On the Predictive Properties of Performance Models Derived through Input-Output Relationships

Mahmoud Awad and Daniel A. Menascé

Computer Science Department, George Mason University
Fairfax, VA 22030, USA
{mawad1,menasce}@gmu.edu

Abstract. Building an analytical performance model is a challenge when little is known about the functionality and behavior of the system being modeled and/or when obtaining model parameters through measurements is difficult. This paper addresses this problem by presenting an approach that derives analytic model parameters by observing the input-output relationships of a real system. More specifically, input (i.e., arrival rates for each job class) and output (i.e., average response time for each job class) measurements are used to estimate the per-class service demands and number of servers for a Queuing Network model of the system. This model, called the computed model (CM), provides the same output values for the same input values used to derive the CM. The important question is whether the CM has predictive power, i.e., can the CM predict the output values that would be observed in the real system for different values of the input? The CM's parameters are obtained by solving a non-linear optimization problem. The paper shows through experiments that the CM is relatively robust and has predictive power over a range of input values.

Keywords: Queuing network models, parameter estimation, non-linear optimization.

1 Introduction

Analytical performance models, such as Queuing Network (QN) models, are essential for performance prediction as well as understanding the qualitative characteristics of a computer system. Developing these models requires intimate knowledge of the computer system and the availability of a number of performance measurements to estimate the model's input parameters. However, there are cases when performance prediction is needed for computer systems whose functionality and behavior are not fully understood.

For example, Internet data centers with virtualized environments, such as cloud computing providers, are capable of hosting multiple heterogeneous application systems of different sizes and complexities. Therefore, it is practically impossible for such data centers to adequately understand the functionality offered by all hosted application systems in order to develop precise performance models.

A. Horváth and K. Wolter (Eds.): EPEW 2014, LNCS 8721, pp. 89–103, 2014.

The issue we address in this paper is the ability to derive QN models for computer systems where little is known about the internal architecture or behavior of the system. We view the system as a "grey" box where input parameters (e.g., arrival rates) and output parameters (e.g., average response time) are known, in addition to some internal structural information about the system (e.g., number and function of each layer in a multi-tiered system, but not necessarily the number of servers of each type). We then derive a QN model, which we call the Computed Model (CM), that closely approximates the computer system. Such model will be of benefit only if it proves capable of predicting the behavior of the real system as the workload intensity changes over time.

Our approach for deriving QN performance models uses a non-linear optimization technique to determine the parameters of the CM. We conducted a number of experiments to evaluate the ability of the CM to predict the behavior of the real system as workload intensity changes. In order to test the viability of our approach, we initially, conducted a number of controlled experiments with an analytical QN model acting as a proxy for a real system. Our results showed that our approach is capable of producing computed QN models that have a robust predictive power. We then conducted two experiments using Apache OFBiz (Open For Business) ERP system; one in which the number of servers per tier is known a priori (Static-N) and one in which the number of servers per tier is inferred by the optimization technique (Variable-N). Both OFBiz experiments show results consistent with the controlled experiments.

A few prior efforts are related to our research. It is important to note that this paper is not about parameterizing QN models of real systems. There is a vast body of literature on that. In that context, the system components are totally visible to the modeler and therefore can be instrumented. What is of interest to the work in this paper is a situation where the modeler either has no access, is unwilling, or does not have the resources to conduct measurements on the internal components of a computer system. In that case, the work in [2,3] used a customized non-linear optimization technique to approximate unknown model parameters when the queuing model is already known. However, their technique is applied to a limited set of single queue models. Also, [11] proposed a black-box approach to calculating unknown input parameters in open multi-class queuing networks given that some service demands are known.

Our optimization technique is simple and efficient and can be applied to open multi-class QN models, where the workload intensity and the response time of the modeled system are known, in addition to minimal internal structure information.

Autonomic computing environments can benefit the most from our approach of deriving performance model parameters dynamically [6]. These environments rely heavily on optimal resource allocation through performance prediction, where the internal architecture of the hosted application systems is not fully understood. The technique presented here does not apply when one is interested in predicting metrics internal to a system (e.g., CPU and disk queue lengths)

The rest of the paper is organized as follows. Section 2 includes the problem definition and notation used throughout the rest of the paper. Section 3 discusses the methodology and algorithm used. The next section shows experimental results. Section 5 discusses some related work. Finally, section 6 presents discussions and concluding remarks.

2 Problem Definition

In model-based performance engineering, real computer systems are abstracted using analytical models, which can be used by performance engineers to answer "what if" questions related to predicting system performance. Queuing network (QN) models have been used quite often for that purpose. Such models have two types of parameters: workload intensity (e.g., arrival rates) and resource service demands (i.e., the average total time spent by a transaction using a resource). Service demands do not include the time waiting to use a resource. See e.g. [12] for details on QN models.

As discussed before, in virtualized environments, the internal architecture of hosted application systems may not be readily available. However, it is a common practice for such environments to provide monitoring tools that can easily record and analyze input and output parameters, such as the transaction arrival and departure rates as well as the time taken to process each transaction. Such monitoring is readily available in operating systems as well as virtualization software, and should be adequate, in our opinion, to develop an overall model that approximates the behavior of the application system.

Figure 1 demonstrates the problem we address in this paper. To derive an approximate analytical model of the system, we treat it as a"gray box," where the input and output parameters are known, as well as minimal information about the internal structure of the system (e.g., number of tiers in a multi-tiered system). The QN Parameter Estimator takes average arrival rates λ (input) and average response times T^{AS} (output) and establishes a relationship between them in order to estimate the parameters of the Computed QN Model (CM). The relationship between the input and output parameters is formulated as a non-linear optimization problem that can be solved using a non-linear solver.

After the Computed QN Model is parameterized, the behavior of the actual system and the corresponding QN model need to be compared frequently to ensure that the QN model accurately represents the actual system, which is important if the model is to be used for performance prediction. This comparison process is demonstrated in Figure 2, where the observed response time, T^{AS}, of the actual system and the computed response time, T^{CM}, of the Computed QN Model are compared for the same arrival rate. The question we pose in our experiments is how accurate is the response time of the CM when compared to the observed response time of the actual system.

This paper considers the problem of deriving and parameterizing a QN model given that the arrival rate (input) and the response time (output) of the actual system can be measured at regular intervals. We focus primarily on multi-tier

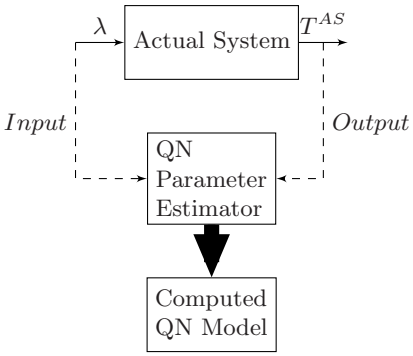

Fig. 1. Parameterizing the Computed QN Model

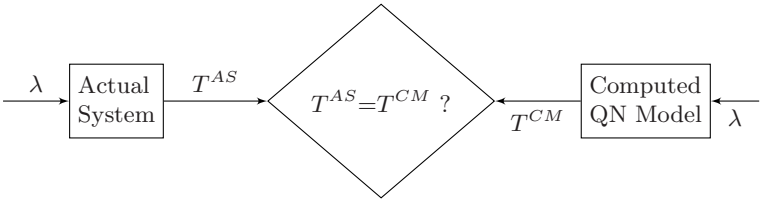

Fig. 2. Maintaining the Accuracy of the Computed QN Model

application systems, in which transactions are processed by one of several servers at each tier and passed on to the next tier. This architecture is typical of online transaction processing systems such as e-commerce application systems. However, our approach can be applied to a host of architectures and application systems.

Figure 3 shows a sample computed QN Model that would be used to represent an actual system with a 3-tier architecture. We will use this architecture to demonstrate our methodology and experimental results. It would be obvious to one skilled in QNs that our approach can be generalized to any number of tiers. In this architecture, transactions are submitted to the web server tier, which may consist of a number of servers that are load balanced. Transactions are passed from the web server tier to the application server tier, which may also consist of a number of load balanced servers. If the transaction needs to access the database, it will be passed on to the database server tier, which will process the transaction and return the results back to the application server tier and then to the web server tier.

We assume throughout the paper that the number of tiers in the actual system is known (we call this internal structural information). But, we assume we do not know the number of servers in each tier, their internal components, nor the service demands at these components for each transaction class. In order to parameterize the computed QN Model, we need to find the service demands for each tier for each class. Consider the following notation:

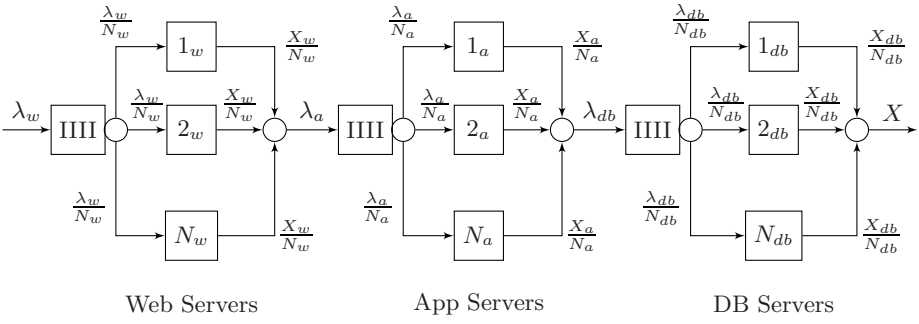

Fig. 3. Computed QN Model Topology in a 3-tier Architecture

- R: number of transaction classes in the queuing network
- λ_r: arrival rate of class r transactions ($r = 1, \cdots, R$)
- N_w, N_a, N_{db}: number of servers in the web, application, and database tiers, respectively
- $D^{\text{AS}}_{w,r}, D^{\text{AS}}_{a,r}, D^{\text{AS}}_{db,r}$: average class r ($r = 1, \cdots, R$) service demands at each of the three tiers for the actual system.
- $D^{\text{CM}}_{w,r}, D^{\text{CM}}_{a,r}, D^{\text{CM}}_{db,r}$: average class r ($r = 1, \cdots, R$) service demands at each of the three tiers for the computed model
- $\mathbf{D}^{\text{AS}}$: matrix of service demands for the actual system. The rows correspond to the web server, application server, and database server tier, respectively, and the columns correspond to the classes.
- $\mathbf{D}^{\text{CM}}$: matrix of service demands for the computed system. The rows correspond to the web server, application server, and database server tier, respectively, and the columns correspond to the classes.
- $T_{w,r}, T_{a,r}, T_{db,r}$: average class r ($r = 1, \cdots, R$) response times at each of the three tiers
- T^{AS}_r: average class r ($r = 1, \cdots, R$) response time for the actual system
- T^{CM}_r: average class r ($r = 1, \cdots, R$) response time for the computed model

3 Methodology

The goal of our methodology is to derive service demands at all tiers and all classes, for the Computed Model (CM) using only the inputs (average arrival rates) and outputs (average response times) of the Actual System (AS).

The average response time of the Actual System for class r transactions is a function f^{AS}_r of the vector $\boldsymbol{\lambda} = (\lambda_1, \cdots, \lambda_R)$ of average arrival rates, the vector $\boldsymbol{N} = (N_w, N_a, N_{db})$, and of the matrix of service demands $\mathbf{D}^{\text{AS}}$.

Hence,

$$T^{\text{AS}}_r = f^{\text{AS}}_r(\boldsymbol{\lambda}, \boldsymbol{N}, \mathbf{D}^{\text{AS}}). \tag{1}$$

The values of $T^{\text{AS}}_r, r = 1, \cdots, R$, can be obtained using standard measuring tools.

The class r response times of the computed model are a function f_r^{CM} of the vector $\boldsymbol{\lambda} = (\lambda_1, \cdots, \lambda_R)$ of arrival rates, the vector $\boldsymbol{N} = (N_w, N_a, N_{db})$, and of the matrix of service demands $\mathbf{D}^{\text{CM}}$. Thus,

$$T_r^{\text{CM}} = f_r^{\text{CM}}(\boldsymbol{\lambda}, \boldsymbol{N}, \mathbf{D}^{\text{CM}}). \tag{2}$$

The function f_r^{CM} in Eq. (2) is the function (or algorithm) used to solve a queuing network model given its parameters [12]. The matrix $\mathbf{D}^{\text{CM}}$ is unknown and it is a goal of this work to estimate $\mathbf{D}^{\text{CM}}$ from $\boldsymbol{\lambda}$ and T_r^{AS} in a way that $T_r^{\text{CM}} \approx T_r^{\text{AS}}$, for $r = 1, \cdots, R$, for a wide range of values of $\boldsymbol{\lambda}$.

Before we discuss how we estimate $\mathbf{D}^{\text{CM}}$, we need to provide a formulation for the function f_r^{CM} for the QN of Fig. 3. We use Seidmann's approximation [14] to model the multiple-server queues in this QN.

This approximation replaces a multiple-server queue by a sequence of a delay device and a load-independent queuing device with properly adjusted service demands. Then, the response time of an N-server single queue with service demand equal to D in each server is approximated as

$$T = D\frac{N-1}{N} + \frac{D/N}{1 - \lambda \times D/N}. \tag{3}$$

Therefore, the function f_r^{CM} with input parameters $\boldsymbol{\lambda}, \boldsymbol{N}$, and $\mathbf{D}^{\text{CM}}$ is

$$T_{w,r} = D_{w,r}^{\text{CM}}\frac{N_w - 1}{N_w} + \frac{D_{w,r}^{\text{CM}}/N_w}{1 - \sum_{r=1}^{R} \lambda_r D_{w,r}^{\text{CM}}/N_w} \tag{4}$$

$$T_{a,r} = D_{a,r}^{\text{CM}}\frac{N_a - 1}{N_a} + \frac{D_{a,r}^{\text{CM}}/N_a}{1 - \sum_{r=1}^{R} \lambda_r D_{a,r}^{\text{CM}}/N_a} \tag{5}$$

$$T_{db,r} = D_{db,r}^{\text{CM}}\frac{N_{db} - 1}{N_{db}} + \frac{D_{db,r}^{\text{CM}}/N_{db}}{1 - \sum_{r=1}^{R} \lambda_r D_{db,r}^{\text{CM}}/N_{db}} \tag{6}$$

$$T_r^{\text{CM}} = T_{w,r} + T_{a,r} + T_{db,r} \tag{7}$$

The problem of obtaining $\mathbf{D}^{\text{CM}}$ can be cast as the following non-linear optimization problem.

Find the service demands in $\mathbf{D}^{\text{CM}}$ (i.e., the values of the variables $D_{w,r}^{\text{CM}}, D_{a,r}^{\text{CM}}, D_{db,r}^{\text{CM}} \ \forall \ r$) that minimize MAXDIFF, the maximum value of the absolute differences between the response times of the actual system and that of the computed model.

$$\text{Minimize MAXDIFF} = \max_{r=1}^{R} \mid T_r^{\text{AS}} - T_r^{\text{CM}} \mid \tag{8}$$

subject to the following constraints:

1. $D_{w,r}^{\text{CM}}, D_{a,r}^{\text{CM}}, D_{db,r}^{\text{CM}} \geq 0 \quad r = 1, \cdots, R$
2. $D_{w,r}^{\text{CM}} + D_{a,r}^{\text{CM}} + D_{db,r}^{\text{CM}} \leq T_r^{\text{CM}} \quad r = 1, \cdots, R$

3. $\sum_{r=1}^{R} \lambda_r \frac{D_{w,r}^{\mathrm{CM}}}{N_w} < 1,\ \sum_{r=1}^{R} \lambda_r \frac{D_{a,r}^{\mathrm{CM}}}{N_a} < 1,\ \sum_{r=1}^{R} \lambda_r \frac{D_{db,r}^{\mathrm{CM}}}{N_{db}} < 1 \quad r = 1, \cdots, R$
4. $T_r^{\mathrm{CM}} = T_{w,r} + T_{a,r} + T_{db,r}$ where
   $T_{w,r} = D_{w,r}^{\mathrm{CM}} \frac{N_w - 1}{N_w} + \frac{D_{w,r}^{\mathrm{CM}}/N_w}{1 - \sum_{r=1}^{R} \lambda_r D_{w,r}^{\mathrm{CM}}/N_w}$, $T_{a,r} = D_{a,r}^{\mathrm{CM}} \frac{N_a - 1}{N_a} + \frac{D_{a,r}^{\mathrm{CM}}/N_a}{1 - \sum_{r=1}^{R} \lambda_r D_{a,r}^{\mathrm{CM}}/N_a}$, and
   $T_{db,r} = D_{db,r}^{\mathrm{CM}} \frac{N_{db} - 1}{N_{db}} + \frac{D_{db,r}^{\mathrm{CM}}/N_{db}}{1 - \sum_{r=1}^{R} \lambda_r D_{db,r}^{\mathrm{CM}}/N_{db}}$

The first constraint says that all service demands must be non-negative, the second constraint says that the response time of each class must be at least equal to the sum of all service demands at all tiers for transactions of that class (this is the zero congestion case). The third constraint indicates that the utilization of the web server tier, application tier, and database tier have to be less than 100%. Finally, the fourth constraint provides the function f_r^{CM} used to compute T_r^{CM} as a function of $\boldsymbol{\lambda}, \boldsymbol{N}$, and the service demands in $\mathbf{D}^{\mathrm{CM}}$.

The above discussion assumes that the number of servers per tier is known a priori. However, one can extend this formulation, as done in the experiments, by including N_{ws}, N_a, and N_{db} as decision variables with the following constraints: $N_{ws} \in \mathbb{N}, N_a \in \mathbb{N}$, and $N_{db} \in \mathbb{N}$ (where it is understood that $0 \notin \mathbb{N}$).

This non-linear optimization problem can be solved using available solvers, including Microsoft's Excel Solver Add-in that uses the Generalized Reduced Gradient (GRG2) method or NEOS solvers (www.neos-server.org/neos/solvers/). We used the Excel Solver in the results reported in this paper.

The solution of this optimization problem provides the necessary service demands to solve the QN model given the arrival rates measured in the actual system. As discussed at the outset of the paper, the question of interest is whether the computed model CM has predictive power over a range of arrival rate values. Given a certain threshold ϵ for the maximum percent absolute relative difference (MPARD) between the actual response time and that predicted by the computed model, the process of recomputing the service demands $\mathbf{D}^{\mathrm{CM}}$ should be repeated when the MPARD exceeds the threshold. Therefore, the computed model may have to be re-calibrated when

$$\mathrm{MPARD} = \max_{r=1}^{R} \{ \mid (T_r^{\mathrm{AS}} - T_r^{\mathrm{CM}})/T_r^{\mathrm{AS}} \times 100 \mid \} > \epsilon. \tag{9}$$

4 Experimental Results

The first subsection of this section discusses experiments, called "controlled experiments," in which we use an open QN model as a proxy for the actual system. The following subsection reports on experiments conducted with a real system.

4.1 Controlled Experiments

The controlled experiments use an open QN model, referred to as the Actual Model (AM), as a proxy for an actual system. The parameters of the AM, such as the service demands at all tiers, are known. Note that these known parameters

of the AM are used only for the purpose of solving the AM and obtaining its response time as a proxy for measuring the response time in the actual system. The service demands of the AM are not used in any way to derive parameters for the CM.

The actual system (represented by the Actual Model) is a 3-tier web-based application system with two classes of transactions and includes a load balancer at each tier (See Fig. 3). In our experiments, we test the ability of the CM to predict the response time of the AM (i.e., the proxy for the AS) with varying average arrival rates and number of load-balanced servers per tier. This is a typical configuration in elastic cloud computing environments in which resources are allocated and de-allocated depending on the workload.

Figures 4-5 show results for 3 servers and 5 servers per tier, respectively. The top graph in each figure shows the value of MPARD (see Eq. (9)) versus the scaling factor used to vary average arrival rates. The scaling factor is a multiplier used to scale up by the same factor the arrival rate of both classes. The initial arrival rates are 0.3 tps and 0.4 tps for classes 1 and 2, respectively. The scaling factor varies over a very wide range: up to 180 for the 2-server case, 210 for the 3-server case, and 400 for the 5-server case. For example, a scaling factor of 20 indicates that the arrival rates for classes 1 and 2 are 6 ($= 20 \times 0.3$) tps and 8 ($=20 \times 0.4$) tps, respectively. The threshold ϵ in the experiments is set to 3% (a low value for the threshold). As shown in the MPARD graphs, re-calibration was able to bring the error rate back to zero after it first surpassed the threshold ϵ.

As Figs. 4-5 demonstrate, the number of times that MPARD exceeds the threshold is pretty low despite the wide variation of the scaling factor. For example, the threshold was exceeded twice for the 3-server case and six times for the 5-server case.

The three bottom graphs of each figure show the average response times with increasing scaling factors, which illustrates the ability of the CM to predict the response times of the AM. The figures show two curves for each transaction class; one for AM and another for CM. The curves show that the CM is capable of tracking very closely (i.e., within the 3% threshold) the response times of the AM. The following ranges of the scaling factor exhibited no need for recalibration: (a) 3-server case: 1-149, 160-200; and (b) 5-server case: 1-50, 60-90, 100-130, 140-290, 300-360, 370-390.

Table 1 shows the service demands at each of the three tiers for classes 1 and 2 for the AM and CM models, 5 servers per tier, and a scaling factor of 180. The table also shows the response times for each class for AM and CM. The results are very close even though the computed and actual service demands are significantly different. The value of MAXDIFF is 1.5×10^{-4}.

We also observed that the timing and frequency of model calibration is hard to predict as the number of servers per tier increases because the individual service demands of the actual system and computed model may be different, as shown in Table 1. Therefore, when the curves for AM and CM response times take longer time to diverge, that is an indication that the corresponding service demands for the different servers in AM and CM are accurate. In Figure 4,

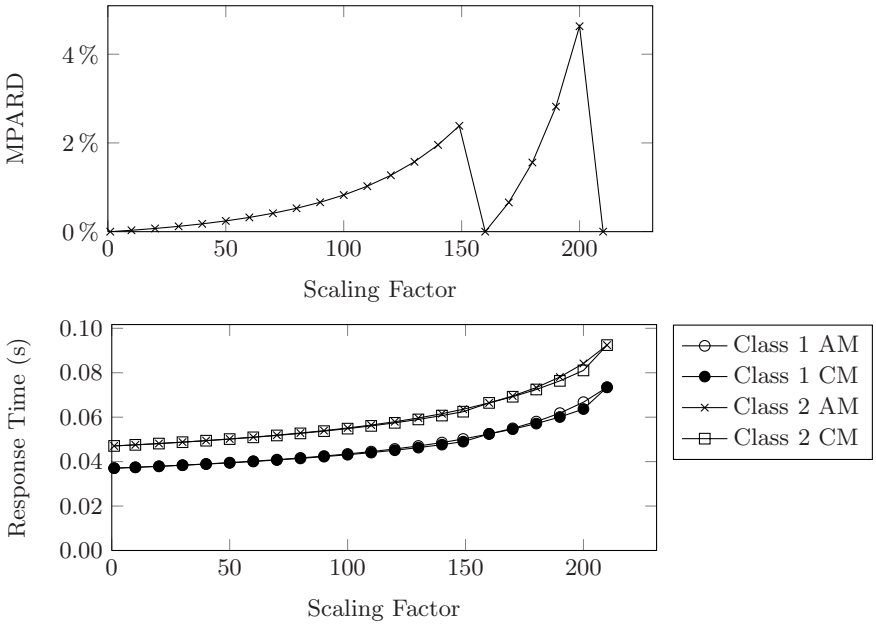

Fig. 4. 3 Servers per Tier. Top: MPARD. Bottom: Response Times for AM and CM for Classes 1 and 2. $\epsilon = 3\%$.

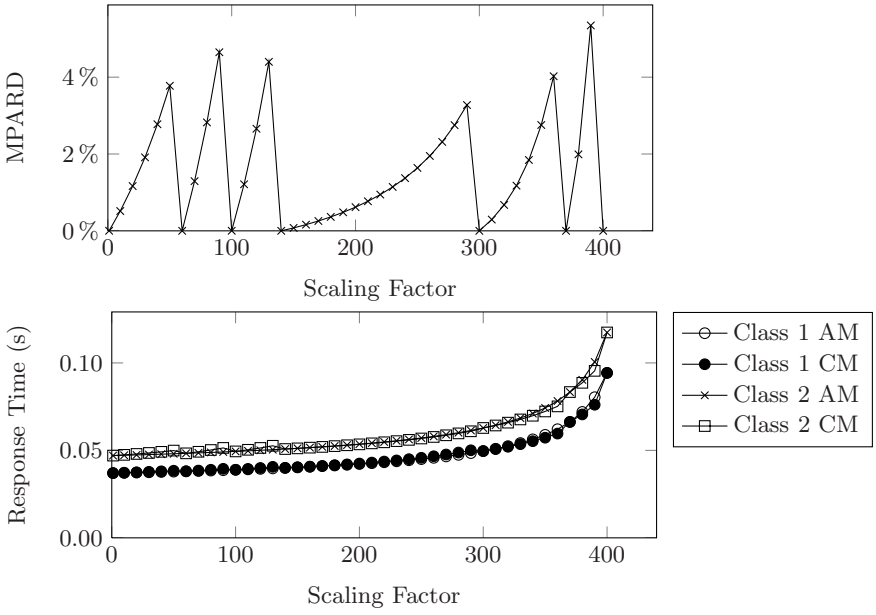

Fig. 5. 5 Servers per Tier. Top: MPARD. Bottom: Response Times for AM and CM for Classes 1 and 2. $\epsilon = 3\%$.

Table 1. Service demands and response times for the AM and CM models for 5 servers per tier and scaling factor equal to 180

	AM		CM	
	Service Demands			
Tier	Class 1	Class 2	Class 1	Class 2
Web	0.010	0.013	0.008	0.016
Application	0.012	0.016	0.021	0.015
DB	0.015	0.018	0.008	0.016
	Response Times			
	0.0414	0.0525	0.0415	0.0524

this occurs when the arrival rates are low, which results in fewer calibrations. When the resources are close to saturation, the frequency of calibration increases though. In Fig. 5, however, the service demands are more accurate right after the fourth calibration, but the frequency of calibration still increases as the system approaches saturation levels.

As noted above, the error threshold ϵ used in the previous experiments was rather low (i.e., 3%). We investigated the impact of increasing ϵ to 5%. Figure 6 shows the variation of MPARD and the average response time for 3 servers per tier and $\epsilon = 5\%$. The corresponding figure for $\epsilon = 3\%$ is Fig. 4. The first recalibration for $\epsilon = 5\%$ occurred for a scaling factor of 200 while for $\epsilon = 3\%$ it occurred much earlier, at a scaling factor of 160.

4.2 Experiments with an Actual System

The controlled experiments helped validate the proposed methodology by varying number of servers per tier and the calibration threshold ϵ while monitoring the response time of the CM compared to the AM representing the actual system. To validate the proposed methodology on a real system we used the Apache OFBiz ERP system and Apache JMeter to generate various workloads. OFBiz was installed on two load-balanced Apache Tomcat servers with two load-balanced MySQL database servers, and a single Apache web server receiving JMeter requests and routing these requests to the load-balanced OFBiz Tomcat servers.

Figure 7 shows the results when the number of servers per tier is static (1 web server, 2 OFBiz Tomcat application servers and 2 MySQL database servers) and the recalibration threshold is set to 25%. In this case, the CM closely predicted the OFBiz response time within the recalibration threshold, and only needed six model calibrations before system resource saturation. Similar to the controlled experiments, the variation in response time between the actual system and the computed model tends to diverge faster as system resources are close to saturation.

Figure 8 shows the results when the number of servers per tier is variable. This test case investigates the ability of our methodology to infer more information about the system architecture components by predicting the optimal number of

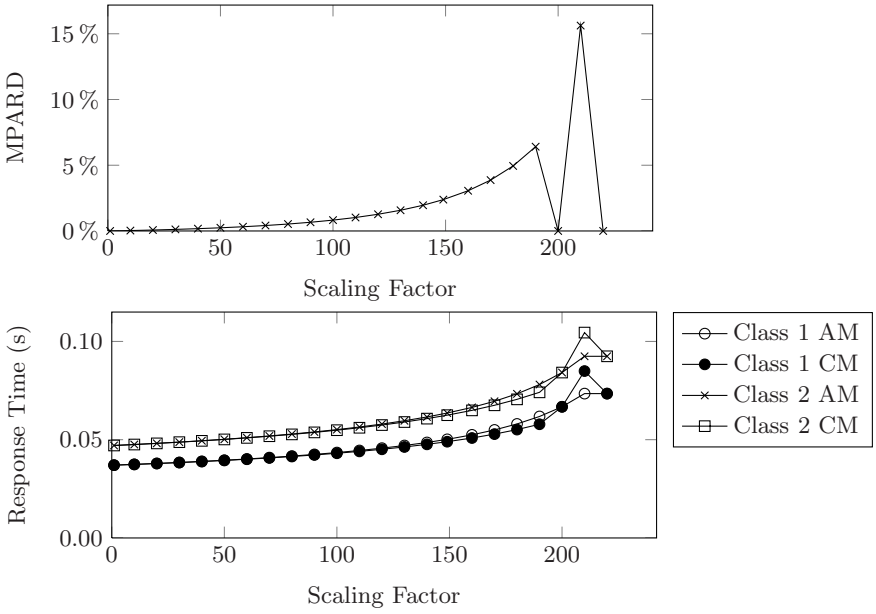

Fig. 6. 3 Servers per Tier. Top: MPARD. Bottom: Response Times for AM and CM for Classes 1 and 2. $\epsilon = 5\%$.

servers per tier that should be used in the CM to accurately represent the actual system. In this experiment, the optimizer was used to perform six model calibrations for the CM, and predicted the following number of web servers, application servers and database servers for each of the six calibrations, respectively: (1,1,2), (1,2,3), (1,2,3), (2,2,2), (2,2,2), (1,2,2), (1,1,2).

5 Related Work

Much of the related work in the fields of performance engineering and capacity planning is focused on dynamic resource allocation when a performance model is fully or partially known a priori. In this paper, we treat the system and its components as black boxes and we try to establish a relationship between system input and output in order to estimate and parameterize an analytical model that closely approximates the behavior of the system.

Some of the prior work that tackled the parameterization of analytical models includes [2,3,4], 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 couple 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.

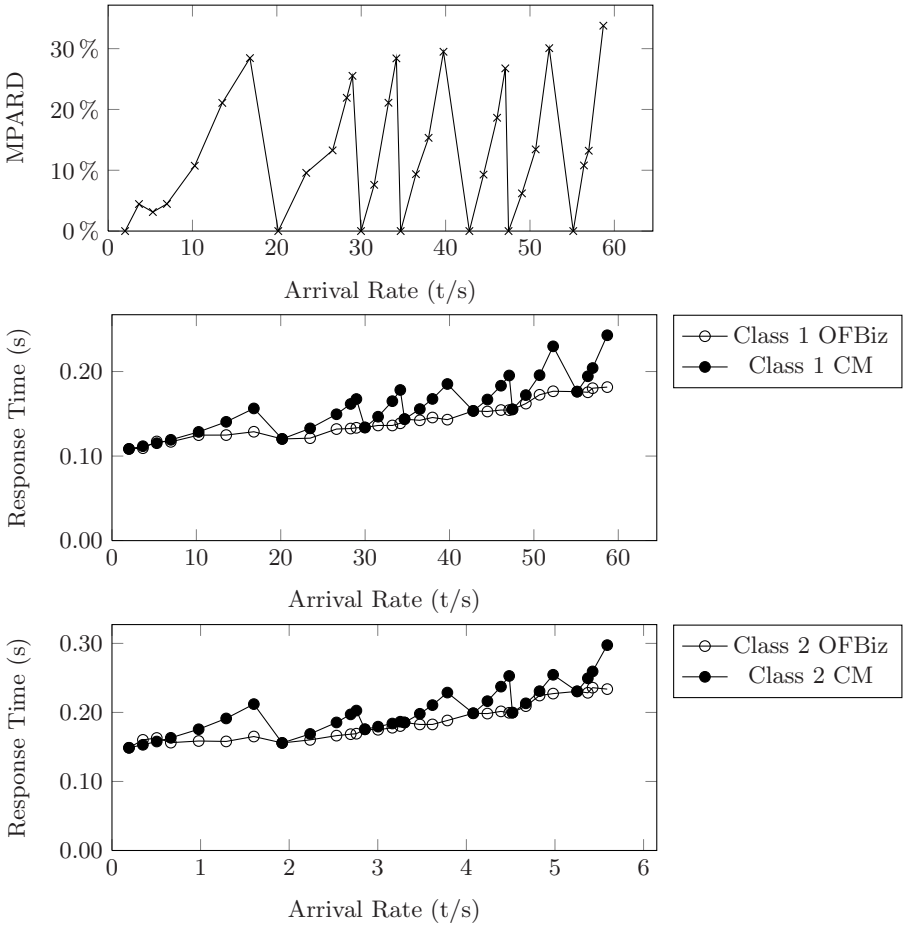

Fig. 7. Static N - Top: MPARD vs. Arrival Rate. Middle: Class 1 Transactions. Bottom: Class 2 Transactions. $\epsilon = 25\%$.

Menascé tackled the issue of model parameterization when some input parameters are already known [11]. 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 presented a survey of performance modeling approaches focusing mainly on business information systems. The authors described the general activities involved in workload characterization, especially estimating service demands, and the various methods and approaches used to estimate it. Some of these methods include general optimization techniques, linear regression, and Kalman filters.

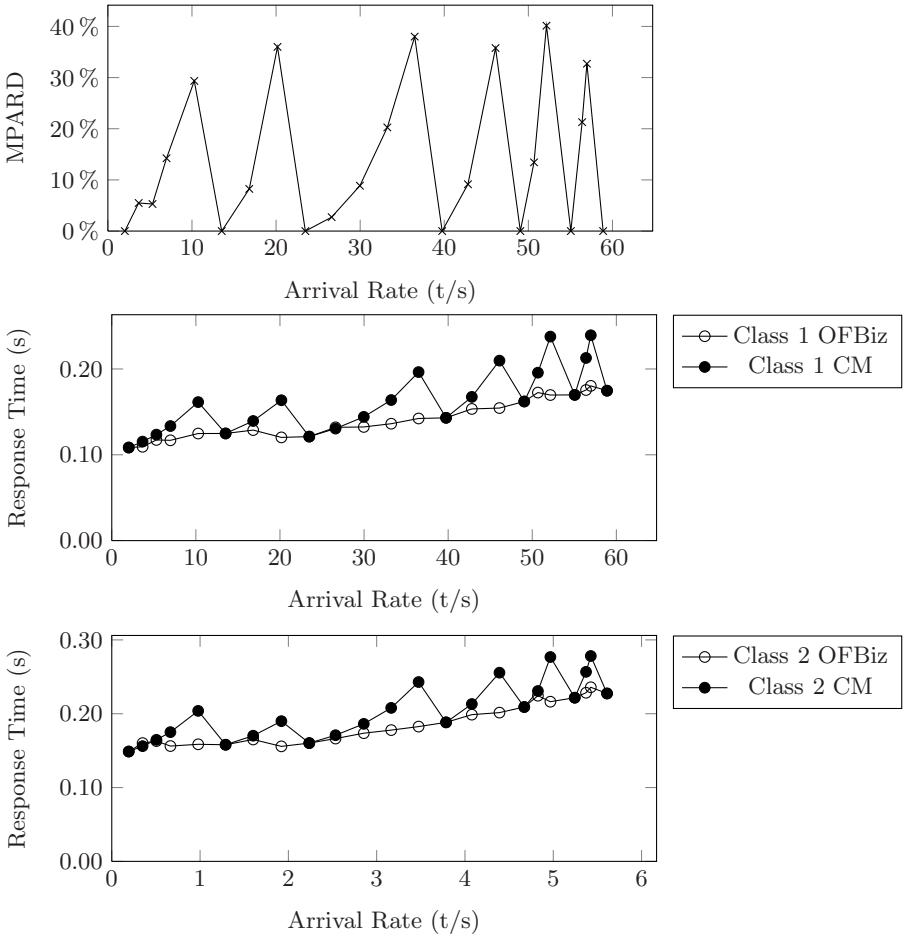

Fig. 8. Variable N - Top: MPARD vs. Arrival Rate. Middle: Class 1 Transactions. Bottom: Class 2 Transactions. $\epsilon = 25\%$.

In [7], the authors presented a method for extracting architecture level performance models in distributed component-based systems using tracing information and instrumentation to infer system components, their connections and the (probabilistic) dependency of their parameters. In contrast, our approach does not require such knowledge of system components or their relationships.

In [13], the authors presented an iterative methodology for building performance models in virtualized environments with a focus on the I/O function of storage systems. The method implemented in that paper focused on the storage component of a particular IBM mainframe system. Our approach addresses a higher level of system abstraction where the internal structure of individual servers, such as the storage system, is unknown.

In [8], the authors presented Modellus; a system for automated web-based application modeling that uses workload characterization, data mining and machine learning techniques. Our focus in this paper is on gray box modeling, where detailed web server logs and database logs may not be available.

The work in [1] 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 [10,15,16] 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 the current input and output values are know and where the decision to recalibrate only depends on the level of divergence between observed measurements and model estimated measurements.

6 Conclusions

Building an analytical performance model is a challenge when little is known about the functionality and behavior of the system being modeled and/or when obtaining model parameters through measurements is difficult. This paper addresses this problem by presenting an approach that derives analytic model parameters by observing the input-output relationships of a real system. This model, called the computed model (CM), provides the same output values for the same input values used to derive the CM. The CM is obtained by solving a non-linear optimization problem. The results showed that the CM is quite robust and has predictive power over a wide range of input values. For example, as the arrival rate of transactions for both classes was scaled by a factor varying from 1 to 400, the response times predicted by the CM only exceed the 3% error threshold six times. When the error threshold was increased to 5% (still a very low value), only two re-calibrations were needed. The ability of the CM to model the number of servers per tier of a multi-tier system is of a particular interest since it proves the ability of the CM to model system components previously unknown to it by knowing only the input and output parameters of a real information system, such as Apache OFBiz.

References

1. Barna, C., Litoiu, M., Ghanbari, H.: Autonomic load-testing framework. In: Proc. 8th ACM Intl. Conf. Autonomic Computing, pp. 91–100 (2011)
2. Begin, T., Baynat, B., Sourd, F., Brandwajn, A.: A DFO technique to calibrate queuing models. Computers & Operations Research 37(2), 273–281 (2010)

3. Begin, T., Brandwajn, A., Baynat, B., Wolfinger, B.E., Fdida, S.: Towards an automatic modeling tool for observed system behavior. In: Wolter, K. (ed.) EPEW 2007. LNCS, vol. 4748, pp. 200–212. Springer, Heidelberg (2007)
4. 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)
5. Bennani, M.N., Menascé, D.A.: Assessing the robustness of self-managing computer systems under highly variable workloads. In: Intl. Conf. Autonomic Computing, pp. 62–69 (2004)
6. Bennani, M.N., Menascé, D.A.: Resource Allocation for Autonomic Data Centers Using Analytic Performance Models. In: 2005 IEEE Intl. Conf. Autonomic Computing, Seattle, WA, June 13-16 (2005)
7. Brosig, F., Huber, N., Kounev, S.: Automated extraction of architecture-level performance models of distributed component-based systems. In: 26th IEEE/ACM Intl. Conf. Automated Software Engineering (ASE), pp. 183–192 (2011)
8. Desnoyers, P., Wood, T., Shenoy, P., Singh, R., Patil, S., Vin, H.: Modellus: Automated modeling of complex internet data center applications. ACM Tr. on the Web (TWEB) 6(2) (2012)
9. Kounev, S., Huber, N., Spinner, S., Brosig, F.: Model-based techniques for performance engineering of business information systems. In: Shishkov, B. (ed.) BMSD 2011. LNBIP, vol. 109, pp. 19–37. Springer, Heidelberg (2012)
10. Litoiu, M., Woodside, M., Zheng, T.: Hierarchical model-based autonomic control of software systems. ACM SIGSOFT Software Engineering Notes 30(4), 1–7 (2005)
11. Menascé, D.: Computing missing service demand parameters for performance models. In: Proc. 34th Intl. Computer Measurement Group Conf., pp. 7–12 (2008)
12. Menascé, D., Dowdy, L., Almeida, V.: Performance by Design: Computer Capacity Planning By Example. Prentice Hall (2004)
13. Noorshams, Q., Rostami, K., Kounev, S., Tuma, P., Reussner, R.: I/O Performance Modeling of Virtualized Storage Systems. In: IEEE 21st Intl. Symp. Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 121–130 (2013)
14. Seidmann, A., Schweitzer, P., Shalev-Oren, S.: Computerized Closed Queueing Network Models of Flexible Manufacturing, Large Scale System. J. North Holland 12, 91–107 (1987)
15. Woodside, M., Zheng, T., Litoiu, M.: The use of optimal filters to track parameters of performance models. In: Second Intnl. Conf. Quantitative Evaluation of Systems, pp. 74–83 (2005)
16. Zheng, T., Yang, J., Woodside, M., Litoiu, M., Iszlai, G.: Tracking time-varying parameters in software systems with extended Kalman filters. In: Proc. 2005 Conf. of the Centre for Advanced Studies on Collaborative Research, pp. 334–345 (2005)