Proc. 2014 Computer Measurement Group Conference, Atlanta, GA, Nov. 3-6, 2014.

Dynamic Derivation of Analytical Performance Models in Autonomic Computing Environments

Mahmoud Awad Daniel A. Menascé
Dept. of Computer Science, MS 4A5
Volgenau School of Engineering
George Mason University
Fairfax, VA 22030, USA
mawad1@gmu.edu menasce@gmu.edu

Abstract

Deriving analytical performance models requires intimate knowledge of the architecture and behavior of the computer system being modeled. In autonomic computing environments, this detailed knowledge may not be readily available (or it may be impractical to gather) given the dynamic nature of these environments. In this paper, we present a framework for dynamically deriving and parameterizing performance models in autonomic systems. Performance models are derived and parameterized by observing the relationships between a real system's input and output parameters (average arrival rates and response times for each job class). The paper shows the results of implementing our approach using Apache OFBiz^{TM} and highlights the predictive power of the derived model.

1 Motivation and Introduction

Computer systems are becoming increasingly complex in terms of hardware and software. As a result of this complexity, it is becoming difficult and costly to manually configure, maintain and protect computer systems, especially in large data centers. For this reason, hardware vendors, software vendors and large hosting providers are incorporating into their products or hosting environments autonomic computing capabilities, where systems become self-configuring, self-healing, self-optimizing and self-protecting [13, 19]. For example, Oracle corporation has implemented self-managing, self-optimizing and self-protecting features into its Database Management System (DBMS) as far back as version 10g, which was released in 2003. Also, public cloud hosting providers, such as Amazon Web Services (AWS), offer elastic storage and elastic load balancing features to dynamically respond to the changing storage and fault tolerance needs of their customers.

Autonomic capabilities in computer systems are implemented using a controller that monitors performance fluctuations and security threats, and responds by performing automatic adjustments to the system's configuration parameters and security controls. Autonomic controllers can be designed using a model-free approach, such as machine learning techniques, or a model-based approach using queuing network modeling techniques (see section 2).

Autonomic controllers that use a closed-loop feedback mechanism along with analytical models or learning techniques tend to produce more efficient resource allocation plans than reactive techniques because of their ability to predict future resource allocation needs [5, 6, 12, 19]. For example, most cloud providers rely solely on reactive, generic and high-level system performance monitors, such as CPU utilization levels, to determine when and how to adjust system resources. In some cases, the adjustments are overly conservative, which is wasteful. In other cases, where the application system is highly dynamic, resource adjustments occur more frequently and in piecewise fashion, which may disrupt application system operations during peak usage.

This paper focuses on the use of model-based analytical performance models in autonomic controllers. One of the main barriers to the adoption and wide-spread use of analytical performance models is that analytical modeling of application systems requires a thorough understanding of the system architecture, software architecture, application behavior, resource usage, and workload characteristics. This knowledge is then provided to highly skilled performance engineers who use specialized modeling techniques to develop analytical performance models tailored to a particular computer system.

Although this knowledge may be available to performance engineers in dedicated data centers, it is not always

readily available in shared public and private hosting environments. For example, public cloud providers offer online tools that allow customers to provision hardware and software resources as needed without any direct interaction with system administrators or performance engineers. The speed and frequency with which customers are able to provision resources and deploy application systems make it virtually impossible to develop performance models tailored to individual applications or even to common classes of applications.

Even when it is possible to develop analytical performance models, there are cases when such models need to be adjusted dynamically to meet the Quality of Service (QoS) requirements of today's dynamic application systems. For example, initial QoS requirements are usually provided based on basic assumptions regarding number of concurrent end-users, throughput and possible spikes in transaction arrival rates throughout the year, such as holiday shopping seasons for e-commerce sites. So, the variability aspects inherent in some computer systems and hosting environments are not fully available or not fully understood when Service Level Agreements (SLA) are signed. Some of these cases include:

1. Unexpected spikes in demand for on-line e-commerce sites, for example, when a new product is released unexpectedly or when a long-awaited (or heavily advertised) feature of the system becomes available.

2. When the data center offers additional hardware or software resources that would positively impact the original deployment configuration of the system without affecting the intended functionality. For example, adding more load balancers or offering new parallel processing capabilities for the middleware or database.

3. When the classification (characterization) of the hosted application system-on which the original performance model was based-is changed. For example, changing from an open queuing model to a mixed model, changing from a single-class transaction system to multi-class, or adding parallel capabilities to the application itself that would require changes in the system's hardware needs, and consequently, changes in the original performance model.

Therefore, the assumptions on which performance models are originally developed may change over time, which may result in inefficient resource allocation. This, in turn, impacts both the data center's and the system owners ability to deliver the agreed-upon SLAs. For this reason, shared data centers need the ability to dynamically modify performance model parameters or select alternative performance models based on changes in the system architecture, software architecture, or major pattern changes in workload intensity.

This paper begins by presenting a framework for the dynamic derivation of analytical performance models in autonomic computing environments. The framework includes a multi-layered approach for workload analysis and performance model generation and parameterization. The approach takes into consideration the availability of system monitoring logs and the balance between producing costly but accurate performance models versus sufficient levels of model accuracy even when detailed system monitoring logs are available. The paper then focuses on one level of this framework, namely the "gray box" level, and presents an approach for deriving Queuing Network (QN) models where input parameters (e.g., arrival rates) and output parameters (e.g., average response time) are known. In addition, the approach utilizes some basic information about the system architecture, most of which can be easily obtained using high-level monitoring tools readily available in cloud environments, such as the number and function of each layer in a multi-tiered system (but not necessarily the number of servers of each tier). We then derive and parameterize a QN model, which we call the Computed Model (CM), that closely approximates the actual computer system.

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 the 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, not shown in this paper, showed that our approach is capable of producing computed QN models that have a robust predictive power. We then conducted experiments, shown in this paper, using Apache's Open For Business (OFBiz^{TM}) ERP system; one in which the number of servers per tier is assumed to be 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^{TM} experiments show results consistent with the controlled experiments.

The rest of the paper is organized as follows. Section 2 includes background on autonomic systems and performance modeling. Section 3 describes the autonomic framework for dynamically deriving analytical models, and the multi-layered approach for determining the appropriate level of performance analytical modeling needed. Section 4 describes the technique used for deriving and parameterizing the CM in the "gray box" level and shows experimental results. Section 5 discusses some related work. Finally, section 6 presents discussions and concluding remarks.

2 Background

This section provides background on autonomic computing systems and performance modeling using QNs.

2.1 Autonomic Systems

Autonomic computing systems implement self-management capabilities using a closed-control loop mechanism where internal and external sensors are deployed to monitor the controlled system and its environment [5, 6, 13, 19]. Data collected from the sensors is analyzed to determine whether or not a change in the system is needed. If change is needed, the autonomic system uses effectors to implement the change based on policies managed by planning components and produced with the help of knowledge management components. This is referred to as the autonomic manager control loop or MAPE-K loop, which stands for Monitor, Analyze, Plan, and Execute based on Knowledge.

Autonomic capabilities, such as self-configuration and self-optimization are provided by a controller. The controller receives input from sensors that monitor performance variables, determines the best configuration parameters to achieve optimal or near-optimal performance and instructs actuators to execute system resource adaptation plans.

2.2 Design of Autonomic Controllers

The main approaches used to design autonomic controllers are control theory, artificial intelligence and queuing networks, and each approach has advantages and limitations.

Control Theory uses a closed-loop mechanism to monitor, plan and adapt systems. However, this model is more applicable to physical systems that interact with their surrounding environments, such as engineering and mechanical systems. Control theory models do not intuitively apply to controlling software and hardware resources in computer system, such as CPU and software threads. Control theory is more applicable to linear systems.

Artificial Intelligence (AI) includes a number of disciplines, such as reinforcement learning, that can be used to develop planning and adaptation techniques as part of a closed-loop mechanism. Planning and adaptation techniques can be developed using machine learning relying on minimal input; for example the system's input and output parameters. However, AI requires constant learning as opposed to other modeling approaches where exact or approximate models of the system can be developed once and used for dynamic adaptation.

Queuing Networks (QN) models computer systems as a network of queues. Analytical QN models are simple and more intuitive than Control Theory and AI models, and work well with autonomic controllers. However, QN models require *a priori* knowledge of the hardware and software configuration as well as of user behavior and patterns of interaction between the users and the system.

The main obstacle to using queuing networks is the *a priori* knowledge required of the system characteristics and software behavior. This problem can be solved by automating the process of building analytical models to be used by autonomic controllers.

2.3 Queuing Network Modeling

Queuing network modeling is a modeling approach in which computer systems are modeled as a network of queues [7, 8, 9, 20]. Each queue represents customers (for example, end-users or data processing requests) and devices (for example, CPU, disk). QN models have two types of parameters: (1) workload intensity, which can be measured by the number of requests in the system (i.e., concurrency level) or by the average arrival rate of requests; and (2) service demands, which are the average total time needed to service a request of a particular class for each of the system devices. Queuing time is not included in the service demand.

Complex QNs represent multiple queues, where customers may join another queue after being served at a previous queue within the network of queues.

2.4 Workload Characterization

Open queuing networks are used to model systems with unbounded request arrival rates and unbounded numbers of requests inside the system. The distribution of the time between successive arrivals is used to characterize the workload, for example, exponential, hyper-exponential and Erlang distributions. Closed queuing networks, on the other hand, are used to model systems with a known number of requests, such as batch processing systems. When dealing with independent transactions, we have to take into account think time. For example, requests submitted to Google search are considered independent of each other, while session-based transactions, such as browsing and purchasing products on an e-business site, such as Amazon, are not independent. Statistical analysis is used to determine the best-fit data distribution for arrival rates.

Another important aspect of system workload characterization is whether or not requests have similar average resource consumption times. Workloads that receive transaction requests with similar CPU and I/O resource needs can be grouped into a single class. Workloads can be categorized as single-class or multi-class using data clustering methods.

Data Clustering is a method by which observed data is divided into homogeneous groups (clusters) in order to minimize variability within the observed data. The resulting groups of observations would tend to exhibit similar characteristics, and consequently, can be classified according to those common characteristics in multi-class queuing networks.

Another useful technique for characterizing workload is Customer Behavior Model Graph (CBMG) [17, 18], which is used to perform clustering at the user session level. This

clustering is then used to produce the average number of visits to each application module, and consequently, the service demands per application module.

3 Framework

We begin by presenting our proposed framework for dynamically inferring analytical performance models within the context of the MAPE-K loop of autonomic systems. The framework consists of the following components as illustrated in figure 1

1. **Monitoring**: In this step, software probes or agents are deployed to monitor performance at different levels of the system architecture. Performance-related data is collected from networks and operating systems as well as application and server logs. Examples include web server logs and database access logs.

2. **Analysis**: The Workload Model Analyzer tries to glean and extract possible performance model input parameters, such as arrival rates, various response times and other metrics by parsing server logs and hardware resource utilization logs.

   (a) Statistical Analysis is performed in order to determine the best-fit data distribution for arrival rates and service times. This would narrow down the possible analytical models to choose from before performing any data clustering or application behavior analysis. For example, exponentially distributed inter-arrival times would indicate the possibility of a single M/M/1 or M/G/1 queuing model. Similarly, discovering multi-core processors, multi-thread software processes or multiple hard disk mount points would favor an M/M/c or M/G/c model.

   (b) Data Clustering is a method by which observed data is divided into relatively homogeneous groups (clusters) in order to minimize variability within the observed data [1]. The resulting groups of observations would tend to exhibit similar characteristics, and consequently can be classified according to those common characteristics in multi-class queuing networks.

   (c) Application Behavior Analysis is used to draw certain conclusions regarding user behavior as well as application behavior. An example of the latter is determining whether the application system exhibits the characteristics of an open or a closed queuing network. An application system whose logs show the same processes running concurrently according to a preset schedule would indicate a closed queuing network. On the other hand, application server logs that show multiple on-line users using random IP addresses accessing various web pages in a certain order would indicate an open queuing network. Application Behavior Analysis can also be used to develop a Customer Behavior Model Graph (CBMG), which is used to perform clustering at the user session level [17]. This clustering is then used to project the average number of visits to each application module, and consequently, the service demands per application module.

3. **Analysis**: The Performance Model Analyzer is an extension of the MAPE-K loop Analysis phase, which focuses on finding the best-fit analytical performance model given the results produced by the Workload Model Analyzer. The Performance Model Analyzer includes a comprehensive repository of performance models and their characteristics. This Analyzer's job is to compare the observed features of the application system with the available performance models in the repository. It will then associate the best-fit analytical performance model with the corresponding application model, and pass on that information to the Adaptation Planner. The Performance Model Analyzer carries out the following steps:

   (a) Model Definition: Gather statistical analysis results from the Workload Model Analyzer and decide whether the system should be modeled as single or multiple queues (QN), single-class versus multi-class performance model, or single request-reply vs. session-based workload.

   (b) Model Parameterization: as described later in section 4.

   (c) Model Validation: Run the selected performance model using different parameters and compare results with application log data sets captured at different times, as described later in section 4.

4. **Planning**: The Adaptation Planner uses the selected performance model and its input parameters to produce adaptation plans in order to maintain compliance with SLAs or respond to future resource needs.

5. **Execution**: The Effector's job is to implement the adaptation plans by scaling up or down hardware or software resources as needed. Additionally, the Workload Model Analyzer should have the ability to determine within a certain threshold that changes to the application environment or application behavior (observed in the sensor data) would call for another round of statistical analysis, data clustering and application behavior analysis.

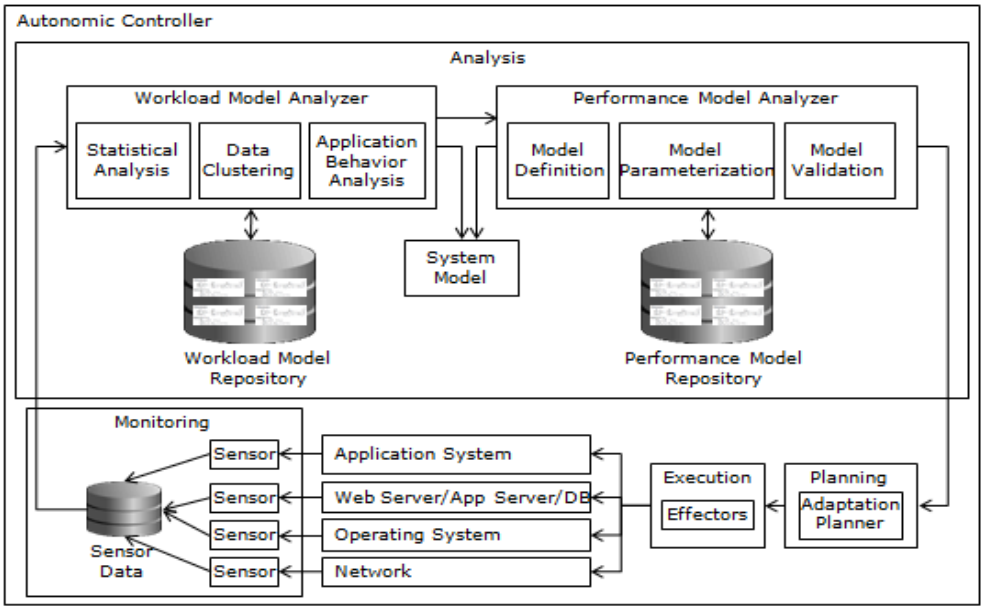

Figure 1: Automatic Model Identification Framework

When implementing the Workload Model Analyzer and Performance Model Analyzer, it is important to determine the sufficient level of abstraction needed when producing the analytical model. For example, when detailed system and application logs are unavailable to the analyzers, this indicates the need for a higher level of system abstraction and, consequently, a much less accurate performance model representing the real system. Our multi-layered approach for workload analysis and model generation takes into account the availability of system monitoring logs and the level of accuracy needed for the system being modeled (see Figure 2.)

At the highest level, systems are viewed as black boxes, where only input and output parameters are known. Model accuracy is verified by comparing the results of the Computed Model (CM) with the actual system. If the error level stays below a certain tolerance value, it means the model is accurate enough and we have reached the optimal level of granularity given the agreed upon QoS or the system complexity and its resource consumption levels.

At the next level of granularity (abstraction), systems are viewed as "gray boxes", where some limited information is available regarding the system architecture or hardware configuration. For example, we may know the number of tiers in a multi-tier architecture and even the number of servers per tier. Lower granularity levels incorporate inferred knowledge about application behavior, user behavior and software architecture from application and system logs.

Our proposed approach is driven by the fact that developing accurate performance models is costly and time-consuming. Even automated performance model generation using QN models, simulation techniques or Artificial Intelligence learning techniques takes a considerable amount of time and computing resources. Therefore, the availability of detailed system architecture and design and usage logs may not justify the time and cost associated with developing a more detailed performance model when a higher level of granularity is adequate.

In the next section, we tackle the gray box level, where input and output parameters (namely arrival rates and response times) are known, but where minimal information about the system architecture is available. The performance model parameters are dynamically adjusted using a QN parameter estimator and the model is recalibrated when its response time deviates from the response time of the real system beyond a certain tolerance level.

4 Methodology

As discussed before, the black box approach for analytical modeling assumes no knowledge of the internal system architecture or application behavior, and therefore, tends to be modeled using a single queue, such as M/M/1, M/G/1, or G/G/1 (approximation only) depending on the distribution of inter-arrival times and the distribution of service times. On the other end of the spectrum, a white box approach is generally used by performance engineers with detailed knowledge of internal system architecture and behavior and detailed logs of response times and queuing delays for the various hardware and software components.

Although the detailed knowledge needed for the white box modeling approach may not be readily available, modern computer systems offer basic monitoring tools capable of tracking input parameters and output parameters, such

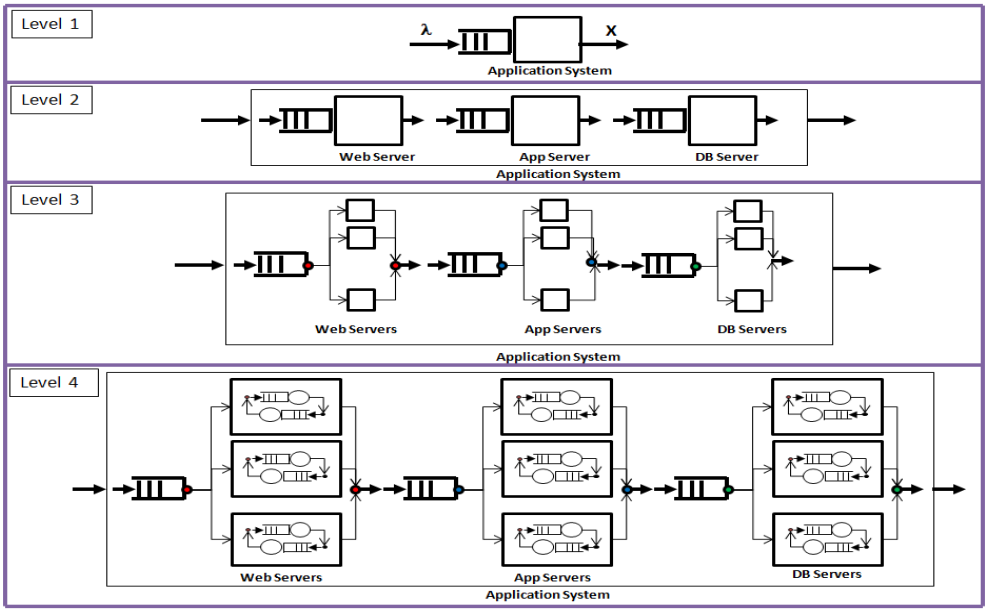

Figure 2: Multi-Layered Approach

as transaction arrival and departure rates as well as the time taken to process each transaction. These tools are currently available in most web servers, operating systems, and virtualization software.

Figure 3 demonstrates the problem we address in this section. In order to derive an analytical model that approximates the structure and behavior of the real system (the actual system in Fig. 3) , we treat the system as a"gray box," where some information about the internal structure of the system is known, such as number of tiers in a multi-tiered system, along with input and output parameters. The QN Parameter Estimator takes average arrival rates λ (input) and average response times of the Actual System (AS) T^{AS} (output) and establishes a relationship between them in order to estimate the parameters of the Computed QN Model (CM). We formulate the relationship between the input and output parameters as a non-linear optimization problem that can be solved using a non-linear solver.

After parameterizing the Computed QN Model, 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. This is important if the model is to produce accurate performance prediction. This comparison process is depicted in Figure 4, 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 accuracy of the response time of the CM compared to the observed response time of the actual system is the issue we address in this section.

In our experiments, we focus primarily on a typical on-

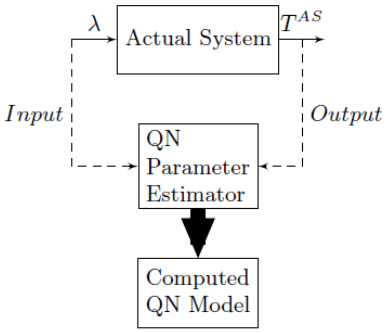

Figure 3: Parameterizing the Computed QN Model

line e-commerce multi-tier application systems, where transactions are processed by one of several servers at each tier and passed on to the next tier. However, our approach can be applied to a wide variety of architectures and application systems.

Figure 5 shows a sample computed QN Model that can be used to represent an actual system with a 3-tier architecture. We use this architecture to demonstrate our approach and experimental results. The number of tiers is used in our approximation formulas to derive the arrival and departure rates of each server within each tier and for determining the level of parallelism in the CM. However, our approach applies to any number of tiers since this number can be passed as a parameter to the performance model parameter estimator.

In this architecture, transactions are submitted to the web server tier, which may consist of a number of servers

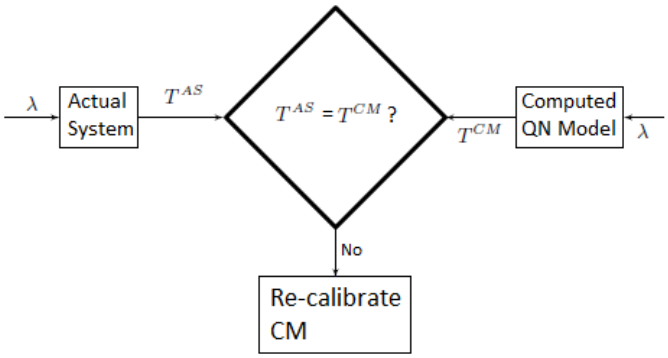

Figure 4: Maintaining the Accuracy of the Computed QN Model

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.

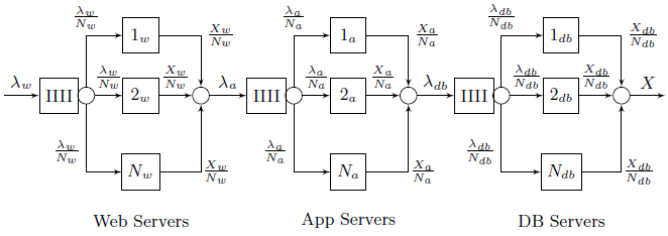

Figure 5: Computed QN Model Topology in a 3-tier Architecture

We assume that we know the number of tiers, but we assume that 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. We use Seidmann's approximation [23] 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, as shown in Figure 6. Instead of using Seidmann's approximation, one could use a queue with load-dependent service rates. However, this would make the optimization model less tractable.

Before we define the problem of obtaining the service demands for the CM, some definitions are in order. Let $D^{\mathrm{CM}}_{w,r}, D^{\mathrm{CM}}_{a,r}, D^{\mathrm{CM}}_{db,r}\ \forall\ r$ be the service demands at the CM at the web server, application server, and database server tiers, respectively, for all classes r. These service demands are stored in the matrix $\mathbf{D}^{\mathrm{CM}}$ where the columns correspond to the classes and the rows to the web server, application

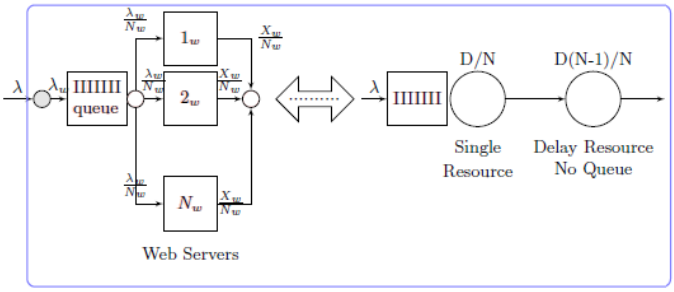

Figure 6: Seidmann's Approximation

server, and database server. Let $\vec{N} = (N_w, N_a, N_{db})$ be the vector that specifies the number of web servers, application servers, and database servers, respectively. Also, let $\vec{\lambda} = (\lambda_1, \cdots, \lambda_R)$ be the vector of arrival rates for classes $1, \cdots, R$.

The optimization problem is then: find the service demands in $\mathbf{D}^{\mathrm{CM}}$ that minimize MAXDIFF, defined as the maximum value of the absolute differences between the response times of the actual system and that of the computed model.

The optimization problem is subject to the following intuitive constraints:

1. All service demands must be non-negative
2. 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).
3. The utilization of the web server tier, application tier, and database tier have to be less than 100%.
4. The function used to compute T^{CM}_r as a function of $\vec{\lambda}, \vec{N}$, and the service demands in $\mathbf{D}^{\mathrm{CM}}$ is:

$$T^{\mathrm{CM}}_r = T_{w,r} + T_{a,r} + T_{db,r} \text{ where}$$
$$T_{w,r} = D^{\mathrm{CM}}_{w,r}\frac{N_w - 1}{N_w} + \frac{D^{\mathrm{CM}}_{w,r}/N_w}{1 - \sum_{r=1}^{R}\lambda_r D^{\mathrm{CM}}_{w,r}/N_w},$$
$$T_{a,r} = D^{\mathrm{CM}}_{a,r}\frac{N_a - 1}{N_a} + \frac{D^{\mathrm{CM}}_{a,r}/N_a}{1 - \sum_{r=1}^{R}\lambda_r D^{\mathrm{CM}}_{a,r}/N_a}, \text{ and}$$
$$T_{db,r} = D^{\mathrm{CM}}_{db,r}\frac{N_{db} - 1}{N_{db}} + \frac{D^{\mathrm{CM}}_{db,r}/N_{db}}{1 - \sum_{r=1}^{R}\lambda_r D^{\mathrm{CM}}_{db,r}/N_{db}}$$

This non-linear optimization problem can be solved using any available non-linear solver. We used Microsoft's Excel Solver Add-in, which uses the Generalized Reduced Gradient (GRG2) optimization method.

Solving this optimization problem provides the necessary service demands needed to solve the QN model given the arrival rates measured in the actual system. 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

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

The computed model may have to be re-calibrated according to the continuous process below.

Step 0 - Initialization: $Starting \leftarrow True$

Step 1: Collect arrival rate ($\vec{\lambda}$) and response time (T_r^{AM}) measurements from the real system.

Step 2: If the arrival rates have not changed significantly since the last measurements were taken go to Step 1.

Step 4: If $Starting$, then run the optimizer to solve the optimization problem described above and obtain $\vec{D}^{\mathrm{CM}}$; $Starting \leftarrow False$

Step 5: Run the CM and compute response times using the computed model: $T_r^{\mathrm{CM}} = f_r(\vec{\lambda}, \vec{N}, \vec{D}^{\mathrm{CM}})$ for $r = 1, \cdots, R$;

Step 6: If $\mathrm{MPARD} = \max_{r=1}^{R}\{|\ (T_r^{\mathrm{AM}} - T_r^{\mathrm{CM}})/T_r^{\mathrm{AM}} \times 100\ |\} > \epsilon$, run the optimizer to solve the optimization problem described above and obtain $\vec{D}^{\mathrm{CM}}$;

Step 7: Go to Step 1

To validate the proposed approach on a real system we used the Apache OFBiz^{TM} ERP system, which is an open source enterprise automation software from Apache and includes an ERP (Enterprise Resource Planning) application, a CRM (Customer Relationship Management) application, an E-Commerce application and an SCM (Supply Chain Management) application among many other business applications. We also used Apache JMeter^{TM} to generate various workloads. OFBiz^{TM} 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^{TM} Tomcat servers, as shown in Figure 7.

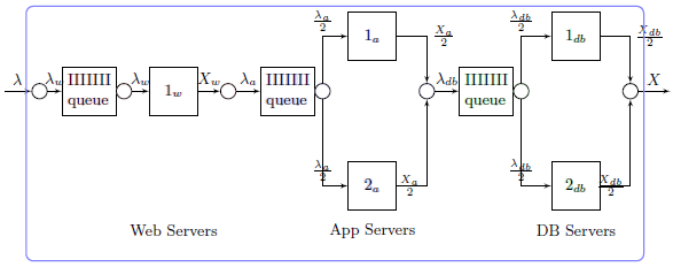

Figure 7: OFBiz^{TM} Experimental Environment

Figure 8 shows the results obtained when the number of servers per tier is static (1 web server, 2 OFBiz^{TM} 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^{TM} response time within the recalibration threshold, and only needed six model calibrations before system resource saturation.The variation in response time between the actual system and the computed model tends to diverge faster as system resources are close to saturation, as expected.

Figure 9 shows the results obtained when the number of servers per tier is variable, i.e., it is assumed to be unknown. This test case investigates the ability of our approach to infer more information about the system architecture components by predicting the optimal number of 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

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.

Menascé tackled the issue of model parameterization when some input parameters are already known [21]. The author proposed a closed-form solution for 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, he did not propose a solution when none of the service demands are known a priori.

In [14], 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.

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

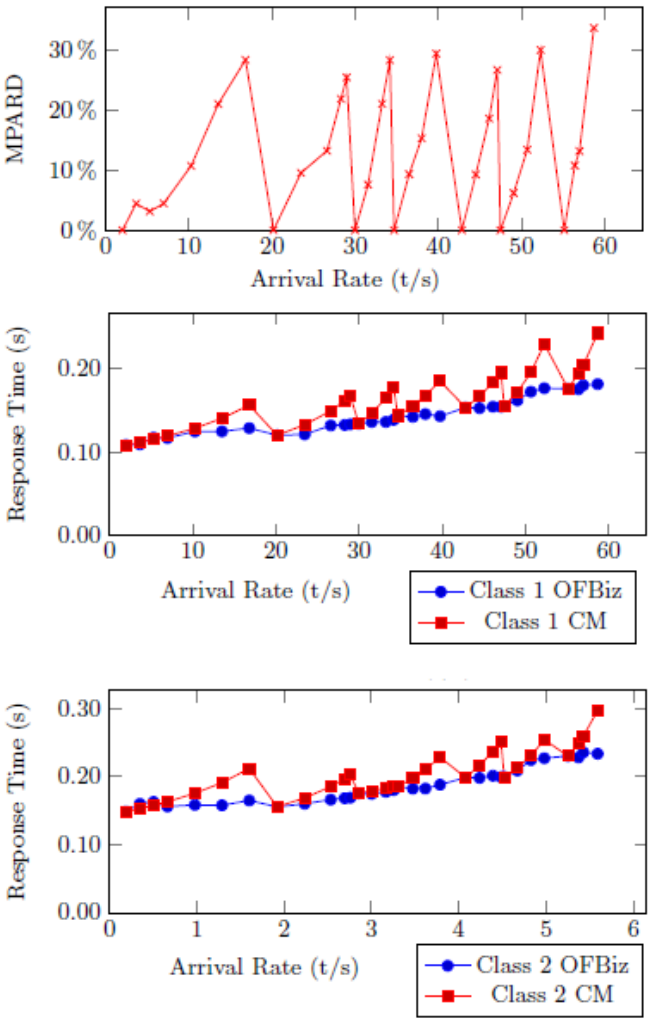

Figure 8: Static N - Top: MPARD vs. Arrival Rate. Middle: Class 1 Transactions. Bottom: Class 2 Transactions. $\epsilon = 25\%$.

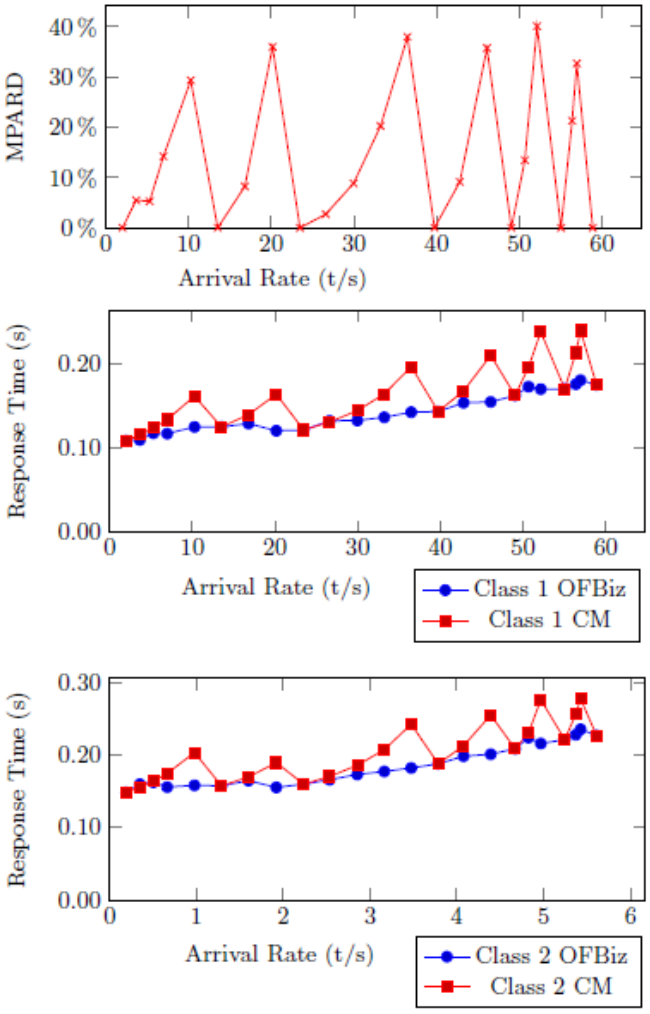

Figure 9: Variable N - Top: MPARD vs. Arrival Rate. Middle: Class 1 Transactions. Bottom: Class 2 Transactions. $\epsilon = 25\%$.

that would eventually saturate a certain system resource in a test environment in order to determine the system's bottlenecks.

The work in [15, 24, 25] addressed the problem of estimating model parameters in highly dynamic autonomic environments in which Service Level Agreements (SLAs) (in the form of 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 known and where the decision to recalibrate only depends on the level of divergence between observed measurements and model estimated measurements.

5.1 Performance Modeling

In [16], the author used a black box approach to illustrate the existence of a modeling shortcut that can be used to describe the system even when its internal structure is unknown. Instead of describing the behavior of a queuing system using Response Time versus Average Utilization graph, the author proposes the use of a system map, which shows Service to Response Time ratio versus Average Utilization. The system map graph is generic and applies to any system. Such graph can be produced using the general formula $M = 1 - \rho^c$. This formula can be used to estimate the shape of the system map of any queuing system even if only one point on the map is provided. c in the formula above represents the number of servers in the system. Therefore, given some measurable performance parameters (e.g., utilization and response time), the number of servers c can be

estimated by obtaining a curve that passes through a particular point on the system map graph. The author offers a method for estimating the system utilization (ρ) when the number of servers c and average number of outstanding requests are known. Using the system map equation above and Little's law, the utilization is bounded from above by nine values, which include the asymptotic value when c = 1, and eight values that approximate the utilization around the knee of the system map.

In [10], the authors introduced a method for performance parameter estimation using Kalman filters and Layer 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 [12], 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 accounts user specified forecasting objectives. The approach presented in this paper can be combined with our proposed framework to forecast future arrival rate for better workload clustering and model derivation.

A similar objective is addressed in [11] where the authors 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 [22], 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.

6 Conclusions

We have presented a framework for the dynamic derivation of analytical performance models in autonomic computing environments. The framework includes a multi-layered approach for workload analysis and performance model generation and parameterization. The approach takes into consideration the availability of system monitoring logs and the balance between producing costly but accurate performance models versus sufficient levels of model accuracy even when detailed system monitoring logs are available. We focused on the "gray box" level of this framework and presented an approach for deriving Queuing Network (QN) models where input parameters (e.g., arrival rates) and output parameters (e.g., average response time) are known. In addition, the approach utilizes some basic information about the system architecture, most of which can be easily obtained using high-level monitoring tools readily available in cloud environments, such as the number and function of each layer in a multi-tiered system (but not necessarily the number of servers of each tier). The derived QN model, which we call the Computed Model (CM), closely approximates the actual computer system. 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. 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^{TM}.

For future work, we are looking into implementing the workload behavior analyzer by studying the impact of dynamic clustering on the ability of the performance model to accurately predict the performance of the real system. We also plan to implement behavioral analysis techniques, such as CBMG, which is used to produce more accurate service demands per application module.

References

[1] Barna, C., Litoiu, M., Ghanbari, H., Autonomic Load-Testing Framework. In: Proceedings of the 8th ACM International Conference on Autonomic Computing, pp. 91–100, 2011.

[2] Begin, T., Brandwajn, A., Baynat, B., Wolfinger, B.E., Fdida, S., Towards an Automatic Modeling Tool for Observed System Behavior. In: Formal Methods and Stochastic Models for Performance Evaluation, pp. 200–212, 2007.

[3] Begin, T., Brandwajn, A., Baynat, B., Wolfinger, B.E., Fdida, S., High-Level Approach to Modeling of Observed System Behavior. In: ACM SIGMETRICS Performance Evaluation Review, 35(3), pp. 34–36, 2007.

[4] Begin, T., Baynat, B., Sourd, F., Brandwajn, A., A DFO Technique to Calibrate Queuing Models. In: Computers & Operations Research 37, no. 2, pp. 273–281, 2010.

[5] Bennani, M., and Menasce, D.A., Assessing the Robustness of Self-Managing Computer Systems under Highly Variable Workloads. In: Proceedings of the International Conference on Autonomic Computing, 2004.

[6] Bennani, M.N., Menascé, D.A., Resource Allocation for Autonomic Data Centers Using Analytic Performance

Models, In: 2005 IEEE International Conference on Autonomic Computing, 2005.

[7] Buzen, J.P. and Denning, P.J., Operational Treatment of Queue Distributions and Mean-Value Analysis. In: Computer Performance, IPC Press, Vol. 1, No. 1, pp. 6–15, 1980.

[8] Buzen, J.P. and Denning, P.J., Measuring and Calculating Queue Length Distributions. In: IEEE Computer, pp. 33–44, 1980.

[9] Denning, P.J. and Buzen, J.P., The Operational Analysis of Queuing Network Models. In: ACM Comp. Surveys, Vol. 10, No. 3, pp. 225–261, 1978.

[10] Ghanbari, H., Barna, C., Litoiu, M., Woodside, M., Zheng, T., Wong, J., Iszlai, G., Tracking Adaptive Performance Models Using Dynamic Clustering of User Classes. In: Proceedings of the 2nd ACM/SPEC International Conference on Performance engineering, 2011.

[11] Gmach, D., Rolia, J., Cherkasova, L. and Kemper, A., Workload Analysis and Demand Prediction of Enterprise Data Center Applications. In: Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization, pp. 171–180, 2007.

[12] Herbst, N.R., Kounev, N.S., and Amrehn, E., Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning. In: Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering, 187–198, 2013.

[13] Kephart, K., and Chess, D., The Vision of Autonomic Computing. In: IEEE Internet Computing, Vol. 36, No. 1, pp. 41–50, January 2003.

[14] Kounev, S., Huber, N., Spinner, S., Brosig, S., Model-Based Techniques for Performance Engineering of Business Information Systems. In: Business Modeling and Software Design, Lecture Notes in Business Information Processing (LNBIP), Vol. 0109, pp. 19–37, 2012.

[15] Litoiu, M., Woodside, M., Zheng, T., Hierarchical Model-Based Autonomic Control of Software Systems. In: ACM SIGSOFT Software Engineering Notes, Vol. 30, No. 4, pp. 1–7, 2005.

[16] McNutt, B., Waiting for a Black Box. In: Computer Measurement Group Conference, 2013.

[17] Menascé, D.A., Almeida, V., Fonseca, R. and Mendes, M.A., A Methodology for Workload Characterization of E-commerce Sites, In: ACM Conference on Electronic Commerce, 1999.

[18] Menascé, D., Almeida, V., Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning. Prentice Hall, 2000.

[19] Menascé, D.A., and M. Bennani, On the Use of Performance Models to Design Self-Managing Computer Systems, In: Proceedings of the 2003 Computer Measurement Group Conference, 2003.

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

[21] Menascé, D., Computing Missing Service Demand Parameters for Performance Models. In: Proceedings of the Thirty-fourth Intl. Computer Measurement Group Conference, pp. 7–12, 2008.

[22] Pérez, J.F., Pacheco-Sanchez, S., Casale, G., An Offline Demand Estimation Method for Multi-Threaded Applications. In: Proceedings of the 2013 IEEE 21st International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (pp. 21-30). IEEE Computer Society, 2013.

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

[24] Woodside, M., Zheng, T., Litoiu, M., The Use of Optimal Filters to Track Parameters of Performance Models. In: Second International Conf. Quantitative Evaluation of Systems, pp. 74–83, 2005.

[25] Zheng, T., Yang, J., Woodside, M., Litoiu, M., Iszlai, G., Tracking Time-Varying Parameters in Software Systems With Extended Kalman Filters. In: Proceedings of the 2005 Conference of the Centre for Advanced Studies on Collaborative research, pp. 334–345, 2005.