Bayesian Service Demand Estimation with Gibbs Sampling

Weikun Wang, Giuliano Casale
Department of Computing
Imperial College London, London, UK
{weikun.wang11,g.casale}@imperial.ac.uk

***Abstract*—Performance modelling of web applications involves the task of estimating service demands of requests at physical resources, such as CPUs. In this paper, we propose a service demand estimation algorithm based on a Markov Chain Monte Carlo (MCMC) technique, Gibbs sampling. Our methodology is widely applicable as it requires only queue length samples at each resource, which are simple to measure. Additionally, since we use a Bayesian approach, our method can use prior information on the distribution of parameters, a feature not available with existing demand estimation approaches.**

The main challenge of Gibbs sampling is to efficiently evaluate the conditional expression required to sample from the posterior distribution of the demands. This expression is shown to be the equilibrium solution of a multiclass closed queueing network. We define a novel approximation to efficiently obtain the normalising constant to make the cost of its evaluation acceptable for MCMC applications. Experimental evaluation based on simulation data with different model sizes demonstrates the effectiveness of Gibbs sampling for service demand estimation.

I. Introduction

The growing trend towards automating quality-of-service (QoS) management in web applications has been supported in recent years by research efforts focused on model-driven software engineering approaches [1], [2], [3], [4]. In these approaches, stochastic models such as queueing systems or Petri nets are defined for a reference system, possibly automatically from UML or Palladio diagrams [5], [6], and the QoS engineer is left with the problem of defining realistic values of the parameters for such models, such as the service demand placed by requests and software layers on the underlying physical resources. In this work, we propose a new Bayesian methodology to support the inference of demands. Our approach is capable of efficiently integrating known information about the parameters during the estimation. This is enabled by a novel approximation technique, proposed in the paper, to efficiently evaluate state probabilities in models of multiclass systems.

We here focus on queueing network models, which are a popular class of stochastic models used in model-driven engineering to evaluate software system performance [7]. Closed models, in particular, are the most popular for software systems since real applications are layered and the interactions between layers typically happen under admission control or finite threading limits [8]. For such models, the service demand at each physical resource needs to be determined from empirical data. Existing approaches perform demand inference in different ways: regression-based estimation procedures [9], nonlinear numerical optimization based on queueing formulas [10], and maximum likelihood methods [11]. Recent comparison analyses indicate that existing methods work in a number of cases, but they are not always accurate and there is no clear winner [11], [12]. Generally speaking, there is still a need for developing new approaches and methodologies for demand estimation from empirical datasets.

Bayesian inference of queueing model parameters has received some attention in the recent literature [11], [13]. Maximum likelihood methods have shown some potential in [11]. However, with the exception of [13], classic Bayesian methods such as Markov-Chain Monte Carlo (MCMC) have not been applied before to the problem of queueing model parameter estimation. Even though the method in [13] is promising, it currently only applies to open queueing networks and single class systems. In this paper, we push the boundary of demand estimation methods by presenting a new procedure based on Gibbs sampling that applies to the broader class of multiclass models.

Compared to previous work, our Bayesian methodology has a number of appealing features. First, it only requires samples of the number of jobs at the resources, which is a quantity not difficult to obtain in many software systems. For example, compared to utilization data, such queue-length information can be obtained also on deployments where the application does not have access to the underlying physical or virtualized resource layer, e.g., platform-as-a-service (PaaS) clouds or deployable PaaS (e.g., CloudFoundry, mOSAIC[1]). Compared to response time data, determining queue-length information can be done by looking only at arrival and departure of requests without tracking their identities, at most tracking their class. This avoids the overheads in production systems for monitoring response times of individual requests. Thus, queue-length based estimation appears more widely applicable than existing utilization-based and response-time based approaches.

We define inference based on queue-length data by looking at the equilibrium state distribution of the underlying queueing network. Differently from response time and utilization distributions, the equilibrium distribution of queue-lengths is well-known and under the BCMP theorem assumptions takes a convenient product-form expression [18]. In order to *efficiently* apply the Gibbs sampling to conditional likelihoods obtained from the BCMP product-form, we show that one needs to obtain an approximation for the normalizing constant of the state probabilities that needs to be amenable for computing integrals. While several approximations exist for *mean* perfor-

[1] http://www.mosaic-cloud.eu

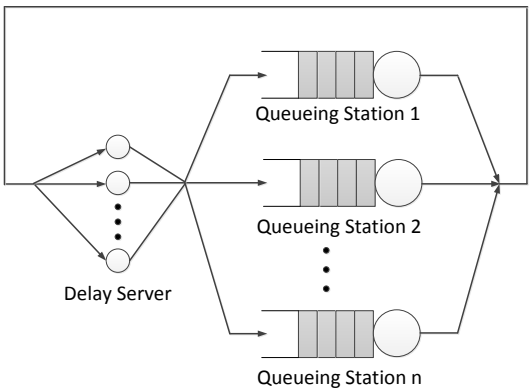


Fig. 1: Queueing network model of an interact application. Queues represent physical resources such as CPUs, the delay server represents user think times, see e.g. [16],[17].

mance metrics, no approximation exists for state probabilities unless one looks at asymptotic limits with many jobs and many queues [14], but existing limits in this regime cannot express think times. We show instead that the problem can be efficiently solved in the general case by applying existing local iterative approaches for mean-values, such as the Bard-Schweitzer algorithm [18] or AQL [15], together with a Taylor expansion of the normalizing constant. This provides, as a by-product, a novel approximate evaluation of the normalizing constant that can be applied for solving queueing network models in general, outside the scope of MCMC.

Summarizing, the main contributions of this work are:

- A novel demand estimation approach that leverages queue-length data, which is more widely available or less expensive to collect than other forms of input data.
- The ability to integrate in the estimation procedure known information by means of prior distributions of parameters in multiclass systems.
- A novel formula for approximating the normalising constant of multiclass product-form queueing networks, which is particularly suited for application in conjunction with the Gibbs sampling, but that can also be used as a stand-alone model evaluation technique.
- Extensive experimentation which shows that the proposed method makes accurate estimates of normalising constants and service demands.

The rest of this paper is organized as follows. Section II gives background and reviews previous work on demand estimation. In Section III a demand estimation based on Gibbs sampling algorithm is proposed together with a novel approach for approximating the normalising constant of multiclass networks. Finally, Section IV evaluates the algorithm and it is followed by concluding remarks.

II. Background

A. Model

We consider systems that can be modelled as the closed queueing network in Figure 1. This is a common model used for web applications [16],[17]. The delay server models inter-arrival times of requests from a population of K users having a known average think time. The model has M queueing stations and R classes of jobs. The queueing stations are the computing resources needed to process the requests. Requests from the users will be dispatched to stations according to a probabilistic policy. Service times and scheduling policies follow the BCMP theorem assumptions, thus: (i) the distribution of think times is general i.i.d.; (ii) queues can be processor sharing with general i.i.d. service times; (iii) queues can be first-come first-serve with exponential service distributions having different means; we point to [18] for additional information on the BCMP theorem assumptions.

Let n_{ij} be the number of jobs of class j at station i, and associate label $i = 0$ to the delay server. Define θ_{ij} to be the demand of class j requests at resource i, which is the product of the mean service time (i.e., execution time excluding contention) of a request and its mean number of visits at the resource before completion. Also, let the service demand matrix be $\boldsymbol{\theta} = (\theta_{11}, \ldots, \theta_{MR})$. Under the above assumptions, the probability for state $\boldsymbol{n} = (n_{01}, \ldots, n_{MR})$ admits the product-form expression [18]

$$P(\boldsymbol{n}|\boldsymbol{\theta}) = \frac{Z_1^{n_{01}}}{n_{01}!} \cdots \frac{Z_R^{n_{0R}}}{n_{0R}!} \prod_{i=1}^{M} n_i! \prod_{j=1}^{R} \frac{\theta_{ij}^{n_{ij}}}{n_{ij}! G(\boldsymbol{\theta})} \qquad (1)$$

where Z_j is the think time of class-j requests at the delay server and $G(\boldsymbol{\theta})$ is the normalising constant[2] of the state probabilities which ensures that $\sum_{\boldsymbol{n}} P(\boldsymbol{n}|\boldsymbol{\theta}) = 1$. Note that, in a Bayesian estimation setting, the state probability (1) may also be seen as the likelihood of observing state $\boldsymbol{n}$ given a particular value of the unknown parameters $\boldsymbol{\theta}$. Throughout this paper, we assume that think times are known, but the approach could be easily extended with minor changes also to the general case where the conditioning is also made on the vector $\boldsymbol{Z} = (Z_1, \ldots, Z_R)$ of think times. We also remark that efficiently computing normalizing constants is crucial for demand estimation using MCMC and may be performed by several recursive algorithms, e.g., [18]. However, none of these existing methods can compute $G(\boldsymbol{\theta})$ in polynomial time as the number of jobs, classes and queues grow large simultaneously. The method proposed in Section III-C addresses this problem.

B. Related work

Previous approaches for demand estimation may be categorised in the following groups.

Regression. In [9], linear regression is introduced for resource demand estimation and [19] proposes to use multivariate linear regression to estimate mean service demands. In [20], a regression based methodology is presented to estimate the CPU demand in multi-tier systems. Experiments with the TPC-W benchmark demonstrate that such approximation is robust to diverse workloads with changing transaction mixes. Moreover, the work in [21] uses Least Trimmed Squares regression based on utilization of the stations and system throughput to estimate service times. The study in [22] considers dynamically estimating CPU demands: the problem is formulated

[2]Notice that for the normalizing constant we explicit the dependence on $\boldsymbol{\theta}$, instead of the more common dependence on the class populations, since $\boldsymbol{\theta}$ is more important for the MCMC methods introduced later.

as a multivariate linear regression of CPU utilization against throughputs. In [23], an evaluation of a class of regression techniques for demand estimation is presented. Then an online regression parameters tuning technique is demonstrated and found to achieve high accuracy. Compared to regression methods, our technique requires only queue-length information and can be more efficient with few samples thanks to the ability of incorporating prior distributions in the analysis.

Machine learning. The work in [24] proposes to use independent component analysis, a machine learning approach, to infer workload characteristics based on CPU utilization and the number of customers. The study in [25] proposes a method based on clustering to estimate the service time. The authors employ density based clustering to obtain clusters and then use clusterwise regression algorithm to estimate the service time. In [13], the authors propose a method to use Bayesian inference for estimation of the demand time in the system. By utilizing MCMC to sample from the posterior distribution, this method proves to be robust to missing data. [26] proposes an algorithm to estimate the service demands with structural changes. A time-based linear clustering algorithm is employed to identify different linear clusters for each service demands. In [27], an estimation of arrival and service rate approach based on queue length is presented. The authors define a Ornstein-Uhlenbeck diffusion to achieve the approximation. Unlike our method, [27] is only for open queueing networks. Compared to the above methods, our approach is the first one to apply MCMC to closed models with multiclass workloads.

Optimization methods. The work in [28] proposes an approach to use quadratic programming to estimate the service demands. End-to-end response time as well as utilization of the stations and system throughput are required in input. Both [29] and [30] use Extended Kalman Filter for parameter tracking. While in [29], a calibration framework based on fixed test configurations is proposed, [30] applies tracking filters on time-varying systems. [30] extends [31] and adapts the Kalman filter to estimate the performance model parameters based on utilization of stations and response time. In [32], a novel iterative approach for resource demand estimation based on linear programming is proposed for multi-tier systems. An extensive evaluation demonstrates effectiveness of the method. Compared to these works, our technique allows to include prior information on the parameters and it requires only queue-length data.

Finally, the work in [12] presents a detailed exploration of the state-of-the-art in resource demand estimation technologies. The accuracy of the algorithms is compared by analysing them in the same environment and varying the parameters of the test environment to see the sensitivity of the methods estimates.

III. Markov-Chain Monte-Carlo Estimation

In this section, we introduce MCMC methods for demand estimation. Markov Chain Monte Carlo [33] methods are a class of algorithms to simulate a target density $f(\boldsymbol{\theta})$ by constructing a Markov chain which has $f(\boldsymbol{\theta})$ as its equilibrium distribution. MCMC methods are often used to draw samples from high dimensional spaces to solve integration and optimisation problems. Classical MCMC methods involve the Metropolis-Hastings algorithm [34] and Gibbs sampling [35]. We now investigate if such methods may be used to infer the service demand matrix $\boldsymbol{\theta}$ from observation of a set of queue-length states $\boldsymbol{N} = \{\boldsymbol{n}^{(1)}, \boldsymbol{n}^{(2)}, \ldots, \boldsymbol{n}^{(D)}\}$ in a system, where $\boldsymbol{n}^{(i)}$ is the i-th state observed.

A. Metropolis-Hastings

1) General algorithm: The Metropolis-Hastings algorithm constructs a Markov chain with continuous state space in order to sample from a target distribution $f(\boldsymbol{\theta})$. A random walk is realised as follows. For iteration $t+1$, the algorithm randomly generates a new parameter $\boldsymbol{\theta}$ from a distribution $q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$ proposed by the user, where $\boldsymbol{\theta}^{(t)}$ is the value sampled at the previous iteration. The algorithm next computes an acceptance probability

$$\alpha = \min\left(1, \frac{f(\boldsymbol{\theta})p(\boldsymbol{\theta})q(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta})}{f(\boldsymbol{\theta}^{(t)})p(\boldsymbol{\theta}^{(t)})q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})}\right) \tag{2}$$

where $p(\boldsymbol{\theta})$ is the prior distribution of $\boldsymbol{\theta}$ and $f(\boldsymbol{\theta})$ is the target density we want to sample from. Then Metropolis-Hastings accepts the candidate $\boldsymbol{\theta}$ with probability α and sets $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}$ if the sample is accepted, or otherwise rejects it by setting $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)}$. After S iterations the algorithm has produced an output sample of size S of vectors $\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(S)}$ distributed according to the target distribution $f(\boldsymbol{\theta})$.

2) Application to demand estimation: If the target distribution is a posterior $f(\boldsymbol{\theta}|\boldsymbol{N})$ where $\boldsymbol{N}$ is an empirical dataset of queue-length observations and $\boldsymbol{\theta}$ is the service demand matrix, then the generated samples will be candidate values for the demands $\boldsymbol{\theta}$. Given the posterior samples $\boldsymbol{\theta}^{(1)}$,...,$\boldsymbol{\theta}^{(S)}$, an estimate of $\boldsymbol{\theta}$ can be obtained by component-wise averaging the samples, possibly after discarding a portion of them to account for the initial burn-in period. Other possibilities to evaluate the posterior include taking the component-wise median of the samples or using the maximum a posteriori principle (MAP), which returns the mode of $f(\boldsymbol{\theta})$. Thus, if we can compute the posterior, Metropolis-Hastings can be easily applied to the demand estimation problem. For example, by using the convolution algorithm [18] one could readily compute (1) and run Metropolis-Hastings. However, we have found some advantages and some issues in the practical application of Metropolis-Hastings to demand estimation.

One advantage of the Metropolis-Hastings algorithm is that it updates all the dimensions at the same time. This can be very efficient for low dimensional spaces, i.e., for small models with few queues and classes. However, this advantage becomes a problem in the context of medium-scale and large-scale multiclass queueing networks as we now explain. In a BCMP network, let

$$f(\boldsymbol{N}|\boldsymbol{\theta}) = P(\boldsymbol{N}|\boldsymbol{\theta}) = \prod_{\boldsymbol{n}\in\boldsymbol{N}} P(\boldsymbol{n}|\boldsymbol{\theta}) \tag{3}$$

be the likelihood that the empirical data is generated by a particular choice $\boldsymbol{\theta}$ of the service demands, where $P(\boldsymbol{n}|\boldsymbol{\theta})$ is the likelihood of observing state $\boldsymbol{n}$, which is simply the state probability in formula (1). From Bayes theorem, we have that the posterior is

$$P(\boldsymbol{\theta}|\boldsymbol{N}) = \frac{P(\boldsymbol{N}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\boldsymbol{N})} = \frac{\prod_{\boldsymbol{n}\in\boldsymbol{N}} P(\boldsymbol{n}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\boldsymbol{N})} \tag{4}$$

Algorithm 1 Gibbs Sampling

Require: $\boldsymbol{\theta}^{(\mathbf{0})}$, $f(\boldsymbol{\theta})$, S
  for $s = 1 : S$ **do**
    for $i = 1 : c$ **do**
      sample $\theta_i^{(s)}$ from $f(\theta_i|\theta_1^s, \ldots, \theta_{i-1}^s, \theta_{i+1}^{(s-1)}, \ldots, \theta_c^{(s-1)})$
    end for
  end for

where $P(\boldsymbol{\theta})$ is the prior distribution of the demands provided by the user. The Metropolis-Hastings algorithm allows to sample from $P(\boldsymbol{\theta}|\boldsymbol{N})$ without knowledge of the normalizing constant $P(\boldsymbol{N})$, since this term simplifies in the formula of the acceptance probability α. Thus, the method is readily applicable if we can choose a suitable prior distribution and we know how to efficiently compute the likelihoods $P(\boldsymbol{n}|\boldsymbol{\theta})$. Unfortunately, the latter problem seems difficult to address in the context of Metropolis-Hastings. The likelihood (1) requires calculating the normalising constant $G(\boldsymbol{\theta})$ for the current value of $\boldsymbol{\theta}$. Then, with the Metropolis-Hastings algorithm we would need to recompute at each iteration the normalising constant since we are altering $\boldsymbol{\theta}$. This is problematic because computational methods for the normalising constant, such as convolution [18], have computational requirements that grow exponentially with the size of the queueing network model; for example, a model with 4 classes or more is normally too expensive to solve. Thus, even a single evaluation of the normalizing constant is problematic on large models. We show in Section IV-C the computational implications of this problem.

Summarizing, a MCMC method like Metropolis-Hastings may not be efficiently applied to closed systems due to the cost of computing the normalizing constant. Later in Section III-C, we propose a method to address the normalizing constant computation problem based on a Taylor expansion of $G(\boldsymbol{\theta})$. This method is efficient in the context of MCMC if $\boldsymbol{\theta}^{(t+1)}$ and $\boldsymbol{\theta}^{(t)}$ differ at most along a single dimension, which is always the case in Gibbs sampling, but not in Metropolis-Hastings. Thus, from now on we focus on Gibbs sampling as a MCMC method to estimate service demands and we limit to present a comparison with Metropolis-Hastings in Section IV.

B. Gibbs sampling

Unlike the Metropolis-Hastings algorithm, Gibbs sampling updates each demand separately, one at a time. We show that in this case the normalising constant can be updated iteratively in an efficient manner that avoids multivariate expansions and multiple integrals. We now describe the approach in general, followed by its application to demand estimation.

1) General algorithm: The general procedure of Gibbs sampling is described in Algorithm 1, where the parameter vector $\boldsymbol{\theta}$ is supposed to have c dimensions, $\boldsymbol{\theta}^{(0)} = (\theta_1^{(0)}, \ldots, \theta_c^{(0)})$ is the starting point, $f(\boldsymbol{\theta})$ is the joint distribution of all θ_i, $1 \leq i \leq c$, and S is the number of samples that the user requires to the Gibbs sampling algorithm. Then, the algorithm will generate S samples with a distribution that will converge to the target density $f(\boldsymbol{\theta})$ [33]. The main difficulty of applying Gibbs sampling to demand estimation is to efficiently compute the conditional distribution $f(\theta_i| \ldots)$.

2) Application to demand estimation: In multiclass queueing networks, $\boldsymbol{\theta}$ will have MR unknowns since we assume the think times known, and $f(\boldsymbol{\theta})$ is the posterior $P(\boldsymbol{\theta}|\boldsymbol{N})$ given in (4). Therefore, in the application of Gibbs sampling to the demand estimation problem, we need to compute efficiently the conditional distribution $P(\theta_{ij}|\boldsymbol{\theta}_{(ij)}, \boldsymbol{n})$, where we denote by $\boldsymbol{\theta}_{(ij)}$ the vector $\boldsymbol{\theta}$ without θ_{ij}. With this conditional distribution we can directly apply the Gibbs sampling procedure by conditioning on all observations $\boldsymbol{N}$ as we now explain.

In order to obtain the conditional distribution for each demand θ_{ij}, we condition as follows

$$P(\theta_{ij}|\boldsymbol{\theta}_{(ij)}, \boldsymbol{n}) = \frac{P(\boldsymbol{\theta}, \boldsymbol{n})}{\int_{\theta_{ij}} P(\boldsymbol{\theta}, \boldsymbol{n}) \mathrm{d}\theta_{ij}} \tag{5}$$

where the integral is on the entire range of definition of θ_{ij} for the prior distribution. Then using Bayes theorem

$$\begin{aligned} P(\theta_{ij}|\boldsymbol{\theta}_{(ij)}, \boldsymbol{n}) &= \frac{P(\boldsymbol{\theta}|\boldsymbol{n})}{\int_{\theta_{ij}} P(\boldsymbol{\theta}|\boldsymbol{n}) \mathrm{d}\theta} = \frac{\frac{P(\boldsymbol{n}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\boldsymbol{n})}}{\int_{\theta_{ij}} \frac{P(\boldsymbol{n}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\boldsymbol{n})} \mathrm{d}\theta} \\ &= \frac{P(\boldsymbol{n}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{\int_{\theta_{ij}} P(\boldsymbol{n}|\boldsymbol{\theta})P(\boldsymbol{\theta}) \mathrm{d}\theta} \end{aligned} \tag{6}$$

Thus, in order to apply Gibbs sampling to the problem under study we need the ability of integrating state probabilities across a possibly large range of feasible demand values for all MR parameters θ_{ij}. Instead of calculating the integral analytically, we propose to approximate it numerically. The integral in the denominator of the last equation may be approximated as

$$\int_{\theta_{ij}} P(\boldsymbol{n}|\boldsymbol{\theta})P(\boldsymbol{\theta}) \approx \sum_{\theta_{ij} \in \boldsymbol{I}} P(\boldsymbol{n}|\boldsymbol{\theta})P(\boldsymbol{\theta})\Delta\theta_{ij} \tag{7}$$

where $\boldsymbol{I}$ is the integral range and $\Delta\theta_{ij}$ is the step size between two values of θ_{ij}. We will assume for simplicity that $\Delta\theta_{ij} = \Delta\theta = \text{const}$ for all the dimensions. In our approach, the values of $\boldsymbol{I}$ and $\Delta\theta$ are defined as input to the demand estimation algorithm. The integral range $\boldsymbol{I}$ can be decided from the prior distribution $P(\boldsymbol{\theta})$. For example, if the prior distribution is an uniform distribution then $\boldsymbol{I}$ could be inside the lower bound and upper bound of the uniform distribution. Finally, the computation of (7) can be readily done provided that we can compute efficiently the normalizing constant in (1), which we discuss in Section III-C. We explain in the next subsection how to calculate the conditional posterior distribution on the entire dataset $\boldsymbol{N}$, i.e. $P(\theta_{ij}|\boldsymbol{\theta}_{(ij)}, \boldsymbol{N})$ in a numerically stable fashion, which is important since such probabilities can be very small.

3) Stable computation of likelihood on $\boldsymbol{N}$: Since the calculation of numerator and denominator in $P(\boldsymbol{n}|\boldsymbol{\theta})$ may involve numbers that exceed the machine floating point range, numerical exceptions arise in practice upon computing $P(\boldsymbol{n}|\boldsymbol{\theta})$. For example, since the normalising constant is a summation of terms on an exponentially large state space, its value often overflows or underflows the floating point range. Furthermore, upon using (3), the repeated multiplications may lead to further exceptions when the probabilities become too small. To tackle this problem, we notice that the structure of (1) allows to work

Algorithm 2 Gibbs Sampling for Demand Estimation

Require: $\boldsymbol{N}, \boldsymbol{I}, \Delta\theta, M, R, \boldsymbol{K}, S, \boldsymbol{Z}$
$\boldsymbol{\theta}^{(0)} = (0, \ldots, 0)$
calculate $\log G(\boldsymbol{\theta}^{(0)})$ by (13)
for $s = 1 : S$ **do**
  for $i = 1 : M$ **do**
    for $j = 1 : R$ **do**
      $\forall \theta_{ij} \in \boldsymbol{I}$, compute by (12)

$$\log G(\theta_{11}^{(s)}, \ldots, \theta_{i(j-1)}^{(s)}, \theta_{ij}, \theta_{i(j+1)}^{(s-1)}, \ldots, \theta_{MR}^{(s-1)});$$

      calculate $P(\theta_{ij}|\boldsymbol{\theta}_{(ij)}^{(s-1)}, \boldsymbol{N})$, $\theta_{ij} \in \boldsymbol{I}$ by (9) and (10);
      randomly generate $\theta_{ij}^{(s)}$ from $P(\theta_{ij}|\boldsymbol{\theta}_{(ij)}^{(s-1)}, \boldsymbol{N})$, $\theta_{ij} \in \boldsymbol{I}$;
      set $\boldsymbol{\theta}^{(s)} = (\theta_{11}^{(s)}, \ldots, \theta_{i(j-1)}^{(s)}, \theta_{ij}^{(s)}, \theta_{i(j+1)}^{(s-1)}, \ldots, \theta_{MR}^{(s-1)})$
    end for
  end for
end for
return mean value of $\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(S)}$ or a subset thereof to filter the burn-in period

out the majority of algebraic operations using only logarithms, for example

$$\begin{aligned}\log(P(\boldsymbol{n}|\boldsymbol{\theta})) = &\sum_{j=1}^{R}\left(n_{0j}\log(Z_j) - \sum_{u=1}^{n_{0j}}\log(u)\right) \\ &+ \sum_{i=1}^{M}\sum_{u=1}^{n_i}\log(u) - \log G(\boldsymbol{\theta}) \\ &+ \sum_{i=1}^{M}\sum_{j=1}^{R}\left(n_{ij}\log\theta_{ij} - \sum_{u=1}^{n_{ij}}\log(u)\right) \quad (8)\end{aligned}$$

is simply the log-likelihood of state state $\boldsymbol{n}$ given the demands $\boldsymbol{\theta}$. Now, combining (6), (7) and (3) in an obvious manner, we can write the logarithm of the conditional posterior distribution for the entire dataset as

$$\begin{aligned}\log(P(\theta_{ij}|\theta_{(ij)}, \boldsymbol{N})) = &\sum_{d=1}^{D}\log(P(\boldsymbol{n}^{(d)}|\boldsymbol{\theta})) + \log(P(\boldsymbol{\theta})) \\ &- \log\left(\sum_{\theta_{ij}\in\boldsymbol{I}}\exp\left(\sum_{d=1}^{D}\log(P(\boldsymbol{n}^{(d)}|\boldsymbol{\theta}))\right)P(\boldsymbol{\theta})\Delta\theta\right)\end{aligned} \quad (9)$$

where the probability $\log(P(\boldsymbol{n}^{(d)}|\boldsymbol{\theta}))$ can be obtained from (8). We compute the final value of the conditional distribution as

$$P(\theta_{ij}|\theta_{(ij)}, \boldsymbol{N}) = \exp(\log(P(\theta_{ij}|\theta_{(ij)}, \boldsymbol{N}))) \quad (10)$$

In our experience, the above passages dramatically reduce the chances of experiencing numerical issues upon working with probabilities and normalising constants. Some problems may arise in (10) when the final result is too small to represent, however since we are computing a probability, which has finite support in [0,1], we can simply set $P(\theta_{ij}|\theta_{(ij)}, \boldsymbol{N}) = 0$ in these cases. This avoids all numerical problems connected to the use of probabilities and normalising constants, which are a well known issue in the context of multiclass BCMP networks.

C. Approximating the Normalising Constant

The main remaining obstacle for applying Gibbs sampling is to efficiently calculate the logarithm of the normalising constant $G(\boldsymbol{\theta})$ in (8). The problem being considered here is that, as the number of classes and queues increases, the direct computation of the equilibrium probability distribution (1) requires an exponentially increasing computational effort, in both time and space. This is because the state space size grows as $\prod_{j=1}^{R}\binom{K_j+M-1}{M-1}$ and, today, no computational techniques exist that have computational costs growing polynomially in all the model parameters. Since the requirement of Gibbs sampling is to perform an integral on a set of normalizing constant for varying values of a parameter θ_{ij} in (7), we need to efficiently approximate the value of the normalizing constant in such a way that we can also perform this integral efficiently.

1) Approximation Method: Our approach consists in applying the following first-order approximation. We perform a Taylor expansion of $G(\boldsymbol{\theta})$ along the dimension that the Gibbs sampling is updating and we use approximate mean-value analysis (MVA) algorithms in the resulting formulas to calculate the first-order derivative appearing in the expansion [15]. To the best of our knowledge, this is the first time that a technique is proposed to approximate normalizing constants without considering asymptotic limits (e.g., many job, or many queues/many jobs limit regimes).

Recall from sensitivity analysis of queueing networks that [36]

$$\frac{dG(\boldsymbol{\theta})}{d\boldsymbol{\theta}_{ij}} = \frac{Q_{ij}}{\theta_{ij}}G(\boldsymbol{\theta}), \quad (11)$$

where Q_{ij} is the mean queue-length of jobs of class j at queueing station i. Consider now a first-order Taylor expansion of $G(\boldsymbol{\theta})$

$$G(\boldsymbol{\theta} + d\boldsymbol{\theta}_{ij}) = G(\boldsymbol{\theta}) + \frac{dG(\boldsymbol{\theta})}{d\boldsymbol{\theta}_{ij}}d\boldsymbol{\theta}_{ij} + o(\theta_{ij}^2)$$

where $d\boldsymbol{\theta}_{ij}$ updates only dimension ij. Using (11) and ignoring higher-order terms we obtain the approximation

$$G(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_{ij}) \approx G(\boldsymbol{\theta})\left(1 + \frac{Q_{ij}}{\theta_{ij}}\right)\Delta\boldsymbol{\theta}_{ij} \quad (12)$$

where $\Delta\boldsymbol{\theta}_{ij} = \Delta\theta \cdot \mathbf{1}_{ij}$ is a vector of length $\Delta\theta$ in direction ij. Note that expression of $\log G(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_{ij})$ is readily obtained from (12). Also, observe that the approximate evaluation of Q_{ij} is efficient, since the Bard-Schweitzer and AQL algorithms typically return it in few seconds or even fractions thereof and their complexities are just $O(MR)$ and $O(MR^2)$ respectively [15]. Based on this observation, we conclude that (12) can be efficiently evaluated in (7) as follows. We perform an iteration starting from any value of θ_{ij} for which $G(\boldsymbol{\theta})$ is known, for example the value $\theta_{ij}^{(s-1)}$ generated at the previous iteration of Gibbs sampling. In computing the summation (7), we simply run the approximate MVA algorithm for each value of θ_{ij} to obtain $G(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}_{ij})$ from the previously computed value $G(\boldsymbol{\theta})$ in the summation. Thus, as long as one knows a initial value for $G(\boldsymbol{\theta})$, the computation of (7) can be done easily; the iterative nature of the Gibbs sampling method always provides the requested initial $G(\boldsymbol{\theta})$, also for $s = 1$ where we will use a closed-form expression of $G(\boldsymbol{\theta}^{(0)})$ described later in equation (13).

TABLE I: $\log G(\boldsymbol{\theta})$: Sensitivity to $\Delta\theta$

step size	Exact	BS	AQL
10^{-1}	-378.45	-445.43	-445.43
10^{-2}		-441.83	-442.01
10^{-3}		-405.37	-405.91
10^{-4}		-382.61	-383.86
10^{-5}		-377.81	-379.23
10^{-6}		-377.27	-378.71

TABLE II: $\log G(\boldsymbol{\theta})$: Sensitivity to K

K/R	Exact	BS	AQL
10	-43.25	-43.26	-43.28
30	-205.68	-205.49	-206.31
50	-378.45	-382.61	-383.86
70	-551.19	-570.10	-571.45
90	-723.87	-770.48	-771.85
110	infeasible	-984.30	-985.65

2) Validation: To illustrate the accuracy of this normalizing constant approximation approach, we report two experiments.

Test 1. We test a case with $R = 3$, $M = 3$. The number of jobs in each class is identical and set to $K/R = 50$, where $K = \sum_j K_j$ is the total number of jobs across all classes. The think time is $Z_j = 1$, for all classes j. The exact value of the demands is

$$\boldsymbol{\theta} = \begin{bmatrix} \frac{1}{50} & \frac{1}{60} & \frac{1}{70} \\ \frac{1}{50} & \frac{1}{60} & \frac{1}{70} \\ \frac{1}{50} & \frac{1}{60} & \frac{1}{50} \end{bmatrix}$$

where rows are for stations and columns for classes. Notice that two queues are identical, which is a pessimistic scenario for approximate MVA accuracy [15]. We use different step sizes $\Delta\theta$ to see the changes in the approximation accuracy for $\log G(\boldsymbol{\theta})$. We use the convolution algorithm to calculate the exact value of $\log G(\boldsymbol{\theta})$. The results are listed in Table I. It is easy to see that the results of our approximations are increasingly better with smaller step sizes. For the AQL algorithm, as the step size increases the results will always be better because it computes a better approximation of the queue length compared to Bard-Schweitzer [15]. For the Bard-Schweitzer algorithm the results are still good considering its much cheaper approximation compared to AQL.

Test 2. We now explore how the results change while varying the number of jobs in the model. In this case, we still use the parameters of the above experiment and fix the step size to 10^{-4}. We keep the fraction of jobs for each class remain identical as we increase the total job population K. The results are shown in Table II. We can see for the same step size, as the model size decreases, both the Bard-Schweitzer and AQL algorithms produces better results, which indicates that they require a small step size for optimal accuracy. When the number of jobs of each class equals 110, we find that using convolution we cannot calculate the normalising constant because the value is too small for the machine floating point range. Instead, the Bard-Schweitzer and AQL algorithms remain feasible. Notice also that the approximation error becomes more visible as K grows large, however the errors become visible only in models that are heavily saturated, with utilisation ranges that are often not realistic in applications (bottleneck queue utilization greater than 99.9%). Even including this error, we show in Section IV that the Gibbs sampling demand estimation is accurate.

TABLE III: Execution time (s)

(a) Different $\Delta\theta$ ($K/R = 50$)

$\Delta\theta$	Exact	BS	AQL
10^{-1}	30.10	0.001	0.0001
10^{-2}		0.19	0.53
10^{-3}		0.57	1.19
10^{-4}		6.18	10.41
10^{-5}		59.80	126.56
10^{-6}		573.37	1679.6

(b) Different K ($\Delta\theta = 10^{-4}$)

K/R	Exact	BS	AQL
10	0.29	5.71	5.1
30	7.31	5.64	8.47
50	30.10	5.64	10.53
70	85.23	5.92	11.38
90	181.66	5.79	12.49
110	335.15	6.05	12.98

The execution time of the three approaches is shown in Table III in which we combine the above two experiments. As the model dimension grows, convolution requires more time to return an answer. Furthermore, if the normalising constant is either too small or too large, as stated in the previous section, convolution must be implemented using symbolic algebra or infinite precisions libraries (e.g., GNU GMP) which results in a large computational overhead. For our proposed approximations, the execution time of the Bard-Schweitzer and AQL based approximations takes seconds at most and the cost of their application mainly depends on the step size $\Delta\theta$. Therefore these two methods are quite effective in approximating the normalising constant for large-scale or medium-scale queueing models. The Bard-Schweitzer takes less time because it is an approximation method and sacrifices accuracy for time. For example, on extremely large models with tens of queues and classes, AQL becomes inefficient and Bard-Schweitzer is much more practical to use. For models with more realistic sizes, AQL yields more accurate evaluations of the normalizing constant at acceptable computational costs.

D. Initialization, prior distributions, final pseudocode

In Gibbs sampling, we set the starting guess for the demands to be the vector $\boldsymbol{\theta} = (Z_1, \ldots, Z_R, 0, \ldots, 0)$, in which case we have $P(K_1, \ldots, K_R, 0, \ldots, 0|\boldsymbol{\theta}) = 1$. For this case, substituting the probability into (1), it is simple to get the initial normalising constant, which is

$$\log G(\boldsymbol{\theta}) = \sum_{j=1}^{R} K_j \log(Z_j) - \sum_{k=1}^{K_i} \log(k) \qquad (13)$$

From this starting point, we are able to use formula (12) to progressively update $\log G(\boldsymbol{\theta})$ as we update θ_{ij} during the computation of (7).

A prior distribution $P(\boldsymbol{\theta})$ is also required for the application of the Gibbs sampling approach. Priors can be important especially when the empirical dataset $\boldsymbol{N}$ is small, since known information about the distribution of parameters can help in better discriminating among the feasible estimates. In this paper, we assume the simplest case where the user chooses a uniform distribution for the prior, thus each parameter θ_{ij} is equilikely in a range $\boldsymbol{I} = [\theta_{ij}^-, \theta_{ij}^+]$. However, our methodology is general and can accept arbitrary prior distributions, typical alternatives are normal or gamma distributions.

Summarizing, in this and in the previous subsections we have defined several steps that enable the efficient execution

of Gibbs sampling as a method for demand estimation. The final pseudocode of the Gibbs sampling method for demand estimation that we have developed is shown in Algorithm 2.

IV. Experimental Evaluation

A. Methodology

We evaluated the performance of Gibbs sampling for service demand estimation by extensive experimental campaigns. Our experiments have been run on a machine with an Intel Core i7-2600 CPU, running at 3.4 GHz and with 16 GB of memory. We have tested single class and multiple classes cases using simulated data from the jSIMgraph simulator of the Java Modelling Tools package [37]. To establish performance across typical scenarios, we have considered topologies where all stations are in series or all stations are in parallel, which are both quite common in models of real applications.

Since one could record millions of records in real life systems, we have used during the experimentation a simple method to compress input data while preserving the empirical probabilities $P(\boldsymbol{n}|\boldsymbol{N})$ of each observed state $\boldsymbol{n} \in \boldsymbol{N}$. First, we analyse the data and estimate the empirical values of $P(\boldsymbol{n}|\boldsymbol{N})$, for all $\boldsymbol{n}$. Then, given a target number L of input samples, we use these empirical probabilities $P(\boldsymbol{n}|\boldsymbol{N})$ to resample the trace obtaining just L samples. Notice that this resampling approach also allows L to be greater than the cardinality D of the input set $\boldsymbol{N}$. We then apply Gibbs sampling to the new dataset.

1) Evaluation criteria: We use the mean absolute percentage error as the evaluation criteria for the accuracy of the estimators, which is

$$\epsilon = \frac{\sum_{i=1}^{M}\sum_{j=1}^{R}\frac{|\theta_{ij}-\theta_{ij}^{*}|}{\theta_{ij}}}{MR} \tag{14}$$

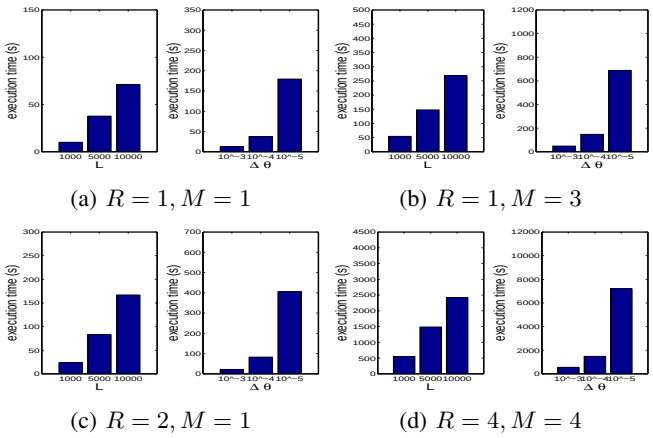
where θ_{ij} is the exact service demand value and θ_{ij}^{*} is the value estimated by our algorithm. We discard the first $S/2$ half of samples in averages to avoid the bias of the initial burn-in. We display the 95% confidence interval of the result as well.

We have not compared our algorithms with others listed in the related work. This is because to the best of our knowledge, our proposed algorithm is the only one capable of using only input samples $\boldsymbol{n}$ without other measurements. Thus, direct comparisons would have little meaning since the methods would perform inference from different input data which we assume unavailable. For calculating the queue length in formula (12) we use the Bard-Schweitzer algorithm.

2) Experiment definition: We first explore how the algorithm behaves for single job class and multiple classes. In these two experiments, we set the prior to be a uniform distribution and ranges from 0 to 0.2 for all experiments. This also defines the integral range $\boldsymbol{I}$. Then we have tested how the prior distribution will change the results when there are only few input data points and thus prior information becomes important for the estimation.

B. Results

1) Single class ($R = 1$): First, we test single class models with a single queueing station and think time $Z = 1$ (for the ease of computation of formula (8), we set the think time equals to 1 in all our experiments), the service demands for the queueing station is $\theta_{ij} = 1/50$, $\forall i, j$, and we simulate 500,000 samples. The default parameter set up for our algorithm is $L = 5000$, $S = 50$, $D = 10,000$ (last D timestamps for the simulation) and $\Delta\theta = 0.0001$. Then we vary the following parameters separately for different number of jobs between 10 and 90 with step 20 (light load to heavy load):



(a) $R = 1, M = 1$ (b) $R = 1, M = 3$

(c) $R = 2, M = 1$ (d) $R = 4, M = 4$

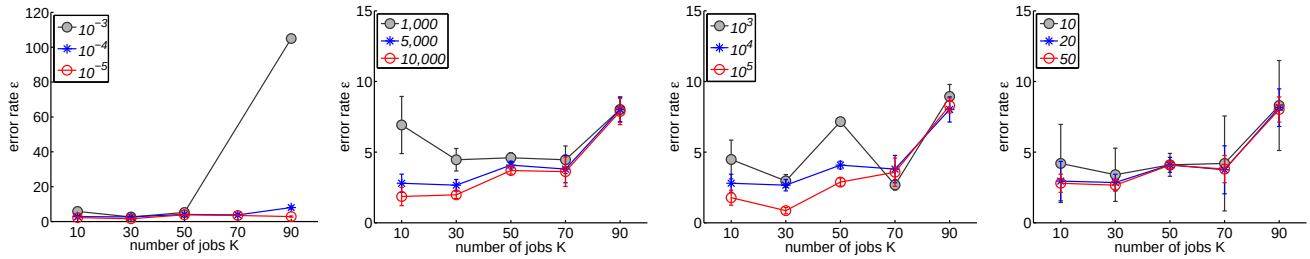
Fig. 5: Execution time

1) $L \in \{1000, 5000, 10000\}$
2) $S \in \{10, 20, 50\}$
3) $\Delta\theta \in \{10^{-3}, 10^{-4}, 10^{-5}\}$
4) $D \in \{1000, 10000, 100000\}$

From Figure 2 we can find that the overall result for the default parameter setup is quite good with an average error rate of 4.27%. With smaller step size, the result is generally better. This is what we expected. It can be noticed that with a step size of 10^{-3}, when the number of customers equals to 90, accuracy degrades. We think that this is because it is an extreme case with server utilization equals to nearly 100%, therefore it requires smaller step sizes to estimate the normalising constant accurately. In applications, this extremely high utilization normally will not happen. For different number of samples L, the trend is similar as the step size since it represents $P(\boldsymbol{n}|\boldsymbol{N})$ more accurately. With more input data the result is generally better too, because $P(\boldsymbol{n}|\boldsymbol{N})$ becomes more accurate. We can also observe that the Gibbs sampling converges very quickly because the result for $S = 20$ samples and $S = 50$ samples are almost the same, but the confidence interval for 50 samples is much smaller as expected.

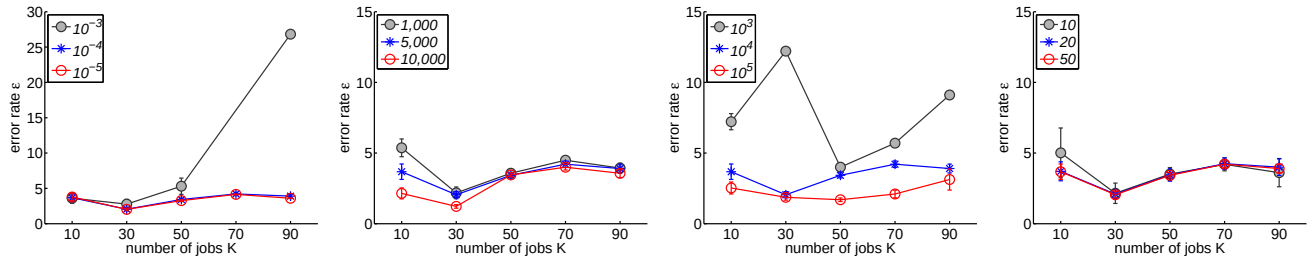
Then we test a single class model with 3 queueing stations in series and $\boldsymbol{\theta} = [1/50; 1/40; 1/40]$. The other default parameters are the same as the above experiment. The results are presented in Figure 3. The general result for the default parameter set up is quite accurate with an error rate of 3.45%. The trend observed in this multiple queues case is the same as in single queue case. In Figure 2c, we find that when $K = 30$ and $D = 10^3$, the result seems inconsistent with other trends. We think this is because the data does not represent well the steady state being too small.

2) Multiple classes ($R > 1$): First, we test two job classes with a single queue. In simulation we set the service demands for the queueing station $\boldsymbol{\theta} = [1/40, 1/50]$ for each class and



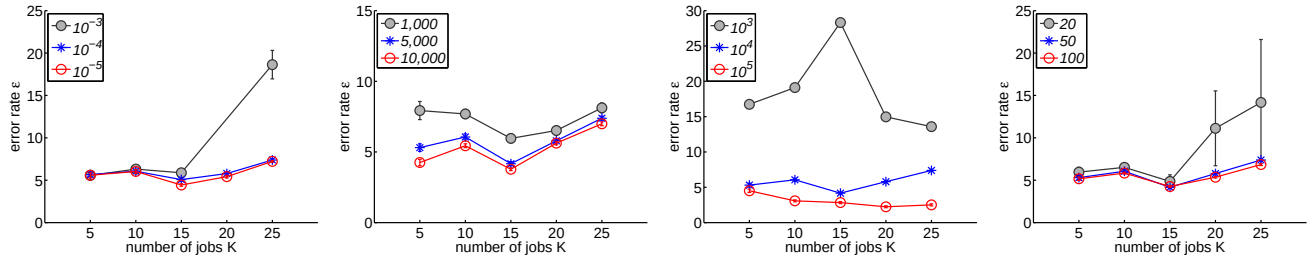
(a) $\Delta\theta$ step size (b) L samples (c) D simulation records (d) S generated samples

Fig. 2: $R = 1$ class, $M = 1$ queueing station



(a) $\Delta\theta$ step size (b) L samples (c) D simulation records (d) S generated samples

Fig. 3: $R = 1$ class, $M = 3$ queueing stations



(a) $\Delta\theta$ step size (b) L samples (c) D simulation records (d) S generated samples

Fig. 4: $R = 4$ classes, $M = 4$ queueing stations

again use 500,000 simulation samples. The default parameters for our algorithm are $L = 5000$, $S = 50$, $\Delta\theta = 0.0001$, $D = 10,000$ timestamps. Then we vary the parameters separately for different number of jobs between 10 and 50 with step 10 for each class. The parameters we vary are the same as the single class case.

The overall accuracy is quite, given a step size of 10^{-4}, we have an average error rate of 3.77% for all the loads. The same trend observed in this case is the same as in single class case. Due to space limitation, we do not present the results here. Next, we set 4 queueing stations in serial in the model with four classes of jobs, The service demand is equal to

$$\boldsymbol{\theta} = \begin{bmatrix} \frac{1}{40} & \frac{1}{50} & \frac{1}{60} & \frac{1}{50} \\ \frac{1}{50} & \frac{1}{30} & \frac{1}{20} & \frac{1}{60} \\ \frac{1}{60} & \frac{1}{40} & \frac{1}{30} & \frac{1}{40} \\ \frac{1}{40} & \frac{1}{60} & \frac{1}{50} & \frac{1}{50} \end{bmatrix}$$

. The default parameters for our algorithm are $L = 5000$, $S = 50$, $\Delta\theta = 0.0001$, $D = 10000$ timestamps. Then vary the following parameters separately for different number of jobs between 5 and 25 with step 5 for each class. The parameters we vary are the same as the previous experiment except that we vary the number of samples $S \in \{10, 20, 50\}$.

From Figure 4, we can find the overall accuracy is generally good, for all the 16 service demand we have an average error rate 5.05% among all the loads given the default parameters. As the step size becomes smaller, the result is better. This is also true for the number of sample and input data. Even this experiment contains 16 service demand to estimate, the algorithm still converges very fast. The average result for the 50 samples and 100 samples case are almost the same, which indicates our algorithm already converges at least around the 25th sample. In Figure 4c, when $K = 15$ and $D = 10^3$ there also exists an outlier. We think this is probably because the data does not represent well the steady state.

We finally run an experiment with the same parameter setup as the above one, but instead of putting all the stations in series we put them in parallel. We set the number of customers between 10 and 50 with step 10 for each class. Due to space limitation, we do not present the result in a figure. The result for the default parameter setup is 7.48%. The same trends are observed as in the serial case. The reason that the result is slightly worse than that of the serial experiment is that parallel topology requires more logger to take records. Therefore for

TABLE IV: Recommended parameter setup

L	S	$\Delta\theta$	D
5000	50	10^{-4}	10000

TABLE V: Results of the recommended parameter setup

Experiment	Time(s)	Average	Upper	Lower
$R = 1, M = 1$	38	4.27%	4.91%	3.64%
$R = 1, M = 3$	148	3.45%	3.75%	3.16%
$R = 2, M = 1$	83	3.77%	4.17%	3.37%
$R = 4, M = 4(Ser)$	1489	5.05%	5.22%	4.88%
$R = 4, M = 4(Pal)$	1460	7.48%	7.16%	7.79%

the same amount of input data, the series topology will have a high accuracy of representing $P(\boldsymbol{n}|\boldsymbol{N})$. However, also in the parallel case the results are highly accurate.

The execution time mainly depends on the following parameters L, S and $\Delta\theta$. Effects of S are easy to understand which is just proportional to the run time of collecting one single sample; $\Delta\theta$ determines the accuracy of approximation and more accurate $\Delta\theta$ value means more iteration in the process; A large L value means that the input data will be large which leads to more computational time. The run time is similar as we vary the load, because in our algorithm load does not have a major impact on the time. Therefore we present the average run time of all loads varying L and $\Delta\theta$.

The average execution time for the above experiments are shown in Figure 5. We do not show the execution time of the parallel four stations experiment here because its run time is similar to the serial case. We find that the algorithm is quite fast for simple cases, i.e., with few dimensions. Besides, considering its fast convergence, we may need only 20 samples to get the final result which will be faster. We can also observe that for the last experiment is quite time consuming (average 1489 seconds for the default parameter setup of all the loads) to generate 50 samples for all the 16 parameters. Since our algorithm is mainly for off-line use, given its typical running costs, this amount of time is acceptable. In addition, if we have good prior distribution for the demand, the algorithm will definitely speed up. This will be discussed in the next subsection.

Overall, we give the recommended parameter set up in Table IV for using the proposed estimation methods. For the number of input records D, in general the more the better. Here we only give the default parameter of D we have used. Summarizing, with this default parameter set up, we report in Table V the execution time, average error and the lower and upper confidence interval obtained for the above five experiments.

3) Prior distribution: Here, we studied how the prior distribution will affect the result and execution time when there are only few input data. For this purpose, we have tested different kinds of prior distributions:

- uniform distribution from 0 to 1 for all demands
- uniform distribution from 0 to 0.1 for all demands
- uniform distribution from θ_{ij} - δ to θ_{ij} + δ for each demand, where θ_{ij} is the exact service demand and δ is 0.03, 0.01, or 0.001.

TABLE VI: Comparison: Gibbs and Metropolis-Hastings

	Error rate	Execution time (s)
Metropolis-Hastings	432%	906
Gibbs	5.03%	470

We run the above tests based on the serial $M = 4, R = 4$ case. The results for using $D = 10$ records and $D = 30$ records and the average execution time are presented in Figure 6. For the first two cases of prior distributions, we find that the result overlaps, both producing the worst accuracy, which means that the prior is still too wide. However, the execution time reduces significantly as the prior narrows. This is because the prior also defines the integral range, leading to a smaller number of iterations as the prior tightens. For the latter three cases, we can see that the accuracy increases gradually as the prior narrows, which indicates that a good prior will definitely improve the result. The execution time also reduces as observed.

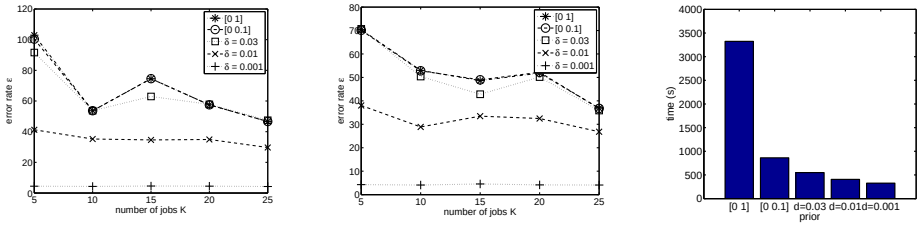
C. Comparing Gibbs sampling and Metropolis-Hastings

Finally, we compare Metropolis-Hastings and Gibbs sampling. We set the queueing model to have $R = 3$ job classes and $M = 3$ queueing stations. The number of jobs is $K/R = 40$ per class, and the think times are $Z_j = 1$, for all classes j. The service demands are

$$\boldsymbol{\theta} = \begin{bmatrix} \frac{1}{50} & \frac{1}{60} & \frac{1}{70} \\ \frac{1}{60} & \frac{1}{50} & \frac{1}{40} \\ \frac{1}{50} & \frac{1}{50} & \frac{1}{60} \end{bmatrix}$$

We use Java Modelling Tools to simulate 500,000 samples and we collect the last $D = 10000$ samples for demand estimation. Then we reduce the data to $L = 5000$ samples. The number of samples requested to the methods is $S = 50$. The prior distribution is a uniform distribution between 0 and 0.2. For Metropolis-Hastings, we set the proposed distribution $q(\boldsymbol{\theta}^{t+1}|\boldsymbol{\theta}^t)$ to be a multivariate normal distribution with means equal to 0.1 and a diagonal covariance matrix with variances all equal to 0.001. For Gibbs sampling, we set the step size to be $\Delta\theta = 10^{-4}$.

We list the results and execution time of both algorithms in Table VI. We find that the result for Metropolis-Hastings algorithm is quite inaccurate and takes a long time compared to Gibbs sampling. We believe that this is due to a number of reasons. First, the high computational times are partly due to the application of the convolution algorithm and partly due to the choice of the prior which can result in slow mixing. The choice of $q(\boldsymbol{\theta}^{t+1}|\boldsymbol{\theta}^t)$ also affects how Metropolis-Hastings progresses. A small variance will lead to slow mixing of the Markov chain while a larger one yields a volatile sample which may reduce the acceptance rate of samples. Summarizing, Metropolis-Hastings involves the additional challenges of choosing an effective distribution $q(\boldsymbol{\theta}^{t+1}|\boldsymbol{\theta}^t)$ together with the problem of approximating the normalizing constant in an efficient manner. No methods are available today to guide these choices. Together, these two problems suggest that Gibbs sampling is a much simpler choice for Bayesian service demand estimation.



(a) $D = 10$ (b) $D = 30$ (c) Execution time

Fig. 6: Prior distribution experiment

V. Conclusion

In this paper, we have presented a service demand estimation methodology based on Gibbs sampling. Differently from existing approaches, our method requires only queue length data, which is easy to collect in real systems. We have shown the effectiveness of the proposed algorithms against simulation.Future research directions could include, among many possible extensions, the estimation also of think times and the evaluation of the impact of different prior distributions.

References

[1] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," *IEEE Computer*, 39(2): 25-31, 2006.

[2] J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai, "Performance model driven QoS guarantees and optimization in clouds," *Proc. of ICSE CLOUD Workshop*, 15–22, 2009.

[3] H. Bruneliere, J. Cabot, *et al.*, "Combining model-driven engineering and cloud computing," *Proc. of MDA4ServiceCloud*, 2010.

[4] D.Ardagna, E. Di Nitto, *et al.*, "Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds," in *Proc. of MISE Workshop*, 2012.

[5] M. Tribastone and S. Gilmore, "Automatic extraction of PEPA performance models from UML activity diagrams annotated with the MARTE profile," *Proc. of WOSP*, 67-78, 2008.

[6] S. Becker, H. Koziolek, and R. Reussner, "The palladio component model for model-driven performance prediction," *Journal of Systems and Software*, 82(1): 3-22, 2009.

[7] V. Cortellessa and R. Mirandola, "Deriving a queueing network based performance model from UML diagrams," *Proc. of WOSP*, 58-70, 2000.

[8] J. A. Rolia and K. C. Sevcik, "The method of layers," *IEEE Trans. Soft. Eng.*, 21(8): 689-700, Aug. 1995.

[9] J. Rolia and V. Vetland, "Parameter estimation for performance models of distributed application systems," *Proc. of CASCON*, 54, 1995.

[10] Y. Liu, I. Gorton, and A. Fekete, "Design-level performance prediction of component-based applications," *IEEE TSE*, 31(11): 928-941, 2005.

[11] S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson, "Estimating service resource consumption from response time measurements," *Proc. of Valuetools*, 48, 2009

[12] S. Spinner, "Evaluating approaches to resource demand estimation," Master's thesis, Karlsruhe Institute of Technology, 2011.

[13] C. Sutton and M. I. Jordan, "Bayesian inference for queueing networks and modeling of internet services," *The Annals of Applied Statistics*, 5(1): 254-282, 2011.

[14] C. Knessl and C. Tier, "Asymptotic expansions for large closed queueing networks with multiple job classes," *IEEE Trans. Computers*, 41(4): 480-488, 1992.

[15] J. Zahorjan, D. L. Eager, and H. M. Sweillam, "Accuracy, speed, and convergence of approximate mean value analysis," *Perform. Eval.*, 8(4): 255-270, 1988.

[16] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi, "An analytical model for multi-tier internet services and its applications," in *Proc. of ACM SIGMETRICS*, 291-302, 2005.

[17] Q. Zhang, L. Cherkasova, and E. Smirni, "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," *Proc. of ICAC*, 27-27, 2007.

[18] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains*, 2nd ed., John Wiley and Sons, 2006.

[19] J. Rolia and V. Vetland, "Correlating resource demand information with arm data for application services," *Proc. of WOSP*, 219-230, 1998.

[20] Q. Zhang, L. Cherkasova, and E. Smirni, "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," in *Proc. of ICAC*, 27-27, 2007.

[21] G. Casale, P. Cremonesi, and R. Turrin, "Robust workload estimation in queueing network performance models," *Proc. of Euromicro PDP*, 183-187, 2008.

[22] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi, "CPU demand for web serving: Measurement analysis and dynamic estimation," *Perform. Eval.* , 65(6): 531-553, 2008.

[23] A. Kalbasi, D. Krishnamurthy, J. Rolia, and M. Richter, "Mode: mix driven on-line resource demand estimation," *Proc. of CNSM*, 1-9, 2011.

[24] A. B. Sharma, R. Bhagwan, M. Choudhury, L. Golubchik, R. Govindan, and G. M. Voelker, "Automatic request categorization in internet services," *SIGMETRICS PER*, 36(2): 16-25, 2008.

[25] P. Cremonesi, K. Dhyani, and A. Sansottera, "Service time estimation with a refinement enhanced hybrid clustering algorithm," *Proc. of ASMTA*, 291-305, 2010.

[26] P. Cremonesi and A. Sansottera, "Indirect estimation of service demands in the presence of structural changes," *Proc. of QEST*, 249-259, 2012.

[27] J. V. Ross, T. Taimre, and P. Pollett, "Estimation for queues from queue length data," *Queueing Systems*, 55(2): 131-138, 2007.

[28] Z. Liu, L. Wynter, C. H. Xia, and F. Zhang, "Parameter inference of queueing models for it systems using end-to-end measurements," *Perform. Eval.* , 63(1): 36-60, 2006.

[29] X. Wu and M. Woodside, "A calibration framework for capturing and calibrating software performance models," *Computer Performance Engineering*, 32-47, 2008.

[30] T. Zheng, C. Woodside, and M. Litoiu, "Performance model estimation and tracking using optimal filters," *IEEE Trans. Software Eng.*, 34(3): 391-406, 2008.

[31] T. Zheng, J. Yang, M. Woodside, M. Litoiu, and G. Iszlai, "Tracking time-varying parameters in software systems with extended kalman filters," *Proc. of CASCON* , IBM Press, 334-345, 2005.

[32] A. Kalbasi, D. Krishnamurthy, J. Rolia, and S. Dawson, "Dec: Service demand estimation with confidence," *IEEE Trans. Software Eng.*, 38(3): 561-578, 2012.

[33] A. R. Webb, *Statistical pattern recognition*. Wiley, 2003.

[34] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," *Biometrika*, 57(1): 97-109, 1970.

[35] S. Geman, D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE T. PAMI*, 6:721-741, 1984.

[36] E. de Sousa e Silva and R. R. Muntz, "Simple relationships among moments of queue lengths in product form queueing networks," *IEEE Trans. on Computers*, 37(9): 1125-1129, 1988.

[37] M. Bertoli, G. Casale, and G. Serazzi, "Java modelling tools: an open source suite for queueing network modelling and workload analysis," in *Proc. of QEST*, 119-120, 2006