Learning Queuing Networks via Linear Optimization

Emilio Incerto
emilio.incerto@imtlucca.it
IMT School for Advanced Studies Lucca
Lucca, Italy

Annalisa Napolitano
annalisa.napolitano@imtlucca.it
IMT School for Advanced Studies Lucca
Lucca, Italy

Mirco Tribastone
mirco.tribastone@imtlucca.it
IMT School for Advanced Studies Lucca
Lucca, Italy

ABSTRACT

The automatic derivation of analytical performance models is an essential tool to promote a wider adoption of performance engineering techniques in practice. Unfortunately, despite the importance of such techniques, the attempts pursuing that goal in the literature either focus on the estimation of service demand parameters only or suffer from scalability issues and sub-optimality due to the intrinsic complexity of the underlying optimization methods.

In this paper, we propose an efficient linear programming approach that allows to derive queuing network (QN) models from sampled execution traces. For doing so, we rely on a deterministic approximation of the average dynamic of QNs in terms of a compact system of ordinary differential equations. We encode these equations into a linear optimization problem whose decision variables can be directly related to the unknown QN parameters, i.e., service demands and routing probabilities. Using models of increasing complexity, we show the efficiency and the effectiveness of our technique in yielding models with high prediction power.

CCS CONCEPTS

• **Software and its engineering → Software performance**; **Software system models**; • **Mathematics of computing → Linear programming**.

KEYWORDS

Software performance, Queuing networks, Automated model extraction, Linear programming

ACM Reference Format:
Emilio Incerto, Annalisa Napolitano, and Mirco Tribastone. 2021. Learning Queuing Networks via Linear Optimization. In *Proceedings of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3427921.3450245

1 INTRODUCTION

Queuing networks (QNs) are a well-known method in performance engineering to gain insights about non-functional properties of computing systems such as throughput, utilization, and response time [5, 30]. Unfortunately, a major problem affecting their widespread adoption in the practice of software engineering is the gap between the mathematical formulation of the model and the real system, which requires significant expertise both in the problem domain and in the techniques and tools used for the analysis [36]. Such a gap widens as the real system becomes more mature, while for early stages of the software development process several methods have been proposed for the derivation of performance models from higher-level designs (see, e.g., the surveys [3, 22]).

In this paper we propose a new method based on linear programming to learn the parameters of a QN. For each station representing a shared resource in the system, we assume that the server multiplicities are already known. This assumption corresponds, for instance, to knowing the number of cores when the shared resource is a CPU; or the thread-pool size when the shared resource is a software entity. Instead, the parameters to be learned are instead the service demands, i.e., the amount of time spent by a job at a service station; and the routing matrix, specifying the probability with which a job visits a new station upon receiving service.

These two kinds of parameters are more difficult to obtain in practice without intrusive instrumentation of the actual system under consideration [33]. This has motivated a line of research on service demand estimation from queue-length data only [29, 33, 34], since this information is more readily available using operating-system facilities, in that, compared to response time measurements, queue lengths can be collected by looking at incoming and outcoming events only. Our method shares the same approach but, differently from most of the literature on this subject, is able to estimate both service demands and the routing matrix at the same time. Formally, this makes the problem nonlinear in general, because the parameters for service demands and the routing probabilities feature as multiplicative factors in the differential equations that give the stochastic dynamics of the QN [5]. In addition, this setting is computationally intractable exactly, since the number of states of a QN's underlying continuous-time Markov process grows combinatorially with the population levels in the network.

These observations have led to treating the estimation of QN parameters as a nonlinear optimization problem where more compact—albeit approximate—dynamical equations are used as constraints and the objective function is to minimize the distance between queue-length observations and their predictions. In [13] this has been recently done by encoding such nonlinear problem as a recurrent neural network which encodes the QN dynamics based on the well-known *mean-field* (or *fluid*) approximation. It essentially consists of one ordinary differential equation (ODE) for each station, which estimates its average queue length [23].

In this paper we show how the same problem—namely, learning routing probabilities and service demands of a QN—can be

ICPE '21, April 19–23, 2021, Virtual Event, France

ACM ISBN 978-1-4503-8194-9/21/04…$15.00
https://doi.org/10.1145/3427921.3450245

achieved by linear optimization, thus considerably reducing the computational cost of the estimation. Our method is still based on the equations derived by the fluid approximation, but it exploits a different interpretation. In most applications to the domain of software performance engineering (e.g., [17–21]), these equations are taken, roughly speaking, as the asymptotically correct deterministic dynamics of the (average) queue lengths in the limiting regime when the population of jobs goes to infinity. Therefore, for any (real) finite situation the fluid equations introduce estimation errors that are rather difficult to bound theoretically (e.g., [12]). The source of error for finite population comes from estimating the expectation of a function of a random variable as the function of the expectations. This is only true when these functions are linear, but, unfortunately, QN equations do feature nonlinearities. In our approach, the constraints used in our linear program involve functions of the random variables (i.e., the queue lengths) of the process. Although closed form ODEs for their expectations cannot be derived, samples of these random variables can be estimated from queue-length data. In other words, our linear program is such that the constraints represent the correct relationship between the random variables as opposed to the approximate ones by a more straightforward encoding of the fluid equations.

We evaluate our method on randomly generated QN models with queue-length observations produced by stochastic simulation. An extensive validation of more than 8000 instances gives prediction errors for queue lengths less than 10%, with runtimes of at most 5 minutes on an ordinary laptop.

Paper organization. Section 2 discusses related works. We provide some background about QNs and the corresponding fluid approximation in Section 3. The linear QNs estimator is presented in Section 4, which discusses how to encode a time-discretized version of the fluid approximation into a linear program whose decision variables represent the model parameters to identify. Section 5 presents the numerical evaluation of the proposed approach on the selected benchmarks. Section 6 concludes.

2 RELATED WORK

This work extends the one presented in the previous paper [17]. Here, we do not settle only for finding service rates but we estimate also the entire network topology, i.e., the connections among stations with the corresponding routing probabilities. Many other service demand estimators rely on Utilization Law [29], using different statistical inference approaches such as linear regression [26], non-linear optimization [24], clustering regression [10], independent component analysis [28], pattern matching [11] and Gibbs sampling [31, 33] based on measured steady-state values of utilization and/or throughput. They require knowledge of measures that could be difficult to precisely obtain, as utilization values of stations. For example in some cloud environments as platform-as-a-service, the developer does not have complete control over the underlying physical or virtual layer and cannot directly measure utilization. We are outperforming all the mentioned approaches by focusing on three key aspects at once: (i) being not-intrusive in the instrumentation, measuring only the queue lengths; (ii) being fast in producing estimations and thus able to be applied with online adaptations if needed; (iii) widening the set of the identified parameters with a topology estimator, able to recover the entire network structure through routing probabilities.

Another comparison is against profilers, as *PerfPlotter* [8], which exploits symbolic execution to generate performance distributions of the program best, worst, and average case, and *Gprof* [16], which samples the program counter in a program run with a certain workload and counts the number of calls and execution times of each procedure. The advantage of profiling is that they are not making assumptions on the distribution of the service rates, while we are assuming them to be exponentially distributed. However, the models that they generate are often a set of metrics [4] or call graphs [2] and lack descriptive and predicting power. The QN instead, once all parameters are well estimated, can predict all future time instants queue lengths at each station. Furthermore, most profilers [15, 35] require *active probing*, i.e., observing the system with different load-tests at different utilization levels. This approach is intrusive because requires extra traffic to be injected in the network, hindering its usage during system runtime.

Most closely related to this paper is the work by Garbi et al. [13]. The authors focus on the same problem of topology and service rates estimation but from another perspective: instead of using the ODEs to solve an optimization problem and to get the value searched basing on some constraints, they train with real traces a recurrent neural network that can learn the values of rates and routing probabilities. In Section 5, we extensively compare with this approach by showing that, thanks to the linear formulation, our proposal sharply outperforms theirs in terms of efficiency and accuracy.

3 BACKGROUND

To make the paper self-contained, in this section we briefly recall QNs definitions. We use the ODEs of the fluid approximation as an estimator of queue-lengths in the optimization problem.

3.1 Queuing Networks

We deal with single-class closed QNs, which have a fixed population of jobs circulating in the system. The extension of this approach to open networks, where there are arrivals and departures of jobs, is straightforward.

Key quantities to describe a single-class closed QN are:

- N is the number of clients in the network;
- M is the number of queuing stations;
- $\mathbf{s} = (s_1, \ldots, s_M)$ is the vector of server multiplicities, where s_i gives the number of independent servers at station i, with $1 \leq i \leq M$;
- $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_M)$ is the vector of exponentially distributed service rates, i.e., $1/\mu_i > 0$ is the mean service demand at station i, with $1 \leq i \leq M$;
- $\mathbf{P} = (\mathbf{P}_{i,j})_{1 \leq i,j \leq M}$ is the routing probability matrix, whose elements $\mathbf{P}_{i,j} \geq 0$ give the probability that a client upon completion of service at station i goes to station j. The routing probability matrix is a *stochastic* matrix, meaning that the sum across each row sums up to one;
- $\mathbf{x}(0) = (\mathbf{x}_1(0), \ldots, \mathbf{x}_M(0))$: the *initial condition*, i.e., $\mathbf{x}_i(0)$ is the number of clients assigned to station i at time 0.

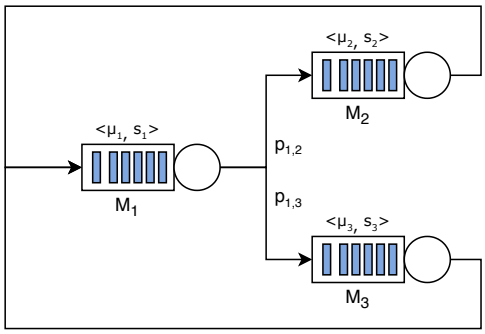
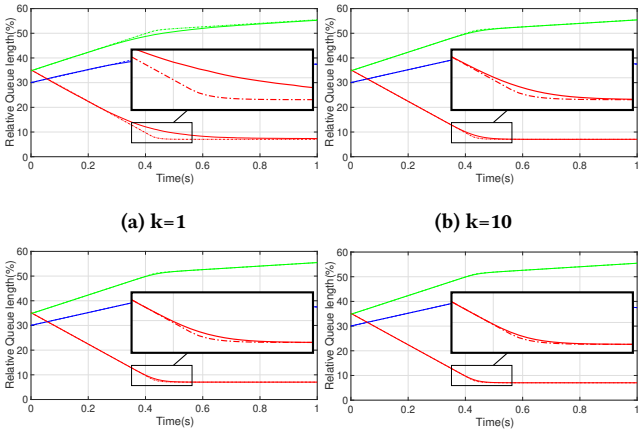


Figure 1: Load balancing example



(a) k=1 **(b) k=10**

(c) k=20 **(d) k=50**

Figure 2: Comparison between the simulated queue lengths evolution (i.e., solid lines) of the CTMC of the QN of Figure 1, averaged over 2000 statistically independent runs, with those obtained by the solution of (2) (i.e., dashed lines) with parameters as in (3)–(6). In each subplot the x-axis denotes the continuous time instants while the y-axis reports the fraction of the total population present at each station.

An example is the load balancer in Fig. 1, where there are $M = 3$ stations: the workload generator M_1 (i.e. user terminals) and the two replicas M_2 and M_3. Replicas in a load balancer system provide the same kind of service and they are distributed so that requests are sent to one replica or the other with the aim of maintaining a balance among queue lengths. Load balancing is a well-known technique in performance engineering to build scalable distributed systems [32]. Here, jobs are distributed among the two stations M_2 and M_3 according to the routing probabilities $p_{1,2}$ and $p_{1,3}$; after service completion they return back to station M_1.

3.2 Fluid Approximation

The fluid approximation allows us to consider only a set of M ODEs for a system that can be modeled as a continuous-time Markov chain (CTMC) [6]. It provides an approximation to the average queue lengths instead of specifically tracking every discrete state according to the following equation:

$$\dot{x}_i(t) = -\mu_i \min\{x_i(t), s_i\} + \sum_{j=1}^{M} p_{j,i}\mu_j \min\{x_j(t), s_j\}, \quad (1)$$

with $1 \le i \le M$, where we use the dot notation to denote time derivative. The *nonlinear* factor $\min\{x_i(t), s_i\}$ of the instant throughput at station i, given by $\mu_i \min\{x_i(t), s_i\}$, is due to the fact that if the station is idle and the queue length $x_i(t)$ is less than the number of servers s_i then $x_i(t) = \min\{x_i(t), s_i\}$ jobs are served in parallel with instant throughput $\mu_i x_i(t)$. Instead, when the station i is busy and $s_i \le x_i(t)$ then s_i jobs are served in parallel and the instant throughput is $\mu_i s_i$.

For example, the fluid approximation for the load balancer depicted in Figure 1 is the following system of ODEs:

$$\begin{aligned}\dot{x}_1(t) &= -\mu_1 \min\{x_1(t), s_1\} + \mu_2 \min\{x_2(t), s_2\} \\ &\quad + \mu_3 \min\{x_3(t), s_3\} \\ \dot{x}_2(t) &= -\mu_2 \min\{x_2(t), s_2\} + p_{1,2}\mu_1 \min\{x_1(t), s_1\} \\ \dot{x}_3(t) &= -\mu_3 \min\{x_3(t), s_3\} + p_{1,3}\mu_1 \min\{x_1(t), s_1\}\end{aligned} \quad (2)$$

The estimations of the average queue lengths tend to the exact values of the expectations of the random variables describing queue lengths, as the populations of jobs and servers grow to infinity, according to Kurtz's theorem [23]. In practice, for finite systems, Eq. 1 will provide only an approximation to the ground-truth stochastic behavior of the model. To graphically visualize the overall accuracy this deterministic QNs representation, Figure 2 depicts the comparison between the simulated queue lengths evolution of the CTMC of the QN of Figure 1, averaged over 2000 statistically independent runs (achieving an error of at most 5% with respect to the true average value with 95% of confidence), with those obtained by the numeric integration of (2) using the following parameters (we set the following values since they represent a randomly generated case in which the approximation errors become more influential):

$$\boldsymbol{\mu} = (21.91, 57.20, 10.49) \quad (3)$$

$$\mathbf{P} = \begin{pmatrix} 0.38, & 0.50, & 0.12 \\ 0.33, & 0.36, & 0.31 \\ 0.15, & 0.73, & 0.12 \end{pmatrix} \quad (4)$$

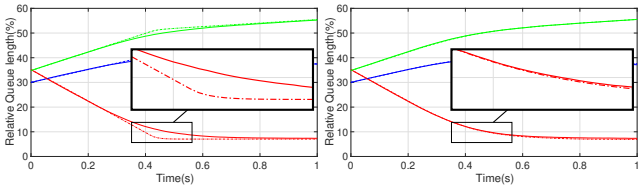
$$\mathbf{s} = k\,(33, 24, 44) \quad (5)$$

$$\mathbf{x}\,(0) = k\,(82, 96, 95) \quad (6)$$

The parameter $k \in \{1, 10, 20, 50\}$ is a scaling factor that determines the total population of jobs and servers in the network. Figure 2 confirms that, in accordance with Kurtz's theorem, the approximation error of the ODE does tend to vanish with increasing values of total population and concurrency levels while it becomes more evident with lower values of the scaling factor k.

The source of error is attributable to the nonlinearities in (2) due to the presence of the minimum function. Indeed, it is well known that the true average queue lengths, denoted by $\mathbb{E}[X_i]$, for $1 \le i \le M$ satisfy the equations:

$$\mathbb{E}[\dot{X}_i(t)] = -\mu_i \mathbb{E}[\min\{X_i(t), s_i\}] + \sum_{j=1}^{M} p_{j,i}\mu_j \mathbb{E}[\min\{X_j(t), s_j\}] \quad (7)$$



(a) not self-consistent ODE **(b) self-consistent ODE**

Figure 3: Comparison between the simulated queue lengths evolution (i.e., solid lines) of the CTMC of the QN of Figure 1, averaged over 2000 statistically independent runs, with those obtained by the solution of (2) (i.e., dashed lines) with parameters as in (3)–(6) and scaling factor $k = 1$ with or without the self-consistent expectation of the minimum function, i.e., b) and a) respectively.

However, these equations are not self-consistent because the expectation of the minimum function in the right-hand side cannot be exactly written in terms of the variables in the left-hand side in general. The approximation leading to (2) essentially consists in the replacement of the expected value of the minimum of two random variables, i.e., $\mathbb{E}[\min\{X_i(t), s_i\}]$, with minimum of the two expectations, i.e, $\min\{\mathbb{E}[X_i(t)], s_i\}$, which is now self-consistent. Although we proved that is error is negligible for self-adaptation purposes [19], it becomes more disturbing in the context of parameter estimation. A strategy for mitigating such an issue is proposed in Section 4 but its beneficial effects, on the case of Figure 2 reporting the maximum errors, are illustrated in Figure 3.

4 LINEAR PROGRAMMING ESTIMATION OF QUEUING NETWORKS

4.1 Discrete-time model

Our learning method is built on a time-discretized version of (1) through the well-known Euler numerical integration schema which approximates time derivatives as $\dot{x}_i(t) \approx (x_i(t+\Delta t) - x_i(t))/\Delta t$, for a given fixed time step Δ. We denote by $\overline{x}_i(k)$ the approximation at the k-th step, i.e., $\overline{x}_i(k) \approx x_i(k\Delta t)$, for $k \geq 0$. It follows that the estimations of the average queue length trajectories can be computed by solving the following system of difference equations:

$$\overline{x}_i(k+1) = \overline{x}_i(k) - \Delta t \mu_i \min\{\overline{x}_i(k), s_i\} + \\ + \Delta t \sum_{j=1}^{M} p_{j,i}\mu_j \min\{\overline{x}_j(k), s_j\} \quad (8)$$

with $k \geq 0$ and $1 \leq i \leq M$ with initial conditions $\overline{x}_i(0) = x_i(0)$.

4.2 Nonlinear QNs estimator

In the most direct estimation approach, (8) are used as constraints in an optimization problem which looks for a solution minimizing the error between the predicted queue lengths, i.e., $\overline{x}_i(k)$, and the measured ones over a given observation window H. Denoting by $\tilde{x}_i(k)$ the measured queue length of station i at time step k, the non-linear estimation problem can be written as follows:

$$\underset{x,\mu,P}{\text{minimize}} \sum_{i=1}^{M}\sum_{k=0}^{H-1} |(\overline{x}_i(k) - \tilde{x}_i(k))| \\ \text{subject to:} \quad (9)$$
$$\text{Eq. (8)}, \quad \text{for } 0 \leq k \leq H-1,\ 1 \leq i \leq M,$$
$$\overline{x}_i(0) = \tilde{x}_i(0), \quad 1 \leq i \leq M.$$

In other words, we search for the optimal vector of service rates μ and routing probabilities matrix P that minimizes the difference between the measurements and the model predictions over all stations and all discrete-time instants of the observation window H, when the model dynamics is initialized with the measured queue lengths $\tilde{x}(0)$.

4.3 Linear programming formulation

Although viable, in principle, the non-linear QNs estimation suffers from well-known scalability issues hampering its usage in practice. In [19], we shown that the runtime of the solution of a globally optimal non-linear program similar to (9) can be orders of magnitude bigger with respect to the solution of the corresponding mixed-integer formulation. In particular, the main difficulties of (9) reside in the nonlinearities of (8), i.e, the multiplicative relation between $\boldsymbol{\mu}$ and $\mathbf{P}$ and the presence of the minimum function. The major technical contribution of this work is to provide an exact linear programming formulation of (9) under the assumption that queue lengths and concurrency levels observations are available for each station of the QN at each considered discrete time instant. Overall, when these conditions are met, the optimization problem (9) admits the following linear formulation.

$$\underset{\mu_i,\, g_{i,j},\, E_{i,k},\, \hat{E}_{i,k}}{\text{minimize}} \sum_{i=1}^{M}\sum_{k=0}^{H-1} \hat{E}_{i,k} \quad (10)$$
$$\text{subject to:}$$
$$E_{i,k} = -\mu_i \gamma_i(k) + \sum_{j=1}^{M} g_{j,i}\gamma_j(k) - \Delta\tilde{x}(k), \quad (11)$$
$$\mu_i = \sum_{j=1}^{M} g_{i,j}, \quad g_{i,j} \geq 0, \quad (12)$$
$$\hat{E}_{i,k} \geq E_{i,k}, \quad \hat{E}_{i,k} \geq -E_{i,k}, \quad (13)$$
$$1 \leq i, j \leq M, 0 \leq k \leq H-1,$$

with $\Delta\tilde{x}(k) = \frac{\tilde{x}_i(k+1) - \tilde{x}_i(k)}{\Delta t}$ and $\gamma_i(k) = \mathbb{E}[\min\{s_i, \tilde{x}_i(k)\}]$. The former is just a syntactic abbreviation for a more compact writing of (11). Instead, the introduction of $\gamma_i(k)$ brings a substantial benefit to the estimation process since, using only the data already available, it enables to account for the error introduced by the fluid approximation, thus increasing the accuracy of the learned parameters (see Section 3.2 for a more detailed discussion about this error).

The key intuition enabling this linear programming formulation is twofold. *i)* Instead of integrating (8) over the whole observation window—as it would be in (9)—each integration step is conducted individually whilst the whole trajectory of the QN is reconstructed by minimizing the absolute value of the single step prediction errors $\hat{E}_{i,k}$, see (10), (11), and (13). *ii)* A change of variables that allows to

replace each nonlinear term, i.e., $p_{i,j}\mu_i$, with a fresh new variable $g_{i,j}$. Then, after the optimization takes place, the estimated routing probabilities can be computed as $p^*_{i,j} = g^*_{i,j}/\mu^*_i$, with $1 \leq i, j \leq M$ where $g^*_{i,j}, \mu^*_i$ are the optimal $g_{i,j}$ and $\mu_{i,}$, respectively.

Concretely, the relation $g_{i,j} = \mu_i p_{i,j}$ is enforced by (12). This is because, if (12) holds, then $0 \leq g_{i,j} \leq \mu_i$; hence $g_{i,j} = p_{i,j}\mu_i$ with $0 \leq p_{i,j} \leq 1$ necessarily. Then by considering that $\sum_{j=1}^{M} p_{i,j}\mu_i = \mu_i$ implies that $\sum_{j=1}^{M} p_{i,j} = 1$, the matrix $\mathbf{P} = (p_{i,j})_{1 \leq i,j \leq M}$ is a valid routing probability matrix. Finally, (13) completes the optimization problem by enforcing $\hat{E}_{i,k} \geq |E_{i,k}|$, which is the standard linear programming formulation for the minimization of the absolute value of a decision variable [7].

5 NUMERICAL EVALUATION

In this section we evaluate the efficiency and effectiveness of the proposed QNs estimation method by considering randomly generated models of increasing complexity. In particular, we generated our dataset by simulating each of the considered QN through the Gillespie's exact simulation algorithm [14], available in the stochastic simulation framework Stochkit [27]. We then used the collected queue length trajectories as inputs of the optimization problem (10)-(13) solved using the well-known CPLEX optimization engine [9]. All the data generation process were performed on an Intel Xeon machine with 48 cores and 60 GB RAM while the learning process has been conducted on a laptop equipped with a 2.8 GHz Intel i7 quad-core processor and 16GB RAM.

5.1 Experimental set-up

For our tests we considered randomly generated closed QNs of different sizes $M \in \{5, 10, 15, 20\}$. For each case, we generated 10 QNs by picking uniformly at random the entries of the routing probability matrices, the service rates in the interval $[1.0, 100.0]$, and the concurrency levels in the interval $[1, 64]$ except for the reference station for which we ensured to assign a number of servers that were greater than or equal to the total number of clients circulating in the network (i.e., in all of the considered models the reference station is an infinite server).

For the generation of the training dataset we evaluated each random QN starting from 100 distinct initial population vectors. We chose the initial number of jobs at each station between 0 and 100 with its total population uniformly distributed between 50 and $100M$. We fixed 50 as the minimum allowed workload to avoid situations in which portions of the network were too little stressed to be learned, i.e., to avoid a very small chance to visit some particular stations making them irrelevant for the optimization; we further comment on this issue later in this section. For each initial condition we collected mean queue length trajectories averaged over 2000 independent repetitions.

Other two important hyper-parameters of the estimation process are the length of each trace, i.e., the time horizon T of the stochastic simulations, and the choice of the discretization interval Δt; the former is related to the amount of knowledge we accumulate from the system, i.e., if longer time horizons lead to larger simulation and optimization runtimes too short traces might not be representative of the full dynamics of the system, the latter should be chosen small enough such that no important events are lost across two consecutive time steps [1]. For our case studies, we set $T = 2s$ and $\Delta t = 0.01$, thus collecting $H = 200$ points per trace.

5.2 Predictive power evaluation

We assess the predictive power of the learned QNs by conducting two distinct "what-if" analyses under unseen configurations (i.e., not included in the traces used for learning) by changing the total number of clients and the concurrency levels of each station, respectively. Similarly to [13], to formally quantify the accuracy of the learned model with respect to the ground-truth QN we rely on the following error function

$$err_i = \frac{\max_{h=1}^{H} |\tilde{x}_i(h) - \overline{x}_i(h)|}{2N} \cdot 100. \tag{14}$$

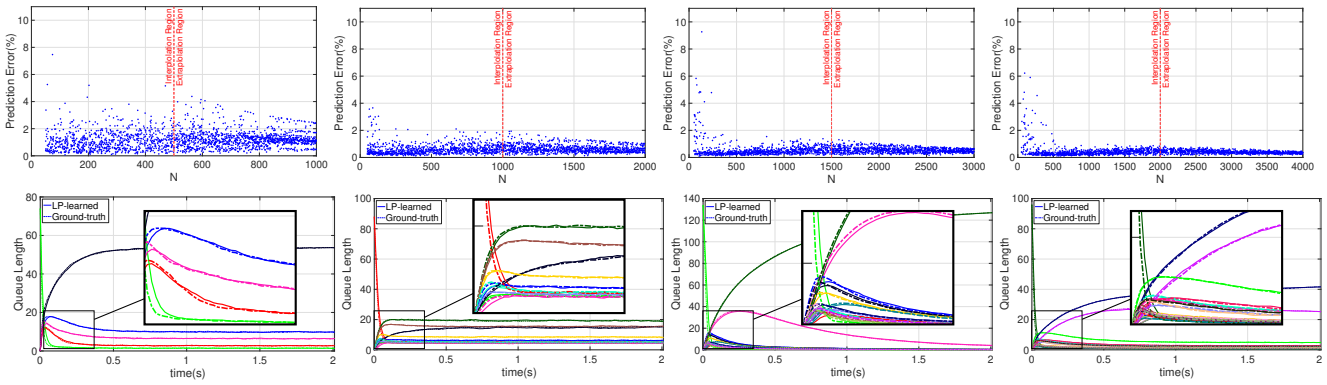
It basically represents the maximum percentage error of station i relative to the total population of clients circulating in the network (since we are studying closed QNs N con be considered constant across the H steps of a trace). We refer the reader to [13] for a more detailed discussion on (14).

What-if analysis over client population. We tested each learned QN with 200 new initial population vectors that were not used for learning. For doing so, the initial number of jobs at each station was chosen between 0 and 200 with its total population uniformly distributed between 50 and $200M$, discarding initial conditions that were included in the learning configurations. We then compared the averages over 2000 independent stochastic simulation of the ground-truth queue-length dynamics with those produced by the LP-learned QN starting from each unseen initial condition.

In the first row of Figure 4 are reported the scatter plots of the prediction errors for each considered QN, for each network size M. The results highlight the high prediction power of the learned models by reporting prediction errors less than 10% in all cases. In particular, there is no significant difference among the interpolation region (i.e., between initial conditions used for learning) and the extrapolation one (i.e., outside the range of initial conditions used for learning) remarking the high accuracy of the learned QNs.

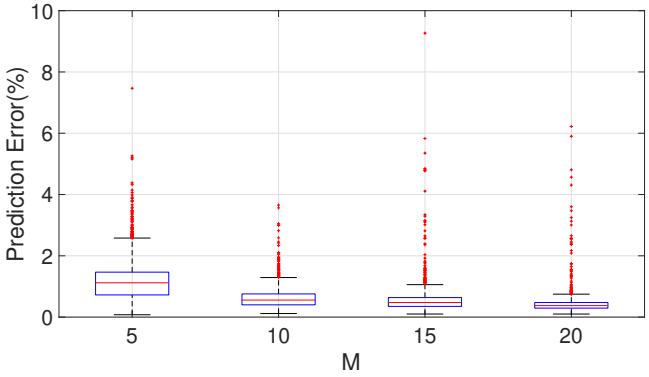
To visualize the impact of the reported errors on the whole queue length evolution, the lowest row of Figure 4 reports the comparison between the ground-truth queue lengths (i.e., dashed lines) and those predicted by the LP-learned QN (i.e., solid lines) on the initial population vector that provoked the maximum prediction errors among the all the what-if over population for each network size (i.e., 7.46%, 3.65%, 9.26%, 6.22% respectively). Even in this case, despite the presence of small deviations, the ground-truth time course evolutions are very well predicted by the corresponding LP-learned QN for both the transient and the steady-state regimes.

Finally, the box-plots in Figure 5 report the summary statistics of the scatter plots depicted in Figure 4. These charts do not seem to show a statistically significant increase in the error as a function of the size of the network; the averages decrease. We intuitively attribute this behavior to the fact that, by enlarging the size of the network, the optimization problem becomes implicitly more constrained, thus increasing the precision of the learned models. We leave as future work a targeted experimentation to give more evidence of this.



(a) M=5 (b) M=10 (c) M=15 (d) M=20

Figure 4: Upper row) Prediction error of the what-if instances where each randomly generated QN is evaluated with 200 unseen initial population vectors, indexed with respect to the network size M. In each chart, the x-axis N represents total number of clients circulating in the network plotted against the prediction error defined in Eq. (14) while the dashed vertical line denotes the boundary of the interpolation (i.e., between initial conditions used for learning) and the and extrapolation region (i.e., outside the range of initial conditions used for learning). Bottom row) Ground-truth queue lengths (dashed lines) and those predicted by the LP-learned QN (solid lines) on the initial population vector that provoked the maximum prediction errors among the what-if over population for each network size M (i.e., 7.46%, 3.65%, 9.26%, 6.22% respectively).



(a)

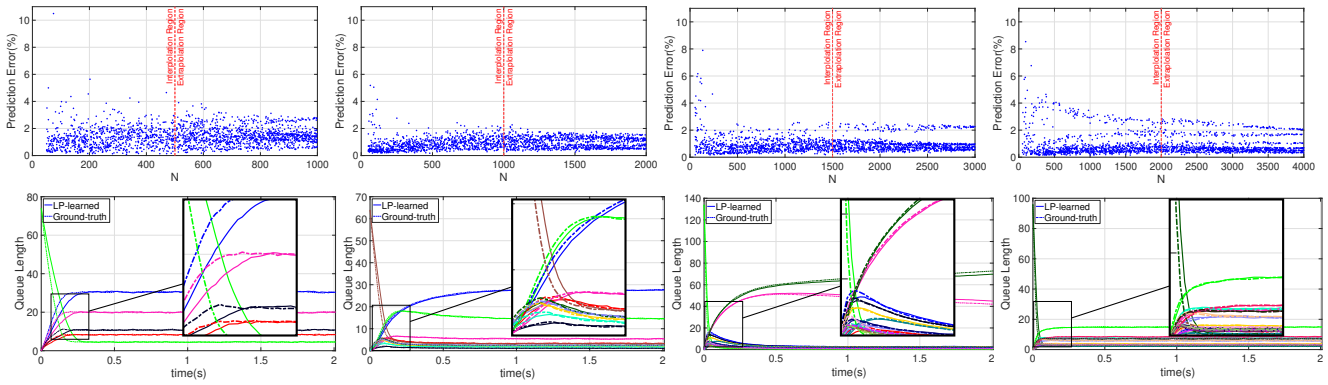
Figure 5: a) Summary statistics on the prediction error for the experiments of Fig. 4. In each box-plot, the line inside the box represents the median error, the upper and lower side of the box represent the first and the third quartiles of the observed error distribution, while the upper and lower limit of the dashed line represent the extreme points not to be considered outliers. The red dots depict the outliers.

What-if analysis over concurrency levels. Here we test the predictive power with respect to modifications to the concurrency levels. For each generated QN we assigned a new concurrency level picked uniformly at random in $[1, 64]$ to each station except the first one, which was left as a think station. Then we compared the dynamics of the ground-truth model (i.e., simulated with the original routing probabilities and rates but with the new concurrency levels) against those obtained by simulating the learned model with the new concurrency levels. As initial conditions, we considered the same points used for the what-if over population of Figure 4. Also for this evaluation we considered the notion of prediction error as shown in Equation (14).

In the first row of Figure 6 are reported the scatter plots of the prediction errors for each considered QN, indexed by the network size M. Also in this case the results highlight the high prediction power of the learned models by reporting predictions error less than 10% in all cases. The impact of the reported errors on the whole queue length evolution is depicted in the lowest row of Figure 6. It reports the comparison between the ground-truth queue lengths (i.e., dashed lines) and those predicted by the LP-learned QNs (i.e., solid lines) on the initial population vector that caused the maximum prediction errors among all the what-if over concurrency level for each network size (i.e., 10.49%, 5.18%, 7.89%, 8.53% respectively). Despite the presence of slightly bigger deviations with respect to Figure 4, the ground-truth time course evolutions are very well predicted by the corresponding LP-learned QN for both transient and steady-state regimes. We justify the small increase of prediction errors by the fact that unlike the previous what-if study over client populations, in this case we changed parameters, i.e., the number of servers, that remained constant in all the traces used for learning.

The box-plots in Figure 7 report the summary statistics of the scatter plots depicted in Figure 6. Analogously to the what-if over population experiments, it does not show a statistically significant increment of the prediction errors with network of increasing size.

Figure 8 depicts the summary statistics of the solution time of the 40 optimization problems we used for our experimentation, grouped by the size of the corresponding QN. Thanks to the linear formulation, the average solution time follows an almost linear growth law as a function of the size of the learned model.



(a) M=5 (b) M=10 (c) M=15 (d) M=20

Figure 6: Upper Row) Prediction error of the what-if instances over concurrently levels. For each of the LP-leaned QN we generated a fresh new concurrency level vector $S = (s_i)_{1 \leq i \leq M}$ where each $s_i, i \geq 1$ is picked uniformly at random in $[1, 64]$ while s_1 is left unmodified. We then evaluated each of these QNs over the same initial population vectors of Fig. 4. Lowest row) Comparison between the ground-truth queue lengths (i.e., dashed lines) and those predicted by the LP-learned QN (i.e., solid lines) on the initial population vector that provoked the maximum prediction errors among the what-if over concurrency level for each network size M (i.e., 10.49%, 5.18%, 7.89%, 8.53% respectively)

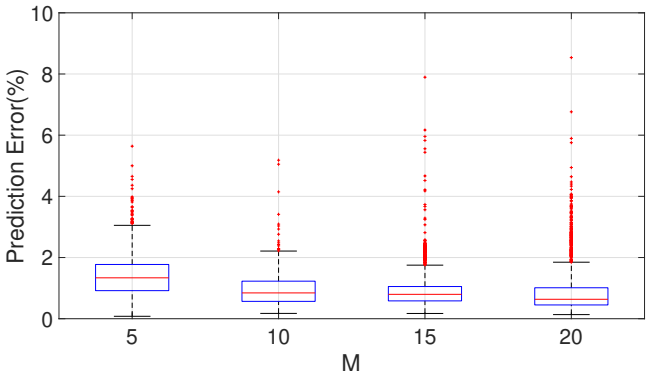


Figure 7: a) Summary statistics on the prediction error for the experiments of Fig. 6. In each box-plot, the line inside the box represents the median error, the upper and lower side of the box represent the first and the third quartiles of the observed error distribution, while the upper and lower limit of the dashed line represents the extreme points not to be considered outliers. Finally, the red dots depict the outliers.

5.3 Comparison with state-of-the-art

For concluding the numerical evaluation, hereafter we compare the predictive power of the proposed approach against that reported by the recursive neural network-based (RNN) estimator recently proposed in [13]. We chose the latter as the object of this comparison since it represents the unique QNs estimator able to simultaneously learn service rates and routing probabilities of an unknown QN relying on concurrency levels and queue lengths observations only (see Section 2 for a more detailed discussion on the related approaches).

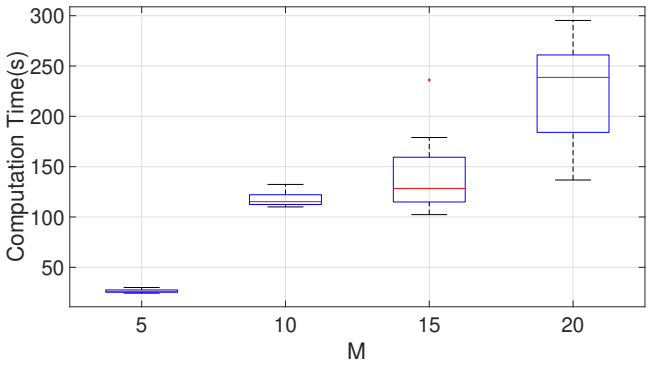
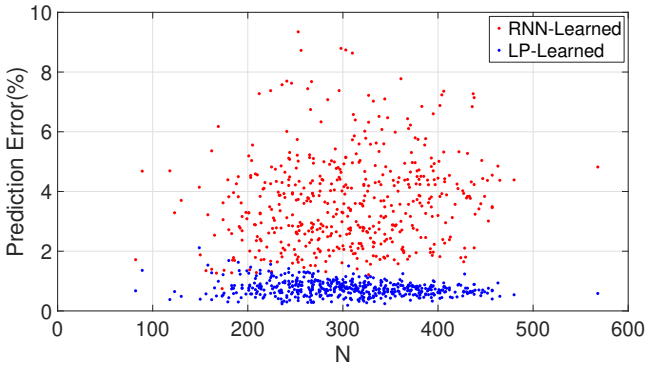


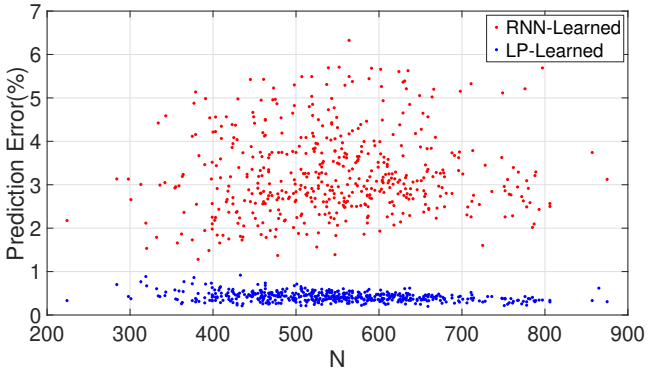
Figure 8: a) Summary statistics on the computation time needed for learning all the 40 QN models (i.e., 10 for each model size) used for the experimentation in Figures 4 and 6.

For the sake of fairness, we repeated the same exact what-if experiments that were conducted in [13] by learning the QNs under study through the traces contained in the corresponding replication package (i.e., 100 randomly generated initial conditions, $T = 10\,s, \Delta t = 0.01$). The only variation consisted in simulating each QN model again for collecting the individual traces necessary to compute each $\mathbb{E}[\min\{s_i, \tilde{x}_i(k)\}]$ that cannot be derived by the already averaged data.

Figure 9 and Figure 10 respectively report the comparison between the predictions errors of the RNN-leaned models and the LP-learned ones, in the what-if over population and concurrency levels scenarios indexed by the corresponding model sizes $M \in \{5, 10\}$ (see [13] for a detailed discussion of the conducted experiment). These numerical results highlight that the LP-based approach (blue points) outperforms the RNN-based one (red points) for all the evaluated scenarios by reporting consistently smaller predictions errors.

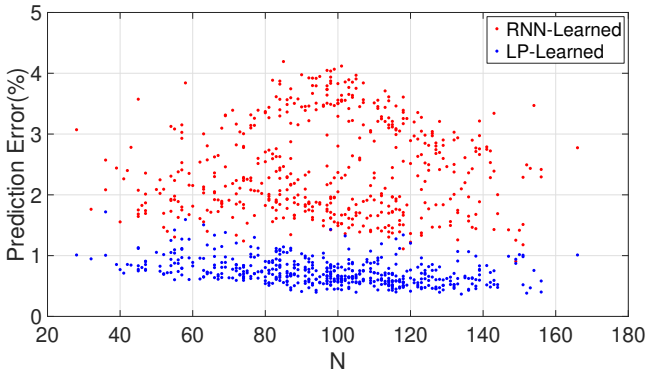


(a) M=5

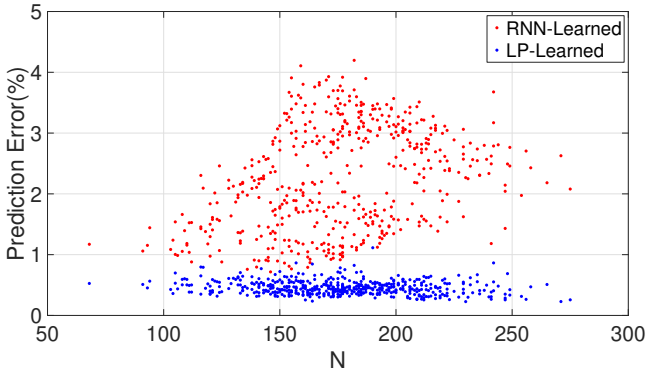


(b) M=10

Figure 9: Comparison between the predictions errors of the RNN-learned models [13] and the LP-learned ones, in the what-if over populations as described in [13] and indexed by the corresponding model sizes $M \in \{5, 10\}$.



(a) M=5



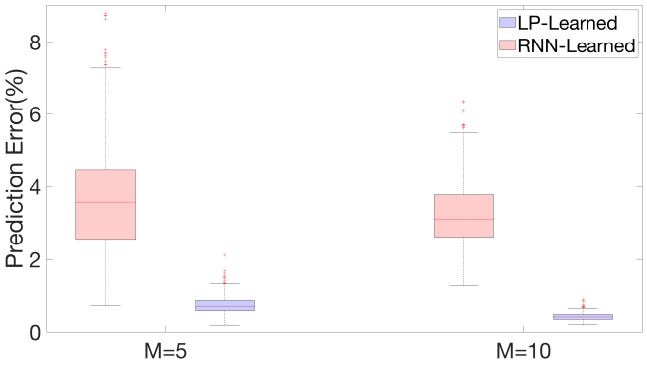
(b) M=10

Figure 10: Comparison between the predictions errors of the RNN-leaned models [13] and the LP-learned ones, in the what-if over concurrency levels as described in [13] and indexed by the corresponding model sizes $M \in \{5, 10\}$.

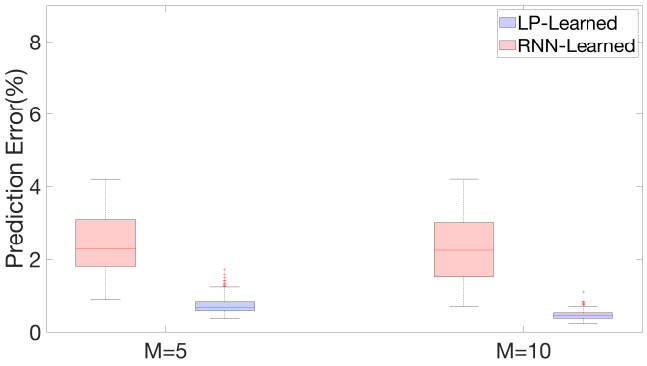
As expected, this fact is also confirmed by the summary statistics of Figure 11. These results strengthen the effectiveness and generality of the proposed approach as it proved able to produce more accurate QNs models (i.e., with a higher prediction power) despite having been learned in a less constrained setting, since in [13] the main diagonal of the routing probabilities matrix is assumed to be known while in our tests it was assumed to be completely unknown.

As a final remark, we empathize that making a reliable comparison on the scalability of the two approaches is very delicate to conduct. This is because of two fundamental aspects: *i)* the conceptually different optimization methods (i.e., local versus global optimization), *ii)* the computational aspects of both approaches are strongly influenced by the choices made in the data collection phase, i.e, number of initial populations, the temporal length of the learning traces T and the discretization step Δt. The former is an intrinsic characteristic that discriminates between non-linear methods and linear programming ones which in general suggests that local methods are more efficient than the global ones [25]. The latter is dependent on many design decisions whose choice can cause the prevalence of one approach over the other in terms of scalability. We leave a definitive answer as to which approach is more efficient for future work, even if we are aware of the fact that, for the reported experimentations, the linear approach seems to outperform even in terms of scalability the RNN based estimator thanks to the fact that a global optimizer can work on shorter traces to learn models with a high degree of accuracy ($T = 2\,s$ in place of $T = 10\,s$, as depicted in Figure 4).

Summarising, the results shown in this section substantially support the fact that QN models of arbitrary topologies and challenging dimensions can be effectively learned by using the succinct linear program (10). The accuracy of the learned models is testified by prediction errors consistently smaller than 10% across all the conducted experiments. We remark the ability of the learned QNs, to provide reliable predictions even in those what-if scenarios in which the varying parameters have been given as constants in the input traces. Strengthening our proposal, the comparison with the state of the art further emphasizes the accuracy of the learned models by reporting a higher degree of accuracy. In addition, the favorable comparison with respect to the state of the art allows us to reasonably expect to have the same accuracy results on real cases as the ones studied in [13], even if the latter have not been explicitly addressed in this work.



(a) RNN-Learned vs LP-Learned accuracy in the same exact what-if over population (See Fig. 9)



(b) RNN-Learned vs LP-Learned accuracy in the same exact what-if over concurrency levels (See Fig. 10)

Figure 11: Summary statistics on the prediction error for the experiments of Fig. 9 and Fig. 10 distinguished by the different model sizes $M \in \{5, 10\}$ and the used learning methods , i.e., RNN and LP, respectively. In each box-plot, the line inside the box represents the median error, the upper and lower side of the box represent the first and the third quartiles of the observed error distribution, while the upper and lower limit of the dashed line represents the extreme points not to be considered outliers. Finally, the red dots depict the outliers.

6 CONCLUSIONS

We presented an efficient and effective methodology for learning queuing networks (QN) models of software systems. The main novelty lies in the formulation of the QN estimation problem as a succinct linear program whose decision variables can be directly related to standard queuing network parameters (i.e., service rates and routing probabilities). We reported promising results on random models of challenging dimensions with a maximum prediction error (i.e., the distance between the dynamics predicted by the learned models and those computed through the ground truth) less than 10% even when the system is evaluated under configurations that are not included in the training set. We plan to extend our work for capturing the performance dynamics of more complex systems, such as multi-class and layered QNs. Moreover, to further enlarge the target systems, we plan to include load-dependent and generally distributed service rates.

ACKNOWLEDGMENTS

This work has been partially supported by the PRIN project "SEDUCE" no. 2017TWRCNB.

REFERENCES

[1] U. Ascher and L. Petzold. 1998. *Computer methods for ordinary differential equations and differential-algebraic equations.* Vol. 61. Siam.
[2] T. Ball and J. Larus. 1996. Efficient path profiling. In *Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture. (MICRO).*
[3] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. 2004. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering* 30, 5 (2004), 295–310.
[4] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan. 2003. Magpie: online modelling and performance-aware systems. In *Proceedings of the 9th conference on Hot Topics in Operating Systems (HotOS).*
[5] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi. 2005. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications.* Wiley.
[6] L. Bortolussi, J. Hillston, Diego L., and M. Massink. 2013. Continuous approximation of collective system behaviour: A tutorial. *Performance Evaluation* 70, 5 (2013).
[7] S. Boyd, S. P. Boyd, and L. Vandenberghe. 2004. *Convex optimization.* Cambridge university press.
[8] B. Chen, Y. Liu, and W. Le. 2016. Generating performance distributions via probabilistic symbolic execution. In *Proceedings of the 38th International Conference on Software Engineering (ICSE).*
[9] IBM ILOG Cplex. 2009. V12. 1: User's Manual for CPLEX. *International Business Machines Corporation* 46, 53 (2009), 157.
[10] P. Cremonesi, K. Dhyani, and A. Sansottera. 2010. Service time estimation with a refinement enhanced hybrid clustering algorithm. In *International Conference on Analytical and Stochastic Modeling Techniques and Applications (ASMTA).*
[11] P. Cremonesi and A. Sansottera. 2014. Indirect estimation of service demands in the presence of structural changes. *Performance Evaluation* 73 (2014).
[12] R. W. R. Darling and J. R. Norris. 2008. Differential equation approximations for Markov chains. *Probability Surveys* 5 (2008).
[13] G. Garbi, E. Incerto, and M. Tribastone. 2020. Learning Queuing Networks by Recurrent Neural Networks. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE).*
[14] D. T. Gillespie. 1977. Exact Stochastic Simulation of Coupled Chemical Reactions. *Journal of Physical Chemistry* 81, 25 (1977).
[15] S. F. Goldsmith, A. S. Aiken, and D. S. Wilkerson. 2007. Measuring empirical computational complexity. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (ESEC/FSE).*
[16] S. L. Graham, P. B. Kessler, and M. K. Mckusick. 1982. Gprof: A Call Graph Execution Profiler. In *Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction (SIGPLAN).*
[17] E. Incerto, A. Napolitano, and M. Tribastone. 2018. Moving horizon estimation of service demands in queuing networks. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS).*
[18] E. Incerto, M. Tribastone, and C. Trubiani. 2016. Symbolic Performance Adaptation. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-managing Systems (SEAMS).*
[19] E. Incerto, M. Tribastone, and C. Trubiani. 2017. Software performance self-adaptation through efficient model predictive control. In *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE).*
[20] M. Kowal, I. Schaefer, and M. Tribastone. 2014. Family-Based Performance Analysis of Variant-Rich Software Systems. In *Fundamental Approaches to Software Engineering (FASE).*
[21] M. Kowal, M. Tschaikowski, M. Tribastone, and I. Schaefer. 2015. Scaling size and parameter spaces in variability-aware software performance models (t). In *30th IEEE/ACM International Conference on Automated Software Engineering (ASE).*
[22] H. Koziolek. 2010. Performance evaluation of component-based software systems: A survey. *Performance Evalutation* 67, 8 (2010).
[23] T. G. Kurtz. 1970. Solutions of ordinary differential equations as limits of pure Markov processes. In *J. Appl. Prob.*, Vol. 7.
[24] Daniel A. M. 2008. Computing Missing Service Demand Parameters for Performance Models. In *International Computer Measurement Group Conference (CMG).*
[25] J. Nocedal and S. Wright. 2006. *Numerical optimization.* Springer Science & Business Media.

[26] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi. 2008. CPU demand for web serving: Measurement analysis and dynamic estimation. *Performance Evaluation* 65, 6-7 (2008).

[27] K. R. Sanft, S. Wu, M. Roh, J. Fu, R. K. Lim, and L. R. Petzold. 2011. StochKit2: software for discrete stochastic simulation of biochemical systems with events. *Bioinformatics* 27, 17 (2011).

[28] A. Sharma, R. Bhagwan, M. Choudhury, L. Golubchik, R. Govindan, and G. M. Voelker. 2008. Automatic request categorization in internet services. *ACM SIGMETRICS Performance Evaluation Review* 36, 2 (2008).

[29] S. Spinner, G. Casale, F. Brosig, and S. Kounev. 2015. Evaluating approaches to resource demand estimation. *Performance Evaluation* 92 (2015).

[30] W. J. Stewart. 2007. Performance modelling and markov chains. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM)*.

[31] C. Sutton and M. I. Jordan. 2011. Bayesian inference for queueing networks and modeling of Internet services. *The Annals of Applied Statistics* (2011).

[32] A. N. Tantawi and D. Towsley. 1985. Optimal Static Load Balancing in Distributed Computer Systems. *J. Acm* 32, 2 (1985), 21.

[33] W. Wang and G. Casale. 2013. Bayesian service demand estimation using gibbs sampling. In *IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*.

[34] W. Wang, G. Casale, A. Kattepur, and M. Nambiar. 2016. Maximum likelihood estimation of closed queueing network demands from queue length data. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (ICPE)*.

[35] W. Wang, N. Tian, S. Huang, S. He, A. Srivastava, M. L. Soffa, and L. Pollock. 2018. Testing cloud applications under cloud-uncertainty performance effects. In *IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*.

[36] M. Woodside, G. Franks, and D. C. Petriu. 2007. The future of software performance engineering. In *Proceedings of the Future of Software Engineering (FOSE)*.