

Exploiting Nonstationarity for Performance Prediction

Christopher Stewart
U. Rochester CS Dept.
stewart@cs.rochester.edu

Terence Kelly
Hewlett-Packard Labs
kterence@hpl.hp.com

Alex Zhang
Hewlett-Packard Labs
alex.zhang@hp.com

ABSTRACT

Real production applications ranging from enterprise applications to large e-commerce sites share a crucial but seldom-noted characteristic: The relative frequencies of transaction types in their workloads are *nonstationary*, i.e., the transaction mix changes over time. Accurately predicting application-level performance in business-critical production applications is an increasingly important problem. However, transaction mix nonstationarity casts doubt on the practical usefulness of prediction methods that ignore this phenomenon.

This paper demonstrates that transaction mix nonstationarity *enables* a new approach to predicting application-level performance as a function of transaction mix. We exploit nonstationarity to circumvent the need for invasive instrumentation and controlled benchmarking during model calibration; our approach relies solely on lightweight passive measurements that are routinely collected in today's production environments. We evaluate predictive accuracy on two real business-critical production applications. The accuracy of our response time predictions ranges from 10% to 16% on these applications, and our models generalize well to workloads very different from those used for calibration.

We apply our technique to the challenging problem of predicting the impact of application consolidation on transaction response times. We calibrate models of two testbed applications running on dedicated machines, then use the models to predict their performance when they run together on a shared machine and serve very different workloads. Our predictions are accurate to within 4% to 14%. Existing approaches to consolidation decision support predict post-consolidation *resource utilizations*. Our method allows *application-level performance* to guide consolidation decisions.

Categories and Subject Descriptors

C.4 [Performance of Systems]: Modeling techniques

General Terms

Experimentation, Management, Measurement, Performance

Keywords

enterprise, internet services, LAR regression, multi-tier, noninvasive, nonstationarity, performance prediction, realistic workloads

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EuroSys'07, March 21–23, 2007, Lisboa, Portugal.

Copyright 2007 ACM 978-1-59593-636-3/07/0003 ...\$5.00.

1. INTRODUCTION

Modern distributed applications continue to grow in scale and complexity. Distributed enterprise applications are furthermore assuming a growing role in business-critical operations. Understanding the performance of such applications is consequently increasingly difficult yet increasingly important due to their economic value. This paper considers the problem of performance prediction in distributed applications: Given forecasts of future application workload, we seek to predict application-level response times. A good solution to this problem will enable operators to explore a wide range of important “what-if” scenarios, e.g., “How will response times change if the the number visitors at my Web site doubles and the buy:browse ratio increases by 50%?” We do not address the complementary problem of workload forecasting, but we show that if accurate workload forecasts are available they can be mapped directly to accurate performance predictions.

The workloads of the real production applications that we seek to model share a crucial but seldom-noted characteristic: the transaction mixes in these workloads are highly *nonstationary* in the sense that the relative frequencies of transaction types vary considerably over time. This is a problem for most conventional performance models, which implicitly assume that transaction mix is stationary, because the system resource demands of different transaction types are usually very different in real applications.

Our approach leverages earlier work that focused on retrospectively *explaining* performance in terms of transaction mix [23]. We incorporate queueing-theoretic extensions into the earlier technique to obtain a method suitable for prospectively *predicting* future performance as a function of transaction mix. One novel feature of our approach is that whereas performance models in prior systems literature include a *scalar* measure of workload intensity, we describe workload using a transaction-mix *vector*. Another novel feature is that we exploit transaction mix nonstationarity to circumvent the need for invasive instrumentation and controlled benchmarking during model calibration.

Our approach is practical for real production systems and can be applied to a wide range of applications. Our models are calibrated using purely passive measurements that are routinely collected in today's real production applications. Furthermore, they work well under a wide range of workload conditions and a wide variety of application architectures, including locally distributed multi-tier E-commerce applications and globally-distributed high-availability enterprise applications.

We compare our proposed method with several alternatives, evaluating their ability to predict response times in two very different real production applications: the Web shopping site of a major retailer and a business-critical internal enterprise application. Our method accurately predicts response times for both applications.

Furthermore our performance models generalize well to regions of workload space very different from those present in the calibration data. We demonstrate that transaction mix models achieve substantially greater accuracy than similar models that employ scalar measures of workload intensity.

Finally, we apply our method to the challenging problem of predicting response times in applications that are *consolidated* onto a shared infrastructure, subject to a severe handicap: we must calibrate our models using only lightweight passive observations of the applications running on dedicated machines prior to consolidation. We evaluate our performance predictions in consolidated environments using a testbed of benchmark applications, since real production applications were unavailable for experimentation. Our predictions are remarkably accurate according to two measures that penalize inaccuracy in very different ways. The current state of the art in consolidation decision support both in practice and in the research literature predicts the *resource utilization* effects of consolidation. We present a practical way to incorporate *application-level performance* into consolidation decision-making.

The remainder of this paper is organized as follows: Section 2 describes the prevalence of transaction mix nonstationarity in real-world workloads, the problems it poses for many conventional performance models, and the opportunities it creates that we exploit. Section 3 presents our approach to performance prediction, defines our main accuracy measure, and describes an accuracy-maximizing model calibration procedure. Section 4 describes the applications used in our tests and presents empirical results on the accuracy of our predictions. Section 5 applies our models to the challenging problem of predicting the performance of applications that are consolidated onto a shared infrastructure. Section 6 reviews related work, and Section 7 concludes with a discussion.

2. TRANSACTION MIX NONSTATIONARITY IN REAL WORKLOADS

It is well known that the *volume* of demand in production applications naturally fluctuates on several time scales (e.g., daily and weekly cycles). Similarly, there is little reason for the transaction *mix* of real applications to remain constant over time. In this section, we describe *transaction mix nonstationarity* in two real production applications (Section 4.1 describes the applications themselves in detail). An investigation into the factors that influence nonstationarity in real applications is orthogonal to our goal of performance prediction, so we leave it for future work.

Figures 1 and 2 illustrate time variations in transaction mix. The first is a scatterplot of the relative frequencies of the two most common transaction types of the “VDR” application in 5-minute time windows. Note that nearly every possible combination is present (the upper right corner of the plot must be empty because the sum of the two fractions cannot exceed 1). Figure 2 is a time series of the fraction of “ACME” transactions that are of type “add-to-cart” in 5-minute windows. It shows that this fraction varies over two orders of magnitude during a four-day period (note that the vertical scale is logarithmic). The transaction mix nonstationarity evident in these figures is not an artifact of 5-minute time windows; it remains when we aggregate measurements into much longer intervals. Figure 4 shows the fraction of VDR transactions due to the most common transaction type in hour-long windows over a period of several days; the fraction ranges from less than 5% to over 50%. Plots using longer aggregation intervals are qualitatively similar.

One implication of transaction mix nonstationarity is that the full spectrum of workloads for which we must predict performance may not be available during model calibration. Performance models

must therefore *generalize* well to workloads that are very different from those used for calibration. Furthermore, a convincing validation of a performance prediction method requires nonstationary workloads, because stationary workloads differ qualitatively from real-world workloads.

Synthetic workload generators used in benchmarking and systems research typically employ first-order Markov models to determine the sequence of transactions submitted by client emulators; examples include the standard TPC-W workload generator [47] and the RUBiS workload generator [38]. This approach cannot yield the kind of markedly nonstationary workloads that we observe in real production applications, because the long-term relative state occupancy probabilities of first-order Markov processes are stationary [43]. Figure 5 shows the relative fractions of the two most common transaction types in the workload generated by the default RUBiS generator during a 5-hour run, in 5-minute windows. *Nearly all of the 60+ data points lie on top of one another.* Plots of different transaction type pairs aggregated into different time windows are qualitatively similar.

What are the implications of nonstationarity for performance modeling? We define a *scalar performance model* as one that ignores transaction mix in workload and instead considers only a scalar measure of workload intensity, e.g., arrival rate. Nonstationarity clearly poses serious problems for scalar performance models. For example, consider an application whose workload consists of equal numbers of two transaction types: type A, which places heavy demands on system resources, and type B, which has light demands. Suppose that we want to predict the application’s performance if the total number of transactions increases by 50%. Scalar models may work well if the relative proportion of the two transaction types remains equal. However such models are unlikely to yield accurate predictions if the transaction mix changes: Performance will differ dramatically if the number of type-A transactions doubles and the number of type-B remains constant, or vice versa. Of course, evaluations of scalar performance models using first-order Markov workload generators will not expose this problem. *Stationary test workloads mask the deficiencies of scalar performance models.*

This paper employs *transaction mix models* that predict application-level performance based on transaction counts by type. These models have a number of attractive features: they are “semantically clear” in the sense that their free parameters have intuitive interpretations; they yield accurate performance predictions under a wide range of circumstances; and the computational procedures used for model calibration are fairly straightforward. However it is nonstationarity that makes our approach particularly practical, because nonstationarity allows us to calibrate our models using only lightweight passive measurements that are collected in today’s real production environments. We describe the opportunities that nonstationarity creates for calibration in greater detail in Section 3.5, after we describe our performance models.

3. TRANSACTION MIX MODELS

This section describes our transaction mix performance models and several variants and alternatives against which we shall later compare them. All models have the same general high-level form:

$$P = F_{\vec{a}}(\vec{W}) \quad (1)$$

where P is a scalar summary of application performance, F specifies the functional form of our models, \vec{a} is a vector of calibrated parameter values, and \vec{W} is a vector of workload characteristics. This section explains the development of our approach. Section 3.1 justifies our basic assumptions in terms of the measured properties

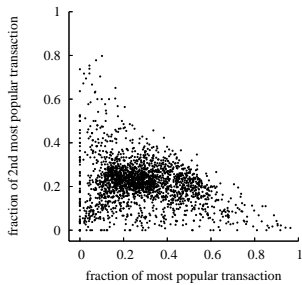


Figure 1: Fractions of VDR transactions, two most common types. Each point represents a different 5-minute interval.

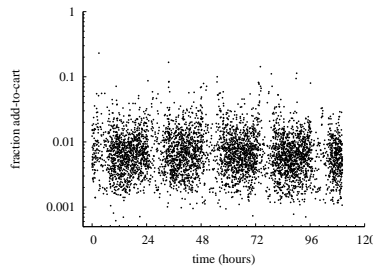


Figure 2: Fraction of ACME “add-to-cart” transactions vs. time. Each point represents a different 5-minute interval.

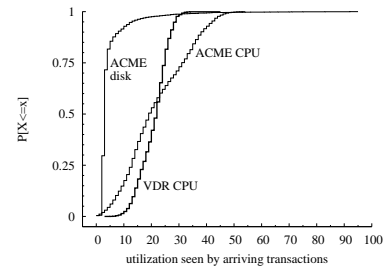


Figure 3: CDFs of resource utilizations encountered by arriving transactions in ACME and VDR applications.

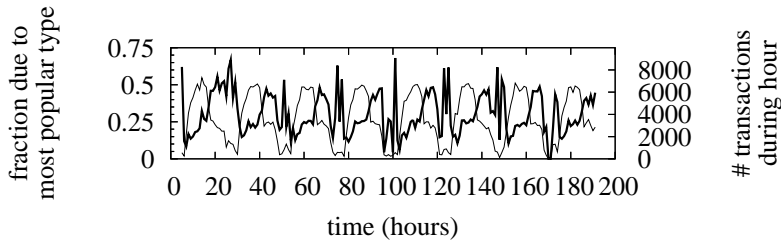


Figure 4: Fraction of VDR transactions due to most common type (heavy line); total transaction volume (light line), 1-hr windows.

of real applications. Section 3.2 presents our performance models. Section 3.4 defines the accuracy measure that we seek to optimize, and Section 3.5 explains how we calibrate our models to maximize accuracy according to this measure.

3.1 Assumptions

We begin with the following observations about modern distributed applications:

1. Workload consists of request-reply *transactions*.
2. Transactions occur in a small number of *types* (e.g., “log in,” “browse,” “add-to-cart,” “checkout” for an E-commerce site).
3. Transaction types strongly influence system resource demands (e.g., “checkout” transactions at an E-commerce site require more CPU than browsing).
4. Resources are adequately provisioned or over-provisioned in business-critical production applications.
5. Transaction mix is nonstationary.

The first two observations apply to every commercially-important distributed production application that we have encountered. The third property arises because transaction types often determine the run-time code path through application logic, which in turn strongly influences resource service demands. The fourth property, adequate resource provisioning, is a fundamental requirement of capacity planning in business-critical applications. By design, allocated capacity is generous relative to offered workload; heavy load and overload represent serious failures of the capacity planning process. Fortunately such failures are rare because capacity planning for intra-enterprise applications can often exploit good estimates of the total user population and anticipated usage patterns.

Even in server-consolidation scenarios where elevating resource utilization is an explicit goal, practitioners are advised to keep *peak* utilizations of resources such as CPU below 70% [14]. In practice, enterprise system operators are typically even more cautious than

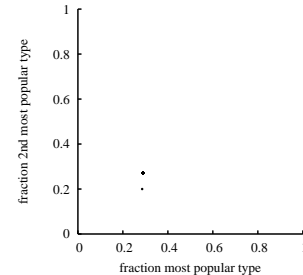


Figure 5: Fraction of RUBiS transactions due to two most common types using default generator. The figure is not empty; note the tight cluster of points at coordinates (0.3, 0.3). This workload is highly stationary.

this conservative guideline. Figure 3 shows cumulative distributions of resource utilizations encountered by arriving transactions in the two distributed production applications used in our investigation, “ACME” and “VDR.” Transactions arriving at these two very different applications operated by two different firms rarely find utilization at any resource in excess of 35%; utilization greater than 50% is almost never encountered. This implies that while *queueing* times at resources such as CPUs and disks should not be ignored, *service* times will often account for much of overall transaction response times. Another implication is that non-queueing congestion effects associated with very heavy load, e.g., cache interference, are likely to be rare in practice.

Together with our first three assumptions, Figure 3 suggests a radically simple performance model that accounts for transaction service times but ignores queueing entirely. Our previous work reports that such a model works surprisingly well in practice. Specifically, the sum of response times across all transactions within a specified time interval is well explained by transaction mix alone [23]. However, Figure 3 also suggests that waiting times are sometimes non-negligible, and our approach in this paper models waiting times.

In summary, we observe that transaction mix alone is a powerful performance predictor; it will be the \vec{W} of Equation 1. We have also seen that queueing can be non-negligible, and our models will explicitly account for waiting times in addition to service times. Our performance measure P will be aggregate transaction response time within short time windows (e.g., 5-minute intervals); this can easily be converted to *average* response time because we know the number of transactions within each window. After specifying the form of our models F and defining our measures of model accuracy, we describe how to obtain accuracy-maximizing parameters \vec{a} by exploiting naturally-occurring nonstationarity.

3.2 Models

This section develops a series of three performance models of increasing sophistication and breadth of applicability. The Basic

model of Section 3.2.1 takes into account transaction mix alone; it is taken from previous work [23]. Section 3.2.2 extends the Basic model to explicitly incorporate queueing delays. The Extended model does not conform to the template of Equation 1, however, because its inputs include resource utilizations as well as transaction mix. While the Extended model may offer improved accuracy when used to retrospectively *explain* performance, it cannot be used to *predict* performance given workload forecasts alone. The Composite model of Section 3.2.3 corrects this deficiency by modeling resource utilizations in terms of transaction mix and incorporating the utilizations thus obtained into the Extended model. Finally, our empirical evaluations will include variants of the Basic, Extended, and Composite models that use only a scalar measure of workload intensity rather than a vector describing transaction mix.

3.2.1 Basic Model

We divide time into short non-overlapping intervals, e.g., 5 minutes. For interval i let N_{ij} denote the number of transactions of type j that began during the interval and let T_{ij} denote the sum of their response times. Our Basic model has the form

$$y_i = \sum_j T_{ij} = \sum_j \alpha_j N_{ij} \quad (2)$$

where y_i is the sum of all transaction response times during interval i . Note that no intercept term is present in Equation 2, i.e., we constrain the model to pass through the origin: Aggregate response time must be zero for intervals with no transactions. Values of model parameters α_j are obtained through model calibration; let a_j denote these calibrated values. Intuitively, calibrated parameters a_j represent typical *service* times for the various transaction types, summed over all service and delay centers on the transaction's execution path.

For given model parameters a_j and observed transaction mix N_{ij} at time i , let

$$\hat{y}_i = F_{\hat{a}}(\vec{N}_i) = \sum_j a_j N_{ij} \quad (3)$$

denote the *fitted value* of the model at time i . If the N_{ij} represent past workload, \hat{y}_i can be interpreted as the model's guess of what aggregate response time should have been during interval i . If instead the given transaction mix is a forecast of future workload, the fitted value represents the model's performance prediction. Note that since the total number of transactions within an interval is known—it is simply $\sum_j N_{ij}$ —one can convert a fitted value \hat{y}_i representing *aggregate* response time into an *average* response time.

Our Basic model can be thought of as an open queueing network containing a single service station with an infinite number of servers. Waiting cannot occur in such a system, and Equation 2 does not explicitly model waiting times.

3.2.2 Extended Model

We extend the Basic model of Equation 2 by adding terms representing waiting times, as follows:

$$y_i = \sum_{j=1}^n \alpha_j N_{ij} + \sum_r \left(\frac{1}{\lambda_i} \cdot \frac{U_{ir}^2}{1-U_{ir}} \right) \cdot \sum_{j=1}^n N_{ij}. \quad (4)$$

The rightmost term represents waiting times in an M/M/1 queue, with one queue per resource; U_{ir} denotes the utilization of resource r during interval i . The naïve approach of adding utilizations as simple linear terms has no basis in queueing theory, but we shall compare our approach with this alternative (see the discussion of Table 4 in Section 4.2.1).

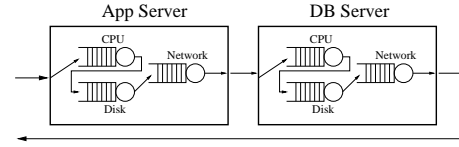


Figure 6: Extended queueing model: One station per resource at each tier.

To derive Equation 4, we note that the term $\frac{1}{\lambda_i} \cdot \frac{U_{ir}^2}{1-U_{ir}}$ represents the average waiting time (per transaction at resource r over interval i) in an M/M/1 queueing model, where λ_i is the arrival rate of transactions of all types in interval i . We multiply this term by the number of transactions of all types in interval i to obtain the sum of waiting times, to agree with the left-hand-side of the equation. Realizing that $\lambda_i = \sum_{j=1}^n N_{ij}/L$ where L is the interval length in seconds, one can further simplify the sum-of-waiting-time term to $\sum_r L \cdot \frac{U_{ir}^2}{1-U_{ir}}$. Finally, since the sum-of-waiting-time term on the right-hand-side of Equation 4 does not involve any unknown parameters α_j , one can regress $z_i \equiv y_i - \sum_r L \cdot \frac{U_{ir}^2}{1-U_{ir}}$ against the transaction mix $\sum_{j=1}^n \alpha_j N_{ij}$.

Figure 6 depicts our Extended model as an open queueing network consisting of a single-server station for each resource (e.g., CPU, disk, network) at each tier (e.g., application server, database server). Although the figure shows only two tiers with three resources each, the model can accommodate additional tiers (e.g., for a Web server) and additional resources. The Extended model captures the distributed aspect of an application by explicitly including network queueing effects.

3.2.3 Composite Model

Because it relies on resource utilizations, the Extended model of Equation 4 cannot be used to predict performance based on transaction mix N_{ij} alone. Our Composite model overcomes this difficulty by estimating utilizations as weighted sums of transaction counts:

$$U_{ir} = \beta_{0r} + \sum_j \beta_{jr} N_{ij} \quad (5)$$

where β_{jr} represents the service demand of transaction type j on resource r . The total service demand placed on the resource is the sum of service demands of all transaction types. As with the Basic response time model of Section 3.2.1, we obtain for each resource r parameters b_{jr} corresponding to the β_{jr} during model calibration. Unlike the model of Equation 2, however, we include an intercept term β_{0r} in our utilization models, because real system resources are not entirely idle even in the complete absence of application workload. Equation 5 generalizes the familiar Utilization Law; specifically, it reduces to the Utilization Law in the special case of one transaction type and no intercept term.

Once we have obtained utilization estimates \hat{U}_{ir} from a calibrated *utilization* model (Equation 5), we substitute these into a calibrated *Extended* model to obtain a *Composite* model of aggregate response time as a function of transaction mix N_{ij} . In rare cases where $\hat{U}_{ir} < 0$ or $\hat{U}_{ir} \geq 1$, we correct the utilization estimate to zero or $1 - \epsilon$, respectively.

3.2.4 Scalar Models

Recent queueing models of distributed application performance rely on a *scalar* measure of workload intensity that ignores transaction types [41, 48]. What additional predictive power do we obtain by using a transaction-mix vector? To address this question, our empirical evaluations will compare our models with Scalar variants

that use only the total *number* of transactions in each time interval. For example, the Scalar variant of the Basic model is

$$y_i = \sum_j T_{ij} = \alpha N_i \quad (6)$$

where $N_i = \sum_j N_{ij}$ is the total number of transactions that occurred during time interval i .

3.3 Discussion

Our approach contains a number of simplifications that deserve mention. We account for waiting times at each resource using an expression for a single-server queue, whereas many real production applications run on systems with multiple CPUs and disks at each tier. Previous work made similar simplifying assumptions [41], and we find that in practice this approach works well. Our model assumes an open network in which requests exit after service. A closed network model would require us to model client “think times,” as in some previous models of distributed applications [28, 48].

Our queueing networks implicitly assume that transactions do not recirculate among resources; our models aggregate all service times from all of a transaction’s visits to a resource, rather than explicitly modeling visits separately. More sophisticated queueing models take into account recirculation among service stations [20], but such models require detailed information about how transactions move among resources, which is often not available in practice. Tools to gather this information exist as research prototypes, e.g., Magpie [7], but few real production systems are currently instrumented to measure fine-grained transaction resource visits. We have designed our models to require for calibration only data that is routinely collected on today’s real production applications.

Our use of open queuing network models is in part motivated by practical considerations: It is often difficult in practice to obtain for real production applications the client session information required to calibrate closed models, and therefore the use of an open model facilitates more thorough empirical validation than would be possible if we employed a closed model. However there are good reasons for preferring open models for their own sake, e.g., they are relatively simple. More importantly, open models are more appropriate if transient overloads are possible because open models do not inherently restrict the number of concurrent transactions in a system. See Schroeder *et al.* for a detailed discussion of the implications of open versus closed models [40].

For all of the production and testbed applications considered in this paper, transaction types are given—they can easily be inferred by, e.g., inspecting the request URL. In our experience such transaction type information is sufficient for our method and is readily available in practice. Our approach remains applicable if transaction types are not given as long as a classifier can be constructed that maps transactions to types that reflect their system resource service demands. Transaction type classification is an orthogonal problem to our modeling interests and has been the subject of extensive research (see Section 6.1).

We have implicitly assumed that an application’s set of transaction types is fixed and the relationship between transaction type and resource demands is stable. This is not a restrictive assumption in practice because the time required to re-calibrate new performance models is short compared to the time scales on which application logic and transaction structure changes. In our experience with real production applications in the enterprise, changes to application structure and configuration are normally rare; stakeholders in business-critical applications do not undertake such modifications lightly or frequently. Even if transaction types or their resource demands change completely, it takes only a day or two to gather suf-

ficient data to calibrate completely new performance models. Less drastic changes are easier to handle: Model calibration itself takes less than a second, and therefore continuous re-calibration (e.g., at the conclusion of every 5-minute measurement interval) can be used to track gradual drift in the workload/performance relationship.

A final simplification is that our models ignore interaction effects across transaction types and implicitly assume that queueing is the only manifestation of congestion. However queueing does not describe certain kinds of resource contention, e.g., cache interference. “Checkout” transactions, for instance, may require more CPU *service* time during heavy browsing if the latter reduces processor cache hit rates for the former. Our models do not account for such effects. The question of whether our simplifying assumptions are *oversimplifications* is ultimately an empirical one, which we address in Section 4.

3.4 Accuracy Measures

If y_i is the actual measured aggregate response time during interval i and \hat{y}_i is the fitted value obtained from a calibrated performance model, let $e_i = y_i - \hat{y}_i$ denote the *residual* (model error) at time i . We measure model accuracy in terms of intuitive functions of the residuals. We cannot use the conventional coefficient of multiple determination R^2 to assess model accuracy; it is not meaningful because Equation 2 and Equation 4 lack intercept terms [33, p. 163].

Our overall figure of merit, normalized aggregate error, generalizes the familiar, intuitive concept of absolute percent error:

$$\text{normalized aggregate error} \equiv \frac{\sum_i |e_i|}{\sum_i y_i} \quad (7)$$

Consider, for example, a *single* (y, \hat{y}) pair: if $y = 100$ and $\hat{y} = 105$, then normalized aggregate error is 0.05, indicating that the model’s prediction is off by 5%. We say that model parameters for Equation 2 or Equation 4 are *optimal* if they minimize error as defined by Equation 7.

In addition to our overall figure of merit we shall also report the distribution of normalized residuals $|e_i|/y_i$, scatterplots of (y, \hat{y}) pairs, and order statistics on the normalized residuals. Each of these measures offers different insight into model accuracy and penalizes inaccuracy in a different way. For example, the distribution of $|e_i|/y_i$ penalizes even small residuals if the corresponding measurements are small, whereas Equation 7 penalizes residuals whose magnitude is large even if $|e_i|$ is small relative to the corresponding y_i . A good model is accurate according to both measures.

3.5 Calibration

The input to calibration is a data set consisting of aggregate response times y_i and transaction mixes $\vec{N}_i = (N_{i1}, N_{i2}, \dots)$. For our Extended and Composite models we furthermore require resource utilizations U_{jr} . These inputs correspond to readily available and purely passive measurements of applications and their underlying system resources. A good rule of thumb is that model calibration requires roughly ten times as many measurement intervals i as transaction types [33]. If measurements are taken at 5-minute intervals, a few days suffice to collect enough data to calibrate our models of the production applications that we study.

The output of calibration is a set of calibrated parameter values a_j corresponding to the α_j parameters of the Basic and Extended models and, for a Composite model, calibrated parameter values b_{jr} corresponding to the β_j parameters of the utilization model (Equation 5).

The goal of calibration is to compute parameters that maximize model accuracy. The denominator in Equation 7 is a constant, so to achieve optimal accuracy a calibrated model must minimize the numerator, i.e., the sum of absolute residuals. This is a special case of linear programming, for which specialized variants of the simplex algorithm have been developed; we use the algorithm of Barrodale & Roberts [8]. The algorithm yields model parameters that optimize retrospective explanatory accuracy with respect to the data used for calibration. This exercise is sometimes known as *least absolute residuals* (LAR) regression. Ordinary least squares (OLS) regression minimizes the sum of *squared* residuals, and it can be shown that a model with OLS parameters can have *arbitrarily worse* accuracy than an optimal model according to the measure of Equation 7. In practice we find that LAR-calibrated models are more accurate than their OLS-calibrated counterparts.

Another advantage of LAR is that it is *robust*, i.e., it resists the influence of extreme values in the calibration data set. By contrast, OLS is far more sensitive to distortion by outliers. A wide variety of robust regression procedures are available; several are variants of OLS and LAR [34, 52]. We prefer plain-vanilla LAR because it guarantees optimal retrospective accuracy, because it is conceptually simple and easy to explain, and because it involves no tunable parameters. The only disadvantage of LAR is that numerical solvers are not as widely available. However, as reported previously, the accuracy gain over OLS outweighs the inconvenience of LAR [23]. A final advantage of using linear programming for model calibration is that it is easy to add additional constraints, e.g., on the values of parameters. Extensions of elementary statistical techniques such as least-squares regression can sometimes achieve similar capabilities, but in our experience they do not offer the generality, convenience, and flexibility of linear programming.

3.5.1 The Role of Nonstationarity

Model calibration in our approach exploits variations in transaction volume, transaction mix, and resource utilization in the calibration data—i.e., the kind of nonstationarity found in real workloads. At the other extreme, it is easy to show that lightweight passive measurements of response times and utilizations collected under perfectly *stationary* workload cannot be used for model calibration. The essential difficulty is that the optimization problem that the regression procedure seeks to solve lacks a unique solution. Typical implementations of OLS regression, for example, fail in such cases because they attempt to invert a singular (non-invertible) matrix.

A simple example conveys intuition for the insurmountable problems created by perfectly stationary transaction mix. Consider the following utilization and transaction mix measurements:

time interval i	utilization U	number of transactions	
		type A	type B
1	u_1	5	7
2	u_2	10	14
3	u_3	15	21
4	u_4	20	28

These data cannot be used to calibrate a utilization model (Equation 5) *regardless of the calibration procedure used*, and *regardless of the utilization measurements u_i* . The problem is that the transaction count information is essentially the same in each row because the counts in rows 2 through 4 are multiples of those in row 1 and the A:B ratio is everywhere the same—i.e., the transaction mix is stationary. This makes it impossible to determine whether A is a heavyweight transaction and B is lightweight or vice versa. The situation is the same if the data include aggregate response times rather than utilization measurements and our goal is to calibrate a Basic model (Equation 2) or an Extended model (Equation 4).

If a first-order Markov model generates workload for calibration and if measurement intervals contain a reasonably large number of requests, the result will almost certainly be *nearly-stationary* workload. This in turn causes *multicollinearity*, a regression pathology that arises when predictor variables are mutually correlated. The net effect is that predictive accuracy will suffer regardless of the regression procedure used. On the other hand, our empirical results show that naturally-occurring workloads have sufficient transaction mix nonstationarity to allow us to calibrate very accurate models using passive measurements of utilizations and response times.

In summary, the nonstationarity of real workloads makes possible our lightweight model calibration approach, which relies on passive measurements and requires no invasive system or application instrumentation or controlled benchmarking. *Nonstationarity allows model calibration to substitute data analysis for invasive measurement.*

4. VALIDATION

We calibrate our models and evaluate their retrospective explanatory accuracy on the first half of each validation data set. We then apply the calibrated models to the transaction mix in each time interval i of the second half to obtain fitted values \hat{y}_i . Finally, we compare these \hat{y}_i with observed y_i to evaluate prospective prediction accuracy. This section first describes the two real production data sets used in our evaluation and an additional data set collected in a lab environment, then presents our results.

4.1 Data Sets

Table 1 summarizes our data sets. The first two were collected on distributed production applications serving real customers and real enterprise users, respectively. Together these data sets severely challenge the ability of our method to generalize along several dimensions. They differ in terms of the nature of the application, the extent of geographic distribution, and workload. Their workloads exhibit the nonstationarities described in Section 2. Due to space limitations we provide an abbreviated description of the data sets here; additional information on all three data sets is available in [24].

Pronounced workload periodicity is present in all of our data sets. Referring to the light data series and right-hand vertical axis in Figure 4, we see that workload ranges from under 100 to roughly 7,300 transactions per hour. The ACME data set shows a similar daily cycle with comparably wide variation. Other periodicities and nonstationarities are present in all of our data sets.

4.1.1 ACME: DotCom-Era Web Shopping

The “ACME” data set is taken from the busiest of seven servers comprising a major US retailer’s Web shopping site. This server accounts for roughly 38% of all ACME transactions during the measurement period. ACME was typical of large E-commerce sites circa 2000. For confidentiality reasons, the researchers who studied the ACME site were not permitted to disclose the details of its hardware and software infrastructure. However, an extensive workload characterization is available [3]. The ACME data set includes measurements of transaction response times and system resource utilizations collected at the application server tier; it does not include requests for static Web pages. Each transaction is furthermore tagged as a cache hit or a miss, and we treat hits and misses of the same transaction type as two different types.

4.1.2 VDR: Modern Enterprise Application

Figure 7 depicts the architecture of the globally-distributed VDR application. VDR is a high-availability business-critical internal

Data Set	Production Dates	Duration	Transactions:		Trans'n's/min		Resp time (sec)		Type of Application
			Number	Types	Mean	Median	Mean	Median	
ACME	July 2000	4.6 days	1,180,430	93	182.2	183.0	0.929	0.437	Web Retail Shopping
VDR	Jan 2005	7.8 days	666,293	37	59.4	56.4	1.289	1.236	Business-critical Enterprise
PetStore	April 2004	38 hours	4,920,642	10	2147.8	3163.8	0.096	0.040	Sample application

Table 1: Summary of production data sets.

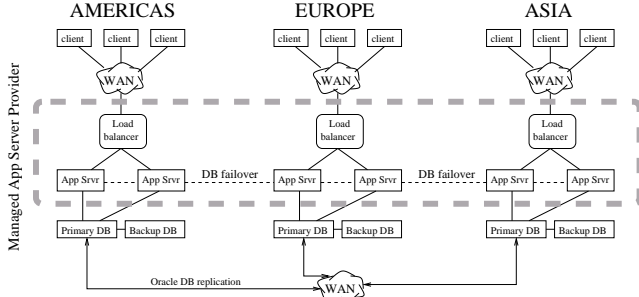


Figure 7: VDR application architecture.

HP application serving both external customers and HP users on six continents. Its system architecture therefore incorporates redundancy and fail-over features both locally and globally. Regional hubs in Atlanta, Swindon, and Singapore respectively serve the Americas, Europe, and Asia regions. All application server hosts are HP 9000/800 servers running HP-UX B.11.11. The Americas region has two such hosts with 16 CPUs and 64 GB RAM each. The European region has three, with 16 CPUs and 32 GB RAM each, and the Asia region has two, with 12 CPUs and 20 GB RAM each. All of the app servers ran BEA WebLogic. We have less detailed information about hosts at the database tier, but we know that they are similar in number and specifications to those at the app server tier and that they ran Oracle 9i.

VDR operators and system architects have told us that VDR transactions are relatively “heavyweight” in the sense that they place substantial demands on system resources. The VDR data set includes both transaction records and system resource utilization measurements collected at *both* application server and database server tiers, derived respectively from application-level logs and OpenView Performance Agent (OVPA).

4.1.3 PetStore: Sample Application

The PetStore data set was collected on a testbed application serving a highly variable and highly nonstationary workload. The total request rate and the ratios of transaction type frequencies vary sinusoidally, and the total rate increases over time so that the peaks represent transient overloads (“flash crowds”) during which offered workload exceeds the system capacity. Our models implicitly assume that load does *not* exceed system capacity, so this data set provides an extraordinary challenge to our approach. The PetStore data set and the environment in which it was collected are described in detail in earlier publications [15, 24]; due to space limitations we do not repeat these descriptions here. The PetStore data set was generated long before the present investigation began and therefore was not (consciously or otherwise) tailored to the methodology proposed in this paper.

4.2 Results

We calibrate our models on the first half of each data set, and also measure their retrospective explanatory accuracy on the first half. We then evaluate the predictive accuracy of the calibrated models using the transaction mixes in the second half of each data set. Transaction mix nonstationarity therefore implies that predic-

	VDR		ACME		PetStore	
	Scalar	TMIX	Scalar	TMIX	Scalar	TMIX
Basic						
OLS	0.1485	0.1002	0.2034	0.1308	0.4403	0.3605
LAR	0.1462	0.0940	0.2029	0.1281	0.3646	0.3230
Extended						
OLS	0.1478	0.0997	0.2012	0.1213	0.2320	0.2897
LAR	0.1454	0.0936	0.2006	0.1185	0.1978	0.2221

Table 2: Retrospective explanatory accuracy: Normalized aggregate error $\sum_i |e_i| / \sum_i y_i$.

tive evaluations will involve workloads different than those used in calibration. Nonstationarities provide a challenging and credible evaluation of the extent to which models generalize beyond the calibration data.

4.2.1 Retrospective Explanation

Table 2 summarizes the explanatory accuracy of eight models on our three data sets according to our overall figure of merit, normalized aggregate error (Equation 7). The table includes both the standard transaction mix model (TMIX) versions of our Basic and Extended models as well as Scalar variants that ignore transaction types and use only the total number of transactions. Both OLS and LAR regression are used to calibrate each model variant.

First, the transaction mix models consistently outperform their scalar counterparts by a wide margin for the production data sets: 15% vs. 10% error for VDR and 20% vs. 13% error for ACME regardless of the calibration procedure. Even for the deliberately overloaded PetStore, which violates flow balance, the Extended transaction mix model calibrated with LAR has only a 22% error. *We achieve substantially greater accuracy by exploiting transaction mix.*

Second, the Extended model, which accounts for queueing, outperforms the Basic model, which does not. Our Basic model achieves nearly the same accuracy for both production applications as the Extended model. This is what we expect because queueing is deliberately minimized in production applications (see Figure 3 and the discussion in Section 3.1). For the heavily-loaded PetStore data, in which queueing is a first-order effect, the Extended model performs substantially better than the Basic model (22% vs. 32% when LAR is used).

Third, normalized aggregate error is lower when LAR regression is used. Outliers (extreme data values) are present in both data sets and are particularly numerous and large in the PetStore data. LAR is a robust estimation procedure and is less sensitive to outliers than OLS, and therefore yields more accurate models according to our accuracy measure.

We also evaluated model variants that include an intercept term in Equations 2 and 4. Such models are “wrong” from a queueing-theoretic perspective because they imply nonzero aggregate response times even when no transactions occur. However an intercept can increase retrospective accuracy but cannot reduce it, so we might be tempted to include one if we are willing to trade “sanity” for accuracy. We found that the benefits of including an intercept are very limited, and that the principled models with no intercept are nearly as accurate.

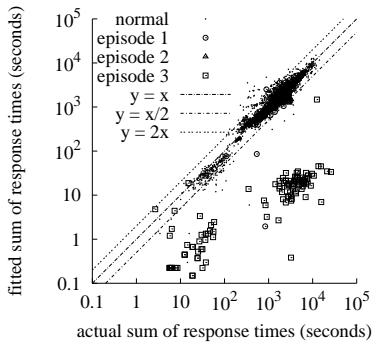


Figure 8: A performance anomaly in the “FT” application.

	VDR		ACME		PetStore	
	Scalar	TMIX	Scalar	TMIX	Scalar	TMIX
$\frac{\sum_i e_i }{\sum_i y_i}$						
Basic	0.1621	0.1226	0.1749	0.1470	0.6528	0.6257
Composite	0.1606	0.1218	0.1723	0.1305	0.3702	0.4173
median						
$ e_i /y_i$						
Basic	0.1418	0.0963	0.1724	0.1378	0.5247	0.5056
Composite	0.1420	0.0931	0.1664	0.1230	0.1708	0.2365

Table 3: Prospective prediction accuracy: Normalized aggregate error $\sum_i |e_i|/\sum_i y_i$ and median $|e_i|/y_i$.

4.2.2 Application: Performance Anomaly Detection

Our previous work explains how accurate retrospective explanatory performance models can be useful [23]. The most obvious application is *performance anomaly detection*, i.e., identifying when performance is surprising, given workload. Knowing whether workload explains performance can guide our choice of diagnostic tools: Ordinary overload might recommend bottleneck analysis, whereas degraded performance *not* explained by workload might suggest a fault in application logic or configuration.

We have shown that a real performance bug episode in a real distributed production application appears as a prominent performance anomaly. Figure 8 shows a scatterplot of (y_i, \hat{y}_i) pairs generated by a Basic model of the “FT” application during a period when application operators reported episodes of a performance bug due to a misconfiguration in a concurrency parameter. One episode corresponds to the prominent clusters of points in the lower right corner of the figure. FT is a globally-distributed enterprise application that resembles VDR in several respects. See [23] for details on the FT application and on this case.

Model calibration takes under one second for large data sets, so our method can be used to detect anomalies in real time by simply recomputing a new model at the conclusion of each time interval (e.g., every 5 minutes) using a large moving window of historical data (e.g., from the previous week or month). The data point corresponding to the most recent interval may then be deemed anomalous if the overall accuracy of the model is good but the most recent performance observation y_i disagrees substantially with the model’s fitted value \hat{y}_i .

4.2.3 Prospective Prediction

Table 3 summarizes predictive accuracy results for our Basic and Composite models calibrated using LAR regression; the table also includes Scalar variants of both models. We do not present results for OLS calibration because they do not alter the qualitative conclusions we drew from Table 2: LAR works better, sometimes by a substantial margin.

In addition to normalized aggregate error, Table 3 shows the alternative accuracy measure discussed in Section 3.4: the median of the distribution of normalized absolute residuals $|e_i|/y_i$. The two measures differ in how they penalize inaccuracy. Normalized aggregate error severely punishes even a single large residual but may “forgive” many small residuals, even those where $|y_i - \hat{y}_i|$ is large in relation to y_i . Our other accuracy measure, median $|e_i|/y_i$, has the opposite tendency: It forgives a large residual if the corresponding y_i is also large, but it penalizes us if the residual is often large in relation to y_i .

Our prospective prediction results are consistent with our retrospective explanation results: First, transaction mix models (TMIX) outperform their Scalar counterparts by a very large margin for both real production applications by both accuracy measures. Even the Basic TMIX achieves normalized aggregate error under 15% for both real production data sets, and its individual performance predictions \hat{y}_i are within 14% of the true value y_i half of the time. The Basic model leaves little room for improvement when applied to real production applications; as we would expect, the Composite model offers relatively modest accuracy improvements for the real production applications, which are intentionally very lightly loaded (Figure 3). Queueing delays are likely to be small in relation to service times in such situations, so we gain relatively little by modeling queueing. However for the deliberately *overloaded* PetStore the situation is very different, as we would expect: The Composite model consistently achieves far better accuracy by both of our accuracy measures.

The relative benefits of incorporating transaction mix and queuing in the testbed application and the production applications are consistent with our expectations: PetStore is heavily loaded and has few transaction types; as we expect, the greatest increase in accuracy occurs when we move from the Basic model to the Composite model. The production applications are lightly loaded and have many transaction types; not surprisingly, the greatest increase in accuracy occurs when we move from a Scalar model to a TMIX model.

The combined benefits of accounting for both queueing and transaction mix in production applications are striking. Compared with the TMIX/Composite models, the Scalar/Basic models have substantially worse normalized aggregate error: 33% greater error for VDR and 34% greater error for ACME. The differences are even larger when we consider median $|e_i|/y_i$: 52% greater error for VDR and 40% greater error for ACME.

Figures 9, 10, and 11 illustrate both retrospective and predictive accuracy for a Composite model of the VDR application calibrated with LAR regression. In all cases, retrospective fitted values or residuals are shown in blue and prospective fitted values are shown in red. Figure 9 presents a time series of observed aggregate response times y_i overlaid on fitted values \hat{y}_i . Overall, the latter track the former quite closely. Residuals are not markedly larger for prospective performance prediction than for retrospective performance explanation. For the VDR data set, our model generalizes well from historical data used for calibration to future workload data used for prediction.

Figure 10 shows a scatterplot of (y_i, \hat{y}_i) pairs. The three straight diagonal lines in the figure are the $y = x$ diagonal, indicating perfect prediction, flanked by $y = 2x$ and $y = x/2$. Although observed values y_i range over three orders of magnitude, fitted values \hat{y}_i almost always agree to within a factor of two, and are usually within 15% of the true y_i value. Finally, Figure 11 shows the full distributions of $|e_i|/y_i$ for both explanatory and predictive models. Our Composite model is highly accurate for retrospectively explaining performance, and the figure shows that residuals remain remark-

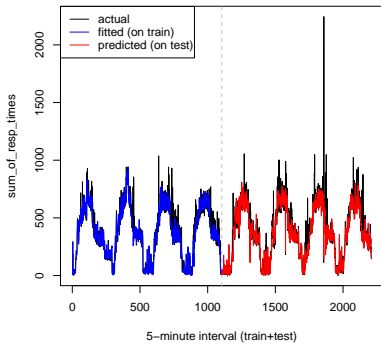


Figure 9: Time series of y_i and \hat{y}_i .

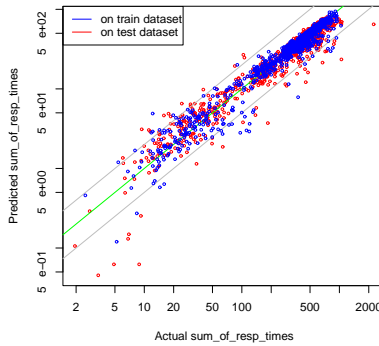


Figure 10: Scatterplot of (y_i, \hat{y}_i) pairs.

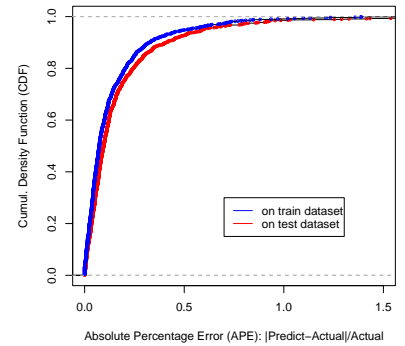


Figure 11: CDF of $|e_i|/y_i$.

	VDR	ACME	PetStore
Basic	0.1226	0.1470	0.6257
naïve U_r	0.1221	0.2193	0.5794
Composite	0.1218	0.1305	0.4173

Table 4: Incorporating utilization naïvely vs. correctly: Normalized aggregate error.

ably small when the model is used to predict performance based on transaction mix.

4.2.4 Generalizing to New Conditions

We performed an additional experiment to evaluate the ability of our Composite model to predict performance in a real application under workload conditions very different from those of the calibration data. We sorted the VDR data set in ascending order on application server CPU utilization. The first half of the resulting re-ordered data set contains time intervals during which CPU utilization on the app server was very low. The second half contains time intervals during which utilization was much higher. Figure 12 shows the distributions of CPU utilizations in both halves of the re-ordered VDR data set.

We calibrated a Composite model on the first half of the re-ordered VDR data, when load was very light (CPU utilization between 4% and 18%). We then evaluated the model’s predictive accuracy on the second half, when load was much heavier (utilization between 18% and 45%). The model’s predictive accuracy was remarkably high under this challenging test: normalized aggregate error $\sum_i |e_i|/\sum_i y_i$ was under 9.7%; most predictions \hat{y}_i were within 8% of the true value y_i . Furthermore, the model’s predictive accuracy did not vary with load: Figure 13 shows a scatterplot of error per time interval $|e_i|/y_i$ versus CPU utilization on the app server. The figure shows that accuracy is not noticeably worse under high utilization (i.e., there is no upward trend to the right).

4.2.5 Modeling Utilization

A nonspecialist in queueing theory might wonder why we do not simply incorporate resource utilizations into our performance model by adding *linear* U_r terms rather than the mysterious $U^2/(1-U)$ terms of Equation 4. Table 4 compares the predictive accuracy of our Basic and Composite models with a model that incorporates utilization in the naïve way. We see that the naïve approach sometimes improves upon our Basic model. However the “correct” approach of our Composite model yields still better accuracy. Several similar cases not reported here tend toward the same conclusion: Embellishing the Basic model in haphazard ways sometimes offers modest advantages, but amendments with sound theoretical justifications (as in our Extended and Composite models) yield better results overall.

5. PERFORMANCE-AWARE APPLICATION CONSOLIDATION

Enterprise systems that comprise multiple applications often execute each sub-application on separate machines to isolate the effects of software faults and workload spikes. Compared with such machine-granularity application boundaries, application consolidation (i.e., executing multiple applications on one machine) has several advantages including better resource utilization and lower management and maintenance overheads. However, workload fluctuations in consolidated environments can have complex effects on application-level performance that reduce the overall predictability of the system. In this section, we use our transaction mix model to predict application-level performance amidst contention for shared physical resources.

5.1 Consolidation Model

We predict system-wide performance in consolidated environments by combining the Composite transaction mix models of each target application running in isolation. We concatenate the transaction mix vectors and sum the resource utilizations in the Composite transaction mix models of the target applications. More formally, the system-wide sum of response times for two consolidated applications in interval i is:

$$\hat{y}_i = \sum_{j=1}^{n'+n''} \alpha_j N_{ij} + \sum_r \left(\frac{1}{\lambda_r} \frac{(U'_{ir} + U''_{ir})^2}{1 - (U'_{ir} + U''_{ir})} \right) \cdot \sum_{j=1}^{n'+n''} N_{ij}$$

where superscripts ($'$ and $''$) distinguish between the two applications. For brevity, we show a unified α_j and N_j which represent concatenation of each application’s individual parameters. The variable n represents the number of transaction types, U_r represents the utilization of resource r , and λ represents the post-consolidation arrival rate of both applications. Note that the unification of several Composite models can trivially be extended to handle the consolidation of more than two applications, though we do not provide empirical results on this more general case. We acknowledge that additive measures of resource utilization may not account for some additional costs of resource sharing (e.g., context switching). We have achieved accurate performance predictions without considering such effects, though their impact may become more significant as the degree of consolidation increases. Likewise, we assume an application’s resource requirements do not decrease after consolidation, which can happen if the target applications interact or share data.

The resulting transaction mix model can be manipulated to derive performance predictions for each application under consolidation by considering the sum of the transaction types corresponding to a target application. Specifically, we extract the per-application

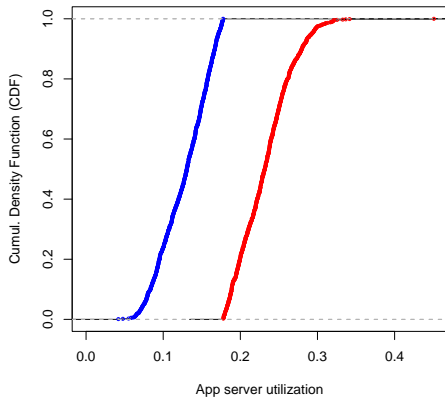


Figure 12: CDFs of app server CPU utilization in both halves of re-ordered VDR data set.

performance under consolidation as follows:

$$\hat{y}'_i = \sum_{j=1}^{n'} \alpha_j N_{ij} + \sum_r \left(\frac{1}{\lambda_i} \frac{(U'_{ir} + U''_{ir})^2}{1 - (U'_{ir} + U''_{ir})} \right) \cdot \sum_{j=1}^{n'} N_{ij}.$$

$$\hat{y}''_i = \sum_{j=n'+1}^{n'+n''} \alpha_j N_{ij} + \sum_r \left(\frac{1}{\lambda_i} \frac{(U'_{ir} + U''_{ir})^2}{1 - (U'_{ir} + U''_{ir})} \right) \cdot \sum_{j=n'+1}^{n'+n''} N_{ij}.$$

5.2 Realistic Workload Generation

Not surprisingly, the ACME and VDR administrators did not allow us to perform consolidation experiments using their applications, so we built our own testbed from benchmark applications. Unfortunately, the workload generators shipped with our benchmark applications produced stationary workloads, which, as we have seen, differ qualitatively from workloads observed in the wild. We developed our own workload generator which applies realistic nonstationary workloads to our testbed by mimicking the transaction type frequencies in the traces of real applications (e.g., ACME and VDR). First, we separately ranked the transaction types in both our real and testbed applications according to their popularity in the real trace and workload generator probabilities respectively. We created a synthetic trace that imitates the nonstationarity of real workloads by replacing each transaction type in the real trace with the transaction type with the same popularity rank in the testbed application. Finally, our workload generator also mimics the seasonality of real workloads by varying the request rate according to a sawtooth pattern. Henceforth references to a real application (i.e., ACME or VDR) preceded by “M-” will indicate a mimicked workload based on the named dataset.

5.3 Evaluation

Our testbed consists of two benchmark applications. The Rice University Bidding System (RUBiS) [38] is an online auction benchmark that consists of 22 transaction types that provide services such as browsing for items, placing bids, or viewing a user’s information. The StockOnline [42] trading system comprises six transaction types corresponding to buying, selling, viewing prices, viewing holdings, updating account information, and creating new users. Each application runs on top of the JBoss 4.0.2 application server [21] and accesses MySQL [32] as the back-end database. The application server and database run on separate machines. Resource consumption at the database is negligible in our testbed, so we focus on the consolidation of the application server tier.

Our experiments are run on a three-machine cluster. Each node consists of four 2.4-GHz CPUs with Intel Hyperthreading technology and 6 GB of main memory. We use the Linux 2.6.9 kernel

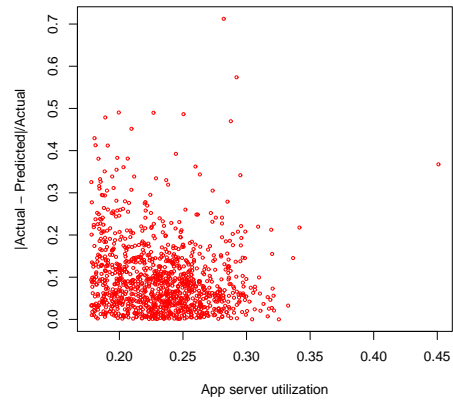


Figure 13: Prediction error $|e_i|/y_i$ versus app server CPU utilization in re-ordered VDR data set.

distributed by Red Hat. All of the experiments mentioned in this section were run for five hours. Matching our observations of the ACME workload, each test was configured to exhibit eight complete sawtooth fluctuations in request rate. Unless otherwise noted, request rates were set for low CPU utilization (15–25%) when each application ran in isolation. Measurements were collected every 30 seconds using the SAR system monitor. Each respective testbed application was calibrated using the M-ACME imitation workload.

Table 5 shows the accuracy of our consolidation predictions using the metrics discussed in Section 4.2.3. We show one consolidation scenario where the calibration and evaluation workloads are the same (M-ACME) and two scenarios in which the workload under consolidation differs from the calibration workload (M-ACME). Under the same calibration and evaluation workloads the aggregate error is 5.5% and 11% for RUBiS and Stock respectively. We also observe that the composition of transaction mix models for consolidation is robust to changes in workload; predicting performance on completely different workloads from the calibration set still yields aggregate errors within 9% and 14%. We note that the normalized errors for the StockOnline application are about twice that of RUBiS. Upon further investigation, we found that StockOnline is developed in a fashion such that transaction type does not provide much distinctive information about resource demands. In this sense, StockOnline violates one of our fundamental assumptions about applications, yet we are still able to report normalized aggregate errors of between 11% and 14%.

Figure 14 shows the cumulative distribution of absolute percent error when the RUBiS and StockOnline applications are consolidated and subjected to the M-VDR workloads. More than 97.6% and 74.2% of RUBiS and StockOnline predictions respectively are within 20% actual response time. The calibration and evaluation of StockOnline on different workloads represents a significant challenge for our model yet 98% of performance predictions are within 40% of the actual response time.

We also wish to demonstrate the robustness of our consolidation prediction under heavy load. We increased the workload intensity by ramping up utilizations such that CPU utilizations reached 70% on the application server. Figure 15 shows the cumulative distribution of the normalized residual error in our heavy-load consolidation scenario. We report normalized aggregate error of 12.8% and 11.7% for RUBiS and StockOnline respectively. While these results indicate the robustness of our consolidation model, we note the limitations of our queueing model: As we noted with the Pet-Store application, our predictions are not intended for systems in which a resource is frequently saturated. Consolidation experiments under such situations gave performance predictions with un-

Consolidated RUBiS Workload	Consolidated Stock Workload	RUBiS Accuracy		Stock Accuracy	
		Normalized Aggregate Error ($\frac{\sum e_i }{\sum y_i}$)	Median of Normalized Absolute Residuals ($ e_i /y_i$)	Normalized Aggregate Error ($\frac{\sum e_i }{\sum y_i}$)	Median of Normalized Absolute Residuals ($ e_i /y_i$)
M-ACME	M-ACME	0.0549	0.0383	0.1064	0.1263
M-ACME	M-VDR	0.0724	0.0706	0.1391	0.1184
M-VDR	M-VDR	0.0810	0.0626	0.1349	0.1145

Table 5: Application-level performance prediction accuracy in consolidated environments. Predictions are based on measurements of each application in isolation. In all cases model calibration employed the M-ACME workload.

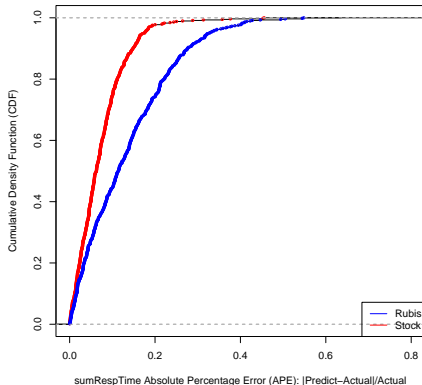


Figure 14: CDF of absolute percent error ($|e_i|/y_i$) under different calibration and evaluation workloads.

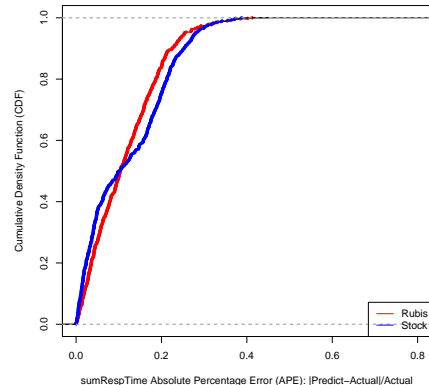


Figure 15: CDF of absolute percent error ($|e_i|/y_i$) under heavy non-saturating workload.

acceptable (41%) normalized standard error; since this result is expected it is not shown here.

6. RELATED WORK

This section reviews literature on several topics related to our work: We begin with a survey of workload characterization studies that have identified regularities in modern application workloads and then consider workload generators used in research and in commercial benchmarking. We then review instrumentation tools and techniques that can be used to calibrate performance models, theoretical performance models themselves, and the application of such models to practical performance prediction problems. Finally, we summarize the state of the art in server consolidation research and practice. A review of literature on an important application of our modeling technique, performance anomaly detection, is available in [23]. A review of literature on LAR regression is available in [24].

6.1 Workload Characterization

Previous research has characterized many aspects of workload in modern transactional applications, e.g., Web server workloads [4] and Web user sessions [2]. Menasce *et al.* propose first-order Markov models of customer accesses at e-commerce sites [30] and describe statistically self-similar arrival patterns at such sites [29]. Some forms of nonstationarity have been observed in workloads, e.g., in Internet traffic [12] and in the diurnal cycles of requests to Web sites [5]. However to the best of our knowledge nonstationarity in the mix of application-level transaction types is not considered in previous research.

Many characterization studies lead directly to prescriptions for improved performance. Breslau *et al.* show that the Zipf-like distribution of Web document access frequencies implies that a certain

Web cache removal policy is optimal [11]. Arlitt & Williamson find support for size-based Web cache removal policies in the size distributions of Web documents [5]. Jung *et al.* characterize transient overload events (“flash crowds”) and suggest ways for system defenses to distinguish them from denial-of-service attacks [22].

Clustering techniques are frequently employed to simplify workload for performance modeling [6]. Often workload units are clustered according to their resource demands as in Magpie [7] and in Esposito *et al.* [18], but occasionally clustering is applied to other aspects of workload, e.g., customers [51].

Our contribution is to recognize that in many modern applications transaction types effectively cluster workload elements according to their resource demands, and that nonstationarity in transaction mix presents an opportunity to calibrate performance models without invasive instrumentation or controlled benchmarking.

6.2 Workload Generation

Commercial synthetic workload generators are used to evaluate the performance of live applications and systems [31] and in standard benchmarks such as TPC-W [47]. To the best of our knowledge, such tools generate workloads with stationary transaction mixes. Schroeder *et al.* survey workload generators used in research, emphasizing the distinction between open and closed generators [40]. These tools allow users to configure arrival rate parameters but do not facilitate the generation of workloads with nonstationary transaction mixes. To the best of our knowledge, the only workload generator that does so is the SWAT tool of Krishnamurthy *et al.* [25, 26]. SWAT employs sophisticated mathematical programming techniques to construct a mix of sessions that conforms to user-specified aggregate workload criteria; nonstationarity follows from the use of recorded sessions from a real production system. SWAT strives to provide both the fidelity of simple trace

replay with the control and configurability of conventional workload generators.

Our contribution is to recognize that the stationary workloads produced by conventional workload generators make parameter estimation for performance modeling *more difficult* than naturally-occurring nonstationary workloads. Indeed, our results show that controlled experiments using synthetic workload generators are not necessary to calibrate performance models. Accurate performance models may be calibrated using the kinds of lightweight passive measurements routinely collected in today’s production systems—transaction logs and utilization logs.

We furthermore report that the simple expedient of replaying transaction logs from real production applications with the transaction types re-named to suit a testbed application (Section 5.2) yields good results.

6.3 Instrumentation & Measurement

A wide range of commercial performance measurement tools are available. Sauers *et al.* provide a candid survey of the strengths and limitations of one vendor’s products [39]. Some tools provide insight into transaction execution paths and resource usage noninvasively—without application source code modifications—via instrumented middleware [19]. If source code is available, applications may be systematically instrumented to record more detailed transaction execution information, e.g., using the ARM instrumentation standard [45]. Instrumentation to characterize transaction resource demands in greater detail and with lower overhead remains an active research area [7, 27, 46].

Our contribution is to recognize that transaction mix nonstationarity in real-world workloads enables us to use very lightweight measurements to characterize the resource demands of transaction types for the purpose of calibrating performance models.

6.4 Queueing Theory

Queueing networks are the subject of a large theoretical literature; see Bolch *et al.* for a lengthy survey [10]. Jain describes applications of queueing theory to computer system performance analysis [20]. The approaches that Jain surveys differ from ours in several key respects: Jain emphasizes the design of controlled experiments for performance analysis, and an underlying assumption throughout much of the book is that systematic benchmarking is possible. Furthermore most of Jain’s queueing network models assume far more detailed information about transaction behavior than is available in many practical situations. For instance, it is frequently assumed that the number of times a transaction visits various resources and the distribution of service times at each station can be measured directly. Our work proceeds from the assumption that lightweight passive measurements of transaction response times and resource utilizations are all that is available.

Operational analysis is a branch of queueing theory that attempts to avoid probabilistic assumptions about system workload (e.g., Poisson arrivals and exponentially-distributed service times) and rely solely upon measurable quantities [17]; Little’s Law and the Utilization Law are classic examples of operational laws. Mean Value Analysis (MVA) restricts attention to the averages (as opposed to the full distributions) of performance measures [35]. Rolia & Sevcik introduce a variant of MVA designed to accommodate *software* servers in addition to conventional hardware service stations [36]. Like conventional MVA, this method pertains to closed queueing networks, whereas we employ open network models.

Generalizations of queueing-theoretic models and MVA address *multiclass* networks [9]. Workload classes are often interpreted as categories such as “batch,” “terminal,” and “transaction,” but

classes can also be used to represent different transaction types. One problem with existing multiclass methods for our purposes is that most assume a closed network—our production traces do not include sufficient information about client sessions for us to employ a closed model. Another problem is that the computational cost of computing exact solutions to MVA using conventional algorithms increases rapidly with the number of classes. A more efficient algorithm has appeared recently [13] but it is formidably complex and difficult to implement. Multiclass models with more than a handful of classes are rarely used in practice due to their complexity and the computational cost of computing solutions.

Our contribution is to introduce a computationally tractable and conceptually simple performance model for open networks that takes transaction mix into account, that models multiple service centers, that is easy to calibrate, and that yields accurate response time predictions for real production applications.

6.5 Applied Performance Prediction

This section reviews in depth two recent papers that apply queueing models to distributed applications, highlighting similarities and contrasts with respect to our work. We refer the reader to their excellent literature reviews for recent, broad, and thorough surveys of related work in this field [41, 48].

Urgaonkar *et al.* model multi-tier Internet services as product-form queueing networks and employ mean value analysis to compute average response times [48]; in some respects this work is similar to that of Liu *et al.* [28]. The Urgaonkar *et al.* model assumptions differ from ours in several details. For instance, Urgaonkar *et al.* explicitly model concurrency limits whereas we do not. We assume an open queueing network whereas Urgaonkar *et al.* assume a closed network. We explicitly model distinct physical resources such as CPUs and disks whereas Urgaonkar *et al.* associate a single queue with each tier. The models differ in their assumptions about how requests recirculate among tiers; compare our Figure 6 with their Figure 3 [48, p. 294]. An important difference is that their method requires more diverse model parameter estimates than ours, including request visit ratios at each tier, service times at each tier, user think times, and certain other parameters related to congestion effects. Urgaonkar *et al.* report that their approach yields accurate average response time estimates for two sample applications (*RUBiS* and *Rubbos*) subjected to stationary synthetic workloads in a testbed environment; they do not report validation results on real production applications.

Stewart & Shen present a performance model of distributed Internet applications based on “profiles” that summarize how application software components and their workloads place demands on underlying system resources [41]. Their model also accounts for inter-component communications and component placement. This work shares some features in common with our approach. For instance, Stewart & Shen account for waiting times at servers using an *M/G/1* model; we employ a similar model in Equation 4. They estimate the resource demands of components by fitting linear models to benchmark data. However, they describe workload by a constant scalar arrival rate, whereas we use a time-varying vector of per-type transaction counts. Stewart & Shen report that their most sophisticated model variant predicts average response times to within 14%. Their validation uses testbed applications (*RUBiS* and *StockOnline*) and stationary synthetic workloads.

An important difference with respect to our work is that the method of Stewart & Shen requires very extensive calibration: The resource consumption profile of each component must be estimated via controlled benchmark experiments, and inter-component communication overheads must also be measured. They place each

profiled component *on a dedicated machine* during calibration and require at least one benchmark run per component. For their full model, $O(N^2)$ benchmark runs are required to estimate pairwise inter-component communication costs [41, p. 75]. We exploit non-stationarity to obtain similar performance profiles using only lightweight passive measurements of running production systems: The coefficients of our utilization model (Equation 5) correspond closely to those in the “component resource profiles” of Stewart & Shen (see Figure 2 and Tables 1 and 2 in [41]). Another difference is that we do not require knowledge of internal application component structure; we use only externally-visible transaction types.

We emphasize two important differences between our evaluation experiments and those presented in Urgaonkar *et al.* and in Stewart & Shen. First, as noted above, we have employed two real production traces for our evaluations; they have used only testbed applications. The workload of our applications is *nonstationary* in several key parameters, including both workload intensity and transaction mix. By contrast, Stewart & Shen and Urgaonkar *et al.* employ synthetic workloads reminiscent of classic steady-state benchmarks both for model calibration *and for evaluation*. The transaction mixes in their synthetic workload (e.g., the buy:browse ratio in their synthetic e-commerce workloads) remain *constant* during both calibration and evaluation. We believe that our nonstationary workload yields a far more challenging and more realistic test of a performance model’s generalizability and predictive accuracy.

Our empirical evaluations could not include comparisons with the methods of Stewart & Shen and of Urgaonkar *et al.* for two reasons: First, the input-output behavior of the three models is sufficiently different to preclude a true apples-to-apples comparison. More importantly, the other two approaches require far more extensive calibration data than is available in our data sets (our current testbed at HP does not permit the same instrumentation as used in Stewart & Shen; e.g., kernel modifications are not allowed). However we do compare our preferred approach with alternatives that, like the models of Stewart & Shen and of Urgaonkar *et al.*, employ a scalar measure of workload intensity (Section 3.2.4). We found that transaction mix models offer substantially higher accuracy than their Scalar counterparts (Section 4.2).

6.6 Consolidation

The computing trends of the 1980s and 1990s led to decentralized IT infrastructures that can be more difficult to manage and less cost-effective than their centralized predecessors. Server consolidation attempts to increase resource utilization while reducing the costs of hardware, data center floor space, power, cooling, and administration. The resource cost benefits of consolidation alone are potentially attractive: Andrzejak *et al.* studied CPU utilization in six enterprise data centers containing roughly 1,000 CPUs and found that consolidation could reduce the number of CPUs needed during peaks by 53% and the mean number required by 79% [1].

Administrators know that systems can be overloaded if the sum of consolidated application resource demands is excessive. Practitioners are advised to rely on rough guidelines for total resource utilization, e.g., “avoid peak CPU utilization over 70%” [14]. Commercial capacity planning decision support aids may employ more sophisticated time-series analysis of pre-consolidation historical data, but their suggestions are based on considerations of resource utilizations [44, 50].

Recent research on consolidation decision support also bases recommendations on resource utilization. Rolia *et al.* analyze historical utilization data to provide statistical guarantees on post-consolidation utilization [37]. Urgaonkar *et al.* profile applications on dedicated nodes to estimate resource demands and “pack” appli-

cations to maximize revenue while controlling the potential for resource overload [49].

The state of the art in research and in practice is to base consolidation decisions on considerations of resource utilizations and either ignore application-level workload or model it as a scalar quantity. This is problematic because the relationship between application-level performance and utilization is complex, and because both depend on transaction mix. Our contribution is a practical way to obtain accurate predictions of *response times* in transactional applications, thus allowing consolidation decisions to consider application-level performance as well as system resource utilization.

7. CONCLUSIONS

The global geographic distribution, organizational decentralization, opaque component structures, and unprecedented scale of modern application architectures confound performance modeling in challenging new ways. Performance prediction in business-critical applications, however, remains an important problem due to the growing economic importance of these applications. This paper presents a practical, versatile, and accurate approach to predicting application-level response times in complex modern distributed applications. Our method exploits naturally-occurring workload non-stationarity to circumvent the need for invasive instrumentation or controlled benchmarking for model calibration. It relies solely on measurement data that is routinely collected in today’s production environments. Our method can be adapted to a wide range of applications, and calibrated models generalize well to new regions of workload/performance space. It is novel in its use of transaction mix to predict performance, and we have shown that transaction mix is a far more powerful predictor of application performance under realistic conditions than scalar workload volume.

Our empirical results show that our method predicts response times in real production applications to within 16% by two very different accuracy measures. A model of a real production application calibrated under light load predicts performance under heavy load to within 10%. Our results show that if accurate workload forecasts are available, they can be mapped directly to accurate performance predictions. Furthermore we predict response times of consolidated applications to within 4% to 14% based on passive pre-consolidation measurements, even when workload changes dramatically between calibration and evaluation. Whereas existing approaches to consolidation decision support consider only resource utilization, our approach enables application-level response times to guide consolidation decisions.

Acknowledgments

Martin Arlitt supplied the ACME data set. Ira Cohen, Julie Symons, and the second author collected the FT and PetStore data sets for separate projects [15, 16]. We thank the operators of the ACME, FT, and VDR production systems for providing anonymized trace data. Hsiu-Khuern Tang answered questions on statistical matters. Arjun Nath provided valuable assistance to our testbed experiments. We are deeply grateful to Narayan Krishnan and Eric Wu for their extraordinarily assistance in setting up and administering the cluster we used for consolidation tests. We thank David Oppenheimer, Jerry Rolia, and Bhuvan Urgaonkar for many insightful discussions of performance modeling and its applications, and we thank Kim Keeton, Kai Shen, Zhikui Wang, Xiaoyun Zhu, Sharad Singhal, and the anonymous reviewers for reading drafts and offering many helpful suggestions. The first author acknowledges support from the U.S. National Science Foundation CAREER Award CCF-0448413.

8. REFERENCES

- [1] A. Andrzejak, M. Arlitt, and J. A. Rolia. Bounding the resource savings of utility computing models. Technical Report HPL-2002-339, HP Labs, Dec. 2002.
- [2] M. Arlitt. Characterizing Web user sessions. *Performance Evaluation Review*, 28(2), Sept. 2000.
- [3] M. Arlitt, D. Krishnamurthy, and J. A. Rolia. Characterizing the scalability of a large web-based shopping system. *ACM Trans. on Internet Tech.*, 1(1):44–69, Aug. 2001.
- [4] M. Arlitt and C. Williamson. Web server workload characterization: The search for invariants. In *SIGMETRICS*, May 1996.
- [5] M. F. Arlitt and C. L. Williamson. Internet Web servers: Workload characterization and performance implications. *IEEE/ACM Trans. on Networking*, 5(5):631–644, Oct. 1997.
- [6] H. P. Artis. Capacity planning for MVS computer systems. *Performance Evaluation Review*, 8(4):45–62, 1979.
- [7] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modelling. In *OSDI*, pages 259–272, Dec. 2004.
- [8] I. Barrodale and F. Roberts. An improved algorithm for discrete L1 linear approximations. *SIAM Journal of Numerical Analysis*, 10:839–848, 1973.
- [9] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM*, 22(2):248–260, Apr. 1975.
- [10] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains*. Wiley, 1998.
- [11] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *INFOCOM*, Mar. 1999.
- [12] J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun. On the nonstationarity of Internet traffic. In *SIGMETRICS*, pages 102–112, June 2001.
- [13] G. Casale. An efficient algorithm for the exact analysis of multiclass queueing networks with large population sizes. In *SIGMETRICS*, pages 169–180, June 2006.
- [14] A. Cockcroft and B. Walker. *Capacity Planning for Internet Services*. Sun Press, 2001.
- [15] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *OSDI*, Oct. 2004.
- [16] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering, and retrieving system history. In *SOSP*, Oct. 2005.
- [17] P. J. Denning and J. P. Buzen. The operational analysis of queueing network models. *ACM Computing Surveys*, 10(3):225–261, Sept. 1978.
- [18] A. Esposito, A. Mazzeo, and P. Costa. Workload characterization for trend analysis. *Performance Evaluation Review*, 10(2), July 1981.
- [19] Hewlett-Packard. OpenView Transaction Analyzer, Sept. 2006. <http://h20229.www2.hp.com/products/tran/>.
- [20] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.
- [21] The JBoss J2EE Application Server. <http://www.jboss.com>.
- [22] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: characterization and implications for CDNs and Web sites. In *WWW*, May 2002.
- [23] T. Kelly. Detecting performance anomalies in global applications. In *USENIX WORLDS*, Dec. 2005.
- [24] T. Kelly and A. Zhang. Predicting performance in distributed enterprise applications. Technical Report HPL-2006-76, HP Labs, May 2006. <http://www.hpl.hp.com/techreports/2006/HPL-2006-76.html>.
- [25] D. Krishnamurthy, J. A. Rolia, and S. Majumdar. SWAT: A tool for stress testing session-based Web applications. In *Computer Measurement Group Conf.*, Dec. 2003.
- [26] D. Krishnamurthy, J. A. Rolia, and S. Majumdar. A synthetic workload generation technique for stress testing session-based systems. *IEEE Trans. Software Engineering*, 32(11):868–882, Nov. 2006.
- [27] J. R. Larus. Whole program paths. In *PLDI*, May 1999.
- [28] X. Liu, J. Heo, and L. Sha. Modeling 3-tiered web applications. In *Proc. MASCOTS*, Sept. 2005. <http://www.cs.mcgill.ca/~xueliu/publications/MASCOTS05Modeling.pdf>.
- [29] D. Menasce, V. Almeida, R. Riedi, F. Ribeiro, R. Fonseca, and W. Meira. In search of invariants for e-business workloads. In *ACM E-Commerce Conf.*, Oct. 2000.
- [30] D. A. Menasce, V. A. F. Almeida, R. Fonseca, and M. A. Mendes. A methodology for workload characterization of e-commerce sites. In *ACM E-Commerce Conf.*, Nov. 1999.
- [31] Mercury Interactive. Loadrunner load tester, Sept. 2006. <http://www.mercury.com/us/products/performance-center/loadrunner/>.
- [32] MySQL database. <http://www.mysql.com>.
- [33] J. Neter, M. H. Kutner, C. J. Nachtsheim, and W. Wasserman. *Applied Linear Statistical Models*. Irwin, fourth edition, 1996.
- [34] C. R. Rao and H. Toutenburg. *Linear Models: Least Squares and Alternatives*. Springer, 1999.
- [35] M. Reiser and S. Lavenberg. Mean value analysis of closed multichain queueing networks. *J. ACM*, 27(2):313–322, Apr. 1980.
- [36] J. A. Rolia and K. C. Sevcik. The method of layers. *IEEE Trans. Software Engineering*, 21(8):689–700, Aug. 1995.
- [37] J. A. Rolia, X. Zhu, and M. Arlitt. Resource access management for a resource utility for commercial applications. In *Int'l Sympos. on Integrated Network Mgmt. (IM)*, Mar. 2003.
- [38] Rice University Bidding System (RUBiS), Mar. 2004. <http://rubis.objectweb.org/>.
- [39] R. F. Sauer, C. P. Ruemmler, and P. S. Weygant. *HP-UX 11i Tuning and Performance*. Prentice Hall, 2004.
- [40] B. Schroeder, A. Wierman, and M. Harchol-Balter. Open versus closed: A cautionary tale. In *NSDI*, pages 239–252, May 2006.
- [41] C. Stewart and K. Shen. Performance modeling and system management for multi-component online services. In *NSDI*, pages 71–84, May 2005.
- [42] The StockOnline Benchmark. <http://forge.objectweb.org/projects/stock-online>.
- [43] D. W. Stroock. *An Introduction to Markov Processes*. Springer, May 2005.
- [44] R. Talaber. Server consolidation assessments with VMware CapacityPlanner, Oct. 2005. <http://download3.vmware.com/vmworld/2005/pac196.pdf>.
- [45] The Open Group. Application response measurement (ARM), Sept. 2006. <http://www.opengroup.org/management/arm>.
- [46] E. Thereska, B. Salmon, J. Strunk, M. Wachs, M. Abd-El-Malek, J. Lopez, and G. R. Ganger. Stardust: Tracking activity in a distributed storage system. In *Proc. SIGMETRICS*, pages 3–14, June 2006.
- [47] Transaction Processing Performance Council. TPC-W benchmark, Apr. 2005. <http://www.tpc.org/tpcw/>.
- [48] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An analytical model for multi-tier Internet services and its applications. In *SIGMETRICS*, pages 291–302, June 2005.
- [49] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *OSDI*, Dec. 2002.
- [50] VMWare. Server capacity planning and consolidation, Oct. 2005. http://www.vmware.com/news/releases/vac_services.html.
- [51] Q. Wang, D. J. Makaroff, and H. K. Edwards. Characterizing customer groups for an e-commerce website. In *ACM E-Commerce Conf.*, May 2004.
- [52] R. R. Wilcox. *Introduction to Robust Estimation and Hypothesis Testing*. Elsevier, second edition, 2005.