Predicting Performance in Distributed Enterprise Applications

Terence Kelly, Alex Zhang
Enterprise Software and Systems Laboratory
HP Laboratories Palo Alto
HPL-2006-76
May 4, 2006*

performance modeling, response time, distributed applications, capacity management

Distributed applications play an increasingly crucial role in business-critical enterprise operations. Understanding the performance of such applications is challenging yet essential due to their growing economic value. A particularly important problem is *performance prediction*: How will application-level performance vary in response to changes in workload? This paper presents a practical and accurate approach to predicting response times as a function of workload mix in complex modern distributed applications. We compare our approach against several alternatives, evaluating their ability to predict the performance of two large, real business-critical production systems and a testbed application subjected to extremely variable synthetic workload. Our results show that our method yields accurate response time predictions under a wide range of conditions, and that our models generalize well to previously-unseen regions of workload/performance space.

* Internal Accession Date Only
Approved for External Publication

Predicting Performance in Distributed Enterprise Applications

Terence Kelly and Alex Zhang*
Hewlett-Packard Laboratories
1501 Page Mill Road m/s 1125
Palo Alto, CA 94304 USA

Abstract

Distributed applications play an increasingly crucial role in business-critical enterprise operations. Understanding the performance of such applications is challenging yet essential due to their growing economic value. A particularly important problem is *performance prediction*: How will application-level performance vary in response to changes in workload? This paper presents a practical and accurate approach to predicting response times as a function of workload mix in complex modern distributed applications. We compare our approach against several alternatives, evaluating their ability to predict the performance of two large, real business-critical production systems and a testbed application subjected to extremely variable synthetic workload. Our results show that our method yields accurate response time predictions under a wide range of conditions, and that our models generalize well to previously-unseen regions of workload/performance space.

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 performance. A good solution to this problem will enable operators to explore a wide range of important "what-if" scenarios, e.g., "How will mean response times at my E-commerce site change if the the number of shoppers doubles and the buy:browse ratio increases by 50%?" While this paper does not address the complementary problem of workload forecasting, we show that if accurate workload forecasts are available they can be mapped directly to accurate performance predictions.

Our approach leverages earlier work that focused on retrospectively *explaining* performance in terms of the mix of transactions in workload [13]. 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 literature include a *scalar* measure of workload intensity, we describe workload using a transaction-mix *vector*.

The advantage of our method is that it is both practical and general: Our performance models can be calibrated using purely passive measurements that are routinely collected in today's real production applications. Furthermore our approach can be applied to 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 experimentally compare our proposed method with several alternatives, evaluating their ability to predict response times in three very different applications: the Web shopping site of a major retailer, a business-critical internal enterprise application, and a testbed application serving extraordinarily heavy and variable workload. Our empirical results show that our method accurately predicts response times for the full range of applications considered. Our method remains accurate under light, moderate, and heavy workload. Furthermore our performance models generalize well to regions of workload/performance space very different from those present in the calibration data. Finally, we demonstrate that transaction mix models achieve substantially greater accuracy than similar models that employ scalar measures of workload intensity.

The remainder of this paper is organized as follows: Section 2 presents our approach to performance prediction, defines our main accuracy measure, and describes an accuracy-maximizing model calibration procedure. Section 3 characterizes the networked applications used in our tests and presents our empirical evaluation results. Section 4 reviews related work, and Section 5 concludes with a discussion.

*kterence@hpl.hp.com alex.zhang@hp.com

$Id: wpm.tex,v 1.59 2006/04/27 06:55:48 kterence Exp $

2 Performance Models

This section describes our preferred performance prediction model, which we call the "Composite Model," and several variants and alternatives that we shall later compare it against. 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 parameters, and $\vec{W}$ is a vector of workload characteristics. This section explains the development of our approach. Section 2.1 justifies our basic assumptions in terms of the measured properties of enterprise applications. Section 2.2 presents our performance models. Section 2.4 defines the accuracy measure that we seek to optimize, and Section 2.5 explains how we calibrate our models to maximize accuracy according to this measure.

2.1 Assumptions

Our approach begins with the following observations about modern distributed enterprise 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 enterprise applications.
5. Crucial aspects of workload are statistically *nonstationary*, i.e., the frequency distributions of key workload characteristics vary dramatically over time.

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 they 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% [8]. In practice, enterprise system operators are typically even more cautious than this conservative guideline. Figure 1 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.

Together with our first three assumptions, Figure 1 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 [13]. However, Figure 1 also suggests that waiting times are sometimes non-negligible, and our approach in this paper models waiting times.

Nonstationarity, our final observation, is an important difference between real-world workloads and conventional synthetic benchmark workloads. Whereas most benchmarks hold the ratios of different transaction types constant over time, we observe that the relative prevalence of different transaction types varies considerably in the wild. For example, Figure 2 shows the fraction of transactions in the ACME application that are of type "add-to-cart" in non-overlapping 5-minute time windows. This fraction ranges over two orders of magnitude, from 0.1% to over 10%, and shows considerable fluctuation on short time scales.

Nonstationarity has important implications for the design, calibration, and validation of performance models. Obviously, models must not require or implicitly assume statistically stationary workload. Furthermore, the full spectrum of workloads for which we must predict performance will *not* be available during calibration. Models must therefore *generalize* well to workloads that are very different from those used for calibration. Finally, a convincing validation requires nonstationary workloads; conventional stationary benchmarks are not adequate.

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

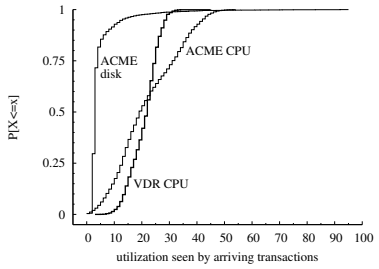

Figure 1: CDFs of resource utilizations encountered by arriving transactions in two real applications.

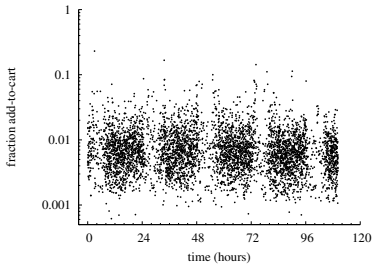

Figure 2: Nonstationarities in ACME workload: "add-to-cart" fraction versus time.

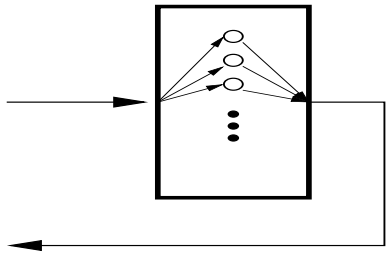

Figure 3: Basic queueing model: single station, infinite servers.

window. After specifying the form of our models F and defining our measures of model accuracy, we describe how to obtain accuracy-maximizing parameters $\vec{a}$.

2.2 Models

This section develops a series of three performance models of increasing sophistication and breadth of applicability. The Basic model of Section 2.2.1 takes into account transaction mix alone; it is taken from previous work [13]. Section 2.2.2 extends the Basic model to explicitly incorporate queueing delays. The Extended model does not conform to the template of Equation 1, however, because it requires 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 alone. The Composite model of Section 2.2.3 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.

2.2.1 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.

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

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

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.

Figure 3 depicts our Basic model graphically 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.

2.2.2 Extended Model

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

$$y_i = \sum_j T_{ij} = \sum_j \alpha_j N_{ij} + \left(\sum_j N_{ij}\right)\left(\sum_r \alpha_r \frac{U_{ir}}{(1-U_{ir})}\right) \quad (4)$$

The rightmost terms represent 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 3.2.1).

Figure 4 depicts our Extended model as an open queueing network consisting of a single-server station for each resource class (e.g., CPU, disk, network) at each tier (e.g.,

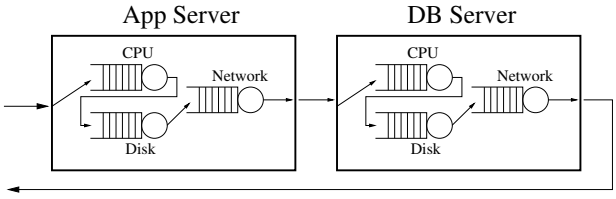

Figure 4: Extended queueing model: one station per resource at each tier.

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.

2.2.3 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 2.2.1, 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.

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 $1 - \varepsilon$, respectively.

2.2.4 Scalar Models

Recent queueing models of distributed application performance rely on a *scalar* measure of workload intensity that ignores transaction types [16,18]. 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.

2.3 Discussion

In its present form, our approach contains a number of simplifications that deserve mention. We account for waiting times at each resource type using an expression for a single-server queue, whereas many real production applications run on systems with multiple CPUs and disks at each tier. Like others who have made similar simplifying assumptions [16], 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 [18].

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 [12], 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 [3], 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.

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 sufficient 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 3.

2.4 Accuracy Measures

If y_i is the 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 the model accuracy; it is not meaningful because Equation 2 and Equation 4 lack intercept terms [14, p. 163].

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

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

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 main 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.

2.5 Calibration

The input to calibration is a data set consisting of aggregate response times y_i and transaction mixes $\vec{N}_i = (N_{i1}, N_{i2}, \ldots)$. For our Extended and Composite models we furthermore require resource utilizations U_{ir}. These inputs correspond to readily available and purely passive measurements of applications and their underlying system resources. The predictive accuracy of our models benefits from variations in transaction volume, transaction mix, and resource utilization in the calibration data. In our experience a few hundred time intervals i are required for good results. If measurements are taken at 5-minute intervals, it takes a day or two to collect sufficient data for calibration.

The output of calibration is a set of parameters a_j for the Basic model. We additionally obtain a_r for an Extended model and, for a Composite model, b_{jr} for the utilization model of 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 [4]. 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 substantially 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 [15, 19]. We prefer plain-vanilla LAR because it guarantees optimal retrospective accuracy, and because it is conceptually simple and easy to explain. The only disadvantage of LAR is that numerical solvers are not as widely available. However, as reported previously, the substantial accuracy gain over OLS outweighs the inconvenience of LAR [13].

3 Empirical Evaluation

Our evaluation methodology is to divide each data set into a prefix consisting of the first 50% of all time intervals in the data set and a suffix consisting of the remainder. We calibrate our models and evaluate their retrospective explanatory accuracy on the prefix. We then apply the calibrated models to the transaction mix in each time interval i of the suffix 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 three very different data sets used in our experiments, then presents our results.

3.1 Data Sets

Table 1 summarizes our three data sets. The first two are distributed production applications serving real customers

Data Set	Production Dates	Duration	Transactions: Number	Types	Trans'ns/min Mean	Median	Resp time (sec) Mean	Median	Type of Application
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
Testbed	April 2004	38 hours	4,920,642	10	2147.8	3163.8	0.096	0.040	PetStore

Table 1: Summary of data sets.

and real enterprise users, respectively. The third was collected in an instrumented testbed running a sample application serving carefully crafted synthetic workload.

Several other data sets of comparable scale and quality were available to us. We chose the three described in this section because together they severely challenge the ability of our method to generalize along several dimensions. The data sets of Table 1 differ in terms of the nature of the application, the time of data collection, the extent of geographic distribution, and workload. The workloads that gave rise to these data sets furthermore exhibit the non-stationarities described in Section 2.1, making them far more difficult to model than data derived from conventional synthetic benchmark workloads.

3.1.1 ACME: DotCom-Era Web Shopping

The "ACME" data set is taken from the busiest of seven servers comprising a major national 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. Load balancers apportioned incoming client sessions across Web servers, which served static page requests directly and passed more complex transactions to application servers and database servers below them. 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 [2].

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 transaction types.

3.1.2 VDR: Modern Enterprise Application

Figure 5 depicts the architecture of the globally-distributed VDR application. VDR is a high-availability business-critical internal HP application serving both external customers and HP users on six continents. Its system architecture therefore incorporates redundancy and

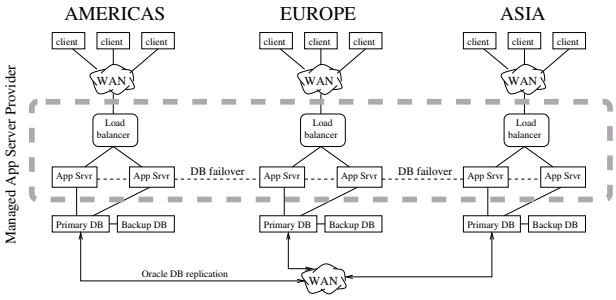

Figure 5: VDR application architecture.

failover features both locally and globally, as shown in the figure. Regional hubs in Atlanta, Swinden, and Singapore respectively serve the Americas, Europe/Middle East/North Africa, and Asia/Pacific regions. All hosts at the application server tier are HP 9000/800 servers running HP-UX B.11.11. The Americas region has two application server hosts with 16 CPUs and 64 GB RAM each. The EMEA region has three app server hosts with 16 CPUs and 32 GB RAM each, and the Asia/Pacific region has two app server hosts 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.

An interesting feature of VDR is that different organizations are responsible for the application itself and for the managed app server infrastructure upon which it runs (the latter is indicated with a dashed rectangle in Figure 5). 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. OpenView Performance Agent (OVPA) collected system utilization metrics; transaction response times are taken from application-level logs.

Pronounced seasonal workload variations are present in both the ACME and VDR data sets. Figure 6 shows

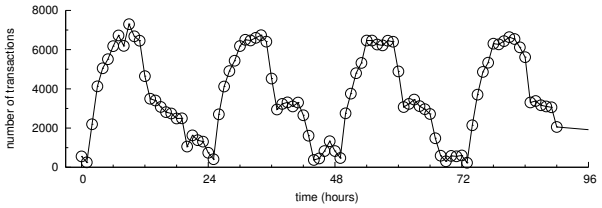

Figure 6: Seasonality in VDR workload.

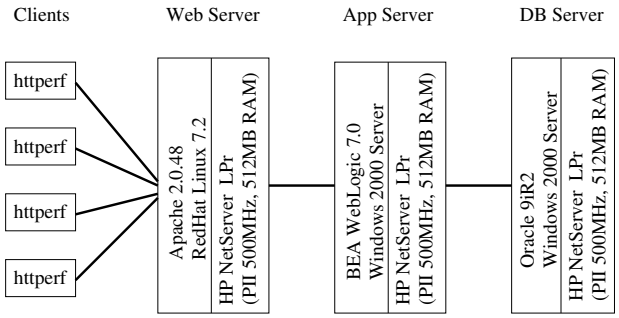

Figure 7: Testbed hardware and software architecture.

hourly transaction counts for the first four days of the VDR data set. 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 in workload levels. Other seasonal variations and nonstationarities are present in both ACME and VDR data sets.

3.1.3 Testbed: Heavy & Nonstationary Workload

The testbed data set is in many ways the most interesting of the three used in our evaluations. The hardware and software components of the testbed are depicted in Figure 7; the testbed has a switched 100 Mbps full-duplex network. The sample application run in the testbed is a Middleware Company version of the Java PetStore. In its default configuration, the application could not support more than 24 user sessions; at higher concurrency levels, a large fraction of transactions aborted with errors. The application configuration had to be manually tuned, after which the PetStore could support over 100 concurrent users [9]. This allowed us to bombard the testbed with very heavy workload from a large number of concurrent client sessions while still receiving meaningful transaction replies.

The workload submitted to the experimental testbed is what makes this data set interesting, and also more challenging for evaluating the accuracy of our performance models than either of our real production application data sets. The workload consists of a steadily increasing number of client sessions whose think times are such that the overall transaction rate reaching the application is a sine

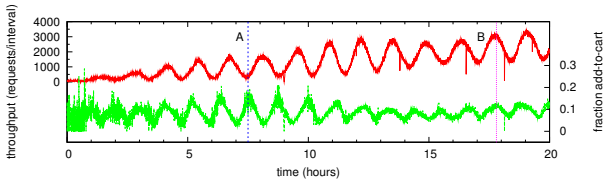

Figure 11: Nonstationarities in testbed workload: throughput and add-to-cart fraction.

wave overlaid on a ramp. The server is deliberately overloaded during workload peaks in the second half of the data set, i.e., *every peak in the second half represents a "flash crowd" or transient overload event*. Because we calibrate our models on the first half of the data set, when load is light, the testbed data set allows us to evaluate how well our models generalize to overload conditions that differ fundamentally from the workload/performance conditions represented in the calibration data.

Figure 8 shows CDFs of CPU utilizations at the app server and database tiers encountered by arriving transactions. Over 25% of transactions arrive when database CPU utilization exceeds 35% and app server CPU utilization exceeds 85%. Comparing Figure 8 with Figure 1, we see that the testbed is far more heavily loaded than our two real production systems. This is important for comparing our Extended and Composite models with the Basic model; the former attempt to account for queueing while the latter ignores it entirely.

Another distinctive feature of the testbed workload is that it is highly *nonstationary* in the sense that transaction mix varies dramatically over time. We achieve this effect by sinusoidally varying two key workload parameters: the probability that a "browse" is followed by an "add-to-cart," and the probability that an "add-to-cart" is followed by a "checkout." The period and amplitude of these two sine waves differ from one another and from the sine wave governing the total number of transactions submitted by clients.

Figure 11 illustrates one of the resulting nonstationarities in our testbed workload (for clarity, only the first 20 hours of the 38-hour data set are shown). The upper time series in the figure shows total transaction throughput and the lower series shows the fraction of transactions that are of type "add-to-cart." Note that the two series are, by design, *not* aligned. Approximately 7.5 hours into the test run (indicated by vertical line "A" in the figure), workload is low but the fraction of add-to-cart transactions is high. Ten hours later (line "B"), the peaks in the two series are roughly aligned. A time series of "checkout" transactions would be aligned with *neither* of the two shown in Figure 11. Figure 9 illustrates the nonstationarities in our testbed workload from a different perspec-

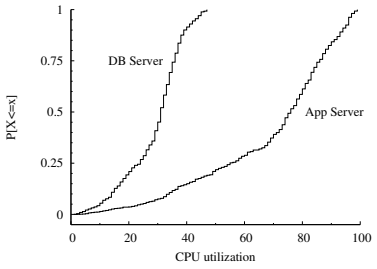

Figure 8: CDFs of testbed CPU utilizations seen by arriving transactions.

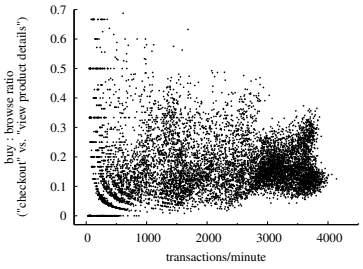

Figure 9: Nonstationarities in testbed workload: throughput vs. buy:browse ratio.

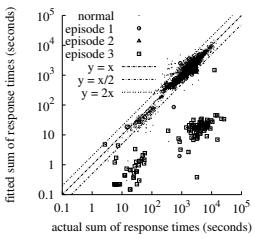

Figure 10: Detecting a performance anomaly in the "FT" application.

tive. The figure shows a scatterplot of throughput versus the buy:browse ratio. For all levels of workload intensity, from under 500 transactions per minute to 4,000 transactions per minute, the buy:browse ratio varies widely.

We calibrate models on a prefix of each data set and evaluate their predictive accuracy on a suffix. Nonstationarities therefore imply that predictive evaluations will involve workloads very different than those used in calibration. We believe that nonstationarities are essential for a challenging and credible evaluation of the extent to which models generalize beyond the calibration data.

3.2 Results

Except where noted otherwise, 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.

3.2.1 Retrospective Explanation

Table 2 summarizes the explanatory accuracy of eight models on three data sets according to our main figure of merit, normalized aggregate error (Equation 7). The table includes both the standard versions of our Basic and Extended models, which model performance as a function of transaction mix, 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. Several results are consistent across all three data sets.

First, we note that our Extended model achieves remarkably high accuracy when calibrated with LAR regression. Normalized aggregate error ranges from under 10% for VDR to roughly 17% for the highly variable and heavily-loaded Testbed application. Our Basic model achieves nearly the same accuracy for the two production applications, but it is noticeably less accurate for the Testbed data set. This is what we expect because the Basic model ignores queueing, which is non-negligible in the heavily-loaded Testbed application.

Second, normalized aggregate error is substantially lower when LAR regression is used. The difference is particularly striking in the case of the Testbed application: Error increases from 17% to 25% if we employ OLS regression instead of LAR in the Extended model.

Third, our transaction mix performance models achieve substantially better accuracy than their Scalar counterparts. For the VDR application, for example, our error measure is over 55% higher for an ExtendedScalar model as compared to the Extended model (14.54% vs. 9.35%) when both are calibrated with LAR.

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 improve retrospective accuracy but not reduce it, so we might be tempted to include one if we are willing to trade "correctness" for accuracy. We found that the benefits of including an intercept are very limited, and that the "correct" models with no intercept are usually almost as accurate.

Our previous work explains how accurate retrospective explanatory performance models can be useful [13]. 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.

Furthermore, we have shown that a real performance bug episode in a real distributed production application appears as a prominent performance anomaly. Figure 10 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. One episode corresponds to the prominent cluster of points in the lower right corner of the figure. FT is a globally-distributed enterprise application that resembles

Data	Basic		BasicScalar		Extended		ExtendedScalar	
Set	LAR	OLS	LAR	OLS	LAR	OLS	LAR	OLS
VDR	0.0940	0.1002	0.1462	0.1485	0.0935	0.0997	0.1454	0.1478
ACME	0.1281	0.1308	0.1609	0.1617	0.1210	0.1239	0.1598	0.1612
Testbed	0.3230	0.3605	0.3646	0.4403	0.1710	0.2493	0.1978	0.2320

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

VDR in several respects. See [13] 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 $\hat{y}_i$.

3.2.2 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: OLS typically yields substantially less accuracy than LAR.

In addition to normalized aggregate error, Table 3 shows an alternative accuracy measure: 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 even a small residual if y_i is also small.

Table 3 shows that our approach yields high accuracy by *both* measures. Even the Basic model 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 performs poorly for the heavily-loaded, highly nonstationary Testbed data, as we would expect, but it leaves little room for improvement when applied to real enterprise applications.

Our Composite model offers far better accuracy than the Basic model for the Testbed, as we would expect. The Composite model offers relatively modest accuracy improvements over the Basic model for the lightly-loaded VDR application. This too is not surprising. Resources are generously provisioned for business-critical enterprise applications like VDR, and resource utilizations are consequently low most of the time (Figure 1). Queueing delays are therefore likely to be small in relation to service times, so we gain little accuracy with a Composite model that accounts for queueing delays that the Basic model ignores. Aggregate response time is easier to predict in lightly-loaded applications than in heavily-loaded ones.

Another intuitive aspect of Table 3 is that both Basic and Composite models have better predictive accuracy on the real production applications than on the testbed data. It is easier to predict aggregate response time in a system with relatively moderate variations in transaction volume and mix than in the wildly variable testbed workload.

As with retrospective explanatory accuracy (Table 2), prospective prediction accuracy is substantially better in the models that exploit knowledge of transaction types than in their Scalar variants. Transaction mix is a remarkably powerful performance predictor.

The only puzzling feature of Table 3 is that the Composite model yields *worse* normalized aggregate error than the Basic model for the ACME application. We suspect that this is due to limitations in the ACME data set. Whereas we have utilization measurements at both app server and database server tiers for VDR and for our testbed, we have only app server measurements for ACME. Furthermore it is suspected that the bottleneck resource in the ACME system during the measurement period was the Java thread pool in the application server [1]. This case illustrates a limitation of our method: If we do not have utilization measurements of performance-critical resources—e.g., database server hardware resources or "soft" resources like the app server thread pool—then calibration will likely *overfit* models to the inadequate input data. Calibration on such deficient data yields models that are accurate in retrospect only accidentally, and that are led astray by irrelevancies when used for prediction.

Figures 12, 13, 14, and 15 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 12 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. Figure 13 shows the difference between the two time series of Figure 12, i.e.,

Data Set	Basic		BasicScalar		Composite		CompositeScalar	
	$\frac{\sum_i \lvert e_i \rvert}{\sum_i y_i}$	median $\lvert e_i \rvert / y_i$	$\frac{\sum_i \lvert e_i \rvert}{\sum_i y_i}$	median $\lvert e_i \rvert / y_i$	$\frac{\sum_i \lvert e_i \rvert}{\sum_i y_i}$	median $\lvert e_i \rvert / y_i$	$\frac{\sum_i \lvert e_i \rvert}{\sum_i y_i}$	median $\lvert e_i \rvert / y_i$
VDR	0.1226	0.0963	0.1621	0.1418	0.1221	0.0933	0.1606	0.1420
ACME	0.1470	0.1378	0.1523	0.1424	0.1566	0.1472	0.1525	0.1431
Testbed	0.6257	0.5056	0.6528	0.5247	0.3269	0.1283	0.3702	0.1708

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

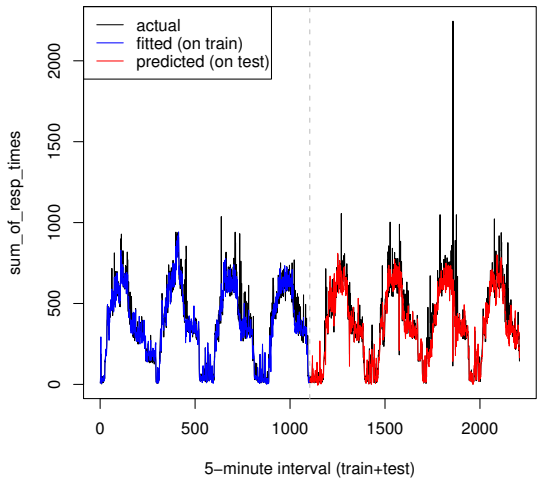

Figure 12: Time series of y_i and $\hat{y}_i$.

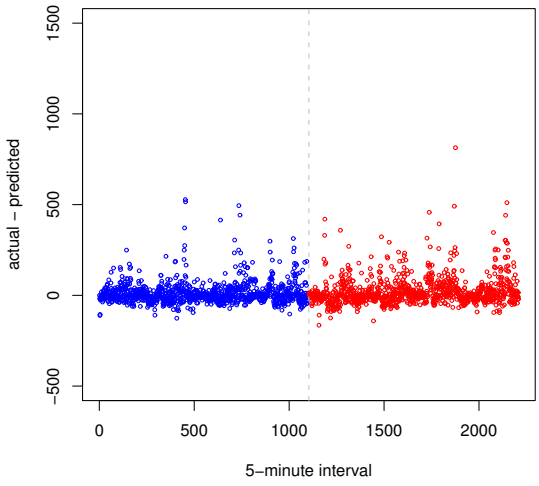

Figure 13: Time series of residuals e_i.

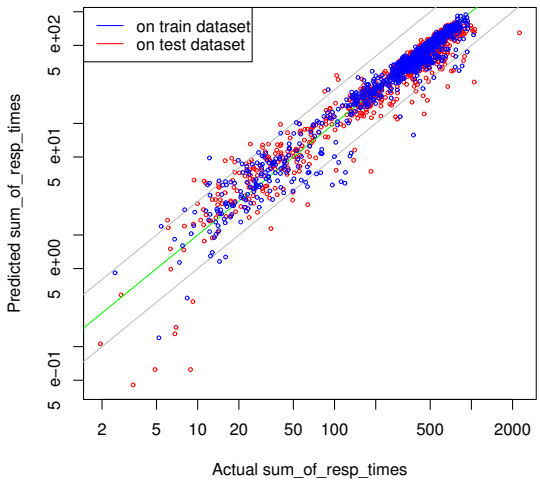

Figure 14: Scatterplot of $(y_i, \hat{y}_i)$ pairs.

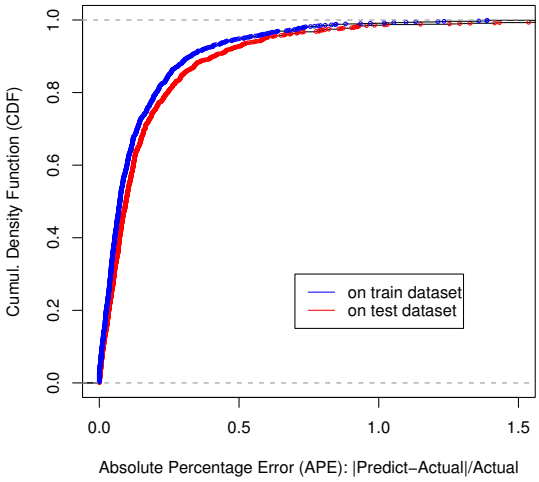

Figure 15: CDF of $|e_i|/y_i$.

the time series of residuals. 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 14 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 15 shows the full distributions of $|e_i|/y_i$ for both explanatory and predictive models. Our Composite model is highly accurate for retro-

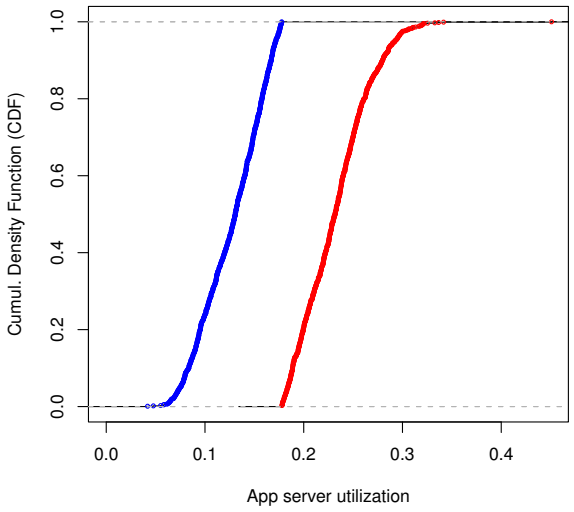

Figure 16: CDFs of app server CPU utilization in both halves of re-ordered VDR data set.

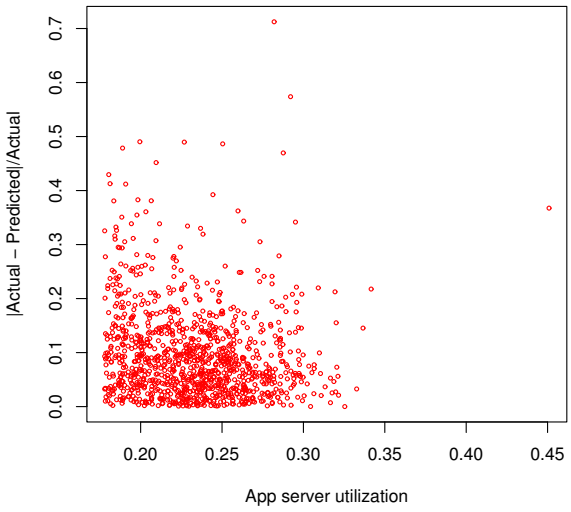

Figure 17: Prediction error $|e_i|/y_i$ versus app server CPU utilization in re-ordered VDR data set.

spectively explaining performance, and the figure shows that residuals remain remarkably small when the model is used to predict performance based on transaction mix.

3.2.3 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 16 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 17 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).

Data set	Basic	naïve U_r	Composite
VDR	0.1226	0.1221	0.1221
ACME	0.1470	0.2193	0.1566
Testbed	0.6257	0.5794	0.3269

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

3.2.4 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 $U_r/(1-U_r)$ 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.

4 Related Work

This section reviews literature on two topics central to our work: queueing models of modern distributed applications and LAR regression. We also briefly discuss

an alternative modeling approach not based on queueing theory. Our previous work reviews at length literature on an important application of our method, performance anomaly detection [13]. A broader and more detailed review of statistical performance modeling for performance debugging is available in Cohen *et al.* [10].

4.1 Queueing Models

Queueing networks are the subject of a large literature; see, e.g., Bolch *et al.* for a lengthy survey [6]. Jain describes applications of queueing theory to computer system performance analysis [12]. The approaches that Jain presents 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 is known, and that the distribution of service times at each station is known. If systems and applications are instrumented very thoroughly, e.g., if a tool such as Magpie [3] can be used, such information might be collected. However our work proceeds from the assumption that lightweight passive measurements of transaction response times and resource utilizations are all that is available.

In the remainder of this section we discuss in depth two recent papers that present queueing models of 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 [16, 18].

Urgaonkar *et al.* model multi-tier Internet services as product-form queueing networks and employ mean value analysis to compute average response times [18]. The model assumptions in this work 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 4 with their Figure 3 [18, 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 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 [16]. 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 simplified 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.

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 estimated. The authors 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 [16, p. 75]. Another difference is that we do not require knowledge of internal application component structure; we use only transaction type information that is visible outside the application. 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 benchmark-like synthetic workloads.

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 in addition to a testbed application serving synthetic workload; they have used only the latter. More importantly, our testbed experiments use a workload that is explicitly designed to be *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. 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 2.2.4). We found that transaction mix models offer substantially higher accuracy than their Scalar counterparts (Section 3.2).

4.2 LAR Regression

Least absolute residuals regression (often known by other names, including "L1" and "least (sum of) absolute deviations" regression) has a long and interesting history, dating back to at least the 1750s [17] and possibly the 1630s [15]. Reasonably convenient algorithms for *unidimensional* LAR regression were available by the nineteenth century. However the general multivariate case of LAR regression is far more computationally difficult than either univariate LAR or multivariate least-squares regression. The relative ease of computing least-squares parameter estimates accounts for much of the popularity of OLS over LAR until recent decades, because multivariate LAR is infeasible without high-speed computers and efficient linear programming algorithms [5].

Statistical considerations sometimes recommend one regression procedure or another. For instance, LAR and least squares provide maximum likelihood parameter estimates under different assumptions about the distribution of random errors in the input data. However, OLS is more popular simply because alternatives are not readily available or widely known. This is unfortunate because in many situations LAR regression optimizes the most natural measure of accuracy. Furthermore LAR is more *robust* than OLS, i.e., less sensitive to extreme data points. Finally, computational complexity no longer precludes the use of LAR: Efficient specialized linear programming algorithms for LAR appeared in the 1970s [4], and LAR parameter estimation remains an active research area [7]. In our experience LAR is a valuable tool and we recommend it for situations where it is appropriate.

4.3 Automated Model Induction

Although we calibrate the parameters of our performance models using measurement data, their functional form—the F of Equation 1—is "hard wired." An alternative approach that has recently attracted considerable attention is to automatically "learn" the very structure of models from data by searching a large space of functional forms [9–11]. "Automated model induction" seeks to eliminate the need for expert knowledge (e.g., of queueing theory in the present case) and thereby reduce the cost of producing models.

The approach we have taken in this paper is closer in spirit to that of conventional queueing models of distributed applications [16, 18]. We begin with a small amount of "expert knowledge": the knowledge that response times consist of service times plus queueing times, the waiting-time terms of Equation 4, and the observations enumerated in Section 2.1. We then embed this knowledge in the functional form of our models.

In our experience the cost of developing flexible, general, accurate, and useful models grounded in elementary queueing theory is not prohibitive. Fixed-form models based on a handful of general observations about distributed application architecture are easy to define, calibrate, validate, explain, and justify from first principles. They furthermore predict performance with remarkable accuracy in a wide variety of real distributed production applications.

5 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 enterprise applications, however, remains an important problem due to the growing economic importance of these applications. This paper has presented a practical, versatile, and accurate approach to predicting performance in complex modern distributed applications. Our method relies solely on measurement data that is routinely collected in today's production environments, it can be adapted to a wide range of applications, and calibrated models generalize well to new regions of workload/performance space.

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%. Even in a testbed application bombarded with extraordinarily heavy and non-stationary synthetic workload, we obtain errors of 13%–33%. Our results show that if accurate workload forecasts are available, they can be mapped directly to accurate performance predictions. Our approach 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 than scalar workload volume.

References

[1] Martin Arlitt, October 2005. Personal communication.

[2] Martin Arlitt, Diwakar Krishnamurthy, and Jerry Rolia. Characterizing the scalability of a large web-based shopping system. *ACM Trans. on Internet Tech*, 1(1):44–69, August 2001.

[3] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using Magpie for request extraction and workload modelling. In *Proc. OSDI*, pages 259–272, December 2004.

[4] I. Barrodale and F.D.K. Roberts. An improved algorithm for discrete L1 linear approximations. *SIAM Journal of Numerical Analysis*, 10:839–848, 1973.

[5] P. Bloomfield and W.L. Steiger. *Least Absolute Deviations: Theory, Applications, and Algorithms*. Birkhauser, 1983.

[6] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains*. Wiley, 1998.

[7] Kenneth L. Clarkson. Subgradient and sampling algorithms for l_1 regression. In *Proc. 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 257–266, 2005.

[8] Adrian Cockcroft and Bill Walker. *Capacity Planning for Internet Services*. Sun Press, 2001.

[9] Ira Cohen, Moises Goldszmidt, Terence Kelly, Julie Symons, and Jeffrey S. Chase. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *Proc. OSDI*, October 2004.

[10] Ira Cohen, Steve Zhang, Moises Goldszmidt, Julie Symons, Terence Kelly, and Armando Fox. Capturing, indexing, clustering, and retrieving system history. In *Proc. SOSP*, October 2005.

[11] Moises Goldszmidt, Ira Cohen, Armando Fox, and Steve Zhang. Three research challenges at the intersection of machine learning, statistical induction, and systems. In *Proc. HotOS X*, June 2005.

[12] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.

[13] Terence Kelly. Detecting performance anomalies in global applications. In *Proc. USENIX Workshop on Real, Large Distributed Systems (WORLDS)*, December 2005.

[14] John Neter, Michael H. Kutner, Christopher J. Nachtsheim, and William Wasserman. *Applied Linear Statistical Models*. Irwin, fourth edition, 1996.

[15] C. Radhakrishna Rao and Helge Toutenburg. *Linear Models: Least Squares and Alternatives*. Springer, 1999.

[16] Christopher Stewart and Kai Shen. Performance modeling and system management for multi-component online services. In *Proc. NSDI*, pages 71–84, 2005.

[17] Stephen M. Stigler. Boscovich, Simpson, and a 1760 manuscript note on fitting a linear relation. *Biometrika*, 71:615–620, 1984.

[18] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. An analytical model for multi-tier internet services and its applications. In *Proc. ACM SIGMETRICS*, pages 291–302, June 2005.

[19] Rand R. Wilcox. *Introduction to Robust Estimation and Hypothesis Testing*. Elsevier, second edition, 2005.