Deriving Parameters for Open and Closed QN Models of Operational Systems Through Black Box Optimization

Mahmoud Awad
Computer Science Department
George Mason University
Fairfax, VA, 22030, USA
mawad1@gmu.edu

Daniel A. Menascé
Computer Science Department
George Mason University
Fairfax, VA, 22030, USA
menasce@gmu.edu

ABSTRACT

Black-box modeling techniques are used when modeling computer systems with unknown internal structure or behavior and/or when it is not feasible or too time consuming to monitor a running computer system. The main challenge in these situations lies in estimating values for the parameters of these models, especially the values of service demands at the various devices for each transaction class. These estimates have to be compliant with the input-output relationships observed through measurements. This means that solving a model of the system with the estimated parameters should yield the same outputs (e.g., response times) for the same inputs (e.g., arrival rates or concurrency level). This paper presents a method for automatically estimating service demands for open, closed, single and multiclass queuing networks (QN). The method is based on casting the estimation problem as a non-linear optimization problem. However, because the solution of closed QNs does not have a closed form, we need to resort to black-box optimization techniques. The parameter estimation method presented here is part of *iModel*, a framework for automatically deriving performance models of systems whose detailed characteristics (structure and behavior) are unknown. Other portions of the framework were discussed in detail in previous publications by the authors. This paper illustrates the ideas through several numerical examples and then applies them to a multi-tiered operational system running OFBiz. The estimated service demands closely satisfy the input-output relationships at various workload intensity levels and can be used for prediction purposes.

CCS Concepts

•**Mathematics of computing** → **Queueing theory;** Cluster analysis; Nonlinear equations; •**Applied computing** → *Operations research;*

ICPE '17, April 22–26, 2017, L'Aquila, Italy.

DOI: http://dx.doi.org/10.1145/3030207.3030208

Keywords

queuing network models; parameter estimation; non-linear optimization; black-box optimization; MVA; AMVA

1. INTRODUCTION

In our previous work, we described a framework called *iModel* for the dynamic derivation of analytical performance models in autonomic computing environments [1, 2, 3]. This work is very important in autonomic computing systems that rely on performance models to dynamically adjust their configurations and system resources to respond to changing workload demands. Our framework infers a system model (queues and interconnections) and a workload model, which are combined to produce an analytical performance model.

The system model is identified using reverse engineering techniques to determine the architectural pattern that best matches the system. The workload model is identified by collecting and analyzing the system and application logs. These logs represent the system's observable behavior, such as arrival rates, system's concurrency level, and response times. Depending on how much can be harvested from the system and application logs, *iModel* determines the best overall analytical model to select (black-box, white-box or the right level of gray in between).

Understanding and analyzing user behavior and identifying user-system interaction patterns is an active research problem in workload modeling. However, the majority of the literature in this area focuses on workload prediction and proactive resource allocation, which are also natural outcomes of our research in automated system model derivation. In [11], the authors presented an on-line workload classification mechanism for optimized forecasting method selection. Based on the level of overhead (noise level) presented in the various time-series-based workload classification methods, the proposed mechanism dynamically selects the appropriate method using a decision tree. The decision tree takes into account user specified forecasting objectives. The authors in [10] proposed a trace-based approach for capacity management. Workload pattern analysis uses a pattern recognition step and quality and classification step, which measure the difference between the current workload and the pattern, computed as the average absolute error. The authors also presented a process for generating a synthetic trace to represent future workload demands. In [14], the authors presented an approach for finding and characterizing workload patterns from data traces and predicting future workload patterns across virtual machines.

In our framework, user-system interaction patterns are

used to produce Customer Behavior Model Graphs (CBMG) [19] and Client-Server Interaction Diagrams (CSID) [19], which are necessary in determining the characteristics of the workload model as well as the system model. These interaction patterns are also necessary to determine whether the system should be modeled as a closed or open QN.

A key aspect of automatically deriving a performance model is the determination of its parameters, which are divided into workload intensity parameters and service demands for each transaction class at each queue. The process of automatically characterizing the workload in *iModel* has been addressed by the authors in a previous publication. Automatically deriving the values of service demands in a non-intrusive way is the goal of this paper. A first attempt at that was presented by the authors in [2]. That approach treated the computer system as a black box and estimated the service demands by solving a non-linear optimization problem in which the service demands are the decision variables.

The approach in [2] dealt with open Queuing Network (QN) only and relied on the fact that open QNs have a closed form expression for their solution. The method described in [2] had another drawback, namely, it required regular calibrations because it relied on a single measured input-output relationship per class. This paper extends that approach in two important ways: (1) service demands for both open and closed QNs are estimated using a black box optimization approach [8], i.e., a method in which the objective function and maybe some of the constraints do not have a closed form expression but just an algorithmic way (e.g., Mean Value Analysis or Approximate Mean Value Analysis) of computing the value of the function and/or constraint. (2) the need for calibration is vastly reduced because the method relies on a set of measured input-output points per class and not on a single one as in [2].

The rest of the paper is organized as follows. Section 2 presents some background on system modeling and our proposed approach. Section 3 provides a high-level description of *iModel* to provide the context of the contribution of this paper. Section 4 describes our service demand estimation method in detail. The closed QN method is described in sub-section 4.1 and the open QN method in subsection 4.2. Section 5 describes the implementation details of the testbed used for validation and shows the results of our experiments using OFBiz, JMeter, and the ELK stack. Section 6 discusses some related work. Finally, Section 7 includes concluding remarks and discusses some of our future plans related to the automatic derivation of system performance models.

2. BACKGROUND

Computer system modeling requires a certain level of understanding of the application and system architecture and behavior of the system being modeled. Just as important is the understanding of how the users interact with the system in terms of workload and patterns of use. When all such details are available to performance engineers, it becomes possible to employ white-box modeling techniques in which detailed and precise models are developed, parameterized, and tested.

When modeling existing operational systems, such information may not be available or easy to gather, possibly because of the system complexity or the inability to conduct detailed analysis of the software architecture and behavior. In such cases, performance engineers resort to black-box or gray-box modeling depending on how much can be gleaned from system documentation, system log traces or from interviews with software developers and system architects.

Black-box modeling is used when none of the internal system details are known (or they are intentionally abstracted) and focuses on the interfaces between system components. Gray-box modeling can be used when white-box modeling is costly or unnecessary while the black-box model is inaccurate or fails to adequately represent the real system.

When modeling an operational system, where few internal details are available, one might resort to reverse engineering techniques, where source code is recovered and analyzed to determine the application behavior and its impact on the hardware resources. Another approach is to analyze system logs and audit traces in order to understand the application's architecture and behavior. Reverse engineering the source code is intrusive and assumes having proper access to the application executables and permission to reverse engineer them. However, system log and audit trace analysis -although less accurate- is less intrusive while providing more insight into the system internals than black-box modeling.

The problem we address in this paper is the ability to derive analytic models of an operational system given only experimental (testing environment runs) or system logs of the system's production environment.

3. THE IMODEL FRAMEWORK

Our framework, *iModel*, aims at automating the process of building an analytic model of an operational system. The automation engine of *iModel* parses and analyzes the system's configuration and log files in order to arrive at the best fit workload model and system model. *iModel* includes a knowledge base of well-known single-queue and queuing network (QN) analytic models with their mathematical formulas or algorithmic solution and parameters. The automation engine determines the best optimization technique and tool to solve the analytic model depending on the system's architecture and observed behavior inferred from the log files. The three main components of *iModel* are:

1. Workload Model Analyzer, which analyzes the user behavior recorded in the various system and application logs in order to develop Customer Behavior Model Graphs (CBMG) [19] and Client-Server Interaction Diagrams (CSID) [19]. CBMGs are used to model the interactions between the user and the system and CSIDs are used to model the interaction between system components, and, as a result, derive the system workload.

2. Performance Model Analyzer, which is the focus of this paper, determines the possible system architecture and application components and their interconnections (communications) by analyzing configuration and log files. Details of this step can be found in [1].

3. Performance Model Repository, which is used to find the best-fit analytic model that matches the system physical characteristics and user behavior. The model is solved using the Performance Model Analyzer's execution engine using solvers that are parameterized using the workload model and the model definition in

the repository (metrics and input parameters and their data types and constraint definitions).

Figure 1 shows the workflow of *iModel*, explained in detail in [1]. The focus of this paper is on steps 7 and 8, in which the model execution engine parameterizes the model using the method described in this paper and solves it.

4. AUTOMATIC DERIVATION OF SERVICE DEMANDS FOR QNS

The problem we address in this paper is that of deriving service demands for a QN model of an *operational* system from measured response times and workload intensity values. We show in this section how that can be done when the goal is to model the operational system as a closed QN and as an open QN. In the case of a closed QN model, the workload intensity is given by the customer population (i.e., average number of customers in the system) and in the case of open QNs by average arrival rates. It is important to emphasize that the fundamental assumption throughout our discussion is that the systems we are interested are in production and cannot be stopped for experimentation purposes nor can they be subject to intrusive measurement techniques. We only rely on data commonly available by system logs generated by such systems.

In general, we can say that a QN model computes the average response R_r for class r transactions through a function f_r described below

$$R_r = f_r(\mathbf{D}, \vec{W}) \tag{1}$$

where $\mathbf{D}$, shown below, is the matrix of service demands where $D_{i,r}$ is the average service demand of class r ($r = 1, \cdots, R$) transactions at queue i ($i = 1, \cdots, K$). The service demand of a transaction at a queue is the total service time of the transaction at the queue and does not include waiting time at the queue.

$$\mathbf{D} = \begin{pmatrix} D_{1,1} & D_{1,2} & \cdots & D_{1,r} & \cdots & D_{1,R} \\ D_{2,1} & D_{2,2} & \cdots & D_{2,r} & \cdots & D_{2,R} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ D_{i,1} & D_{i,2} & \cdots & D_{i,r} & \cdots & D_{i,R} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ D_{K,1} & D_{K,2} & \cdots & D_{K,r} & \cdots & D_{K,R} \end{pmatrix} \tag{2}$$

The vector $\vec{W} = (w_1, \cdots, w_r, \cdots, w_R)$ is the vector of workload intensities per class. In the case of open QNs, $w_r = \lambda_r$, which is the average arrival rate of class r transactions. In the case of closed QNs, $w_r = N_r$, i.e., the customer population for class r (i.e., the concurrency level of class r).

The techniques described in this section are used to derive the matrix of service demands $\mathbf{D}$ given a series of values of response times R_r and the workload intensity levels for which they were obtained. Numerical examples discussed in this section show how response times can be successfully predicted by an analytic model when the workload intensity values are different from the ones used to derive the service demands. In other words, the parameterized model has predictive capability.

4.1 Closed QNs

In this subsection, we show how we can derive the service demands that can be used to build a closed QN model of an operational system for which we assume that we can measure the response time and throughput of transactions submitted to the system.

To illustrate our ideas, we start with a single-class closed QN and then we generalize for multiple classes. In the single-class case, the matrix of service demands $\mathbf{D}$ becomes the vector $\vec{D} = (D_1, \cdots, D_i, \cdots, D_K)$ of service demands.

Consider that time is divided into M intervals $t_1, t_2, \cdots, t_j, \cdots, t_M$ all of duration τ, that R_j^a is the average response time of the transactions that completed during interval j and that C_j^a is the number of transactions that completed during that interval. Thus, according to Little's Law [16], the average number of transactions in the system during interval t_j is $n_j = R_j^a \times X_j^a$ where the throughput X_j^a during interval t_j is C_j^a/τ. The superscript a is used throughout the paper to indicate actual values, i.e., values obtained through measurements of the system as opposed to values predicted by a model.

Let f described below be a function that computes the computer system's average response time $R(n)$ when the average number of transactions in the system is equal to n. Besides n, the function takes as a parameter the vector $\vec{D} = (D_1, \cdots, D_i, \cdots, D_K)$ of service demands for the K queues in the QN. As indicated above, service demands do not include waiting time and therefore are independent of the system's load, represented by n.

$$R(n) = f(\vec{D}, n). \tag{3}$$

There is no closed form equation for the function f, only computational algorithms such as Mean Value Analysis (MVA) and Approximate Mean Value Analysis (AMVA) [20, 18].

Our problem can then be be formalized as follows. Given the sequence $\{(n_1, R_1^a), \cdots, (n_j, R_j^a), \cdots, (n_M, R_M^a)\}$ find the vector of service demands $\vec{D}$. For each interval j we can define the error ξ_j as

$$\xi_j = R(n_j) - R_j^a = f(\vec{D}, n_j) - R_j^a. \tag{4}$$

Equation (4) indicates the diference between the response time computed by function f for a concurrency level equal to n_j given service demands $\vec{D}$ and the actual response time R_j^a measured during that interval. We can find $\vec{D}$ by solving the following optimization problem.

$$\begin{aligned} \text{Minimize} \quad & \sum_{j=1}^{M} \xi_j^2 \\ \text{s.t.} \quad & \sum_{j=1}^{M} \xi_j = 0 \\ D_i \quad & \geq \quad 0 \quad i = 1, \cdots K \\ \sum_{i=1}^{K} D_i \quad & \leq \quad \min_{j=1,\cdots,M} \{R_j^a\}. \end{aligned} \tag{5}$$

The decision variables for the above problem are the service demands in $\vec{D}$, the objective function to be minimized is the sum of square errors, the first constraint forces the sum of the errors to be equal to zero, the second constraint

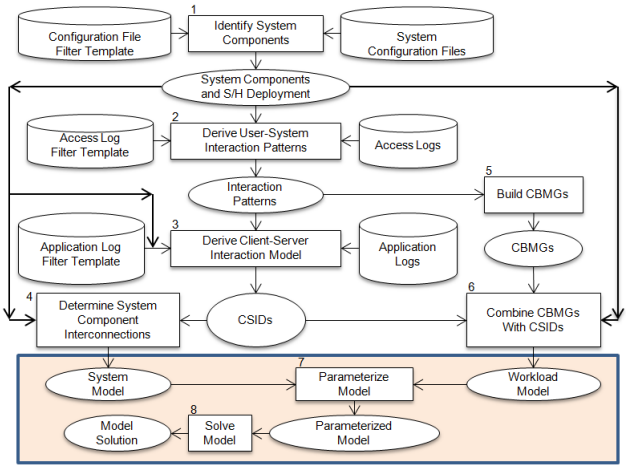

Figure 1: *iModel* Flowchart

indicates that the service demands cannot be negative, and the last constraint says that the sum of the service demands is the lowest possible value for the response time, i.e., the response time when there is no congestion.

This is a non-linear optimization problem that does not have a closed form solution for the objective function nor for its first constraint. Thus, we resort to black-box optimization methods.

Solving the optimization problem above for the data in Table 1 using Excel's Generalized Reduced Gradient (GRG) non-Linear solver provides the following values for service demands: $D_1 = 1.512$ sec, $D_2 = 1.781$ sec, and $D_3 = 2.008$ sec. It is important to note that the errors are all exactly equal to zero even though we do not have a constraint that requires that.

To check if the above service demands provide predictive capabilities, we solve an MVA model using these service demands for $n = 7$ and 11 and obtain the following response time values: 16.291 sec and 23.850 sec, respectively.

Table 1: Sequence of values of n_j, R_j^a, and X_j^a.

n_j	R_j^a (in sec)	X_j^a
2	7.09	0.2821
4	10.71	0.3736
5	12.53	0.3991
6	14.36	0.4177
8	18.06	0.4430
9	19.92	0.4518
10	21.79	0.4589

Figure 2 shows measured response times as a function of the concurrency level according to Table 1 (black triangles) and the two points, for $n = 7$ and 11 (unfilled squares). As it can be seen, the points computed by a model that uses the service demands estimated by our method match the curve obtained through measurements.

Figure 3 shows the measured throughput as a function of the concurrency level according to Table 1 (black triangles) and the two points, for $n = 7$ and 11 (unfilled squares) predicted by a closed QN model that uses the derived service demands resulting from the solution of the optimization

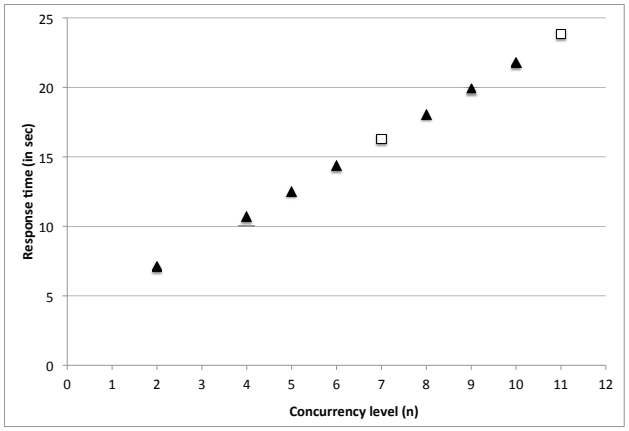

Figure 2: Response time (in sec) vs. concurrency level. Triangles: measured response time. Squares: predicted by a model.

problem described above. As it can be seen and as expected, the points computed by the QN model that uses the service demands estimated by our method match the curve obtained through measurements.

As another example of the power of the technique described here, we present another example in which the system is balanced (i.e., the service demands are the same for all devices). The graph in Fig. 4 shows throughput values measured from a system (black triangles) for concurrency levels equal to 2, 4, 6, 8, 9, 11, and 14. Applying our method yielded the following demands: $D_1 = 1.503$ sec, $D_2 = 1.501$ sec, and $D_3 = 1.496$ sec. The squares in the figure show the throughputs obtained by using a closed QN model that takes as inputs the service demands obtained by solving the optimization model. The figure shows that the throughput values obtained for concurrency levels equal to 1, 3, 5, 7, 10, and 13 (i.e., values not used in solving the optimization model) match exactly what they should be.

We now provide the algorithm that can be used to compute the average response time R_j^a, the average concurrency level n_j, and the throughput X_j^a for the intervals t_j from an

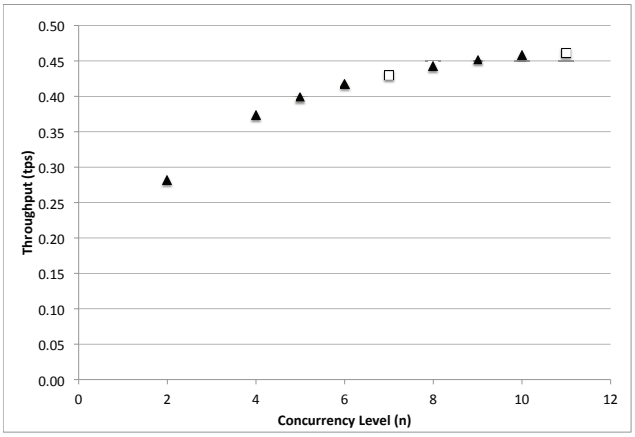

Figure 3: Throughput (in tps) vs. concurrency level. Triangles: measured throughput. Squares: predicted by a model.

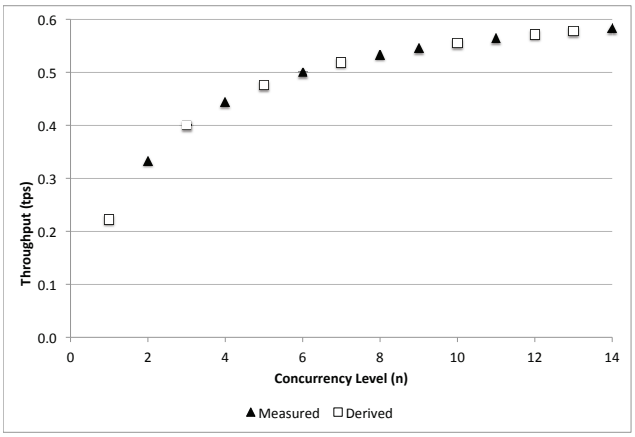

Figure 4: Throughput (in tps) vs. concurrency level for a balanced system. Triangles: measured throughput. Squares: predicted by the model.

operational system. We describe the algorithm for a single class case (i.e., similar transactions) but assume that while a system is in operation, its workload intensity and, consequently, the response time and throughputs will vary over time. The generalization for multiple classes is straightforward by segregating transactions based on their type (as indicated in operational logs, most of the time by the URL). The goal of the algorithm below is to determine time intervals during which there is very little variability (within a given tolerance η) in the average concurrency level n_j during the interval. Assume that an operational log of the operational system contains one entry per completed transaction. Entry i is of the type (t_i, R_i) where t_i is the timestamp of the entry and R_i is the response of the transaction that finished at time t_i.

Step 1: initialization TotalRespTime $\leftarrow 0$; StartTime $\leftarrow 0$; $n_j^{\text{prev}} \leftarrow 0$; $i \leftarrow 0$;

Step 2: accumulate response time $i \leftarrow i+1$; TotalRespTime $\leftarrow$ TotalRespTime $+ R_i$;

Step 3: compute concurrency level (see [18])

$$n_j^{\text{curr}} \leftarrow \text{TotalRespTime}/(t_i - \text{StartTime}) \quad (6)$$

Step 4: check tolerance If $\mid n_j^{\text{curr}} - n_j^{\text{prev}} \mid /n_j^{\text{curr}} > \eta$ then $n_j^{\text{prev}} \leftarrow n_j^{\text{curr}}$ and go to Step 2;

Step 5: outputs $n_j \leftarrow n_j^{\text{curr}}$; $R_j^a \leftarrow \text{TotalRespTime}/i$; $X_j^a = i/(t_i - \text{StartTime})$.

An example of the execution of the algorithm above is given in Table 2 that was generated from snippets of an actual analysis of the log of a real operational system. The table shows two identified intervals, one including entries numbered 29-36 and the other for entries numbered 46-54. The average concurrency level for the first interval shown in the table is 5.32 and is equal to 7.36 for the second. Their coefficients of variation are 0.05 and 0.09, respectively, i.e., very small. The average response times are 8.3 sec and 12.4 sec, respectivey, in these intervals.

Having presented the problem description for a single-class closed QN case, we generalize the problem for multiple classes. The equations below are an obvious generalization of equations (3)-(5).

$$R_r(\vec{N}) = f_r(\mathbf{D}, \vec{N}) \quad (7)$$

$$\xi_{j,r} = R_r(\vec{N}_j) - R_{j,r}^a = f_r(\mathbf{D}, \vec{N}_j) - R_{j,r}^a \quad (8)$$

where $\vec{N}_j$ is the population vector observed at interval t_j and the optimization problem is

$$\begin{aligned} \text{Minimize} \quad & \sum_{r=1}^{R}\sum_{j=1}^{M} \xi_{j,r}^2 \quad (9) \\ \text{s.t.} \quad & \sum_{r=1}^{R}\sum_{j=1}^{M} \xi_{j,r} = 0 \\ D_{i,r} \quad & \geq \quad 0 \quad i = 1, \cdots K \text{ and } r = 1, \cdots, R \\ \sum_{i=1}^{K} D_{i,r} \quad & \leq \quad \min_{j=1,\cdots,M}\{R_{j,r}^a\} \quad r = 1, \cdots, R. \end{aligned}$$

where $f_r(\mathbf{D}, \vec{N})$ is computed using AMVA [18].

4.2 Open QNs

Similarly to the closed QN case, time is divided into intervals $t_1, \cdots, t_j, \cdots, t_M$. For each interval t_j, we can measure the average response time per class, $R_{j,r}^a$, and the average arrival rate $\lambda_{j,r}^a$ per class.

The equations above are easily transformed into the open QN case by changing the workload intensity vector from $\vec{N}$ into the vector of arrival rates $\vec{\lambda} = (\lambda_1, \cdots, \lambda_r, \cdots, \lambda_R)$. We added a last constraint to the optimization problem to guarantee that the utilization $U_i = \sum_{r=1}^{R} \lambda_r \times D_{i,r}$ of device i for all devices $i = 1, \cdots, K$ is less than one.

$$R_r(\vec{\lambda}) = f_r(\mathbf{D}, \vec{\lambda}) \quad (10)$$

$$\xi_{j,r} = R_r(\vec{\lambda}_j) - R_{j,r}^a = f_r(\mathbf{D}, \vec{\lambda}_j) - R_{j,r}^a \quad (11)$$

Table 2: Example of interval computation based on real data for $\eta = 0.005$

i	Completion time (t_i)	Response time (sec) (R_i)	TotalRespTime	n_j^{curr}	n_j^{prev}	Error	$\bar{n}_j^{\text{curr}}$
29	27.52	13.23 0	132.872	4.83	4.62	0.0431	-
30	27.63	3.336	136.208	4.93	4.83	0.0207	-
31	29.06	18.77	154.978	5.33	4.93	0.0755	-
32	28.95	3.663	158.641	5.48	5.33	0.0267	-
33	30.3	4.01	162.651	5.37	5.48	0.0207	-
34	30.49	3.197	165.848	5.44	5.37	0.0132	-
35	32.6	16.311	182.159	5.59	5.44	0.0264	-
36	33.39	4.102	186.261	5.58	5.59	0.0017	5.32
-	-	-	-	-	-	-	-
46	41.42	5.129	260.197	6.28	6.39	0.0164	-
47	41.85	6.563	266.76	6.37	6.28	0.0144	-
48	42.19	29.899	296.659	7.03	6.37	0.0936	-
49	42.28	4.993	301.652	7.13	7.03	0.0144	-
50	42.10	22.807	324.459	7.71	7.13	0.0744	-
51	44.10	29.812	354.271	8.03	7.71	0.0405	-
52	43.87	5.583	359.854	8.20	8.03	0.0206	-
53	47.01	3.722	363.576	7.73	8.20	0.0606	-
54	47.33	3.038	366.614	7.75	7.73	0.0016	7.36

where $\vec{\lambda}_j$ is the vector of arrival rates at interval t_j and the optimization problem is

$$\begin{aligned} \text{Minimize} \quad & \sum_{r=1}^{R}\sum_{j=1}^{M} \xi_{j,r}^2 \qquad (12) \\ \text{s.t.} \quad & \sum_{r=1}^{R}\sum_{j=1}^{M} \xi_{j,r} = 0 \\ D_{i,r} \quad & \geq \quad 0 \;\; i = 1, \cdots K \text{ and } r = 1, \cdots, R \\ \sum_{i=1}^{K} D_{i,r} \quad & \leq \quad \min_{j=1,\cdots,M}\{R_{j,r}^a\} \quad r = 1, \cdots, R. \\ U_i = \sum_{r=1}^{R} \lambda_r \times D_{i,r} \quad & < \quad 1 \quad i = 1, \cdots, K. \end{aligned}$$

In the case of open QNs, the function $f_r(\mathbf{D}, \vec{\lambda})$ is computed through the following well-known closed-form equation [18].

$$f_r(\mathbf{D}, \vec{\lambda}) = R_r(\vec{\lambda}) = \sum_{i=1}^{K} \frac{D_{i,r}}{1 - \sum_{r=1}^{R} \lambda_r \times D_{i,r}}. \qquad (13)$$

Figure 5 shows an example in which the (arrival rate, response time) pairs shown by the solid triangles were used to solve the optimization problem described in Eq. (12) for an open QN. The solution to this problem yields the following set of service demands: $D_1 = 2.30$ sec, $D_2 = 1.60$ sec, $D_3 = 1.60$ sec, and $D_4 = 1.58$ sec. We then used a single class open QN model to compute the response times for the following values of the arrival rate (in tps), which were not used as input to solve the optimization problem: 0.04, 0.10, 0.14, 0.16, 0.22, 0.26, 0.30, 0.32, 0.36, 0.40, and 0.41. The values of the response times predicted by the model are plotted in Fig. 5 as unfilled triangles. As the figure illustrates, the predicted values match perfectly with the curve that underlies the measured points.

We now show an example with two classes and three devices. Table 3 shows the measured arrival rates and measured response times per class for six measurement intervals.

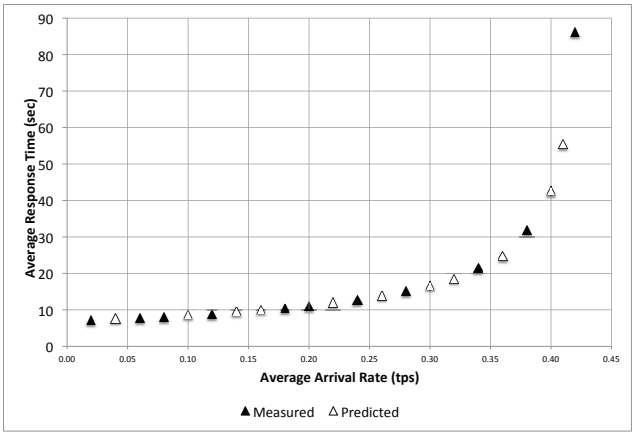

Figure 5: Average response time (in sec) vs. average arrival rate (in tps). Solid triangles: measured response times. Unfilled triangles: predicted by a model.

Table 4 shows the matrix of service demands that resulted from the solution of the optimization problem above.

Figures 6 and 7 use solid markers (triangles and squares) to show the average response times for classes 1 and 2, respectively, for the six intervals shown in Table 3. Additionally, the unfilled markers in these figures show response time values predicted for the average arrival rates shown in Table 5 and computed by Eq. (13). The two predicted values in each figure do not show a perfect continuity because the values of the arrival rates for all classes impact all other classes through the utilization of all devices (see denominator of Eq. (13)). Nevertheless, since the arrival rates for which the prediction is shown in Table 5 are within the arrival rates for two successive intervals, one can see how the

Table 3: Measured arrival rates (in tps) and average response times (in sec) per class.

	Class 1		Class 2	
Interval	$\lambda^a_{j,1}$ (tps)	$R^a_{j,1}$ (sec)	$\lambda^a_{j,1}$ (tps)	$R^a_{j,2}$ (sec)
t_1	0.030	3.300	0.015	2.827
t_2	0.050	3.475	0.045	2.977
t_3	0.100	3.832	0.080	3.282
t_4	0.300	5.792	0.150	4.910
t_5	0.350	7.375	0.220	6.241
t_6	0.450	13.992	0.320	11.656

Table 4: Derived service demands (sec).

Queue	Class 1	Class 2
CPU	1.00	0.80
Disk 1	1.25	0.80
Disk 2	0.90	1.10

predicted values fall within the measured values. For example, the first pair of arrival rates for which predictions are shown in Table 5 is 0.2 tps and 0.09 tps for classes 1 and 2, respectively. In both cases, these arrival rates fall within those of intervals t_3 and t_4. The corresponding predicted response times also fall between the measured response times for these intervals.

5. EXPERIMENTAL VALIDATION

This section presents a validation of the method described in this paper on an experimental testbed. Our implementation used a number of supporting tools and technologies, including Apache OFBiz (ofbiz.apache.org) and Apache JMeter (jmeter.apache.org) for running the experiment, and LogStash (www.elastic.co) for parsing system logs. We used jMetal (see http://jmetal.sourceforge.net) as the optimization framework for solving the non-linear optimization problem. This framework allows one to use various optimization algorithms. After experimenting with several, we settled on MOEA, which is an open-source evolutionary computation library for Java that specializes in multi-objective optimization. We also used an in-house implementation of the Approximate Mean Value Analysis (AMVA) algorithm.

Apache OFBiz™ is an open source ERP (Enterprise Resource Planning) system from Apache. Logstash, which is part of the ELK stack (Elasticsearch, Logstash, and Kibana,) is an open source tool used to collect, organize, and parse various system logs and uses templates defined using regular expressions to describe the log structure and parse its contents. The output of Logstash is stored in Elasticsearch, which is a document store and a search server, where documents are stored in JSON format (Java Script Object Notation).

Apache JMeter is an open source tool used to record, generate, and run multiple workloads. It uses different techniques for spawning processes representing system users, and it has many extensions for recording experiment results in various formats.

Table 5: Computed average response times (in sec) per class.

Class 1		Class 2	
λ_1 (tps)	R_1 (sec)	λ_1 (tps)	R_2 (sec)
0.20	4.466	0.09	3.805
0.40	9.132	0.25	7.680

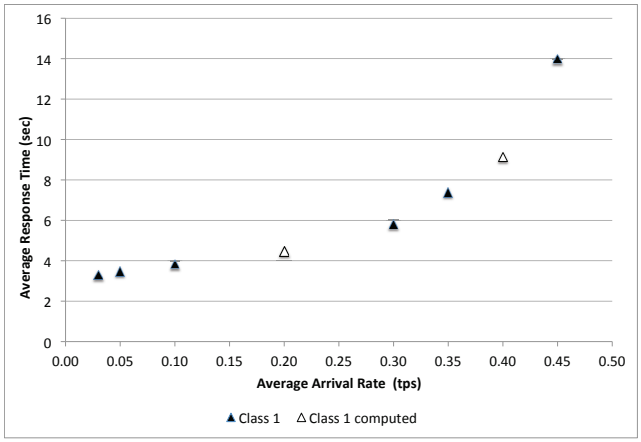

Figure 6: Average response time (in sec) for class 1 vs. average arrival rate (in tps). Solid triangles: measured response time. Unfilled triangles: predicted by a model.

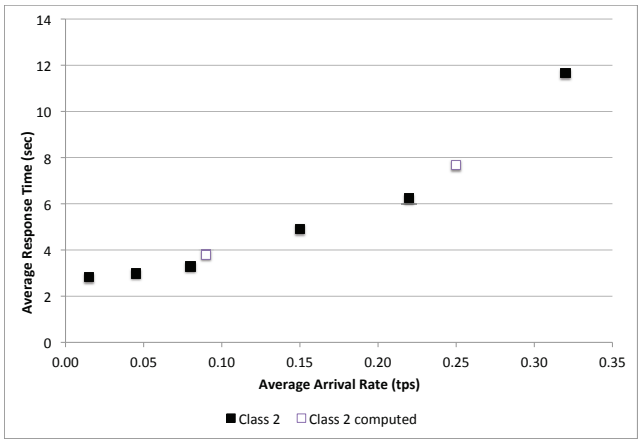

Figure 7: Average response time (in sec) for class 2 vs. average arrival rate (in tps). Solid squares: measured response time. Unfilled squares: predicted by a model.

5.1 Experimental Conditions and Assumptions

Our testbed uses Ubuntu Linux servers to host the OFBiz ERP software suite, which was installed on a Tomcat 7.0 application server, and to host the MySQL database supporting the OFBiz suite. Apache JMeter was used to record and run a test plan that simulates different groups of users performing the business function Order Entry.

Table 6 shows the Order Entry sequence of steps, called workflow heretofore, performed by the users and captured by JMeter. Most of the workflow steps are interactive, such as login (step 2) and initorderentry (step 6). Other steps are

performed by the system and do not require think time, such as image loading and search lookahead, as in steps 4, 5 and 8, where the system performs multiple AJAX (Asynchronous JavaScript And XML) requests while the user is typing a customer name or product name. Steps 4, 5, 8 and 9 are repeated multiple times for a total of 21 transactions in a workflow.

Table 6: Order Entry Workflow

no.	Workflow Step
1	/ordermgr/main
2	/ordermgr/login
3	/ordermgr/orderentry
4	/ordermgr/LookupUserLoginAndPartyDetails (2)
5	/ordermgr/LookupCustomerName (3 times)
6	/ordermgr/initorderentry
7	/ordermgr/setOrderCurrencyAgreementShipDates
8	/ordermgr/LookupProduct (4 times)
9	/ordermgr/additem (2 times)
10	/images/GZ-1001/small.png
11	/images/GZ-9290/small.png
12	/ordermgr/quickcheckout
13	/ordermgr/updateCheckoutOptions/quickcheckout
14	/ordermgr/checkout

We used JMeter to run the workflow shown in Table 6 using 5, 10, 15, 20, 22, 25, 27, 30, 32, 35, 40 and 45 users, called threads heretofore, under the following conditions:

1. Response time is defined as the time it takes to perform all 21 steps of the workflow.

2. JMeter's "Ramp-Up Period" is set to 30 seconds, which is the time JMeter takes to ramp up to the full number of threads.

3. JMeter's "Delay Thread Creation Until Needed" is set to true to eliminate the overhead of starting all threads at the beginning of the experiment.

4. JMeter's Loop Count is set to 10, which means that, for each experiment, threads ran 10 times consecutively. For example, an experiment with 10 users means the workflow in Table 6 ran 100 times producing 2100 transactions.

5. Experiment ramp-up and ramp-down times. In order to capture measurements during the system's steady state, we defined the measurement start time, T1, and end time, T2, for each experiment as follows:

   T1 = maximum start time of all workflows for all threads.

   T2 = minimum end time of all workflows for all threads.

   For each of the 10 experiments, all threads that started or ended prior to T1 or after T2 were removed from the final measurements. By doing that, we eliminated all partial workflows that might skew the average response time or average throughput given our definition of response time.

6. Think time between workflow steps uses a uniform random timer between 0 and 3 seconds. The random think time is only used between interactive steps. For example, think time is added between all steps except steps 10 and 11 (image loading) and between the multiple AJAX calls of steps 5 and 8 (LookupCustomerName and LookupProduct).

To validate our method we performed the following steps:

Step 1 - Measurements Perform several measurements on a system running the OFBiz ERP software suite for several intervals to obtain the throughput and response time. The results of our measurements are reported in Table 7 for intervals t_1-t_{12}. Each line shows the number of clients generating requests in each interval. The number of workflows generated in each interval varied from 42 to 362 excluding the ramp-up and ramp-down times (actual total workflows varied from 50 to 450.) We used Little's Law [16] to obtain the concurrency level n_j for each interval t_j as the product of the measured throughput and the measured response time (i.e., $n_j = X_j^a \times R_j^a$). Figure 8 shows the average response time per workflow execution and Fig. 9 shows the average throughput in workflows per second (wps).

Table 7: Apache OFBiz Results

Interval	X_j^a (in wps)	R_j^a (in sec)	n_j
t1 (5 users)	0.130	4.406 ± 0.145	0.575
t2 (10 users)	0.255	4.481 ± 0.149	1.141
t3 (15 users)	0.389	4.602 ± 0.149	1.788
t4 (20 users)	0.516	5.234 ± 0.123	2.700
t5 (22 users)	0.565	5.337 ± 0.141	3.016
t6 (25 users)	0.622	5.751 ± 0.398	3.574
t7 (27 users)	0.659	6.722 ± 0.168	4.427
t8 (30 users)	0.738	7.193 ± 0.191	5.307
t9 (32 users)	0.746	8.430 ± 0.225	6.286
t10 (35 users)	0.773	10.077 ± 0.257	7.785
t11 (40 users)	0.805	15.002 ± 0.372	12.076
t12 (45 users)	0.806	20.462 ± 0.465	16.497

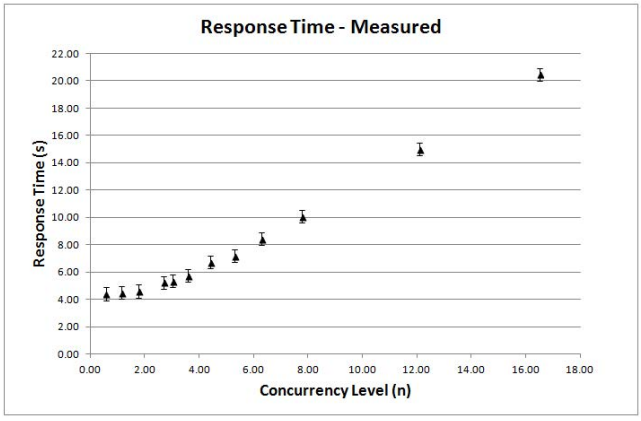

Figure 8: Average response time (in sec) vs. concurrency level.

Step 2 - Estimating Service Demands Apply the optimization technique described in the previous sections to estimate the service demands for the closed QN

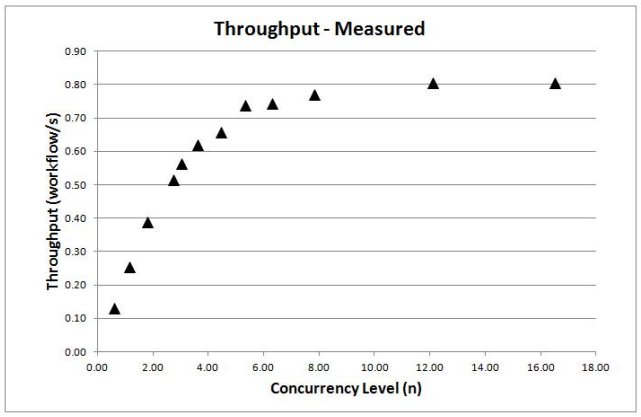

Figure 9: Average throughput (in wps) vs. concurrency level.

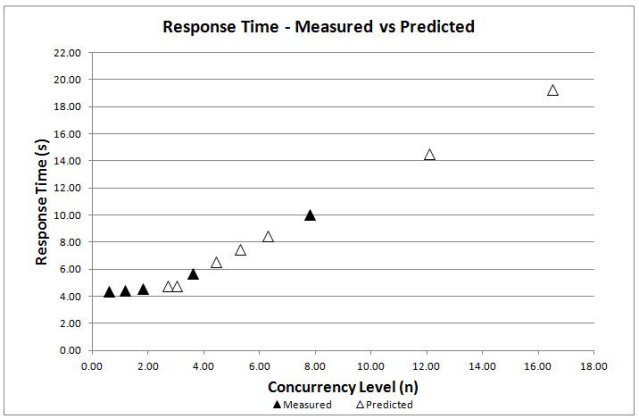

Figure 10: Response time (in sec) vs. concurrency level. Solid Triangles: measured response time. Unfilled Triangles: predicted by a model.

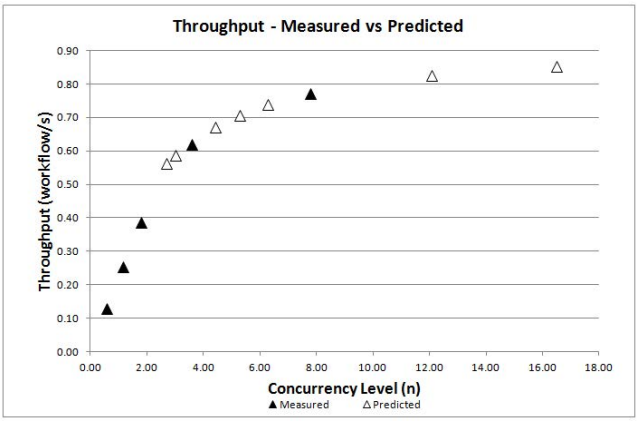

Figure 11: Throughput (in workloads per sec) vs. concurrency level. Solid Triangles: measured response time. Unfilled Triangles: predicted by a model.

model that represents the experimental testbed. In our example, we built a closed QN model with three queues representing the Web server, the application server, and the database server. The service demands derived by solving the optimization model, using as inputs the data for intervals t_1, t_2, t_3, t_6 and t_{10} are 0.937 sec, 1.114 sec and 1.002 sec, respectively. Because the concurrency levels for those five intervals are not integer, we used Approximate MVA, which uses the Bard-Schweitzer approximation to solve closed QNs with non-integer concurrency levels [22].

Step 3 - Prediction Using the closed QN model with the service demands obtained in step 2, compute the throughput and response time for concurrency levels not used in step 2. See the X^{model} and R^{model} columns for intervals t_4, t_5, t_7, t_8, t_9, t_{11} and t_{12} in Table 8.

Step 4 -Validation Compare the response time values predicted by the model in step 3 with measured response times for the same concurrency levels. If the values are relatively close (i.e., on the order of 10% difference), the service demand estimation process is validated and the resulting closed QN model can be considered to have predictive power. As it can be seen in the last two columns of Table 8, the percent absolute error defined as $100 \times (measurement - model)/measurement$ for intervals t_4, t_5, t_7, t_8, t_9, t_{11} and t_{12} is small (less than 10% and—in most cases—less than 5%), which indicates that the derived model has predictive power for concurrency levels not used in the derivation of the service demands. It is important to note that the largest errors occurred for small concurrency levels (2.7 and 3.0 workflows) due to the fact that we used AMVA, which does not provide a good approximation for low concurrency levels. Also, production systems rarely operate at such low concurrency levels.

Figure 10 shows five measured response times as a function of the concurrency level according to Table 8 (see black triangles) and the seven points t_4, t_5, t_7, t_8, t_9, t_{11} and t_{12} (white triangles). As it can be seen, the points computed by a model that uses the service demands estimated by our method match the curve obtained through measurements.

Figure 11 shows the measured throughput as a function of the concurrency level according to Table 8 (see black triangles) and the seven points t_4, t_5, t_7, t_8, t_9, t_{11} and t_{12} (white triangles) predicted by a closed QN model that uses the derived service demands resulting from the solution of the optimization problem described above. As it can be seen and as expected, the points computed by the QN model that uses the service demands estimated by our method match the curve obtained through measurements.

6. RELATED WORK

We have already referenced before our own prior work [1, 2, 3]. In this section we discus related prior work by others. Some of the prior work that tackled the parameterization of analytical models includes [5, 6, 7], where the problem of estimating known model parameters is treated as an optimization problem that is solved using derivative-free optimization. The objective function to be optimized is based on the distance between the observed measurements and the corresponding points derived from the model. The authors point out that the main problem is determining how to cou-

Table 8: Validation with experimental results obtained using OFBiz Order Entry transactions.

Period	No. Users	X_j (in Wps)	R_j^a (in sec)	n_j	X^{model} (in Wps)	R^{model} (in sec)	% Rel. Error Throughput	% Rel. Error Resp. Time
t_1	5	0.130	4.406 ± 0.145	0.575	NA	NA	NA	NA
t_2	10	0.255	4.481 ± 0.149	1.141	NA	NA	NA	NA
t_3	15	0.389	4.602 ± 0.149	1.788	NA	NA	NA	NA
t_6	25	0.622	5.751 ± 0.398	3.574	NA	NA	NA	NA
t_{10}	35	0.773	10.077 ± 0.257	7.785	NA	NA	NA	NA
Comparison between measurements and model predicted results								
t_4	20	0.516	5.234 ± 0.123	2.700	0.56	4.80	-9%	8%
t_5	22	0.565	5.337 ± 0.141	3.016	0.59	4.84	-4%	9%
t_7	27	0.659	6.722 ± 0.168	4.427	0.67	6.58	-2%	2%
t_8	30	0.738	7.193 ± 0.191	5.307	0.71	7.49	4%	-4%
t_9	32	0.746	8.430 ± 0.225	6.286	0.74	8.51	1%	-1%
t_{11}	40	0.805	15.002 ± 0.372	12.076	0.83	14.60	-3%	3%
t_{12}	45	0.806	20.462 ± 0.465	16.497	0.85	19.32	-6%	6%

ple these two sets of points in order to arrive at an objective function to be minimized. The proposed approach is applied to a small set of single-queue models, whereas our approach applies to single and multiclass open and closed QNs.

The work in [4] used Kalman filters to estimate resource service demands for the purpose of system performance testing. The authors attempted to find the workload mix that would eventually saturate a certain system resource in a test environment in order to determine the system's bottlenecks.

The work in [15, 25, 26] addressed the problem of estimating model parameters in highly dynamic autonomic environments in which Service Level Agreements (SLAs) (in the form of Quality of Service (QoS)) have to be maintained while offering optimal use of data center resources. The authors proposed the use of a model-based estimator based on Extended Kalman Filters, where the current state depends on prior knowledge of previous states. Our approach, on the other hand, relies on solving an optimization problem where only current input and output values are known.

Menascé tackled the issue of model parameterization for open QNs when some input parameters are already known [17]. The author proposed a closed-form solution to the case when a single service demand value is unknown, and a constrained non-linear optimization solution when a feasible set of service demands are unknown. However, that work did not propose a solution when none of the service demands are known a priori.

In [9], the authors introduced a method for performance parameter estimation using Kalman filters and Layered Queuing Models. They used a clustering algorithm to determine the optimal number of classes such that the LQN closely tracks the behavior of the real system.

In [21], the authors presented three service demand estimation methods (RPS, MLPS and MINPS) for multi-threaded applications. RPS is used in single-processor systems, MLPS is used in multi-processor systems and MINPS consolidates both methods. Both RPS and MLPS methods over-estimate the service demand in the type of workload they handle. MINPS runs both methods and chooses the method that produces the smaller estimated mean service time.

In [24], the authors presented a survey of service demand estimation techniques, such as Kalman filter, optimization, machine learning, and linear regression. The authors introduced a classification scheme based on the input parameters, output metrics and the ability of the estimation technique to tolerate input data anomalies. In order to evaluate the accuracy of the various estimation techniques, the authors generated two data sets; one using an M/M/1 queuing simulator, and another using a real system. MATLAB functions were used to implement the estimation techniques, such as least-squares regression and constrained non-linear optimization. The accuracy of the estimation technique was assessed using the mean relative demand error, relative utilization error or relative response time error. The results presented in the paper show how service demand estimation accuracy is impacted by the length of sampling interval, number of samples, number of workload classes, load level, delays during processing and execution time. The experimental results presented in the paper are meant to help performance engineers decide which estimation technique to use given the impact of variations in sampling intervals, workload classes and execution time on the accuracy of the technique's service demand results. An example of the results is that estimation techniques that rely on response time measurements are not as negatively impacted by missing workload classes as techniques that rely on utilization measurements.

The work in [13] proposes an approach based on queuing networks to represent system configurations. They use symbolic analysis and satisfiability modulo theory (SMT) to find a model that fits continuous changes in run-time conditions.

In[12], the authors used the previously introduced Descartes Modeling Language (DML) to illustrate how self-aware systems can manage their resources using a closed-feedback loop (MAPE-K control loop). Their proposed online performance prediction process takes into consideration the system's performance objectives, constraints and overhead. Online performance prediction is achieved by answering performance queries that specify the performance metrics desired and the constraints and overhead that would impact the adaptation plan selection. Performance queries are implemented using a declarative (and expressive) query language. Using the DML instance of the system and the performance query, the performance prediction process produces a model that is either fine grained, coarse grained or black-box and—eventually—solves the model, thereby producing performance query results. The framework uses a

number of solving techniques to solve the generated model, including bound analysis techniques based on Little's law and utilization law, LQN solver for solving LQN's and a combination of QNs and Colored Generalized Stochastic Petri Nets. The experimental results show how parameterizing the model at lower load levels (20% CPU utilization) can be used to predict the system's performance at higher load levels within 5% error in resource utilization and 20% error in mean response time.

7. CONCLUDING REMARKS AND FUTURE WORK

This paper presented and validated a method to automatically derive service demand parameters of open and closed multiclass queuing network models. The input to the method is a set of measured response times for different levels of workload intensity (arrival rates or concurrency levels). Then, a non-linear optimization problem is setup where the objective function is the sum of square errors between the measured response times and the response times computed by the QN model. The problem is that there is no closed form expression for response times for closed QNs. To overcome this problem we resorted to black box optimization techniques. We validated our method by setting up an experimental testbed to run the OFBiz suite. We then took measurements, derived the service demands using the method presented in this paper and then used the automatically parameterized model to predict the response time for non-measured workload intensity levels.

The method described in this paper can be applied to situations in which the machines are multi-core. This can be done by using Seidmann's approximation [23] in which a service center with a single queue and m servers (e.g., m cores) each with service demand D can be replaced in the QN by a single server queue with service demand D/m followed by a delay server with service demand $D(m-1)/m$. The use of this approximation transforms an open or closed QN with multiserver queues into one with only single server queues.

8. ACKNOWLEDGMENTS

This work was partially supported by the AFOSR grant FA9550-16-1-0030. The authors thank Ms. Noor Bajunaid for having developed the Java code for the implementation of AMVA.

9. REFERENCES

[1] Awad, M. and Menascé, D.A., "Performance Model Derivation of Operational Systems Through Log Analysis," IEEE Intl. Symp. Modeling, Analysis and Simulation of Computer Systems and Telecommunication Systems (MASCOTS 2016), Imperial College, London, UK, Sept. 19-21, 2016.

[2] Awad, M. and Menascé, D.A., "On the Predictive Properties of Performance Models Derived Through Input-Output Relationships", Proc. European Performance Engineering Workshop (EPEW 2014), Venice, Italy, 2014.

[3] Awad, M. and Menascé, D.A., "Dynamic Derivation of Analytical Performance Models in Autonomic Computing Environments", Proc. Computer Measurement Group Conf. Performance and Capacity, Atlanta, GA, 2014.

[4] Barna, C., Litoiu, M., Ghanbari, H., "Autonomic load-testing framework," Proc. 8th ACM Intl. Conf. Autonomic Computing, Karlsruhe, Germany, 2011, pp. 91–100.

[5] Begin, T., Brandwajn, A., Baynat, B., Wolfinger, B.E., Fdida, S., "Towards an automatic modeling tool for observed system behavior," Formal Methods and Stochastic Models for Performance Evaluation, pp. 200–212. Springer Berlin Heidelberg (2007)

[6] Begin, T., Brandwajn, A., Baynat, B., Wolfinger, B. E., Fdida, S., "High-level approach to modeling of observed system behavior," ACM SIGMETRICS Performance Evaluation Review, 35(3), 34–36 (2007)

[7] Begin, T., Baynat, B., Sourd, F., Brandwajn, A., "A DFO technique to calibrate queuing models," Computers & Operations Research, 37, no. 2, 2010, pp. 273–281.

[8] Durillo,J.J. and Nebro, A.J., "jMetal: a Java Framework for Multi-Objective Optimization," Advances in Engineering Software, 42(2011), pp. 760-771.

[9] Ghanbari, H., Barna, C., Litoiu, M., Woodside, M., Zheng, T., Wong, J., Iszlai, G., "Tracking Adaptive Performance Models Using Dynamic Clustering of User Classes," Proc. 2nd ACM/SPEC Intl. Conf. Performance Engineering, Karlsruhe, Germany, March 14-16, 2011.

[10] Gmach, D., J. Rolia, L. Cherkasova, and A. Kemper. "Workload Analysis and Demand Prediction of Enterprise Data Center Applications," Proc. 2007 IEEE 10th Intl. Symp. Workload Characterization (IISWC '07), Boston, MA, Sept. 27-29, 2007.

[11] Herbst, N.R., N. Huber, S. Kounev, and E. Amrehn. "Self-adaptive workload classification and forecasting for proactive resource provisioning." Proc. 4th ACM/SPEC Intl. Conf. Performance Engineering (ICPE '13), Prague, Czech Republic, April 21-24, 2013.

[12] Huber, N., F. Brosig, S. Spinner, S. Kounev, and M. Bahr, "Model-Based Self-Aware Performance and Resource Management Using the Descartes Modeling Language," to appear in the IEEE Tr. Software Engineering, DOI 10.1109/TSE.2016.2613863

[13] Incerto, E., M. Tribastone, and C. Trubiani, "Symbolic Performance Adaptations" Proc. 11th Intl. Symp. Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2016), May 16-17, 2016, Austin, TX.

[14] Khan A., X. Yan, S. Tao and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," 2012 IEEE Network Operations and Management Symposium, Maui, HI, 2012

[15] Litoiu, M., Woodside, M., Zheng, T., "Hierarchical model-based autonomic control of software systems," ACM SIGSOFT Software Engineering Notes, Vol. 30, No. 4, pp. 1–7 (2005)

[16] Little, J.D.C., "A Proof for the Queuing Formula: L = λ W," Operations Research. 9 (3), 1961, pp 383–387.

[17] Menascé, D., "Computing Missing Service Demand Parameters for Performance Models," Proc.

Thirty-fourth Intl. Computer Measurement Group Conf., Las Vegas, NV, Dec. 7-12, 2008, pp. 7–12.

[18] Menascé, D.A., Almeida, V.A.F., and Dowdy, L., "Performance by Design: Computer Capacity Planning By Example." Prentice Hall, 2004.

[19] Menascé, D.A. and Almeida, V.A.F., "Scaling for E-Business: technologies, models, performance, and capacity planning," Prentice Hall, 2000.

[20] Reiser, M. and Lavenberg, S. S. "Mean-Value Analysis of Closed Multichain Queuing Networks," J. ACM., 27(2), April 1980, pp. 313–322.

[21] Pérez, J.F., Pacheco-Sanchez, S., Casale, G., "An Offline Demand Estimation Method for Multi-Threaded Applications," Proc. 2013 IEEE 21st Intl. Symp. Modeling, Analysis and Simulation of Computer and Telecommunication Systems (pp. 21-30), IEEE Computer Society, 2013.

[22] Schweitzer, P. "Approximate analysis of multiclass closed networks of queues." Proc. Intl. Conf. Stochastic Control and Optimization, 1979.

[23] Seidmann, A., P. Schweitzer, and S. Shalev-Oren, "Computerized Closed Queueing Network Models of Flexible Manufacturing," Large Scale System J., North Holland, Vol. 12, 1987, pp. 91–107.

[24] Spinner S., G. Casale, F. Brosig, and S. Kounev., "Evaluating Approaches to Resource Demand Estimation," Performance Evaluation, 92:51 - 71, October 2015, Elsevier B.V.

[25] Woodside, M., Zheng, T., Litoiu, M., "The use of optimal filters to track parameters of performance models," Second Intl. Conf. Quantitative Evaluation of Systems, Torino, Italy, Sept. 19-22, 2005, pp. 74–83.

[26] Zheng, T., Yang, J., Woodside, M., Litoiu, M., Iszlai, G. "Tracking time-varying parameters in software systems with extended Kalman filters," Proc. 2005 Conf. of the Centre for Advanced Studies on Collaborative Research, 2005, pp. 334–345.