An Statistical Approach for Estimating CPU Consumption in Shared Java Middleware Server

Wei Wang
State Key Laboratory of Software Engineering, Wuhan University,
Wuhan 430072, P.R. China
Technology Center of Software Engineering, Chinese Academy of Sciences
Beijing 100190, P.R. China
wangwei@otcaix.iscas.ac.cn

Xiang Huang, Yunkui Song, Wenbo Zhang, Jun Wei, Hua Zhong, Tao Huang
Technology Center of Software Engineering, Chinese Academy of Sciences
Beijing 100190, P.R.
{huangxiang, songyunkui06, zhangwenbo, wj, zhongh, tao}@otcaix.iscas.ac.cn

Abstract—**Middleware sharing is one of the important resource sharing approaches which enables sharing of costs across a large pool of users. However, the shared Java middleware server easily causes interference on performance between concurrent user requests. A key requirement to an effective performance isolation is the knowledge of the resource consumption of the various kinds of use requests classified according to different application context information. Direct measurement of resource consumption requires instrumentation which is impractical. In this paper, we demonstrate that CPU consumptions of various kinds of user requests on a given hardware can be approximated by a proposed Kalman filter based approach. Experimental results derived from testing the approach by using the TPC-W e-commerce suite deployed on a widely-used Java middleware server (Tomcat) illustrate the potential of this approach.**

Keywords-Performance Isolation; CPU Consumption; Kalman Filter

I. INTRODUCTION

Middleware sharing is one of the important resource sharing approaches in SaaS system [1][2]. However, the shared middleware approach easily causes interference on performance between concurrent user requests. This affects infrastructure resources as well as applications and services that are hosted on shared resources but need to be made available in multiple isolated instances. This is called performance isolation requirement. A key requirement to an effective performance isolation is the knowledge of the resource consumption of the various kinds of use requests classified according to different application context information, such as session classes or access roles.

Java middleware server is one of the most commonly used middleware servers. However, the Java virtual machine (JVM) lack resource accounting mechanism, which could be used to measure resource consumption. Unlike other resources, CPU consumption is probably the most challenging resource type to measure in Java middleware server. Prevailing approaches rely on native code libraries, or program transformations. The native code based approach measures the CPU consumption by native agent which probes CPU for calculation of cycles by sampling [3][4]. Most of the web-based system is transactional and the CPU consumption of each request execution is very tiny. Therefore, we need a high sampling rate provide accurate results. Moreover, this requires platform-specific features (such as access to special timers) and hence hamper portability. The program transformation based approach measures the CPU consumption by tracking the number of executed JVM bytecode instructions and then transforms them to CPU consumption [5]. This approach is independent of any particular JVM or underlying operating system. However, these transformations incur more than 30% performance overhead at runtime [6]. Another limitation of this approach is that it can not measure resource consumption for native code execution.

In this paper, we show that CPU consumptions of various kinds of user requests on a given hardware can be approximated by a Kalman filter [7] based estimation. This estimation solution is simplicity, portable and nonoverhead. Experimental results derived from testing the proposed approach by using the TPC-W e-commerce suite [8] deployed on a widely-used Java middleware server (Tomcat) illustrate the potential of the approach.

The rest of paper is organized as follows. Section II presents our Kalman filter based approach. Then, Section III shows several the experiment cases for validation. Related work is given in Section IV. Conclusions and future work are outlined in Section V.

II. Statistical Approach for CPU Consumption Estimation

A. Formalizing the Problem

The problem is to design a statistical estimation approach to approximate the CPU consumptions according to different application context information based on the observer data. In the present work, we use session class as an example in the rest of this paper. The session class is one kind of application context information to classify user requests from different organizations. Prerequisite to applying estimation is that a statistical analysis engine periodically collects Java middleware server access log that reflects all throughputs according to different session classes and the total CPU utilization of Java middleware server.

We observe transactions of each session class over monitoring windows of fixed length *T*. The total CPU utilization of Java middleware server utilization is aggregated at the end of each monitoring window. Assuming that there are totally *N* session classes serviced by the server, we use the following notation:

• *T* is the length of the monitoring window;

• N_i is the number of completed transactions of the *i*-th session class, where $1 \le i \le N$;

• U_{CPU} is the average CPU utilization of Java middleware server during this monitoring window;

• S_i is the average service time (average CPU consumption per transaction) of transactions of the *i*-th session class, where $1 \le i \le N$.

From the utilization law [11] in the classic queueing theory, one can easily obtain (1.a) for each monitoring window.

$$U_{\mathrm{CPU}}T = \frac{\sum_i N_i S_i}{m} \qquad (1.a)$$

where *m* is the number of CPU.

Because it is practically infeasible to get accurate service time S_i, let C_i denote the approximated S_i for $1 \le i \le N$. Then, an approximated utilization U'_{CPU} can be calculated as

$$U'_{\mathrm{CPU}} = \frac{\sum_i N_i C_i}{mT} \qquad (1.b)$$

To solve for C_i, one can choose a statistical analysis method from a variety of known methods in the literature. A typical objective for a statistical analysis method is to minimize the error between $U_{\mathrm{CPU},k}$ and $U'_{\mathrm{CPU},k}$, where k is the index of the monitoring window over time. In this paper, we will be interested in the relative error between the estimated service time C_i and the measured service time S_i.

A linear regression-based method can be chosen to solve for S_i. However, the accuracy of the regression results critically depends on the quality of monitoring data used in the regression analysis. If the collected data exhibits a changing performance characteristic over time, i.e., the average service time S_i under different transaction mixes are very different [9], then this can significantly impact the derived service time and can lead to an inaccurate approximation result [10]. To avoid inaccurate estimation, the statistical approach should be able to track the changing parameter S_i.

B. The Kalman Filter Estimator

Kalman filter [7] has been widely used in the area of autonomous or assisted navigation [12][13]. One of the main advantages of the filter is that it can estimate hidden parameters indirectly from observed data and can integrate data from as many measurements as are available, in an approximately optimal way.

The Kalman filter addresses the general problem of estimating the state *X* of a discrete-time controlled process which is governed by the linear stochastic difference equation (with discrete monitoring window indexed by *k*):

$$X_k = AX_{k-1} + Bu_{k-1} + w_{k-1} \qquad (2.a)$$

with a measurement *Z* that is:

$$Z_k = H_k X_k + v_k \qquad (2.b)$$

in which *A* is a transform matrix from monitoring window *k-1* to *k*, u_{k-1} represents a known vector, *B* is a control matrix, w_{k-1} represents the process noise whose covariance is Q_{k-1}. H_k is a matrix about the relation of Z_k and X_k, v_k represents the measurement noise whose covariance is R_k.

C. Adapting the Kalman Filter to Estimate the CPU Consumptions of Multiple Session Classes

For CPU consumptions of different session classes, we model *X* as a vector of *N* classes' average service time that drift randomly. Thus:

$$X_k = X_{k-1} + w_{k-1} \qquad (3.a)$$

in which $X_k = [C_1^k, C_2^k, \dots C_N^k]$ represents the average service time of each session class within monitoring window *k*. Also, the measured total CPU consumption is modeled by the result of the equation (1.b) calculation, with added errors of estimation:

$$Z_k = \frac{\sum_i N_i^k C_i^k}{mT} + v_k \qquad (3.b)$$

in which Z_k represents the measured total CPU utilization, N_i^k is the measured throughput of session class i, and H_k is defined as $\left[\frac{N_1^k}{mT}, \frac{N_2^k}{mT}, \dots \frac{N_n^k}{mT}\right]$.

The estimated CPU consumptions are computed recursively, beginning from an initial estimate vector $\hat{X}_0$, and an initial error covariance matrix P_0. Each recursive step can be summarized as follows:

1) project the state ahead with $w_{k-1} = 0$:

$$\hat{X}_k^- = \hat{X}_{k-1} \qquad (4.a)$$

2) project the error covariance P_k^- ahead:

$$P_k^- = P_{k-1} + Q_k \qquad (4.b)$$

3) compute the Kalman gain K:

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \quad (4.c)$$

4) update the state of *X*:

$$\hat{X}_k = \hat{X}_k^- + X_k (Z_k - H_k \hat{X}_k^-) \quad (4.d)$$

5) update the error covariance *P*:

$$P_k = (I - K_k H_k) P_k^- \quad (4.e)$$

Step (4.d) is a key step for the estimation, in which the equation can be simplified as

$$X_{new} = X_{old} + Ke \quad (4.f)$$

which means the old service time would be updated with the latest errors.

The time complexity of a recursion in our approach is $O(N^3)$ where N is the number of session class. In detail, the complexity of equation (4.c) is just with the complexity of seeking a reciprocal instead of inversing of a matrix. Therefore, the most complex calculation left is in equation (4.e), which is a matrix multiplication. Thus, the total complexity of our approach is $O(N^3)$.

D. *The Influence of the monitoring window length*

The measurements, including N_i^k, which is the measured throughput of session class *i*, and Z_k, which is the total CPU utilization of Java middleware server, are most easily measurable in the system. They are averages over the estimator's monitoring window of length *T*. Therefore, a longer monitoring window gives more accurate measurements, but allows greater drift and more delay before the next update. This will reduce the adaptability of the approach to service time variation. A shorter *T* tracks changes more closely at the cost of inaccurate measurement. We define the mean length of change period by T_{change}. It is expected that large value of T_{change}, such as $T_{\text{change}} \gg T$, will be necessary for good tracking and accuracy. In Section III-A, we explore the impact of the monitoring window length on the accuracy of the filter solution under a continuously changing workload mix.

III. EVALUATION

A. *Experimental environment*

We built a test-bed of an e-commerce application that simulates the operation of an on-line bookstore, according to the classic TPC-W benchmark [8]. Figure 1 depicts the experimental environment. We also extended the load generator to simulate online behaviors of multiple session classes [14]. The system under test is a two-tier architecture paradigm including a Java middleware server (Tomcat [15]) and a database server run on two different servers. The statistical analysis engine gathers metrics from the Java middleware server over a sequence of execution intervals. Specifics of the software/hardware used are given in TABLE I.

We simulate the case that the TPC-W on-line bookstore application serves 3 session classes, identified as "Class 1", "Class 2" and "Class 3". The experiment explores the influence of monitoring window size, workload changes and native code execution on the accuracy of this approach.

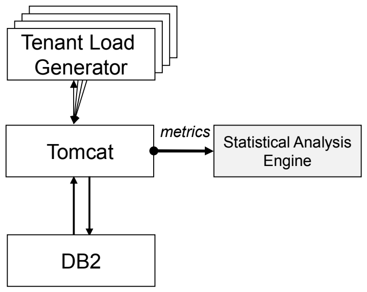

Figure 1. E-commerce experimental environment with multiple session classes

TABLE I. SOFTWARE/HARDWARE COMPONENTS

Server	Processor	RAM	Number
Session classes (Emulated-Browsers)	Pentium IV/1.8GHz	2GB	3
Java Middlerware Server-Tomcat6.0	Pentium IV/2.8GHz	2GB	1
Database Server-DB2-V9.5	Pentium IV/3.9GHz	2GB	1
Statistical Analysis Engine	Pentium IV/1.8GHz	2GB	1

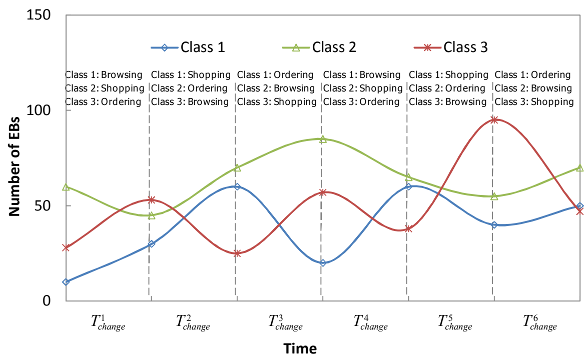

Figure 2. Changing workloads

For every 30 seconds during experiments, we validate the accuracy in the relative error of CPU consumption estimation:

$$e_i = \frac{|C_i - S_i|}{S_i},$$

where C_i is the latest estimated service time, and S_i represents the latest service time measured by the approach used in [16].

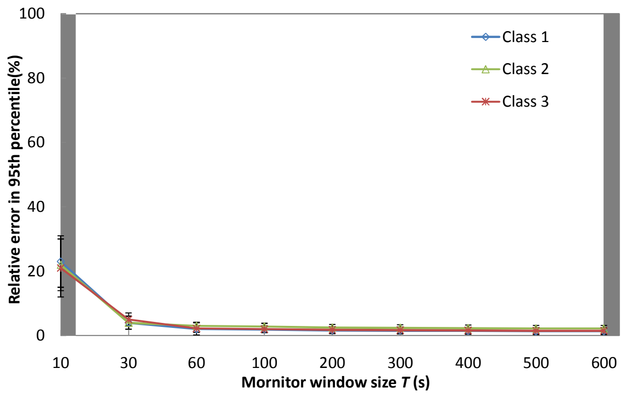

Figure 3. Relative errors in 95th percentile under persistent workload mix and different T

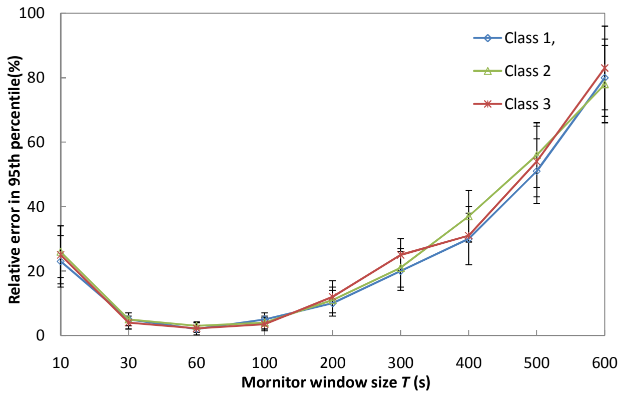

Figure 4. Relative errors in 95th percentile under changing workload mix and different T

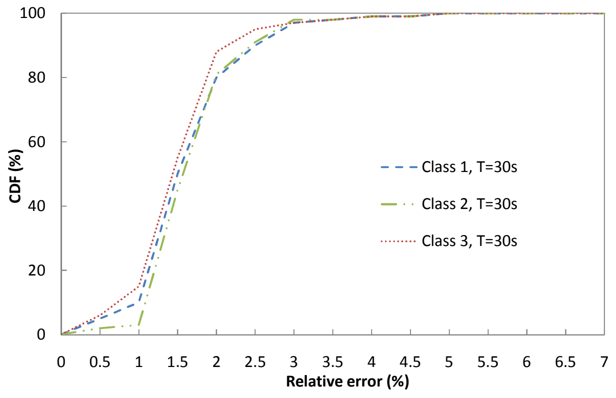

Figure 5. CDF of relative error of estimation results (native code execution injected)

Figure 2 shows changing workloads of three session classes (the lines labeled “Class 1”, “Class 2”, “Class 3”). X axis is time, Y axis is concurrent emulated browsers (EBs). As shown in Figure 2, the workload size changes over time. We can run experiments under changing workload mix or persistent workload mix. The former means the workload mix of each session class changes over changing interval of fixed length T_{change}(as shown in Figure 2), whereas the later means the workload mix of each session class will start with the same mix and not change every T_{change}.

B. *Sensitivity of estimation to T*

We examine the sensitivity of estimation results to T, using a short workload changing interval $T_{\text{change}} = 5\ mins$. For comparison, we also run experiments under persistent workload mix (shopping mix).

Figure 3 and 4 show the relative errors in 95th percentile under persistent shopping mix and changing workload mix respectively, using different monitoring window sizes: $T = 10s, 30s, 60s, 100s, 200s, 300s, 400s, 500s, 600s$. Comparing these results, we can summarize the observations as follows:

- A longer monitoring window achieves higher accuracy under persistent workload mix, as show in Figure 3, however, may reduce the adaptability of the filter to workload mix variation, i.e., as shown in Figure 4, when $T > 200s$, the relative errors continue to increase under changing workload mix condition.
- A shorter T tracks changes more closely and still deliver acceptable estimation accuracy, i.e., as shown in Figure 4, when $T < 100s$, the relative errors are less than 15%. However, when the T is too short to reach accurate measurements, the relative errors increase, i.e., when $T = 10s$, the relative errors exceed 20%.

The above observations imply that, even under continuously changing workload mix condition, our approach tracks changes and accurately approximate CPU consumption of each session class, especially with appropriately tuned monitoring window size taken into account.

C. *Estimation of native code execution*

In this subsection, we modified the TPC-W application by injecting some CPU sensitive native code execution into transactions of two session classes and explore the impact of the native code execution on the accuracy of estimation. Figure 5 shows the Cumulative Distribution Function (CDF) of the relative errors of estimation results using a 30s monitoring window size. As shown in Figure 5, more than 95% of relative errors of the CPU cost approximation of all three session classes are less than 4%. Experiment results indicate that the CPU consumption of native code execution, which is counted as part of the CPU consumption of Java middleware server process, is well estimated by the filter.

IV. Related Work

Unlike other resources, CPU consumption is probably the most challenging resource type to measure in Java middleware server. Prevailing approaches to measure CPU consumption in Java middleware server mainly rely on native code libraries, or program transformations.

Native code based approach are widely used in previous works for CPU consumption measurement [3][4][17][18]. For example, Jordan et al. [4] uses native code to obtain application's CPU consumption, according to which, their approach isolates applications inside a JVM. The native code based approach measures the CPU consumption by native agent which probes CPU for calculation of cycles by sampling. However, most of the web-based system is transactional and CPU consumption of each request execution is very tiny. Therefore, we need a high precision timestamps to provide accurate results. For example, Magpie [18] uses Event Tracing for Windows [16] to measure CPU usage with the processor cycle counter in Windows operating system. Similar approaches on other operating systems include the Linux Trace Toolkit [19] and Solaris DTrace [20]. This kind of approach requires platform-specific features, such as access to special timers, and hence hampers portability. In contrast, our approach is nonintrusive and hence independent of operating system.

Binder et al. [5] proposed a portable CPU-management framework for Java. Their proposed approach measures the CPU consumption by tracking the number of executed JVM bytecode instructions and then transforms them to CPU consumption. However, this approach incurs more than 30% performance overhead at runtime [6]. In contrast, our approach is based on monitoring data, such as throughput and Java middleware server process CPU utilization, that are typically available in enterprise production environments, and therefore does not incur any performance overhead. Another limitation of the transformation based approach is that it can not measure CPU consumption for native code execution. Our experiments show that the CPU consumption of native code (such as DLL), which is counted as part of the CPU consumption of Java middleware server process, is well estimated by Kalman filter.

In [21][22] the authors use multiple linear regression techniques to derive the average service time of applications. However, the regression-based method depends on groups of samples collected from a long period. If the collected data exhibits a changing performance characteristic over time, this can significantly impact the derived service time and lead to inaccurate approximation results. Previous studies [10][22] have shown that the accuracy of regression techniques suffer if the service time for a system are not deterministic.

Woodside et al. [23] proposed a mechanism to update the multiple performance models' parameter based on extended Kalman filter. Their approach investigates the use of queuing models to deduce resource consumptions. It relies on measured response times from a system and a performance model for the system. Resource consumptions are estimated such that the model's mean response time prediction closely matches the mean of the measured response times. The cost of their approach is the product of the number of state X and the total time for calculation. And the cost for calculation is directly related to the complexity of the performance model, which usually takes more time [24]. In this paper we proposed liner Kalman filter approach, the time complexity is $O(N^3)$ where N is the number of state X in one server. It means this method can calculate the resource consumptions that only takes several milliseconds, and hence can be efficiently used as a part of on-line resource evaluation method and more suitable for further performance isolation control.

V. Conclusion and future Work

The shared middleware easily causes interference on performance between concurrent user requests. A key requirement to an effective performance isolation is the knowledge of the resource consumption of the various kinds of use requests classified according to different application context information. Direct measurement of resource consumption in the shared Java middleware server requires instrumentation which is impractical. In this paper, we establish the feasibility of CPU consumption estimation in the shared Java middleware server using a Kalman filter based approach. We illustrate the effectiveness of the proposed approach by using the TPC-W e-commerce suite deployed on a widely-used Java middleware server (Tomcat). Experiments were performed under different settings of changing workloads to evaluate the effectiveness of the approach and the requirements for the filter settings. Our approach is based on monitoring data that are typically available in enterprise production environments, and therefore does not incur any performance overhead. Furthermore, it is nonintrusive and independent of operating system.

One focus of our continuing work is to validate this approach under more complex application workload situations, such as frequent application updates and batch-style applications. We are also working to "close the loop" for automated performance isolation. In this paper, we concentrate on the CPU consumption estimation. We believe that this approach can be efficiently applied for estimating other shared system resources. Thus, this approach can be used for performance anomaly detection, performance isolation and resource provision in the shared Java middleware server environment.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant No.61003029; the National Grand Fundamental Research 973 Program of China under Grant No. 2009CB320704; the National Science and Technology Major Project of China under Grant No. 2009ZX01043-002-003, 2010ZX01045-001-010-4, and the State Key Laboratory of Software Engineering(SKLSE).

REFERENCES

[1] Zhang, Q., Cheng, L. & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications, Vol. 1, No. 1, pp. 7—18.

[2] Develop and Deploy Multi-Tenant Web-delivered Solutions using IBM middleware: Part 2: Approaches for enabling multi-tenancy: URL http://www.ibm.com/developerworks/webservices/library/ws-multitenantpart2/index.html

[3] Sun Microsystems, Inc. JVM Tool Interface (JVMTI): URL http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/

[4] Mick Jordan , Grzegorz Czajkowski , Kirill Kouklinski , Glenn Skinner, Extending a J2EE™ server with dynamic and flexible resource management, Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware, October 18-22, 2004, Toronto, Canada.

[5] W. Binder and J. Hulaas. A portable CPU-management framework for Java. IEEE Internet Computing, 8(5):74–83, Sep./Oct. 2004.

[6] Hulaas, J., Kalas, D.: Monitoring of Resource Consumption in Java-based Application Servers. In: Proceedings of the 10th HP OpenView University Association Plenary Worksop (HPOVUA 2003), Geneva, Swizerland (2003).

R.E.Kalman, A New Approach to Linear Filtering and Prediction Problems, Transactions of the ASME-Journal of Basic Engineering, 1960.

[7] TPC-W Benchmark: URL http://www.tpc.org

[8] Daniel Menascé, Virgílio Almeida, Rudolf Riedi, Flávia Ribeiro, Rodrigo Fonseca, Wagner Meira, Jr., In search of invariants for e-business workloads, Proceedings of the 2nd ACM conference on Electronic commerce, p.56-65, October 17-20, 2000, Minneapolis, Minnesota, United States.

[9] L. Cherkasova, Kivanc Ozonat, Automated Anomaly Detection and Performance Modeling of Enterprise Applications, ACM Transactions on Computer Systems, Vol. 27, No. 3, November 2009.

[10] Lazowska ED, Zahorjan J, Graham GS, Sevcik KC. Quan-titative System Performance: Computer System Analysis Using Queueing Network Models. Upper Saddle River: Prentice-Hall, Inc., 1984.

[11] Catlin, Donald E.Estimation, control, and the discrete Kalman filter. New York : Springer-Verlag, c1989.

[12] Evensen, Geir. Data assimilation: the ensemble Kalman filter. Berlin; New York : Springer, c2007.

[13] Apache Tomcat: URL http://tomcat.apache.org

[14] Bench4Q: URL http:// forge.ow2.org/projects/jaspte

[15] Microsoft. Event tracing for windows: URL http://www.microsoft.com/whdc/devtools/tools/EventTracing.mspx

[16] G. Czajkowski and L. Dayn`es. Multitasking without compromise: A virtual machine evolution. In ACM Con-ference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'01), pages 125–138, Tampa Bay, Florida, Oct. 2001.

[17] P.Barham, A.Donnelly, R.Isaacs, R.Mortier. Using Magpie for request extraction and workload modelling December 2004 6th Symposium on Operating Systems Design and Implementation (OSDI'04),2004.

[18] K. Yaghmour and M. R. Dagenais. Measuring and charac-terizing system behavior using kernel-level event logging. In Proc. USENIX Annual Technical Conference, June 2000.

[19] B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal. Dynamic instrumentation of production systems. In Proc. USENIX Annual Technical Conference, pages 15–28, June 2004.

[20] Q. Zhang, L. Cherkasova, G. Mathews, W. Greene, and E. Smirni. R-capriccio: A capacity planning and anomaly de-tection tool for enterprise services with live workloads. In Middleware, 2007.

[21] J. Rolia, V. Vetland. Correlating Resource Demand Information with ARM Data for Application Services. In Proc. of the ACM Workshop on Software and Performance, 1998.

[22] M. Woodside, T. Zheng, and M. Litoiu, "The Use of Optimal Filters to Track Parameters of Performance Models," Proc. Second Int'l Conf. Quantitative Evaluation of Systems, Sept. 2005.

[23] Woodside, M., Franks, G., Petriu, D.C.: The future of software performance engineering. In: FOSE 2007: 2007 Future of Software Engineering, Washington, DC, USA, pp. 171–187. IEEE Computer Society, Los Alamitos (2007)