RC23175 (W0404-049) April 6, 2004
Computer Science

IBM Research Report

Parameter Inference of Queueing Models for IT Systems Using End-to-End Measurements

Zhen Liu, Laura Wynter, Cathy H. Xia, Fan Zhang
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Parameter Inference of Queueing Models for IT Systems using End-to-End Measurements

Zhen Liu, Laura Wynter, Cathy H. Xia, Fan Zhang
IBM TJ Watson Research Center
Yorktown Heights, NY 10532
{zhenl,lwynter,cathyx,fzhang}@us.ibm.com

March 12, 2004

Abstract

Performance modeling has become increasingly important in the design, engineering and optimization of Information Technology (IT) infrastructures and applications. However, modeling work itself is time consuming and requires a good knowledge not only of the system, but also of modeling techniques. One of the biggest challenges in modeling complex IT systems consists in the calibration of model parameters, such as the service requirements of various job classes. We present an approach for solving this problem in the queueing network framework using inference techniques. This is done through a mathematical programming formulation, for which we propose an efficient and robust solution method. The necessary input data are end-to-end measurements which are usually easy to obtain. The robustness of our method means that the inferred model performs well in the presence of noisy data and further, is able to detect and remove outlying data sets. We present numerical experiments using data from real IT practice to demonstrate the promise of our framework and algorithm.

1 Introduction

Performance modeling has been of great theoretical and practical and importance in the design, engineering and optimization of computer and communication systems and applications for several decades. A modeling approach is particularly efficient in providing architects and engineers with qualitative and quantitative insights about the system under consideration.

However, as Information Technology (IT) matures and expands in the scope of available applications, IT systems increase at a fascinating rate in both size and complexity. For example, today, a typical Web service hosting center may have hundreds of nodes and dozens of different applications simultaneously running on it. Each of the nodes in turn has often multiple processors and layered caches. These nodes make use of both local and shared storage systems. The size and complexity of such systems make performance modeling much more difficult, if at all tractable. Detailed modeling, fine tuning and accurate analysis can be carried out only on very small systems or very small components in a system.

In addition, due to the rapid evolution of hardware technology, components in these systems are upgraded at a much higher pace than in the recent past, in order to meet demand and to improve the Quality of Service (QoS) parameters of performance and availability. Hence, performance modeling must be done in a very short time frame in order for the model and analysis to be relevant.

These constraints made performance modeling work on modern IT systems very expensive, and oftentimes unaffordable. In order to obtain relatively accurate performance evaluation results with a short turnaround time, i.e., before the system under consideration becomes obsolete, heavy investments are necessary in human and compute power.

On the other hand, IT systems have become critical in most businesses. Losses of millions of dollars per minute when a company's IT system goes down are well-documented. Thus, it is natural that users impose more and more stringent QoS requirements on their systems. In the case of IT outsourcing, service-level agreements (SLA) signed between the parties stipulate, among other things, the service quality guarantees, often with associated penalties in case of violations. As a consequence, predictive modeling is truly vital in the capacity planning and QoS management of such systems.

It is therefore urgent that researchers develop performance modeling methodologies that can cope with large systems in an efficient and effective way. In this paper, we propose one such method based on queueing networks. Queueing network models have been and continue to be the most popular paradigm for the performance analysis of such systems, cf. e.g. [8, 11]. When one uses queueing networks for the performance modeling, the first thing to do is to specify the system configuration and component dependence relationships using a network of queueing stations with an appropriate number of servers and service disciplines in the queues. The customers in the queueing network represent the jobs processed by the system components. These queueing stations can model both hardware resources (such as routers, machines or cache, etc.) and software elements such as threads or thread pools. Depending on the desired level of modeling detail, this step of queueing network specification will be more or less complex. Once the queueing network is built, performance predictions and what-if analyses can be done through analytical approaches or simulations by varying the queueing network parameters such as traffic intensity, service time requirements, number of servers, queueing disciplines, etc.

While many parameters, such as the queueing discipline, the number of servers in a queueing station, buffer size, etc., of the queueing network can be easily set by the modeler, the service requirements (i.e. job processing times) of customers are much more difficult to estimate. However, without an accurate estimation of the service requirements, performance predictions can vary wildly. In other words, a principal difficulty in building a valid queueing network of an IT system is the fine tuning of the service requirements. This step requires benchmarking of the real system followed by an adjustment of the queueing parameters in order to obtain coherency between measurement and predictions.

In many cases, however, measuring job processing times is either technically difficult, far too time consuming, and/or very intrusive. Given that a computer system is typically multi-threaded, with many internal input and output queueing effects, it is often impossible to measure such processing times (without the queueing delay) of various transactions. As a result, direct measurement of these parameters is in general too costly. Furthermore, measurement data, when available, are often noisy or biased. In short, determining service requirements is probably the greatest impediment to using queueing networks for the performance modeling of IT systems.

In this paper, we propose an optimization-based *inference* technique to tackle this important yet highly challenging problem. It is formulated as a parameter estimation problem using a general (Kelly-type, [7]) queueing network. We consider the case where aggregate and end-to-end measurement data (i.e. system throughput, utilization of the servers, and end-to-end response times) are available. Note that such data are typically easy to obtain. Each set of measurements, that is, a set of service requirements in which the working environment (load, scripts,. . .) is constant, is referred to henceforth as an *experiment*.

Our contribution in this paper is three-fold.

- First, we formulate the overall problem as a set of tractable, quadratic programs, one for each set of end-to-end measurements.
- Then, based upon that formulation, we present a novel and highly robust method for solving the problem, the *self-adjusting nested estimation procedure*, which makes explicit use of the underlying problem's structural properties. In particular, we use the non-uniqueness of the solution to each quadratic program and the presence of multiple experiments, to obtain queueing network parameters that maintain a representation of the entire set of solutions to the data. The robustness of the method means the model performs well in the presence of noisy data, and further is able to detect and remove outlying experiments within the procedure itself. This robustness comes at a very low computational cost.
- Finally, we show that for data coming from real IT system measurements, typically subject to measurement errors and other bias, the methods we propose here are superior to other available methods for obtaining parameter values from multiple, multi-dimensional experiments. In particular, we compare our algorithm with an adaptation of the bundle-adjustment method used in 3-D image reconstruction to the queueing network inference setting; our method is shown to be clearly more robust and yet it requires a only modest increase in computation time.

We emphasize that the inference methodology for developing performance models is applicable in a much more general context than the specific Kelly-type network model considered here. We restrict our attention to Kelly-type network in this paper so as to explore fully the potential of the quadratic program structure. In fact, our numerical results demonstrate that the approach works quite well in most real system practice, which indicates that product-form queueing network models can serve as good approximations to capture the high-level queueing dynamics of real systems.

Though our modeling approach is very useful in doing prediction, what-if analysis, etc., it does not solve every problem in this space. For example, this model is not intended to predict the impact of major functional changes, or software defects, although we claim later in the paper that our outlier detection component will assist in *identifying* functional changes or software defects.

Earlier work on queueing parameter estimation can be found in [12] and [14]. In [12], a tandem queueing network with First-Come-First-Serve (FCFS) servers was considered. Various equations were established relating different queueing variables (such as queue length and response times). In [14], a tandem queueing network with two queueing stations was considered, one with FCFS discipline and the other with Processor-Sharing (PS) discipline. Other related inference work can be found in the network tomography research area, see e.g. [10, 5, 4, 2, 15]. Most work has focused on designing smart probing schemes and using statistical methods such as EM algorithms or Monte Carlo methods to infer either the network loss probability or the delay distribution.

In the next section, we present background on the relevant characteristics of IT systems, followed by the nature of the queueing model we shall derive, and the inference problem. In Section 3, the queueing dynamics are defined. Section 4 presents the derivation of the mathematical program for a single experiment, multiple experiments, and its properties. Section 5 introduces our algorithm, its theoretical basis, characteristics, and a comparison with a related algorithm from the 3-D image reconstruction literature. In Section 6, we demonstrate the workings of our algorithm on both constructed and real data sets, and compare it with the related algorithm described in Section 5. Conclusions and suggestions for further work on the topic can be found in Section 7.

2 The Modeling Framework

2.1 Background on IT Systems

Rapid development in e-business and information technology has made today's IT environment quite complex. As illustrated in Figure 1, a typical IT system has multiple interconnected layers composed of many software and hardware components, such as networks, caching proxies, routers, load balancers, high-speed links, firewalls, and various types of e-business servers .

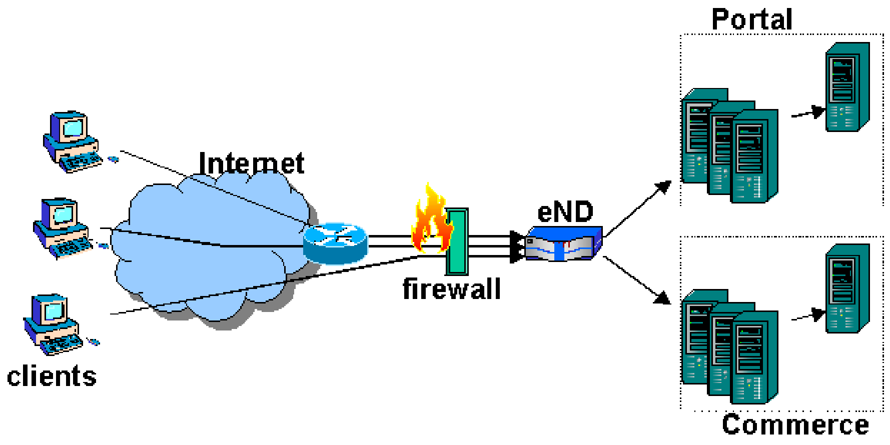

Figure 1: A Typical IT Environment

The e-business servers are often organized to have a multi-tiered architecture. Each tier has a cluster of machines and may handle a particular set of functions. There are two tiers in Figure 1, for example. The first tier is composed of Web and authentication servers, referred to as the Portal. The second tier, referred to as the Commerce tier, is composed of a cluster of application servers that process most of the business transactions. Incoming requests are routed to the Portal or the Commerce tier based on their functional requirements. Within the Commerce (or Portal) cluster, again there are multiple layers, a front-end layer with several servers working in parallel, and a back-end layer composed of one or several database servers. The front-end servers are responsible for obtaining static/dynamic pages for clients. The back-end database server(s) handles database transactions. A network dispatcher (eND) is commonly used to route incoming requests in a weighted round robin fashion, so as to evenly distribute the load to the parallel front-end servers. The processed pages are usually returned directly to the clients.

The e-business workload, composed of transactions and requests to the e-business servers, is also quite complex. Consider for example a typical enterprise online shopping scenario. It contains authentication transactions such as login, and business transactions such as browsing the catalog, searching for products, adding items to a shopping cart, proceeding to check out, etc. Each of these transactions uses the site's resources differently. Transactions such as browsing may only involve the front-end application server to fetch static pages, which is relatively inexpensive, while other transactions such as searching or checking out may involve composition of a dynamic page or multiple queries to the database that require a large amount of processing time and involve both the front-end application server and the back-end database server. In addition, user navigational patterns vary dramatically from person to person. Some users may spend all their time browsing and searching, while some frequent buyers may directly jump in for buying.

It is thus a challenging task to assess an IT system's capability of delivering end-to-end performance assurance across the entire IT environment, given the variety of system architectures, numerous applications with different functions, and the vast diversity in user behaviors.

2.2 Queueing Model

In order to ensure the feasibility of the modeling framework yet still capture the fundamentals of the complex e-business infrastructure, we require the queueing model to be neither too detailed nor too general, and to rely on a controllable number of parameters.

We therefore use a high-level multi-class queueing network model. This form of model captures major resource and delay effects and provides good traceability between the performance measures and the system architecture. Each resource component that incurs non-negligible delays will be modeled by a *generic* service station with queueing effect. Such a generic service station could have anywhere from one to an infinite number of servers. For example, if we think the delay incurred at the firewall is constant and non-negligible, one could then model the firewall as an infinite server station with constant service time.

The next step is to characterize and profile the transactions into different classes, so that requests within each class would follow similar paths through the various server stations and require similar service demands at each station along the path. Such profiling can be based on prior engineering knowledge or after careful workload analysis. For example, a login transaction is normally different from a buy transaction and they would visit different set of server stations and make different resource demands. Figure 2 shows the queueing network model corresponding to the IT environment in Figure 1.

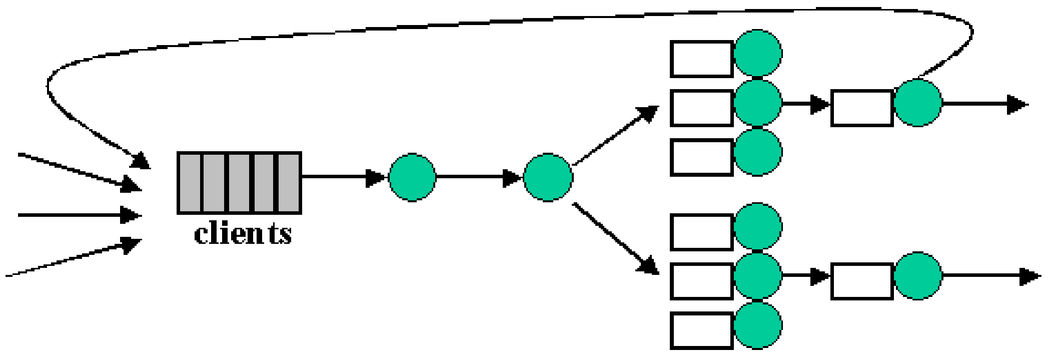

Figure 2: An Example of Queueing Network Model

Suppose there are I generic service stations, and J different job classes in the corresponding queueing model. However, the arrival rate of a particular job class may or may not be known.

We define the *end-to-end delay* of a particular job (or transaction) to be the total response time that a user experiences from the time she issues that transaction to the time she receives the complete response as desired. For example, if a buy transaction traverses the firewall, network dispatcher, transaction server, database server, and the network, its end-to-end delay is the sum of the delays incurred at each hop along the path.

Different generic service stations can serve jobs under different service disciplines. For example, to the model the user think time between two consecutive transactions, a delay center with infinite servers can be used. to model the multi-threaded and time-shared processing/computing environment in most application servers, one can assume the corresponding queues are served under the Processor-sharing service discipline. Processor-sharing is considered as a limiting approximation to time sharing in which the quantum length tends to zero. Hence, if there are n jobs in the system, they each simultaneously receive $1/n$ of the resource.

The detailed service time information at each station is yet to be determined.

2.3 The Inference Problem

One of the biggest challenges in modeling complex IT systems using queueing network models consists in the calibration of the queueing network parameters, such as the service requirements of the various job classes at each station. The ideal way to capture these parameter values is to directly measure the resource consumption of each class at each device. In real IT systems, however, it is very difficult to measure the actual service time, without queueing delay, of a job. For example, in the Apache HTTP server, a job may be admitted, but then it actually stays in the accept queue, waiting to be picked up by a server thread. Because there are many such queueing effects inside a computer system, it is impossible to measure each service time in detail.

We therefore have to rely on performance metrics that *are* measurable and are also relatively inexpensive. For example, it is relatively easy to collect server utilization data or system throughput. End-to-end delay, defined as the time from initiation of a request from the user until the time that the user receives a response, can also be measured easily. In fact, quite often end-to-end delays are continuously monitored in order to make sure that the service level delivered in accordance with service-level agreement (or service contract). Usually such contracts stipulate performance (such as end-to-end delay) guarantees. Server utilization information is another metric frequently used for IT system performance monitoring.

In practice, experiments are often set up to collect above end-to-end measurements. These experiments could be aimed at measuring the performance under different load combinations, different load intensity, or a down-scaled or up-scaled configuration. Each of these experiments can be expensive and time consuming, requiring both efforts in setting up the experiments and data collection. Therefore one cannot afford to carry out too many of these experiments.

Given that such experiments are carried out under different conditions (in terms of the traffic intensity, transaction flows, system configuration, or system maintenance), naturally there are some experiments more trustworthy than others. We will refer to the less trustworthy experiments as *outliers* as they would simply produce inconsistent conclusions thus needed to be filtered out. Such outliers maybe caused by some unintentional human faults in setting up one experiment, or simply by some temporary environmental changes (e.g. system maintenance) while running the experiment. In addition, we consider the fact the measurement data within a particular experiment may be *noisy*.

In what follows, we will refer to *an experiment* as a set of end-to-end measurements collected under a particular load combination. We assume we have a finite and limited number of such experiments. Using these N different set of experiments as input, we present a tractable, optimization framework to infer the optimal set of service requirement parameters so that the resulting performance predicted by the model is the closest to the end-to-end measurements. That is, for given measurements (E^m, ρ^m) where E^m denotes the end-to-end delay measurements, and ρ^m denotes the observed server utilization, the optimal parameter setting will produce an estimation (E^e, ρ^e) that is closest to the measured under certain distance metric:

$$\min \ \|(E^m, \rho^m) - (E^e, \rho^e)\|$$

Here the distance metric $\|\cdot\|$ is quite general, e.g. absolute difference, maximum difference, weighted sum of absolute differences, or weighted sum of least square errors,etc.

3 Queueing Dynamics

In this section, we describe the queueing model and the underlying dynamics in more detail. Consider an open Kelly-type network [7] with I service stations and J different job classes. Jobs of class j visit the stations along a deterministic path

$$h(j,1), h(j,2), \cdots, h(j,L_j). \tag{1}$$

If $h(j,l) = i$, we say that a class j job visits station i at hop l. In total, a job j makes L_j hops before it exits the system. Let $H(j)$ be the set of all server stations that class j jobs visit. Denote v_{ji} the number of times that a class j job visits station i.

We assume that the J different job classes have independent Poisson arrival processes, with rate λ_j for class j jobs. In addition, suppose jobs of class j incur i.i.d. random service demands S_{ji} at station i, where S_{ji} is a general distribution, with mean $s_{ji} = E[S_{ji}]$. Let R_{ji} be the mean response time of class j jobs at station i. Denote by E_j the mean end-to-end delay of a class j job after it visits all of the stations along its path (1).

To make the model tractable, we assume the generic service stations are either delay centers, typically modeled modeled as an *infinite server*(IS) queues (and will later be referred as IS-stations), or finite capacity service stations operated under the processor-sharing (PS) service discipline (will later be referred as PS-stations).

Denote $\lambda_{ji} = v_{ji}\lambda_j$ as the arrival rate of class j jobs at station i. Then, the total job arrival rate at station i is

$$\lambda^{(i)} = \sum_{j=1}^{J} \lambda_{ji} \quad i = 1, ..., I.$$

Similarly, let $\rho_{ji} = \lambda_{ji} s_{ji}$. Then, total traffic intensity ρ_i at station i is

$$\begin{aligned}\rho_i &= \sum_{j=1}^{J} \rho_{ji} \\ &= \sum_{j=1}^{J} v_{ji}\lambda_j s_{ji}, \quad i = 1, ..., I.\end{aligned} \tag{2}$$

Throughout the paper, we shall assume that

$$\rho_i < 1, \quad \text{for all } i = 1, ..., I,$$

so that the queueing network is stable.

Since all service stations are assumed to be either infinite server (IS) or processor-sharing(PS) stations, (both are symmetric queues), the underlying queueing network is a quasi-reversible network [7]. We further have the following properties:

a) The state variables for each station are independent of those for other stations; hence the product form holds;

b) The arrival process for every class of jobs at each station has the PASTA property.

It is well-known [13] (§6-8 Theorem 26) that the number of jobs at PS station i has the same distribution as that of the corresponding M/M/1 queue. Moreover, the probability that a job belongs to class-j is ρ_{ji}/ρ_i. Hence, the mean number of jobs L_i at PS station i is

$$L_i = \frac{\rho_i}{1 - \rho_i},$$

and the mean number of class-j jobs L_{ji} at PS station i is

$$L_{ji} = \frac{\rho_{ji}}{1 - \rho_i}.$$

Applying Little's law, the mean response time R_{ji} for class j jobs at station i is then given by

$$R_{ji} = \frac{L_{ji}}{\lambda_{ji}} = \frac{s_{ji}}{1 - \rho_i}. \tag{3}$$

Clearly if station i is an IS queue (i.e. a delay center), we have that

$$R_{ji} = s_{ji}. \tag{4}$$

Therefore, the end-to-end delay for class j jobs can be derived as follows,

$$E_j = \sum_{i \in H(j), i \in IS} s_{ji} + \sum_{i \in H(j), i \in PS} \frac{s_{ji}}{1 - \rho_i}.$$

4 A quadratic program

4.1 Notation

Here we summarize the notation we use in the queueing inference problem. The following input parameters and performance measures are assumed to be given

$$\begin{aligned}
J :=& \text{number of job classes;}\\
I :=& \text{number of service stations;}\\
\lambda_j :=& \text{arrival rate for class } j \text{ jobs;}\\
v_{ji} :=& \text{number of times that a class } j \text{ job visits station } i\text{;}\\
E_j^m :=& \text{measured end-to-end delay of class } j \text{ jobs;}\\
\rho_i^m :=& \text{measured utilization of service station } i.
\end{aligned}$$

The following parameters and performance metrics are to be estimated:

$$\begin{aligned}
s_{ji} :=& \text{mean service requirement of class } j \text{ jobs at station } i\text{;}\\
R_{ji} :=& \text{mean response time of class } j \text{ jobs at station } i\text{;}\\
T_{ji} :=& v_{ji}R_{ji}, \text{ total mean response time of class } j\\
& \text{jobs (sum of multiple visits) at station } i\text{;}\\
E_j^e :=& \text{estimated end-to-end delay of class } j \text{ jobs;}\\
\rho_i^e :=& \text{estimated utilization of service station } i.
\end{aligned}$$

Denote further in matrix format that:

$$\begin{aligned}
\Lambda &= \{\lambda_j\}_{J\times 1}; \quad \text{(Arrival Rate)}\\
S &= \{s_{ji}\}_{J\times I}; \quad \text{(Service Times)}\\
T &= \{v_{ji}R_{ji}\}_{J\times I}; \quad \text{(Multiple-visit Response Time Matrix)}\\
E^m &= \{E_j^m\}_{J\times 1}; \quad \text{(Measured End-to-end Delay)}\\
E^e &= \{E_j^m\}_{J\times 1}; \quad \text{(Estimated End-to-end Delay)}\\
\rho^m &= \{\rho_i^m\}_{I\times 1}; \quad \text{(Measured Server Utilization)}\\
\rho^e &= \{\rho_i^m\}_{I\times 1}; \quad \text{(Estimated Server Utilization)}
\end{aligned}$$

4.2 Single Experiment

We shall focus on a single experiment first in order to derive the corresponding quadratic parameter inference program. The extension to multiple experiments is presented below.

Let us suppose that the matrices Λ, ρ^m, E^m are given. We can estimate the total response times (sum of multiple visits) for a class-j job at a PS-station i as follows:

$$T_{ji} = v_{ji} \cdot \frac{s_{ji}}{1 - \rho_i^m}, \text{ if } i \text{ is a PS-station.} \tag{5}$$

Similarly, the total response times (sum of multiple visits) for a class-j job at an IS-station i is given by

$$T_{ji} = v_{ji} \cdot s_{ji}, \text{ if } i \text{ is an IS-station.} \tag{6}$$

We define $\beta = \{\beta_i\}_{I\times 1}$, where

$$\beta_i = \begin{cases} \frac{1}{1-\rho_i^m}, & \text{if } i \text{ is a PS-station} \\ 1, & \text{if } i \text{ is an IS-station} \end{cases}$$

Let us define further: $Z = \{z_{ji}\}_{J\times I}$ where

$$z_{ji} = v_{ji}s_{ji}.$$

Therefore, $z_{ji} = 0$ whenever $v_{ji} = 0$. Then in matrix format we have

$$T = Z \cdot D(\beta),$$

where $D(\beta)$ stands for the diagonal matrix of β.

Therefore the estimated total end-to-end delay is given by

$$E^e = T \cdot \mathbf{1}_{I\times 1} = Z \cdot D(\beta)\mathbf{1}_{I\times 1} = Z \cdot \beta. \tag{7}$$

Similarly, based on the fact that $\rho_i^e = \sum_{j=1}^{J} \lambda_j v_{ji} s_{ji}$, we have that the estimated server utilization is:

$$\rho^e = (\Lambda)^T Z D. \tag{8}$$

The parameter estimation optimization problem is to thus to find the service requirements $S = \{s_{ji}\}$ such that the weighted least squared error is minimized. We have rewritten this problem so that the the differences between measures and estimated values of the parameters appear in the *definitional* constraints, and we minimize the sum of squared deviations from those differences. In other words, we solve a quadratic program subject only to non-negativity constraints on the service requirements, s_{ji}, i.e.,

$$\min_s \quad \sum_{j=1}^{J} w_j \delta_j^2 + \sum_{i=1}^{I} \epsilon_i^2 \tag{9}$$

$$s.t. \quad \rho_i^e(s) - \rho_i^m = \epsilon_i, \quad i = 1, ..., I. \tag{10}$$

$$E_j^e(s) - E_j^m = \delta_j, \quad j = 1, ..., J, \tag{11}$$

$$s_{ji} \geq 0, \quad j = 1, ..., J; i = 1, ..., I. \tag{12}$$

Here we use the weighted least square as our distance metric in this paper, where $w_j = \frac{\lambda_j}{\sum_{j=1}^{J} \lambda_j}$, $\forall j$, define the weights over the job classes based on the arrival rates. Denote in matrix format

$$W = \{w_j\}_{J\times 1} = \frac{\Lambda}{\Lambda^T \mathbf{1}_{J\times 1}}.$$

Note that both $E^e(s)$ and $\rho^e(s)$ are linear in s, and that the variables ϵ and δ are unrestricted in sign, as we are minimizing the squared values.

Let us now define a new labeling of the unknowns, which we shall refer to as x, where

$$\mathbf{x} = \begin{bmatrix} \tilde{\mathbf{x}} \\ \epsilon \\ \delta \end{bmatrix},$$

and $\tilde{\mathbf{x}}$ is a column vector form for S, i.e. $\tilde{x}_k := s_{ji}$ if $k = (j-1) * I + i$. $\epsilon = \{\epsilon_i\}$ and $\delta = \{\delta_j\}$. The optimization problem can then be re-stated as the following quadratic program in standard form:

$$\textbf{(QP)} \quad \min \quad \frac{1}{2} x^T H x \tag{13}$$

$$s.t. \quad Ax = b; \tag{14}$$

$$x_k \geq 0, \quad k = 1, ..., I \times J. \tag{15}$$

where

$$A = \begin{bmatrix} A_s \\ A_\epsilon \\ A_\delta \end{bmatrix}, \qquad b = \begin{bmatrix} b_s \\ b_\epsilon \\ b_\delta \end{bmatrix},$$

and

$$H = \begin{bmatrix} H_s & 0 & 0 \\ 0 & H_\epsilon & 0 \\ 0 & 0 & H_\delta \end{bmatrix}$$

with

$$\begin{aligned} H_s &= 0_{IJ \times IJ} \\ H_\epsilon &= 2I_{I \times I} \\ H_\delta &= 2D(W). \end{aligned}$$

Let L be the total number of v_{ji}'s such that $v_{ji} = 0$. Then,

$$\begin{aligned} A_{\epsilon ik} &= \begin{cases} \lambda_j v_{ji},\ i = 1, ..., I;\ k = (j-1) \times I + i \\ -1,\ i = 1, ..., I;\ k = I \times J + i \\ 0, o.w. \end{cases} \\ A_{\delta jk} &= \begin{cases} T_{ji},\ j = 1, ..., J;\ k = (j-1) \times I + i \\ -1,\ j = 1, ..., J;\ k = I \times J + I + j \\ 0, o.w. \end{cases} \\ A_{slk} &= \begin{cases} 1,\ l = 1, ..., L;\ k = (j-1) \times I + i \text{ and } v_{ji} = 0 \\ 0,\ o.w. \end{cases} \\ b_{\epsilon i} &= \rho_i^m,\ i = 1, ..., I \\ b_{\delta j} &= E_j^m,\ j = 1, ..., J \\ b_{sl} &= 0,\ l = 1, ..., L. \end{aligned}$$

4.3 Multiple Experiments: General Formulation

The previous subsection showed that the inference, or parameter estimation, problem may be reduced to that of minimizing the sum of squared deviations from the measured parameter values. For any particular set of experimental results, the above model may be solved, to obtain the appropriate parameter set for that experiment.

However, as is the case with any parameter estimation problem arising from a series of experiments, it is insufficient to simply solve the above quadratic inference problem for a single experiment. In practice, typically, multiple experiments can be carried out (e.g. under different load conditions). It is thus important to include the entire set of experiments in the parameter estimation step to obtain as much data on the full range of the parameters as possible, especially in the presence of nonlinear queueing effects.

Suppose that we have data from N experiments, each providing an estimate of the matrix, H^n and constraint values, X^n, based on the input parameters given within that experiment, $\rho^n, E^n, \ldots$.

The goal of the inference problem over the set of multiple experiments is therefore to find a robust vector of parameters, x that *best* represents the parameters obtained by solving the inverse parameter estimation problem for each experiment, yet is able to detect and eliminate outlying experiments.

For notational simplicity, we shall make the following simplifying assumption, although our method applies in general without this assumption.

Assumption 1 *The network configuration, in terms of servers present (used), is identical across all experiments.*

The problem of reconstructing a 3-D scene from multiple 2-D images shares many features with the problem of queueing network parameter inference, in that the multiple images, like the multiple experiments here, each contain some information on the true value (location) of each 3-D point based on a different 2-D configuration (camera angle).

In the context of computer vision, a method called bundle adjustment is widely used to find the set of 3-D points that best corresponds to all of the 2-D images. Bundle adjustment proposes a single least-squares framework in which the distance from the observed and the measured positions of each point is minimized. In effect, all of the 2-D images are treated together, and both the best 3-D positions of each point and the best 2-D-to-3-D projection matrix are determined simultaneously. (See, for example, [9]).

The analog to bundle adjustment in the context of queueing network parameter inference is to combine, in a single quadratic program, the data from all N experiments.

Bundle-adjustment-type algorithm: A single, large QP.

$$
\begin{aligned}
\textbf{(single-QP)} \quad \min \quad & \frac{1}{2} x^T \sum_{n=1}^{N} H^n \, x \\
s.t. \quad & A^n x = b^n; \;\; n = 1, \cdots, N \\
& \quad x_k \geq 0, \;\; k = 1, ..., I \times J.
\end{aligned}
$$

We use this single, large QP approach that combines all experiments into a single data set as a benchmark in our numerical experiments, comparing it with our algorithm in terms of solution accuracy, robustness to noise and outlying experiments, and computation time.

Before we present the algorithm, let us first examine the properties of the model.

4.4 Properties of the model

We have the following properties of the queueing-based quadratic parameter inference problem.

Lemma 1 *The Hessian matrix H is symmetric and positive semi-definite for all x.*

Proof: Notice that H is diagonal and $H_{kk} \geq 0$ for any k. Thus, for any x, we have

$$x^T H x = \sum H_{kk} x_k^2 \geq 0$$

and H is positive semi-definite.

Note that the objective function of (QP) is convex in x and the constraint set is convex and closed. The (QP) is therefore a convex program. It is well known, however, that the following property holds for convex programs.

Theorem 1 *Consider any convex minimization problem,* min $f(x)$ *where $x \in X \subset R^n$, f is a finite convex function and X a closed, convex, nonempty set. Then, the set of solutions of the convex minimization problem is convex and any local solution is a global solution.*

Proof: The set of solutions $\mathcal{S}$ to min $f(x)$ over $x \in X$ is given by

$$\mathcal{S} = \{x : x \in X, f(x) \leq f(x^*)\}, \tag{16}$$

which is a convex set, since f and X are convex, and the intersection of convex sets is convex.

Corollary 1 *The solution set, $\mathcal{S}$, given by (16), where f is given by (13) and X by (14)–(15), defines a set of linear-quadratic equalities and inequalities.*

Proof: Follows from the definitions of X, and $f(x)$.

Consider the optimization problem for an individual experiment, and recall that the optimal solution is non-unique. Without any a prior knowledge, it is hard to tell whether one solution would be better than any other. However, as the data set includes multiple experiments, and therefore multiple solution sets for the parameter values, we have additional information by which to reduce the size of the optimal solution set. For example, consider two optimal different solutions (parameter vectors) for experiment 1. We can say that one solution is 'better' than another if it produces better objective value when used in conjunction with data from experiment 2, and so on.

In other words, we can use the optimality condition from Theorem 1 to devise a *nesting* procedure for solving successively the quadratic minimization programs from each experiment, incorporating at each minimization the set of feasible and optimal solutions from the previous experiment(s).

Before doing so, however, note from the Corollary above that the solution set (16) is comprised of linear-quadratic inequalities and equalities, as opposed to the original feasible set, X. The following result shows that we can express the optimal solution set for the quadratic program without sacrificing the linearity of the constraints.

Theorem 2 *Consider as before any convex minimization problem,* min $f(x)$ *where* $x \in X \subset R^n$, f *is a finite convex function and* X *a closed, convex, nonempty set. Let* x^* *be some optimal solution. Then, the set of solutions of the convex minimization problem is convex and can be characterized by*

$$\mathcal{S} = \{x : x \in X, \nabla f(x^*)^T (x - x^*) = 0,\ \nabla f(x^*) = \nabla f(x)\}. \tag{17}$$

Proof: See [1][Thrm. 3.4.4].

Corollary 2 *The solution set,* $\mathcal{S}$, *given by (17), where* f *is given by 13 and and* X *by 14–15, defines a set of linear equalities and inequalities. That is, we have that when* f *is quadratic and given by* $f(x) = \frac{1}{2}x^T Hx + h^T x$, *and the feasible set* X *is polyhedral, the set of optimal solution set to* min $f(x)$, $x \in X$ *is polyhedral, and is given by*

$$\mathcal{S} = \{x : x \in X, H(x - x^*) = 0,\ h^T (x - x^*) = 0\}.$$

Proof: Substitute the gradient $\nabla f(x)$ into the solution set defined in (17). See also [1][Corr 3.4.4.2].

This latter characterization of the optimal solution set of each experiment's inference problem will be of direct use in defining the algorithm for obtaining a robust parameter vector from the set of multiple experiments.

5 The self-adjusting nested optimization method

The algorithm that we propose is based upon a nesting of successive solution sets for the parameter values across experiments. Indeed, as mentioned above, the solution vector for any experiment, n, here referred to as $(x^n)^*$, need not be unique, although the value of the quadratic program *at that solution*, $f((x^n)^*)$, is unique. This nonuniqueness makes the combination of experimental results across experiments quite challenging since, at each experiment, a set of optimal parameter values exists. What we would like therefore is that, once we solve for one such set of optimal values from one experiment, we consider, in the next experiment, not only those parameter vectors that are optimal (and feasible) for that next experiment, but also the full set of solutions that are optimal from the first experiment. This idea serves three objectives: on the one hand, we impose coherence across the multiple experiments, in terms of the parameter vectors that they produce from their inference problems, and, on the other hand, we successfully reduce the size of each solution set at successive experiments, by adding additional constraints to each inference problem. Lastly, we are provided with an inexpensive way in which to detect (and therefore remove) outlying experiments. The additional constraints are precisely the optimality of the parameter vector to all (or some) of the previous experiments' inference problems.

5.1 Basic Idea of the Method

The fundamental idea of the nested procedure is that we would like to *nest* the solution sets, one after the other, in the successive parameter estimation problems, so that in each experiment, $n > 1$, we consider a *subset* of the set of solutions to problem $n+1$. The subset that we are interested in is precisely the *intersection* of set of feasible solutions to the current $n+1^{st}$ problem and the set of *optimal* solutions to the previous, n^{th}, problem, and so on.

Proceeding in this manner, while the solution set for the final experiment, that is the N^{th} parameter set, may still not be a singleton, it will be considerably reduced in size through the intersection with all other optimal solution sets. In addition, it will be both optimal and feasible for all experiments. Furthermore, we will have guaranteed coherence across the multiple experiments' solutions.

Hereafter, for the n-th experiment's quadratic minimization problem, $n = 1, \dots, N$, we refer to the feasible set as X^n, the optimal solution set as $\mathcal{S}_n^*$, and the optimal solution value as f_n^*.

For simplicity in motivating the algorithm, let us first make an additional assumption to ensure feasibility of the results of performing the nesting procedure. We shall relax this assumption in the when we present the complete algorithm.

Assumption 2 *The intersection of the sets of optimal solutions to each experiment's quadratic minimization problem, $\mathcal{S}_n^*$, is nonempty, in other words, $\cap_{n=1\dots N}\mathcal{S}_n^* \neq \emptyset$.*

Under Assumptions 1 and 2, we are in a position to devise a basic, self-adjusting method for combining data from the N experiments with the aim of obtaining a robust set of parameters that best takes into account the results of all experiments. Based upon Theorem 2 and Corollary 2, the self-adjusting nested optimization method involves the addition of polyhedral constraints only, with respect to each experiments original inference problem.

1. Solve the quadratic minimization problem from experiment 1. Obtain an optimal solution, $x^{1,*}$.

2. Solve the quadratic minimization problem from experiment number ℓ, $\ell = 2 \dots N$, adding the following linear constraints at each iteration, ℓ:

$$x \in \cap_{n=1..\ell} X^n \tag{18}$$

$$H^n(x - x^{n,*}) = 0, \; n = 1..\ell - 1. \tag{19}$$

$$(h^n)^T(x - x^{n,*}) = 0, \; n = 1..\ell - 1. \tag{20}$$

Proposition 1 *Under the Assumptions 1, 2, the order in which the experiments are considered does not change the final solution set of the nested optimization procedure.*

Proof: Suppose the above nested optimization procedure is applied on N experiments in the order $l = 1, \dots, N$. Denote the optimal solution set to the nested optimization problem at step l as $\hat{\mathcal{S}}_l$. By construction of the nested method, we have $\hat{\mathcal{S}}_l \subseteq X_l \cap \hat{\mathcal{S}}_{l-1}$, for $l = 2, \dots, N$. Thus, $\hat{\mathcal{S}}_l = \hat{\mathcal{S}}_1 \cap \dots \cap \hat{\mathcal{S}}_l$. We claim that $\hat{\mathcal{S}}_l = \mathcal{S}_1^* \cap \dots \cap \mathcal{S}_l^*$, for all l. By Assumption 1 the solution set $\mathcal{S}_n^*$, $l = 1 \dots N$ exists, and by Assumption 2, the intersection $\mathcal{S}_1^* \cap \dots \cap \mathcal{S}_N^*$ is nonempty. It is therefore equivalent to consider the experiments in any order, since it does not change the value of the overall intersection of optimal solution sets, $\mathcal{S}_1^* \cap \dots \cap \mathcal{S}_N^*$. It suffices to show the claim for $l = 2$. The arguments are similar for larger numbers. Clearly, $\hat{\mathcal{S}}_1 = \mathcal{S}_1^*$. For any $y \in \mathcal{S}_1^* \cap \mathcal{S}_2^*$, we have $y \in X_2 \cap \hat{\mathcal{S}}_1$, and $f_2(x) \geq f_2(y)$ for all $x \in X_2$. We then must have $y \in \hat{\mathcal{S}}_2$. Therefore, $\mathcal{S}_1^* \cap \mathcal{S}_2^* \subset \hat{\mathcal{S}}_2$. To show the converse, suppose on the contrary that there exists $z \in \hat{\mathcal{S}}_2$ such that $z \notin \mathcal{S}_1^* \cap \mathcal{S}_2^*$. Since $\hat{\mathcal{S}}_2 \subset \hat{\mathcal{S}}_1 = \mathcal{S}_1^*$, it follows that $z \notin \mathcal{S}_2^*$. Thus for all $y \in \mathcal{S}_1^* \cap \mathcal{S}_2^*$, we must have $f_2(z) > f_2(y) = f_2^*$. Since y is also a feasible point in $X_2 \cap \hat{\mathcal{S}}_1$, z cannot be optimal for the nested problem, contradicting to the fact that $z \in \hat{\mathcal{S}}_2$. We therefore must have $\hat{\mathcal{S}}_2 = \mathcal{S}_1^* \cap \mathcal{S}_2^*$.

Corollary 3 *If any one of the Assumptions 1, 2 is not satisfied, then the order of the nesting of the multiple experiments has an effect on the final solution set.*

Proposition 1 implies that, under Assumptions 1, 2 we may in fact identify the intersection of all optimal solution sets in any order, whereas when the data is significantly noisy, or contains outliers, the ordering of the experiments will be apparent in the outcome. It is precisely this fact that enables the method, at a low computational surcharge, to *self-adjust*, i.e. to detect (and allow removal of) outlying or highly noisy experiments.

Indeed, the method is able to self-adjust, with minimal cost, to significantly noisy data, outlying experiments, by monitoring the residual error after each level of the nesting procedure, that is, each experiment, and detecting any significant jumps in the objective value. (Recall that the objective value is precisely the residual error of the estimation). By making use of a user-defined threshold, the experiment that caused the large jump in objective value can be eliminated from the data set and the procedure continued. The cost of this self-adjustment to noisy data is limited to the detection test performed, and hence is minimal.

5.2 Steps of the Self-adjusting Nested Estimation Method

The self-adjusting feature of the method requires that the user define a given tolerance, or a way to compute a tolerance depending upon the residual error of the estimation. The residual error is obtained at no addition cost, as it is precisely the objective value at each nest.

Let us refer to this (possibly state-dependent) tolerance as τ_ℓ, for state, or nest, ℓ.

When the Assumption 2 is not satisfied, it is necessary to *relax* the feasible set, that is, to allow an optimal solution to the n^{th} problem to be infeasible with respect to the intersection of the optimality set number $(n-1)$ and the feasibility set of the n^{th} problem. The idea is then to *minimize* the amount of this infeasibility. When a feasible solution exists, therefore, this amount will be minimized to zero, and we recover the original nested optimization method.

1. Solve the quadratic minimization problem from experiment 1. Obtain an optimal solution value, $x^{1,*}$ and the optimal solution, $f_1(x^{1,*})$ which we refer to as f_1^*.

2. Solve a modified version of the quadratic minimization problem from experiment number ℓ, $\ell = 2\dots N$, adding the following additional linear constraints:

$$\begin{aligned} x &\in \cap_{n=1..\ell} X^n(y_1), && (21)\\ H^n(x - x^{n,*}) &= y_2,\ n = 1..\ell-1, && (22)\\ (h^n)^T(x - x^{n,*}) &= y_3,\ n = 1..\ell-1, && (23)\\ y_1 &\geq 0. \end{aligned}$$

   where $X^n(y_1)$ is the feasibility set for experiment n, relaxed uniformly by the scalar y_1; that is, for any general set of constraints $X = \{g(x) \leq 0\}$, the relaxed feasible set in y_1 is given by $X(y_1) = \{g(x) \leq y_1\}$. We have thus the following dimensions for the relaxation parameters: $y_1 \in \Re_+$, $y_2 \in \Re_+^{\ell * I * J}$, $y_3 \in \Re_+^{\ell}$, and the following modified quadratic objective:

$$RE_\ell = \min_{x,y}\ f_\ell(x) + y_1 + y_2^2 + y_3^2.$$

3. If, at nest $\ell > 2$, the residual error, $RE_\ell \geq \tau_\ell$, then experiment ℓ is an *outlier*, and is discarded. Return to step 2 and set $\ell = \ell + 1$. If $\ell = 2$, then set $\ell = \ell + 1$ and return to step 1.

The self-adjustment step, 3, discards an experiment whose inclusion in the nesting procedure causes the residual error to increase more than the tolerance, τ_ℓ. If the first or second experiment (nest) causes this increase, it is impossible (without further testing) to determine which of the two is the outlier. The simplest solution, which is proposed in the algorithm, is to simply discard both experiments 1 and 2.

5.3 Further Uses of the Characterization of the Nested Optimality Set

Since the most common scenario in queueing network parameter inference is that the number of variables greatly exceeds the number of measurements available, the solution sets at each nest will likely not be singletons. In this case, it is of great interest to have a measure of how large the solution sets are, and in particular, how large the final nested solution set is. Indeed, in practice, it is often more important to give an *optimal range* of parameter values than one single value.

Using the self-adjusting nested method, it is straightforward to obtain a measure of the *size of the final solution set* (after having eliminated any outlying experiments), i.e., that of the N^{th} experiment, $\mathcal{S}_N^*$, by solving a sequence of linear programs with particular objectives, described below. Solving this sequence of LPs gives in effect the smallest multi-dimensional bounding box that contains the solution set.

To do so, solve the following pair of linear programming problems for every $i = 1 \ldots \dim(x)$:

$$\max\ x^T e^i,$$
$$\min\ x^T e^i$$

each objective subject to$x \in \mathcal{S}^*$, where $e^i = (0, \ldots, 1, \ldots 0)$ and the 1 is in the i^{th} row of the (column) vector, $\mathcal{S}^*$ is the solution set of the N^{th} experiment, $\mathcal{S}_N^*$, and $\dim(x)$ is the dimension of the vector x.

This provides an *outer approximation* of the final intersection of all (non-outlying) optimality sets, which can be used e.g. to provide a range of input data to the queueing network, after calibration.

Furthermore, note from Step 3, above, that the method detects outliers at no additional cost. An additional use of the outlier detection step is in identifying malfunctions or configuration/software defects. To do so, one need only collect the outliers and information about the experiments from which they were drawn. Using either problem determination or autonomic management tools, one can then attempt to identify if a fundamental problem has caused the measurement data to differ dramatically from the other measurement data by analyzing the patterns of the outliers. Indeed, while in most cases, an outlying experiment will be a random occurrence, it may be that a more structural change has occurred in the system.

6 Analysis and numerical experience

In this section, we illustrate the use of the algorithm on both synthetic and real data. On the synthetic data, the numerical tests demonstrate the accuracy of the estimates produced by the method, and its robustness to both noise and insufficient data. By insufficient data, we mean that the number of experiments performed is quite low as compared to the number of parameters to estimate. On the data obtained from a real IT system, the tests validate the use of the proposed queueing network inference approach.

6.1 Comparative results on synthetic data

As mentioned previously, we also compare our method with a version of the bundle-adjustment method used widely in 3-D image reconstruction, in which all experiments are lumped together as a single, large data set; in that approach, the problem is formulated as a single QP. We compare the two approaches in terms of estimation quality and computation time. Solution quality can be defined in a number of ways; here we consider two such metrics: the sum of squared error of the end-to-end measurements, and the values predicted by the model with the estimated parameters.

To test the robustness of the algorithm, it is necessary to vary the noise level in the data. Here, a noise level of 0 means that the (synthetic) data has no bias. A noise level of 1% means we have generated experiments with uniform noise throughout, at 1% deviation from the true values. The quality of the solution is evaluated with respect to the measured data, in particular, the first quality metric is defined by $\sum_{n=1}^{N}(\sum_{j=1}^{J} w_{j,n} \left(E_{j,n}^e - E_{j,n}^m\right)^2 + \sum_{i=1}^{I} \left(\rho_{i,n}^e - \rho_{i,n}^m\right)^2)$, that is, the total weighted least square error across all experiments. The second quality metric is measured with respect to the true parameter value; since this first set of tests have been run on generated data, we know the true values of the S. Thus we can also measure solution quality as $||S_{true} - S_{predicted}||/||S_{true}||$, i.e., the percentage deviation with respect to the true values of the parameters.

Several random sets of data were generated for each error level, and the solution quality obtained was averaged over those instances with the same error characteristics. In Figures 3 to 5, all problems were of the same size, with $I = J = 10$, and therefore 100 problem variables (i.e., not including dunny variables that permit handling infeasibility). The 6th figure illustrates the computation time for our algorithm and the benchmark method as problem size increases, from 9 to 625 problem variables.

Let us investigate how the the quality of the algorithm degrades with noise. Figure 3 gives a first indication of the robustness of the method. When the noise level is 0 and if we have a sufficient number of experiments, we should be able to determine the parameters exactly, i.e., no least squares estimation is needed in this case. Indeed, observe that the least squares error is 0, as shown in Figure 3. Note further that the least squares error increases very gradually with increasing noise. Indeed, even when the measurements are badly degraded by noise, the quality of the estimation produced is still very good, with a residual error of less than 0.1.

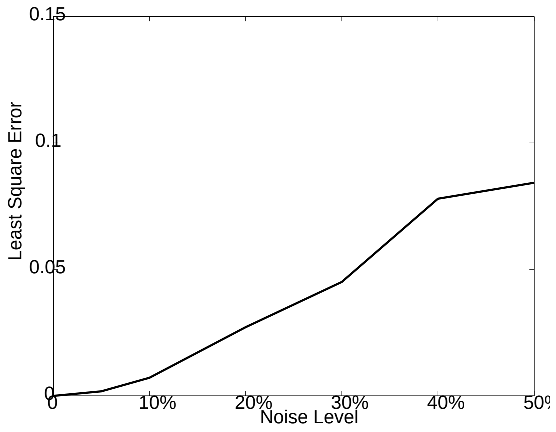

Figure 3: The auto-filtering nested algorithm scales well in the presence of (uniform) noise

Figure 4 compares the quality of the algorithm. in terms of the least squares error, with the single QP approach when one of the experiments is an "outlier", i.e., significantly biased with respect to the other experiments. We use a set of 6 experiments in total. The outlier has a bias of $||S_{outlier} - S||/||S|| = 1.5$. The position of the outlier in the 6 experiments is randomized. We plot the ratio of least square error(LSE), i.e., $LSE_{singleQP}/LSE_{nestedQP}$. The larger the value of the ratio is, the larger benefit we get by using the nested method vs the single QP method. As before, the data is averaged over several different runs to provide reliable statistics.

Note that outliers especially degrade the quality of the single QP approach, since all experiments are considered simultaneously and it is impossible to isolate the effect of a single bad experiment on the estimation. In this respect, the sequential, nested approach of our algorithm is really much more robust, as experiments are added one-by-one and the effect of any bad experiment is immediately visible. Further, with the self-adjustment capability, the nested algorithm automatically removes the bad experiment resulting in a much improved estimate. In fact, we find in these cases that the least squares error is 2 orders of magnitude better using our algorithm than that of the single QP, even when there is significant noise in all of the experiments.

Figure 5 compares the robustness of the auto-filtering nested algorithm with that of the single QP method when insufficient data is used. That is, the number of experiments available to estimate the parameters increases from 1 to 5, where 5 experiments provide an equal number of equations as there are unknowns. Furthermore, note that when there is only 1 experiment, the nested method and the single QP are identical, as is the case, in the absence of noise, when there is complete data (5 experiments in this example). Here the quality of the solution is measured with respect to the second metric, using the true parameter value.

We consider two scenarios. First, we consider an ideal scenario, i.e., the measurement is not degraded by noise and there is no outlying experiments. The second scenario is more realistic. We generate a set of 5 experiments with fixed 10% uniform noise and an 100% outlier, i.e., $||S_{outlier} - S||/||S|| = 1.0$. Again, the position of the outlier is randomized.

In Figure 5, the normalized difference in solution deviation is traced as the amount of data increases. As expected, the difference is null when there is 1 experiment for both cases and when there is complete data for the first case. However, of interest is what happens between those extremes; observe that the nested algorithm gives quality improvements of up to more than 30% for the first case and more than 60% for the second one.

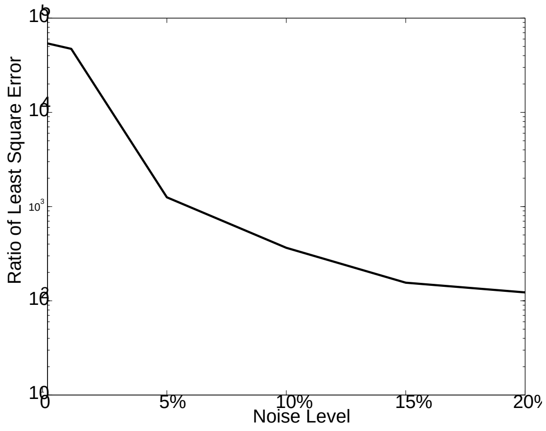

Figure 4: When an outlying experiment is included in the data, the solution quality of the auto-filtering nested algorithm is significantly better than the single QP approach

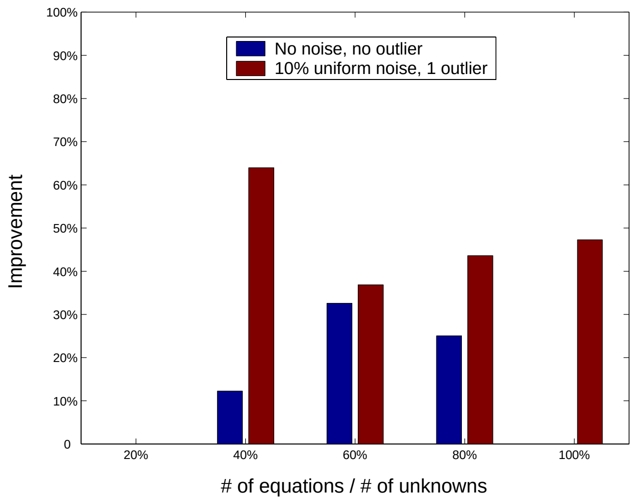

Figure 5: Normalized quality difference across the two methods as the amount of available data increases. At both extremes, the two methods are identical, in between the auto-filtering nested algorithm is superior.

Figure 6 compares the computation times of the two procedures. Here we have fixed the number of experiments at 3, and have varied the size of the optimization problem in each experiment from 9 to 625 problem unknowns (not including the dummy variables used to ensure feasibility). One would expect that the single QP is faster, since solving a single quadratic program, even if it has N times as many constraints and variables, would be faster than solving N separate quadratic programs; this is indeed true for the most part. The important point to take away from Figure 6 is that the improvement in robustness of the auto-filtering nested algorithm comes at a very modest computational surcharge (roughly a doubling of the computation time over the benchmark algorithm) yet it does not break down in the presence of outliers and non-uniform noise. Note that this behavior scales with problem size; as the problem gets larger, the more robust, nested method remains at roughly double the computation time of the single QP approach.

6.2 Validation Results from a real IT system

The proposed modeling framework and inference procedure have been tested as well on measurement data collected from a real IT environment for Web services with similar architecture to that shown in Figure 1.

Figure 7 illustrates the modeling results. In this figure, the light-gray bars represent the measured

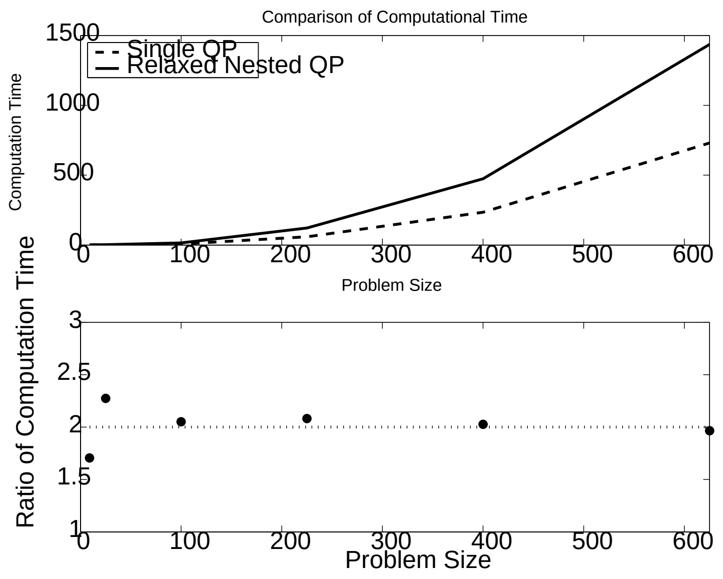

Figure 6: Computation times (in seconds) across methods on noise-free data as problem size increases. The second figure shows that the ratio of computation times of the two methods remains constant with increasing problem size.

end-to-end response time for each transaction. The dark-gray bars are those predicted by the model. Note that the values on the y-axis have been suppressed for reasons of confidentiality.

We have obtained the model parameters based on the measurements of one experiment when the system is experiencing 100% of its average workload. The input measurement data used for the experiment includes the end-to-end response times of all requests and utilization numbers at the different layers of the Web service architecture. Based on the inferred model, we then plot the mean response time for each transaction side by side with the measured mean response time in Figure 7. One can observe that the model matches well to all the measurements.

Figure 8 and Figure 9 illustrate the validating results against measurements. Here we use one experiment when the system is experiencing 150% of its average workload. Based on the inferred model we calibrated with the previous experiment, we calculate (and predict) the mean end-to-end response times for all transactions. The prediction results, shown as the dark-gray bars in Figure 8, are compared side-by-side along with those we measured which are shown in light-gray bars. The predicted versus the measured resource utilization at various servers in the Web service architecture are shown in Figure 9. Note that the model matches well not only all the transactions, but also the utilization at all the servers, to the measurements even though they are not used for calibration of the model. If we define relative accuracy $RA = 1 - \frac{|prediction - measurement|}{measurement}$, then more than 50% of time the relative accuracy is above 95%, and more than 90% of time the relative accuracy is higher than 80%. The quality of these results on real system data further validates the use of the *Kelly-type network assumption*; although the conditions for its applicability were not met, the system behaves closely enough so that it gives a very accurate prediction to the real system behavior.

7 Conclusions and future research directions

We have presented a formulation for estimating the parameters of queueing networks with processor sharing queues, using a quadratic programming framework. Based upon the properties of the quadratic program, we proposed a novel and robust algorithm for obtaining the best parameter values across a set of possibly incompatible experiments. The method is validated on a number of randomly-generated problem instances, and compared and contrasted against a benchmark algorithm used widely for the related problem of 3-D image reconstruction. It is demonstrated that at a modest computational surcharge, it is possible to get significantly more robust estimates through the proposed algorithm. The approach of

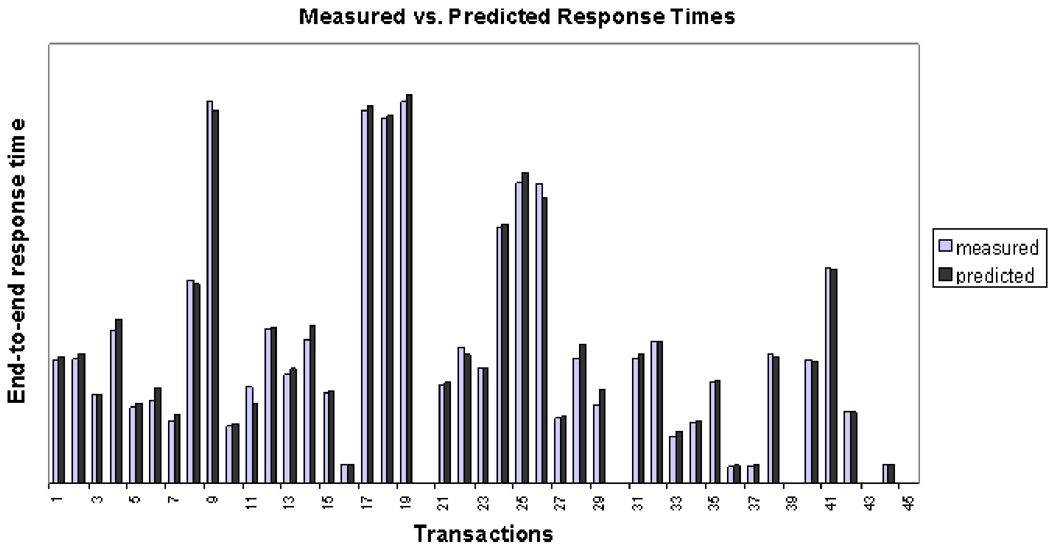

Figure 7: Modeling effect

using queueing network models to infer parameters of a typical IT system is then further validated on data coming from real IT systems, as they are used in practice.

Several areas for potentially valuable future research have emerged from this work. One such direction is an exploration into the use of the outer approximation presented in 5.3. The advantage of being able to describe each solution set in terms of simple box constraints is that any point can be very quickly said to lie inside or outside the box. Thus, rapid heuristics may be developed based upon that representation of the parameter solution sets, and of their intersection. Note that these methods are clearly heuristic since there are points in the boxes that are not in the true solution sets. For the purpose of practical studies, however, this approximation would be sufficient, and would allow very rapid analysis.

Second, we have raised the possibility of using the outlier-detection scheme within a problem determination or autonomic management role, in order to assist in detecting structural changes to the IT system. It would be of value to develop the methods to make best use of this data.

Third, note that we have assumed that the system is of the Kelly-type; however, observe that our work can readily be extended to the case of general Kelly-type networks with possibly FCFS queues. More generally, it can be extended to networks where the end-to-end delays can be decomposed to local delays.

Finally, another direction of research is the inference of higher moments of the service requirements, or even their distributions. For this, there will be a need of more detailed measurement data including higher moments of end-to-end response times.

References

[1] M.S. Bazaraa, H.D. Sherali, and C.M Shetty, *Nonlinear programming: Theory and Algorithms*, Second Ed., Wiley, New York, 1993.

[2] S. Alouf, P. Nain and D. Towsley, "Inferring network characteristics via moment-based estimators", in *Proceedings of the IEEE Infocom 2001 Conference*, April 2001.

[3] C. Bishop, *Neural Networks for Pattern Recognition*, Morgan Kaufmann, San Mateo, 1995.

[4] Mark Coates and Robert Nowak, "Sequential Monte Carlo Inference of Internal Delays in Nonstationary Communication Networks", *IEEE Transactions on Signal Processing*, Special Issue on Monte Carlo Methods for Statistical Signal Processing, 2001.

[5] N.G. Duffield, F. Lo Presti, "Multicast Inference of Packet Delay Variance at Interior Network Links", *IEEE Infocom 2000.*

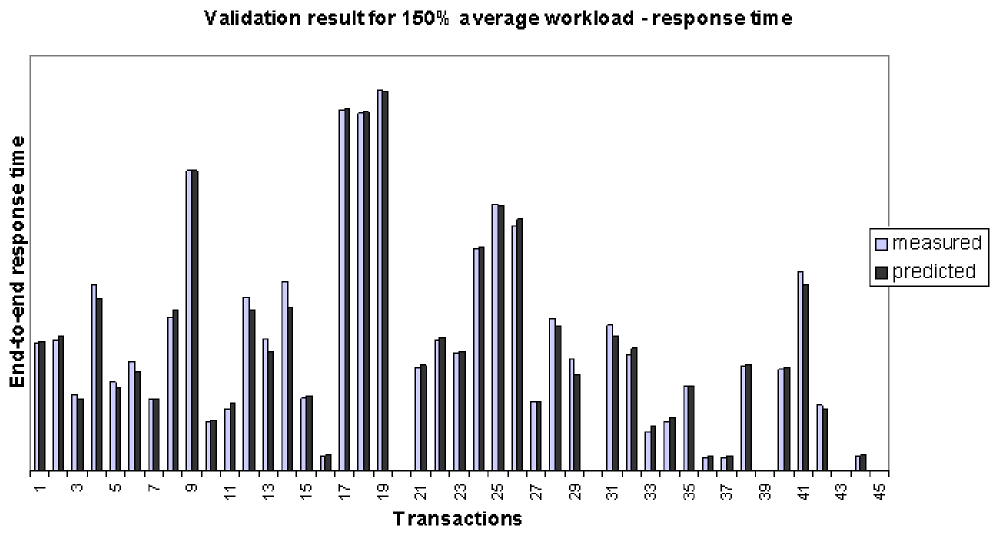

Figure 8: Validation against measured end-to-end delays

[6] M. Goldszmidt, D. Palma and B. Sabata, "On the quantification of e-business capacity", in *Proceedings of the 3rd ACM conference on Electronic Commerce*, p.235-244, October 2001, Tampa, Florida.

[7] F.P. Kelly, *Reversibility and Stochastic Networks*, Second Ed., John Wiley & Sons; 1979.

[8] L. Kleinrock, *Queueing Systems Volume II: Computer Applications*. John Wiley and Sons, 1976.

[9] B. Liu, M. Yu, D. Maier, and R. Männer, "Accelerated bundle adjustment in multiple-view reconstruction", in proceedings of the *7th Intnt'l Conf. on Knowledge-Based Intelligent Information and Engineering Systems (KES2003)*, 1012–1017, Oxford, UK, Sept, 2003.

[10] F. LoPresti, N.G. Duffield, J. Horowitz, and D. Towsley, "Multicast -Based Inference of Network-Internal Delay Distributions", UMass Computer Science TR99-55, Nov. 1999. To appear in *IEEE/ACM Trans. on Networking.*

[11] D. Menasce and V. Almeida, *Capacity Planning for Web Performance*, Prentice hall, 1998.

[12] V. Sharma, R. Mazumdar, "Estimating traffic parameters in queueing systems with local information", *Performance Evaluation*, 32:217-230, 1998.

[13] R.W. Wolff, *Stochastic Modeling and the Theory of Queues*, Prentice Hall, 1989.

[14] L. Zhang, C.H.Xia, M.S. Squillante and W.N. Mills III. Workload Service Requirements Analysis: A Queueing Network Optimization Approach, In *10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'02).*

[15] Y. Zhang, M.Roughan, N.G. Duffield, A.Greenberg, "Fast Accurate Computation of Large-Scale IP Traffic Matrices from Link Loads", *ACM Sigmetrics* 2003.

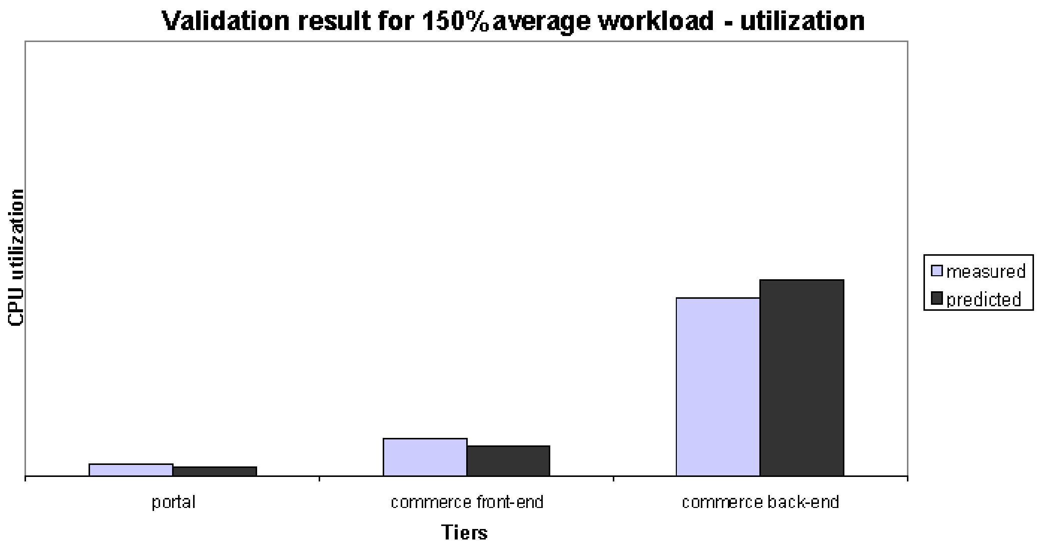

Figure 9: Validation against measured utilization