RC24894 (W0911-025) November 4, 2009
Computer Science

IBM Research Report

Enhanced Inferencing: Estimation of a Workload Dependent Performance Model

Dinesh Kumar, Li Zhang, Asser Tantawi
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Enhanced Inferencing: Estimation of a Workload Dependent Performance Model

Dinesh Kumar, Li Zhang, Asser Tantawi
IBM T.J. Watson Research Center, Hawthorne, NY, USA
{kumardi,zhangli,tantawi}@us.ibm.com

ABSTRACT

Performance modeling of software systems is vital for predictive analysis of their performance and capacity planning of the host environment. Robust performance prediction and efficient capacity planning highly depend on an accurate estimation of the underlying model parameters. AMBIENCE, which is a prototype tool developed at IBM Research, makes use of the powerful Inferencing technique to generate a workload-independent parameters based performance model. However, modern software systems are quite complex in design and may exhibit variable service times and overheads at changing workloads. In this work, we extend the Inferencing technique for generating *workload-dependent* service time and CPU overhead based performance models. We call this extended form as Enhanced Inferencing. Implementation of Enhanced Inferencing in AMBIENCE shows significant improvement of the order of 26 times over Inferencing. We further present a case study where Enhanced Inferencing provides a quantitative performance difference between consolidated and partitioned software system installations. Ability to carry out such evaluations can have significant impact on capacity planning of software systems that are characterized by workload-dependent model parameters.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques

General Terms

Queueing Theory, Inferencing, Estimation

Keywords

service time, CPU overhead, performance prediction

1. INTRODUCTION

Performance modeling is a crucial step in capacity planning of computer software systems. Performance models of complex software and hardware architectures can be very helpful in accurately predicting their performance for varying traffic patterns and workloads. In this paper, we are concerned with performance modeling of transaction-based, distributed software systems that process transactional workload of request/response type, such as HTTP workload. An example being an e-commerce based online shopping system. User workload processed by this system includes authentication transactions such as login, and business transactions such as browsing a catalog, searching for products, adding items to a shopping cart, proceeding to check out, etc. Each of these transactions use the e-commerce system resources differently and can be classified into different classes. Figure 2 in Section 2 depicts an example transaction-based, distributed software system. The 12 icons on left represent 12 different transaction classes that are generated by the 'Client' machine. These transactions are processed by the 'Web/CQ' machine that hosts a Webserver (e.g., Apache) and the ClearQuest [2] application. Processing of these transactions may trigger further requests to the 'DB' machine that hosts a database.

Classical queueing theory based performance models [5] require the knowledge of parameters such as *service times* for different transaction classes and *CPU overheads* for different machines. These parameters can be used to compute and predict performance metrics such as average transaction response time, average number of jobs/transactions waiting to be processed, etc. There are other existing techniques that make use of simulations and manual caliberations to compute similar performance metrics [17]. However, none of these techniques can be practically applied if the service times and CPU overheads are unknown. Instrumenting software applications with probes in order to actually measure the service time and overhead parameters can be intrusive, requires extensive manual coding [1] and is time consuming. In fact, source code of a standard, commercialized software system may not even be accessible. Moreover, instrumentation is an iterative procedure and is difficult to pursue in a dynamically changing environment (see [8] and references there in). The system parameters must therefore be estimated using only readily available measurement data. AMBIENCE [18, 6], which is a prototype tool developed at IBM Research, makes use of a powerful Inferencing algorithm [7] to estimate a service time, delay and CPU overhead based performance model. Inferencing allows one to compute these three parameters from readily available measurement data on end-to-end response times, CPU utilizations and workload arrival rates. It does not require any instrumentation

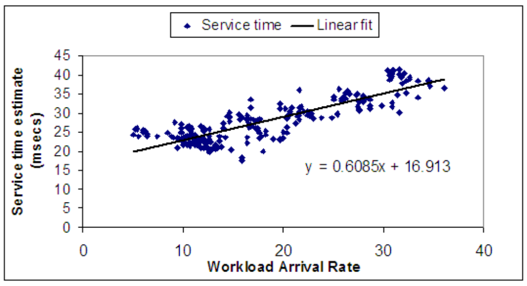

Figure 1: Rough service time estimates

or modification of the software system. Inferencing however models service time and CPU overhead as *constant* parameters, independent of the workload.

Modern software systems must continuously self-reconfigure their components to adapt to run-time changes in the host and network environments [12]. This is especially the case for Internet based online applications that operate in a highly dynamic environment with fast changing user workloads and browsing patterns. However, autonomic reconfiguration [12] in modern software systems can lead to *workload-dependent* service times and overheads (see [3, 16] and references there in). Moreover, state of the art transaction-based software systems are quite complex in design. They may incur extra processing overheads when mean transactional workload is high, as compared to when it is low. Due to this end-to-end transactions may incur variable service times depending on the total arriving workload (discussed in detail in Section-3). As an example, for a single transaction class, Figure 1 shows rough service time estimates from measurements gathered from a software system similar to that of Figure 2. The service times clearly show a linear dependence on the workload arrival rate. More evidence and discussion can be found in [8, 3] and references there in. As a consequence, Inferencing may not work reliably for workload-dependent parameters since it assumes constant model parameters.

1.1 Related Work

A prototype implementation of a performance management system for multi-tiered web applications deployed on clustered web servers is decribed in [9]. The management system allocates server resources dynamically in order to optimize the expected value of a system-wide utility function. Authors there use a closed queueing network model to predict the response time of requests for different resource allocations. However, service time parameters of the queueing network model are obtained using an ad-hoc approach based on the argument that response time at each node is close to the service time on the same node when workload is light. Approximating response time at light workload as a measure of the service time lacks a sound analytical and theoretical basis. This approach may not work with multiple classes of transactions. The reason being that experiments with hypothetically light workloads for different classes, whether individually or all classes collectively, may not incorporate the interaction between multiple classes of transactions. Since presence or absence of a particular class of transaction may impact service time of another, this may lead to incorrect service time estimates. Moreover, light load response time approximation is not a feasible approach for software systems with workload-dependent parameters.

Urgaonkar et al. [16] develop another closed queueing network model of a multi-tiered system. Estimates of the model parameters, such as visit ratios and load-dependent service times, are obtained off-line through analyzing various measurement logs in the system. Their methodology lacks a sound analytical model for parameter estimation. Pacifici et al. [8] consider the problem of dynamically estimating CPU demands of applications using CPU utilization and throughput measurements. Using a linear model, they formulate the problem as a multivariate linear regression problem and analyze measurement data in order to extract realistic traffic properties. With this approach however, they admit facing several practical issues such as insignificant flows, co-linear flows, space and temporal variations, etc. They present several ad-hoc techniques to deal with these issues. Their experimental results demonstrate that the approach is viable only for a *rough* estimation of the dynamically changing CPU demands. They provide many future work guidelines to deal with various other issues that they were not able to address in their work. Authors in [15, 10] disclose a method for estimating the response time of a typical transaction request. However, they do not have any mathematical basis for the estimation of response time and rely solely on simple measurement data. Moreover, they do not explicitly model service time. The work in [14] creates profiles for application resource consumption based on OS level measurements. It uses regression techniques to correlate CPU resource, remote invocation overhead and inter-component communication versus the request load. However, it requires kernel level modifications to collect usage data. Authors in [13] express response times as linear combinations of the service times with the coefficients derived from measured utilizations and request volumes. It then applies various regression techniques to estimate the response times based on data collected from different time intervals. This approach can not be used to predict the resource utilizations for given request arrival volumes. It also does not address the possible non-linear scaling behaviors between resource utilizations and request volumes due to workload-dependent service times.

Authors in [11] use simple curve fitting (regression) techniques to fit the utilization and response time curves independent of each other. Although it is relatively easy to pick the family of curve for utilization, the significant non-linear behavior of response times makes it a challenging task to select their curve family. Furthermore, modeling and fitting the utilization and response time independent of each other clearly misses the intrinsic linkage and coupled relationship between the two. On the other hand, Inferencing first employs a queueing theory based model that relates the utilization and response time, and then uses a least-squares framework for estimating the model parameters.

To summarize and to the best of our knowledge, even though performance modeling has been a research topic since long time, no systematic and analytically sound methodology for estimating *workload-dependent* service times and CPU overheads has been published in literature before. Most of the techniques cited above, including the Inferencing algorithm [7], consider only constant service times and overheads. Moreover, all above cited techniques (except Inferencing) either require instrumentation of the software system, lack a sound analytical model and require ad-hoc techniques, or are based on simple regression techniques failing to link the response time and utilization metrics.

1.2 Summary of Contribution

In this work, we extend the Inferencing technique for generating workload-dependent service time and CPU overhead based performance models. We call this extended form of Inferencing as *Enhanced Inferencing.* Enhanced Inferencing works by increasing the *degrees of freedom* of the performance model parameters and representing service times and overheads as functions of the sum of total arriving workload. Different functional representations are possible depending on the nature of the software system. In particular, they may be polynomial functions of either the simple sum of workload, the exponential of sum of workload or the logarithm of sum of workload. Degrees of the chosen polynomials represent the amount of increase in degrees of freedom for the service time and CPU overhead parameters. To the best of our knowledge, ours is the first attempt to propose a method for performance modeling of software systems that process multiple classes of transaction and exhibit workload-dependent characteristics.

Rest of the paper is organized as follows. In Section 2, we first provide a brief background in which the notation used in this paper is introduced and the Inferencing technique is summarized. It is also shown that Inferencing does not give good results for an example software system with workload-dependent service times. In Section 3, further motivation for workload-dependent performance modeling is provided and Enhanced Inferencing is presented. Some insights for choosing the right number of experiments is discussed. Implementation of Enhanced Inferencing into the AMBIENCE tool is shown to provide significantly improved results for the same example software system. Section 4 provides a case study with a consolidated and partitioned software system. We finally conclude in Section 5.

2. BACKGROUND

Consider a single transaction class processed by a single server machine. Transactions with similar characteristics are grouped into a single class. For example, transaction classes may include transactions for searching, buying, logging in, etc. Buying transaction class may include transactions that are responsible for purchasing a particular product and so on. Different transaction classes typically have different service requirements. The single server machine is modeled as an $M/G/1$ queue with Processor Sharing (PS) service discipline [5]. Then from queueing theory it is known that for an open model [5] of an $M/G/1$ queue with PS service discipline,

$$R = \frac{s}{1-u}, \tag{1}$$

where,

$$u = \lambda\, s. \tag{2}$$

In the above formulae, s is the service time of given transaction class, R is its response time, λ is the arrival rate of all such transactions and u is the server utilization. The above equations can be repeatedly applied to a given software system that includes multiple server machines and multiple classes of transactions.

2.1 Notation

All entities, $\tilde{\cdot}$, with a tilde on their top are actual *average* values of readily available measurement data that can be obtained by conducting suitable experiments with the software system. Let there be K transaction classes and M physical machines in a given system topology. Let $\mathcal{C}$ denote the index set of all transaction classes that may have different service requirements. Let $\mathcal{L}$ denote the index set of all physical machines that are part of a given topology. Let the rate at which total workload of a class $c \in \mathcal{C}$ transaction arrives from an external source into the network (through any machine) be denoted by $\tilde{\gamma}^c$. Let the rate at which workload of a class $c \in \mathcal{C}$ transaction arrives at machine $i \in \mathcal{L}$ from with in the network be denoted by λ_i^c. It represents the *effective* workload of class c arriving at machine i, where as $\tilde{\gamma}^c$ denotes the total workload of class c being generated by an external source. Denote, $\boldsymbol{\lambda}^c = [\lambda_1^c, \lambda_2^c, \ldots, \lambda_M^c]$. Also, in the vector $\boldsymbol{\gamma}^c = [0, 0, \ldots, \tilde{\gamma}^c, \ldots, 0]$, $\tilde{\gamma}^c$ is located at the jth position ($j \in \mathcal{L}$) when class c transactions enter the network through machine j. One may then compute the effective workloads by solving the open Jackson network traffic balance equations [5] given by,

$$\forall c \in \mathcal{C}, \qquad \boldsymbol{\lambda}^c = \boldsymbol{\gamma}^c (I - P^c), \tag{3}$$

where, $P^c = [P_{ij}^c]$ is the traffic routing probability matrix, i.e., probability that a class c flow leaves machine i and goes to machine j is given by P_{ij}^c. It is assumed that P^c is well defined for a given network architecture, i.e., the values P_{ij}^c are given.

2.2 Inferencing

Here we briefly describe the Inferencing technique from [18, 6] that allows one to generate a service time, network delay and CPU overhead based performance model. The three model parameters are assumed to be constant and independent of the arriving workload. Consider the case of stationary workload or assume that measurements are taken from the system when arriving workload stays within a given stationary regime. Let s_i^c denote the service time of class c transaction at machine i, d_{net}^c denote the total network delay incured by class c transaction and o_i^{cpu} denote the total CPU overhead at machine i. The queueing theory based formulae described earlier in Equations 1 and 2 can be repeatedly applied to estimate the model parameters using Inferencing technique as follows. For any given system topology, expressions for response time, R^c, of traffic class c and utilization, u_i, of machine i can be written as,

$$\forall c \in \mathcal{C}, \qquad \sum_{i \in L} \alpha_i^c \frac{s_i^c}{1 - \tilde{u}_i} + d_{net}^c = R^c, \tag{4}$$

$$\forall i \in \mathcal{L}, \qquad \sum_{c \in \mathcal{C}} \frac{\lambda_i^c}{P_i} s_i^c + o_i^{cpu} = u_i, \tag{5}$$

where, $\alpha_i^c = \lambda_i^c / \tilde{\gamma}^c$ and P_i is the total number of processors (CPUs) in machine i. One may then seek to minimize the sum of squares of relative errors between the analytical entities given by Equations 4 and 5 and their corresponding mean measurement values obtained from experiments conducted with the real software system. The corresponding optimization problem will comprise a quadratic objective function in the set of variables $\{s_i^c, d_{net}^c, o_i^{cpu}, e^c, e_i | c \in \mathcal{C}, i \in \mathcal{L}\}$,

$$\min \sum_{c \in \mathcal{C}} \left(\frac{e^c}{\tilde{R}^c}\right)^2 + \sum_{i \in \mathcal{L}} \left(\frac{e_i}{\tilde{u}_i}\right)^2, \tag{6}$$

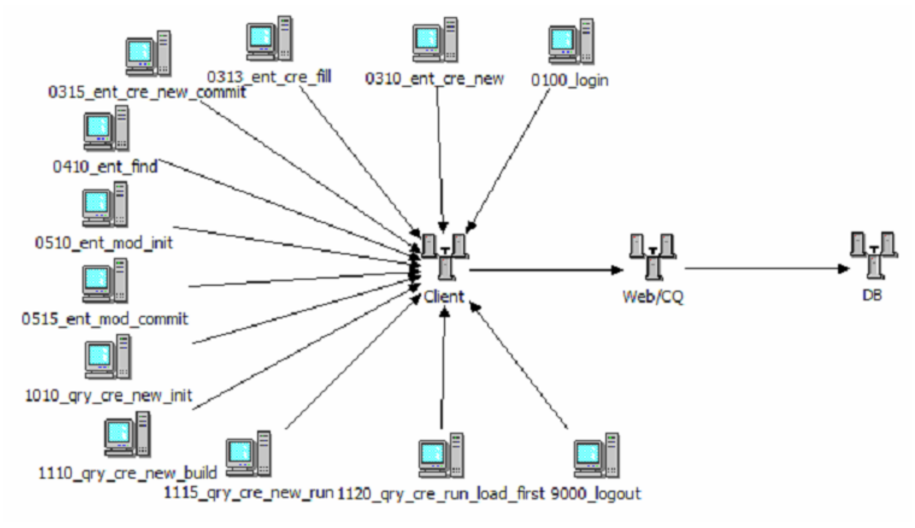

Figure 2: Consolidated topology for ClearQuest software system

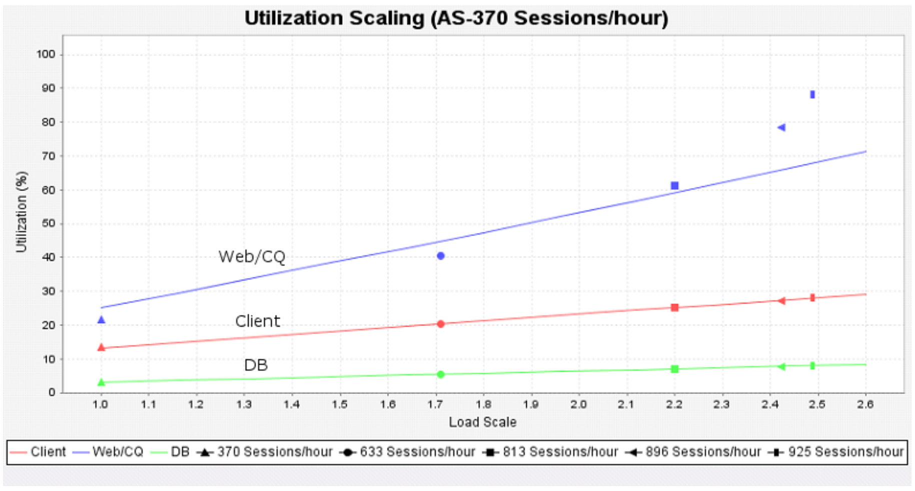

Figure 3: Utilization scaling with Inferencing

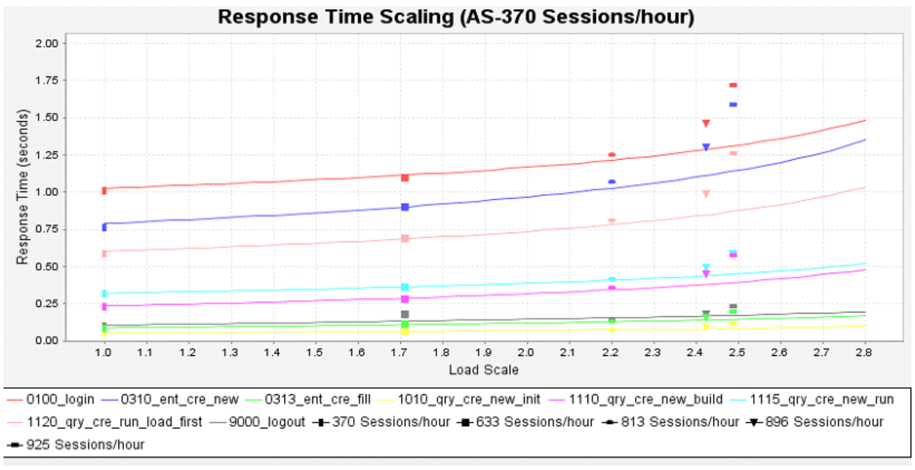

Figure 4: Response time scaling with Inferencing

with the following set of linear constraints,

$$\forall c \in \mathcal{C}, \qquad R^c + e^c = \sum_{i \in L} \alpha_i^c \frac{s_i^c}{1 - \tilde{u}_i} + d_{net}^c + e^c = \tilde{R}^c, \quad (7)$$

$$\forall i \in \mathcal{L}, \qquad u_i + e_i = \sum_{c \in \mathcal{C}} \frac{\lambda_i^c}{P_i} s_i^c + o_i^{cpu} + e_i = \tilde{u}_i. \quad (8)$$

e^c and e_i are slack variables and denote the error in response time and utilization, respectively. The above optimization problem is a standard Quadratic Programming (QP) problem with a linear set of constraints [4]. It can be solved using standard optimization algorithms for QP such as those described in [4]. The solution to this optimization problem will give us the service times, s_i^c, network delays, d_{net}^c, and CPU overheads, o_i^{cpu}, required as part of the solution to the Inferencing problem.

2.3 Conducting Experiments

In the previous sub-section, it was assumed that measurements are taken from the software system when arriving workload stays within a given stationary regime. These measurements will include average response times $\tilde{R}^c$ for each class $c \in \mathcal{C}$, average CPU utilizations $\tilde{u}_i$ for each machine $i \in \mathcal{L}$ and average arrival rates λ_i^c computed from γ^c for each $c \in \mathcal{C}$, $i \in \mathcal{L}$. A single experiment can thus be carried to gather these measurements in order to construct a single set of constraints given by Equations 7 and 8. The Inferencing optimization problem can be solved along with this single set of constraints to obtain the solution for model parameters. In a similar manner, multiple experiments can be conducted at varying stationary regimes of workload to formulate multiple sets of constraints given by Equations 7 and 8. The Inferencing optimization problem can then again be solved with these multiple sets of constraints to obtain the solution for model parameters. This time however, the solution will be representative of the constant model parameters valid across variable workloads for which the multiple experiments are conducted. In the following discussion, insights for deciding the number of experiments to be conducted are provided.

Consider the linear system of equations formed by Equations 4 and 5. Assume R^c and u_i there are replaced by their known measurable values from experiments. In general, if there are K classes and M machines then there will be $KM + K + M$ unknown variables in that system, i.e., service time (s_i^c) for each class at each machine (KM variables), delay (d_{net}^c) for each class (K variables), and CPU overhead (o_i^{cpu}) for each machine (M variables). For every experiment conducted, the number of linearly independent available measurements will be $K + M$, i.e., response time (R^c) for each class and CPU utilization (u_i) of each machine. Thus, a total of r experiments will lead to a system of $(K + M)r$ equations as per Equations 4 and 5. Workload arrival rate measurements are simply considered as known entities and will not increase the total number of equations. In order to obtain a unique solution for the $KM+K+M$ unknown variables, exactly the same number of equations are required. In other words, the following relationship must hold, $(K + M)r = KM + K + M$. This implies that the number of experiments required to obtain a unique solution is given by,

$$r = \left\lceil \frac{KM}{K + M} + 1 \right\rceil. \quad (9)$$

Any number of experiments less than that defined by Equation 9 will lead to an under-determined system of equations and hence multiple solutions. Any number of experiments more than that defined by Equation 9 will lead to an over-determined system of equations and possibly no solution. However, Equation 9 only provides a guideline for deciding the number of experiments to be conducted. In reality, greater number of experiments may be required, or lesser may be sufficient. Actual measurement data is always noisy due to background processes and other unpredictable factors with in the software system. This may result in experiment data sets that are not linearly independent, requiring additional experiments to be carried out. Alternatively, noisy data may reduce the set of feasible solutions. In the least-squares estimation problem this may eventually lead to a unique *optimal* solution in spite of an under-determined system with lesser number of experiments. The fact that the system of equations formed by Equations 4 and 5 may not be exactly solved using noisy measurement data for R^c and u_i, is the exact reason why a least-squares estimation framework was setup in the previous sub-section.

2.4 Inferencing Example

Consider the ClearQuest software system of Figure 2. Details of this system were described earlier in Section 1. We are interested in generating a performance model of this system using AMBIENCE tool which uses the Inferencing algorithm. For this, 5 different experiments were carried out at 5 different transactional workloads. Though Equation 9 requires only 4 experiments, one additional experiment was carried out due to reasons cited in the above discussion. The workload intensities were 370 sessions/hour, 633 sessions/hour, 813 sessions/hour, 896 sessions/hour and 925 sessions/hour, in all cases the workload comprising a certain proportional mix of the 12 transaction classes. Measurements obtained from these experiments were input into AMBIENCE and the model was built to infer (estimate) the various service times, CPU overheads and network delays. Once these model parameters are estimated, AMBIENCE plots CPU utilization and response time scaling curves based on Equations 4 and 5. Figures 3 and 4 show the CPU utilization and response time scaling charts, respectively. The dots in the charts represent average utilization and response time measurements for different machines and transaction classes, respectively, at different workloads. The lines in the charts are utilization and response time predictions generated by the model, based on Equations 5 and 4, respectively, using the infered (estimated) service times and CPU overheads. On x-axis 'Load Scale' represents ratio between the transactional workload for different experiments and the base workload. The experiment with smallest workload is considered as the base workload experiment and is indicated in the heading on top of the chart. Thus, for the 370 sessions/hour experiment the Load Scale is 1.0. In Figure 3 we see that the model predicted utilization is *linear* in variation and matches well with the measured utilization for 'Client' and 'DB' machines. Observing Equation 5, this indicates that the service times and CPU overheads for these two machines are constant and workload independent. However, for 'Web/CQ' machine we see that the model predicted and measured utilizations do not match well. Since service times are modeled as constant parameters in Inferencing, the model predicted utilizations do not scale non-linearly (see

Equation 5) to match the non-linear trend in the measurement points. The non-linear variation in utilization measurement points indicates that the actual service times may be workload dependent. In Figure 4, we again see that the model predicted and measured response times match well for some transactions, but not for others.

3. ENHANCED INFERENCING

As discussed in Section 1, complex design of state of the art software applications may lead to variable service times depending on the total arriving workload. This is because computer machines may incur extra processing overheads when mean transactional workload is high as compared to when it is low. In a multi-tasking operating system environment, higher average transactional workload may cause increased context-switching of various application modules and components in the CPU and memory. Over a given period of time the CPU may end up spending more time in context-switching than processing the actual transaction jobs. This may lead to a slower transaction processing rate by the CPU and increased service times for a given transaction class. In other words, it may lead to a slower server. Also, higher transaction workload processed by the NIC (Network Interface Card) can force it to generate higher number of interrupts at the CPU for data transfers between the NIC and main memory. This will cause fewer CPU cycles being allocated to the processing of actual transaction jobs and the CPU will spend more time in serving the interrupts. Service times of transactions will be further affected due to this phenomenon. It is clear from the results in Section 2.4 and the above discussion that we must consider service times and CPU overheads as some function of the arriving workload and not constants. In other words, service time for a particular transaction class at a given machine and total CPU overhead in a particular machine must be modeled so that they can take greater values at higher mean workloads as compared to at lower mean workloads. We therefore extend the Inferencing technique to incorporate arriving workload dependent service times and CPU overheads and call it Enhanced Inferencing.

Enhanced Inferencing works by increasing the *degrees of freedom* of the service time and CPU overhead model parameters and representing them as functions of the workload. We have empirically observed that the rate at which service times increase with increasing arriving workload does not follow a uniform trend. It rather depends on the complex modular design of software application systems, configuration of application deployment environment, etc. We have futher observed that the rate of increase of service times and CPU overheads is usually either a polynomial, exponential or logarithmic function of the arriving workload. The empirical evidence is provided through a case study in Section 4. Based on this evidence and observation, intuitively it would then make sense to consider the service times and CPU overheads as increasing polynomial, exponential or logarithmic functions of the total arriving workload. Moreover, we have observed that the *sum* of arriving workload over all transaction classes works as a fairly good approximation in order to incorporate dependence of service times and CPU overheads on the entire workload vector. This motivates us to consider them as polynomial functions of either the simple sum of workload, the exponential of sum of workload or the logarithmic of sum of workload. However, note that this dependency on arriving workload may also be characterized as, service times and CPU overheads being functions of any arbitrary or generic representation of the workload vector. In other words, the service times and CPU overheads may be expressed as $s_i^c(f(\{\lambda_i^c | c \in \mathcal{C}\}))$ and $o_i^{cpu}(g(\{\lambda_i^c | c \in \mathcal{C}\}))$, respectively, for *any* given functions f and g. Here, as an example we consider these functions to be $f(\{\lambda_i^c | c \in \mathcal{C}\}) = \sum_c \lambda_i^c$ and $g(\{\lambda_i^c | c \in \mathcal{C}\}) = \sum_c \lambda_i^c$. Therefore, denoting $x = \sum_c \lambda_i^c$, for a given transaction class c the service time at machine i is expressed in one of the following ways,

$$s_i^c(\sum_c \lambda_i^c) = s_i^c(x) = a_{0,i}^c + a_{1,i}^c x + a_{2,i}^c x^2 + \cdots + a_{n,i}^c x^n \tag{10}$$

$$s_i^c(\sum_c \lambda_i^c) = s_i^c(x) = a_{0,i}^c + a_{1,i}^c e^x + a_{2,i}^c e^{2x} + \cdots + a_{n,i}^c e^{nx} \tag{11}$$

$$s_i^c(\sum_c \lambda_i^c) = s_i^c(x) = a_{0,i}^c + a_{1,i}^c \log x + \cdots + a_{n,i}^c (\log x)^n \tag{12}$$

where, n is the degree of the chosen polynomial. Note that one may choose either one of the formulations in Equations 10, 11 or 12 for each computer machine i, separately. However, service times of all transaction classes in a given machine will have identical formulations. Similarly, the total CPU overhead at machine i may be expressed in one of the following ways,

$$o_i^{cpu}(\sum_c \lambda_i^c) = o_i^{cpu}(x) = b_{0,i}^c + b_{1,i}^c x + b_{2,i}^c x^2 + \cdots + b_{m,i}^c x^m \tag{13}$$

$$o_i^{cpu}(\sum_c \lambda_i^c) = o_i^{cpu}(x) = b_{0,i}^c + b_{1,i}^c e^x + b_{2,i}^c e^{2x} + \cdots + b_{m,i}^c e^{mx} \tag{14}$$

$$o_i^{cpu}(\sum_c \lambda_i^c) = o_i^{cpu}(x) = b_{0,i}^c + b_{1,i}^c \log x + \cdots + b_{m,i}^c (\log x)^m \tag{15}$$

where, m is the degree of the chosen polynomial. Again, either one of the formulations in Equation 13, 14 or 15 may be chosen for each computer machine i, separately. We may now rewrite Equations 4 and 5 as,

$$\forall c \in \mathcal{C}, \qquad \sum_{i \in L} \alpha_i^c \frac{s_i^c(\sum_c \lambda_i^c)}{1 - \tilde{u}_i} + d_{net}^c = R^c \tag{16}$$

$$\forall i \in \mathcal{L}, \qquad \sum_{c \in \mathcal{C}} \frac{\lambda_i^c}{P_i} s_i^c(\sum_c \lambda_i^c) + o_i^{cpu}(\sum_c \lambda_i^c) = u_i \tag{17}$$

The optimization problem will again comprise a quadratic objective function, but this time in the set of variables $\{a_{p,i}^c, b_{q,i}, d_{net}^c, e^c, e_i \mid p \in \{0, 1, \ldots, n\}, q \in \{0, 1, \ldots, m\}, c \in \mathcal{C}, i \in \mathcal{L}\}$,

$$\min \sum_{c \in \mathcal{C}} \left(\frac{e^c}{\tilde{R}^c}\right)^2 + \sum_{i \in \mathcal{L}} \left(\frac{e_i}{\tilde{u}_i}\right)^2, \tag{18}$$

with the following set of linear constraints,

$$\forall c \in \mathcal{C}, \quad R^c + e^c = \sum_{i \in L} \alpha_i^c \frac{s_i^c(\sum_c \lambda_i^c)}{1 - \tilde{u}_i} + d_{net}^c + e^c = \tilde{R}^c, \tag{19}$$

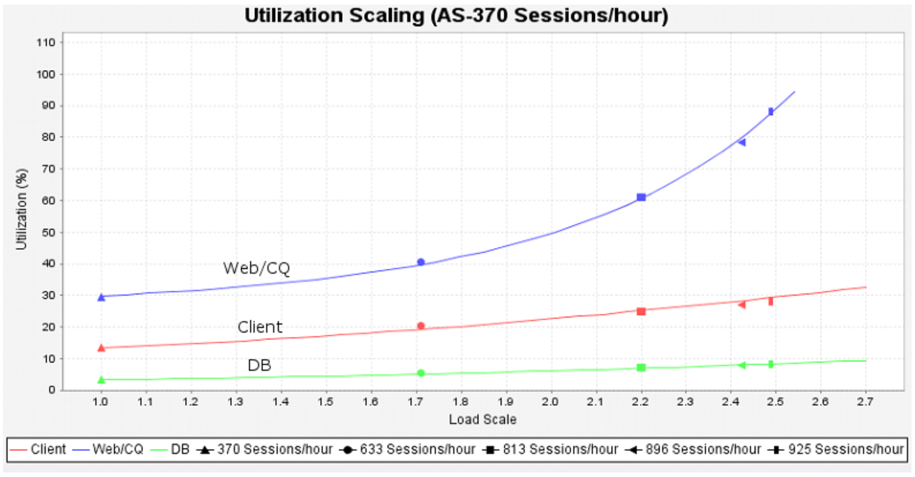

Figure 5: Utilization scaling with Enhanced Inferencing

$$\forall i \in \mathcal{L}, \quad u_i + e_i = \sum_{c \in \mathcal{C}} \frac{\lambda_i^c}{P_i} s_i^c(\sum_c \lambda_i^c) + o_i^{cpu}(\sum_c \lambda_i^c) + e_i = \tilde{u}_i. \tag{20}$$

The model now has increased degrees of freedom through the multiple constant coefficients in the polynomials defined above. But, the main advantage of our approach is that the above optimization problem still remains a standard Quadratic Programming (QP) problem with a linear set of constraints [4]. Any increase in computational complexity for this new QP problem over that of Inferencing is limited by the degrees of the chosen polynomials. It can still be solved using standard algorithms for QP [4]. The solution to this problem will give us the service time and CPU overhead polynomial coefficients, $a_{p,i}^c$ and $b_{q,i}$, and network delays, d_{net}^c, required as part of the solution to the Enhanced Inferencing problem. Note that Inferencing described in Section 2.2 is only a particular case of Enhanced Inferencing with constant, zero-degree polynomials.

3.1 Choosing Function Type and Degree

The choice of function type and its degree for Enhanced Inferencing is not hand-picked. It rather depends on the design complexity and transaction processing characteristics of the software system under consideration. Some software systems may exhibit a simple polynomial variation in the model parameters, while others may exhibit an exponential or logarithmic variation. Given the type of variation, some software systems may exhibit a higher order variation, while others may exhibit a lower order variation. Degrees of the chosen polynomials, n and m, represent the required amount of increase in the degrees of freedom for the service time and CPU overhead parameters. The exact type and degree of variation which is appropriate for a given software system can be naturally derived from the solution to the QP problem. To begin with, an arbitrary function type and an arbitrarily high value (e.g., 5 or 6) is chosen for the degrees. The QP problem is solved with this choice. Then, depending on the software system characteristics and measurement data, the residual error, i.e., the minimal value of the objective function of Equation 18, will be acceptable or not based on a given error tolerance. If the residual error is not acceptable then the optimization algorithm is iterated over other choices of function types. The function type that yields minimal residual error is the optimal choice. The solution vector with the optimal choice of function type will yield non-zero values for the lower order coefficients $a_{p,i}^c$ and $b_{q,i}$, and possibly zero or close to zero values for the higher order coefficients. In this manner, appropriate values of the degrees, n and m, can be inherently derived from the experimental data and they will be dictated by the processing complexity of the software system and the solution to the QP problem itself.

3.2 Conducting Experiments

We provide here insights for deciding the number of experiments to be conducted for Enhanced Inferencing. This discussion is in the lines of Section 2.3. Due to Equations 10 through 15, there will be a total of $n+1$ coefficient variables for each service time and $m+1$ coefficient variables for each CPU overhead. Therefore, for enhanced inferencing if we have K classes and M machines we will have a total of $(n+1)KM + K + (m+1)M$ unknown variables to be estimated, i.e., $n+1$ coefficients $(a_{p,i}^c)$ of service time for each class at each machine ($(n+1)KM$ variables), delay (d_{net}^c) for each class (K variables), and $m+1$ coefficients $(b_{q,i})$ of CPU overhead for each machine ($(m+1)M$ variables). However, for every experiment conducted, the number of linearly independent available measurements will still be $K+M$, i.e., response time for each class and CPU utilization of each machine. Thus, a total of r experiments will again lead to a system of $(K+M)r$ equations as per Equations 16 and 17. Workload arrival rate measurements are still simply considered as known entities and will not increase the total number of equations. Again, in order to obtain a unique solution for the $(n+1)KM + K + (m+1)M$ unknown variables, exactly the same number of equations are required. This implies that the number of experiments required is given by,

$$r = \left\lceil \frac{(n+1)KM + K + (m+1)M}{K+M} \right\rceil. \tag{21}$$

Similar to the arguments provided in Section 2.3, Equation 9 should not be considered accurate for deciding the number of experiments. In reality, greater or lesser number of experiments may be required.

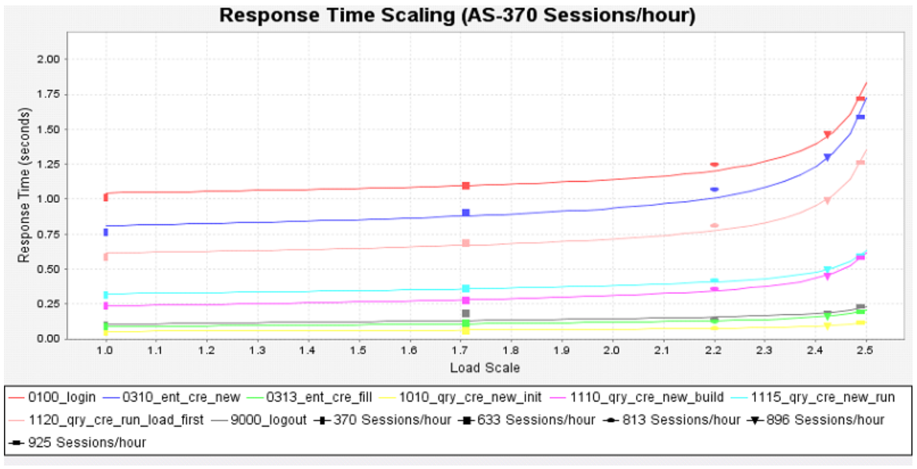

Figure 6: Response time scaling with Enhanced Inferencing

The above discussion is clearly related to the insights provided in Section 3.1. The choice of number of experiments given by Equation 21 is dependent on the derived degrees of polynomials or functions. In turn, the derived values of degrees and function types depend on the chosen set of experiments and measurement data. Each independently conducted experiment yields a different set of measurement data due to background noise and other unpredictable factors with in the software system. Though it can be envisaged to perform greater number of experiments than required before-hand, the exact choice of the subset of experimental data will lead to an iterative procedure for solving the Enhanced Inferencing problem. For a given error tolerance, this procedure can be automatized with in the AMBIENCE tool using standard conditional logic and loops.

3.3 Example Revisited with Enhanced Inferencing

AMBIENCE tool was modified to implement the Enhanced Inferencing algorithm described above. Service times and CPU overheads are now represented by the polynomial coefficient vectors instead of zero-degree constant values. We regenerated a performance model for the example discussed previously in Section 2.4 using the modified AMBIENCE tool. Same set of experimental data was used as in Section 2.4. Even though 6 experiment data sets are required as per Equation 21, 5 sets turned out to be sufficient for obtaining a unique solution (see discussion in Section 2.3). The automatized iterative procedure resulted in the choice of 'exponential' function types and a degree of $n = 1$ and $m = 1$ for both service time and CPU overhead curves.

Figure 5 shows the CPU utilization scaling chart for the regenerated performance model. We see that the model predicted utilization is now non-linear in variation for the 'Web/CQ' machine and qualitatively, matches well with the measured utilization. For the 'Client' and 'DB' machines the utilizations match well just like before. Figure 6 shows the response time scaling chart for different transactions. This time we see that model predicted and measured response times match well for all the transaction classes. This updated model resulted in a residual error of 0.081, where as the original Inferencing based model of Section 2.4 resulted in a residual error of 2.145. Quantitatively, this leads to a highly significant improvement of the order of 26 times or about 96% decrease in residual error. We see that with Enhanced Inferencing the AMBIENCE tool gives significantly better results.

4. A CASE STUDY

Here we present another case study in relation to the example studied in Sections 2.4 and 3.3. Similar to the consolidated topology shown in Figure 2 where the Webserver and ClearQuest application are deployed on a single machine, Figure 7 shows a partitioned topology with the Webserver and ClearQuest application deployed on two separate machines. The modified AMBIENCE tool with Enhanced Inferencing was employed to generate a performance model for the partitioned topology of Figure 7. Data from 5 different experiments carried out at 5 different transactional workloads was used. The workload intensities were 684 sessions/hour, 1087 sessions/hour, 1292 sessions/hour, 1360 sessions/hour and 1432 sessions/hour, in all cases the workload comprising the same proportional mix of 12 transaction classes as in Sections 2.4 and 3.3. Figures 8 and 9 show CPU utilization scaling charts corresponding to performance models for the consolidated and partitioned topologies, respectively. Figure 8 is a simple replot of Figure 5 with the base workload experiment changed to 633 sessions/hour instead of 370 sessions/hour in Figure 5. Notice the change in load scales on x-axis for the various measurement points due to the change in base workload experiment. The base workload in Figure 9 is also set to 633 sessions/hour and not 684 sessions/hour. These changes were done to make Figures 8 and 9 easily comparable.

As mentioned earlier in Section 3.3, the overall utilization for 'Web/CQ' machine in the consolidated topology scales exponentially (Figure 8). The CPU overhead is infered to be a degree 1 polynomial in exponential of sum of total arriving workload. Explicitly, it is estimated as, $0.2966 + 0.00267exp$(sum of arrival workload). We also observe that the 'Web/CQ' machine reaches 100% utilization bottleneck at about 950 sessions/hour. For the performance model of the partitioned topology, the overall utilizations for 'Web' and 'CQ' machines are infered to scale quadratically (see Figure 9). The CPU overheads are infered to be degree 2

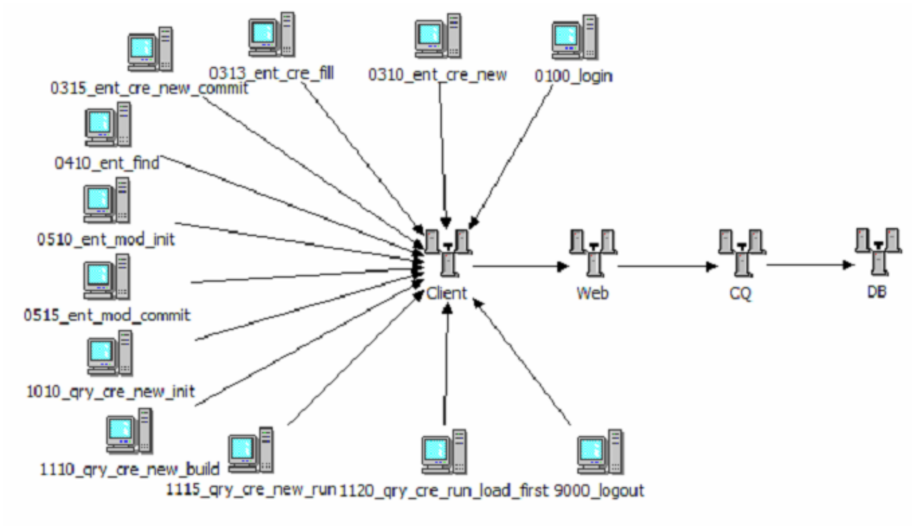

Figure 7: Partitioned topology for ClearQuest software system

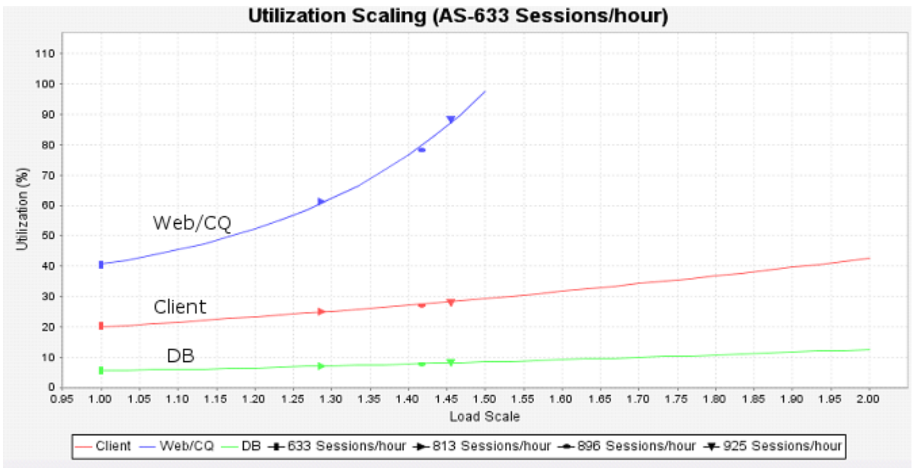

Figure 8: Utilization scaling for 'Web/CQ' machine in the consolidated topology

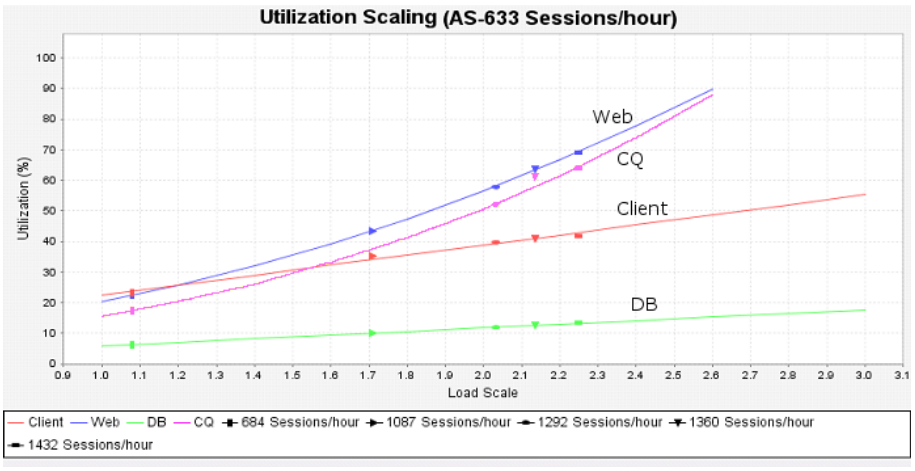

Figure 9: Utilization scaling for 'Web' and 'CQ' machines in the partitioned topology

polynomials in simple sum of total arriving workload. They are explicitly estimated as, 0.07851+0.008728(sum of arrival workload)^2 and 0.01251+0.000992(sum of arrival workload)^2 for the 'Web' and 'CQ' machines, respectively. Also note in Figure 9 that each of the 'Web' and 'CQ' machines reach 100% utilization bottleneck at about 1680 sessions/hour.

We notice above that separating the Webserver and ClearQuest application and deploying them on two different machines instead of a single one, can increase the transaction processing capacity of the system from about 950 sessions/hour to 1680 sessions/hour. Moreover, we can explicitly quantify the CPU overhead scaling variations as discussed above. The fact that different function types were obtained for different system topologies, serves as empirical evidence for our discussion in the second paragraph of Section 3. Such a performance evaluation study of a software system to predict its performance under consolidated and partitioned topologies can be extremely helpful for capacity planning purposes. Since the Webserver and ClearQuest application show a clear transition from exponential to quadratic scaling behavior in terms of the CPU overhead, this study could not have been carried out accurately with out employing Enhanced Inferencing.

5. CONCLUSION

Enhanced Inferencing is a novel approach for performance modeling of software systems with workload-dependent parameters. It is based on simple least-squares estimation, but makes use of a queueing theory model to incorporate the critical interdependence between response time and utilization metrics. Dependence of service time and CPU overhead model parameters on total arriving workload in various natural forms: polynomial, exponential and logarithmic, allows Enhanced Inferencing to be used for a wide variety of software systems with varying design complexities. It is thus a generic modeling methodology that can be readily used for transaction-based systems to produce a reliable and accurate performance model. The modified AMBIENCE tool with Enhanced Inferencing has considerable potential to improve the accuracy and efficiency of capacity planning with minimal intervence.

6. REFERENCES

[1] *Application Resource Measurement - ARM.* http://www.opengroup.org/tech/management/arm/.

[2] *IBM Rational ClearQuest.* http://www-01.ibm.com/software/awdtools/clearquest/.

[3] R. Bekker, S. C. Borst, O. J. Boxma, and O. Kella. Queues with workload-dependent arrival and service rates. *Queueing Syst. Theory Appl.*, 46(3-4):537–556, 2004.

[4] Z. Dostál. Optimal Quadratic Programming Algorithms. Springer, 2009.

[5] D. Gross and C. M. Harris. Fundamentals of Queueing Theory (Third Edition). Wiley-Interscience, 1998.

[6] Z. Liu, C. H. Xia, P. Momcilovic, and L. Zhang. AMBIENCE: Automatic Model Building using InferEnce. In *Congress MSR03*, France, 2003.

[7] Z. Liu et al. Method and apparatus for automatic model building using inference for it systems. *US Patent no. 7296256.*

[8] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi. Cpu demand for web serving: Measurement analysis and dynamic estimation. *Performance Evaluation*, 65(6–7):531–553, June 2008.

[9] G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef. Performance management for cluster based web services. *IEEE Journal on Selected Areas in Communications*, 23(12):2333–2343, December 2005.

[10] Z. Qin et al. Application response time prediction. *US Patent no. 6393480.*

[11] D. Saghier et al. Method for modeling system performance. *US Patent no. 7107187.*

[12] B. Solomon, D. Ionescu, M. Litoiu, and M. Mihaescu. A real-time adaptive control of autonomic computing environments. In *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 124–136, 2007.

[13] C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 31–44, New York, NY, USA, 2007. ACM.

[14] C. Stewart and K. Shen. Performance modeling and system management for multi-component online services. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 71–84, Berkeley, CA, USA, 2005. USENIX Association.

[15] T. Turicchi et al. Method for service level estimation in an operating computer system. *US Patent no. 2002/0082807.*

[16] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. Analytic modeling of multitier internet applications. *ACM Transactions on The Web*, 1(1):1–35, May 2007.

[17] L. Zhang, Z. Liu, A. Riabov, M. Schulman, C. Xia, and F. Zhang. A comprehensive toolset for workload characterization, performance modeling, and online control. *Computer Performance Evaluations, Modelling Techniques and Tools (Springer-Verlag LNCS)*, 2794:63–77, 2003.

[18] L. Zhang, C. Xia, M. Squillante, and W. M. III. Workload Service Requirements Analysis: A Queueing Network Optimization Approach. In *10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS)*, 2002.