered to the latter steps of the process. The workload variation allows the process to formulate the correlations between load increase and system performance, which are then used to discover the limitations causing an SLO to become violated. The SLO policy describes behavior that causes violation, such as average response time for an interaction exceeding a certain threshold, which are then translated to form a starting point for staging: an initial workload and response time levels (per interaction type) that satisfy the SLO.

The second step of the bottleneck detection process is to train a machine learning classifier with previously accumulated metric data allowing the classifier to form correlations between the performance of the system and the resulting SLO satisfaction. A machine learning classifier is trained with multiple training instances (one for each staging trial) where each instance has a set of attributes along with a prediction attribute. Each metric corresponds to an attribute where the training attribute value is calculated by taking the delta of that metric value with the previous staging trial's metric value. Finally, the trained classifier model will reveal correlations between metric trends and SLO satisfaction.

The initial set of collected metrics consists of 220 application-specific and system-level metrics. In order to reduce the amount of extraneous non-correlated metrics, we apply a correlation coefficient function introduced in [2],

Manuscript received February 28, 2006. This work was partially supported by NSF/CISE IIS and CNS divisions through grants IDM-0242397 and ITR-0219902, DARPA IPTO through grant FA8750-05-1-0253, an IBM SUR grant, and Hewlett-Packard. We would like to thank Sharad Singhal of HP Labs for his valuable insight and comments during the development of this paper.

J. Parekh, G. Jung, G. Swint, and C. Pu are with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 USA. (e-mail: jason.parekh@cc.gatech.edu, gueyoung.jung@cc.gatech.edu, galen.swint@cc.gatech.edu, calton@cc.gatech.edu).

A. Sahai is with Hewlett-Packard Laboratories, Palo Alto, CA 94304 USA. (e-mail: akhil.sahai@hp.com).

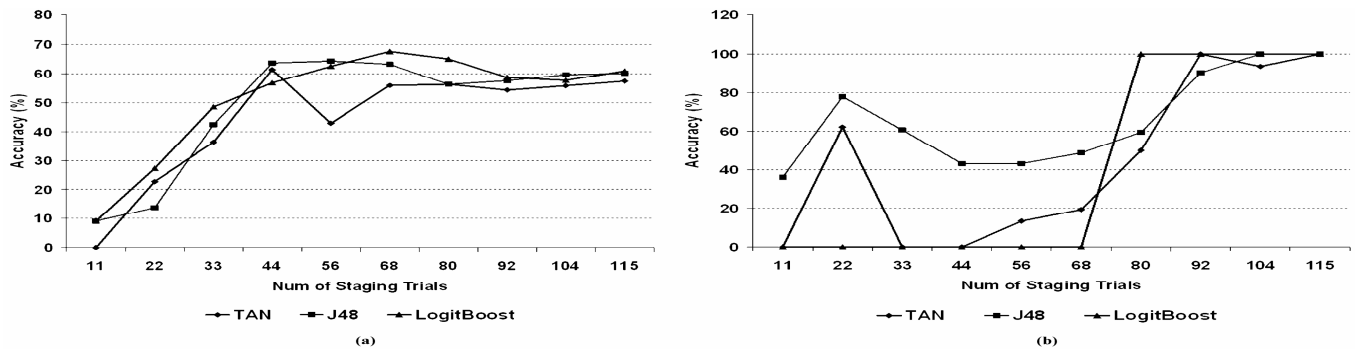


Fig. 1. The convergence speed of a classifier reveals how lengthy of a staging phase is needed to obtain accurate results from a classifier. (a) shows the prediction accuracy (based on 10-fold cross-validation). (b) shows the final bottleneck identification accuracy.

comparing metrics to the overall response time. From this function, we apply a threshold to reduce our working set of metrics for training.

The final step of the bottleneck detection process is to query a trained classifier using an approach that discovers candidate bottlenecks. The process queries the classifier by creating a test set consisting of test instances that are similar to training instances, with the only difference being that the prediction attribute is not included in the test instance since this is the information the classifier predicts using its model.

### III. EVALUATION RESULTS

We evaluated our bottleneck detection process using RUBiS on a three-tier system: Apache as the web tier, JOnAS/Tomcat as the application server tier, and MySQL as the database tier. The main classifiers studied from the WEKA toolkit are: Tree-augmented naïve (TAN) Bayesian network, LogitBoost, and C4.5 decision tree (the J48 implementation).

Our experiments show the CPU usage of application server radically increases and saturates at 100% when the SLO satisfaction goes down to around 85%. In contrast, while the SLO satisfaction starts to decrease, the trends of other tiers' CPU usages are almost flat. Also, the memory usages of both the application server and database server are under utilized. The memory usage of HTTP server is somewhat high, but its trend is almost flat. The other metrics are below thresholds.

In our comparative study of machine learning classifiers, we define the accuracy of each classifier in two ways: the cross-validation accuracy and the bottleneck identification accuracy. The former accuracy determines the accuracy in a pure machine learning sense, oblivious to the actual application of the classifier, by testing the trained classifier's generated model on how well it can predict unseen instances. The latter accuracy determines the accuracy as applied to our bottleneck detection process.

The convergence speed of a classifier defines how many trials of staging (with each trial varying in workload and thus training set size) is needed to obtain strong accuracy. This speed becomes relevant as we measure the efficiency of each classifier, which is the minimal amount of staging required for the classifier to identify bottlenecks in the system.

Figure 1 (a) displays the convergence speed for the prediction accuracy. Each of the classifiers in the graph show similar behavior stabilizing at 44 staging trials, although once stabilized LogitBoost has a slightly higher prediction

accuracy. While these accuracies could be stronger, our experiments show that the classifiers' bottleneck identification accuracies are high. Figure 1 (b) displays the convergence speed for the bottleneck detection accuracy. The period from 10 to 40 staging trials shows erratic behavior, which can be explained by the under developed classifiers (shown in the Fig. 1 (a)). The period from 40 staging trials onwards shows positive results as each classifier increases toward 100% bottleneck identification accuracy. In terms of overall reliability, the J48 classifier seemed to provide strong results due to its steady increase toward 100% bottleneck detection accuracy and its higher accuracy throughout a majority of the variations in number of staging trials.

### IV. CONCLUSION

In this work, we explore the performance of various machine learning classifiers with regard to bottleneck detection in enterprise, multi-tier applications governed by service level objectives. This builds on our previous work which used a J48 decision tree to assist tuning the TPC-W application. Specifically, in this paper, we demonstrate the effectiveness of three classifiers, a tree-augmented Naïve Bayesian network, a J48 decision tree, and LogitBoost, using our bottleneck detection process, which delves into a new area of performance analysis based on the trends of metrics (first order derivative) rather than the metric value itself. Furthermore, we illustrate the efficiency of each classifier by measuring the convergence speed, or the number of staging trials required in order to provide positive results. Using RUBiS, we test our bottleneck detection process on a set of 220 combined system-level (CPU, memory, etc.) and application-level metrics (open database connections, EJB pool size). Finally, we show the effectiveness of the classifiers used in our bottleneck detection process as each classifier strongly identifies the enterprise system bottleneck.

### REFERENCES

- [1] Elba project. <http://www-static.cc.gatech.edu/systems/projects/Elba>.
- [2] M. Raghavachari, D. Reimer, and R. D. Johnson, "The deployer's problems: configuring application servers for performance and reliability", in *Proc. 25<sup>th</sup> International Conference on Software Engineering (ICSE)*, Portland, OR, USA, May 2003.