Exact Analysis of Performance Models by the Method of Moments

Giuliano Casale

Imperial College London
Department of Computing
180 Queen's Gate, London SW7 2AZ
g.casale@imperial.ac.uk

Abstract

Performance evaluation of distributed systems and service-oriented architectures is often based on stochastic models, such as closed queueing networks which are commonly solved by the Mean Value Analysis (MVA) algorithm. However, the MVA algorithm is unable to solve models with hundreds or thousands of users accessing services of multiple classes, a configuration that is often useful to predict the performance of many real-world applications. This paper introduces the Method of Moments (MoM), the first exact algorithm for solving closed queueing networks with large population sizes.

Compared to the MVA algorithm, which is based on a recursive evaluation of mean queue-lengths, MoM defines a recursion on higher-order moments of queue-lengths that is solved at each step by a linear system of equations. This approach dramatically decreases the costs of an exact analysis compared to the MVA algorithm. We prove that MoM requires log-quadratic time and log-linear space in the total population size, whereas MVA complexity expressions grow combinatorially as the product of class populations. This extends the feasibility of exact methods to a much larger family of multiclass performance models than those that can be solved by the MVA algorithm.

Key words: Performance analysis, capacity planning, multiclass systems, closed queueing networks, exact evaluation

1 Introduction

Queueing networks are an established class of analytical models for capacity planning, performance evaluation, optimization, and design exploration of software and hardware systems [17, 28]. A typical performance analysis

based on queueing networks consists in studying the growth of performance indexes such as throughput, response time, or server utilization, under an increase of the number of users of the system. To overcome state-space explosion problems, product-form queueing networks are often used by researchers and practitioners as an efficient formalism to model the scalability of real-world systems [1, 31, 38]. The main advantage of product-form models is that the continuous-time Markov chain underlying the queueing network enjoys a closed-form expression of the equilibrium state probabilities which makes the analysis computationally tractable. This enables efficient computation of system performance metrics.

We here focus on closed networks, which are models of systems where a constant population of users is served by a network of servers with heterogeneous processing speeds [1]. Closed networks have been intensively investigated in performance evaluation [28] and are frequently used in capacity planning for static and dynamic provisioning of modern multi-tier and service-oriented architectures [31, 32, 39]. Despite the availability of many exact analysis techniques for closed networks [2,4,10,11,13,14,16,20,21,30,34,35], a long-standing issue of these models is their inability to evaluate exactly systems with many users belonging to several workload classes, a case that often arises in applications because transactions of different types are often modeled as distinct workload classes. For instance, recent measurements on large scale J2EE web applications suggest that exploring the behavior of a real multi-tier application from low to heavy load requires modeling many hundreds of users and at least five workload classes [24]. However, models of this dimension are prohibitive for exact analysis algorithms like the Mean Value Analysis (MVA) [35], which is the de-facto standard for solving closed queueing networks. In practice, MVA is scalable up to models with 3 to 4 classes since complexity expressions grow combinatorially as the product of class populations. This difficulty of the MVA approach has been addressed in previous work only by approximate techniques such as local iterative methods [9, 15, 37], bounds and asymptotic expansions [30, 40], numerical inversion methods [11], diffusion approximation [23], and Monte Carlo integration [25], but it is difficult to validate the robustness of these methods on large multiclass networks because of the lack of exact solution methods that are scalable on models with many users and many classes. That is, no efficient algorithm exists that can provide exact values of the queueing network performance indexes under large multiclass workloads.

In this paper, we overcome the computational limitations of MVA and established exact solution algorithms by introducing the Method of Moments (MoM), a new exact analysis technique which is able for the first time to solve efficiently models with large multiclass populations. The basic idea of MoM is to solve queueing networks by recursively computing a set of *higher-order moments* of the stations' queue-length distributions. This is in contrast with the classic MVA approach where one evaluates mean queue-lengths only, but

we find that these higher-order moments equivalently determine the equilibrium solution of the model. The advantage of higher-order moments is that they can be computed efficiently in a recursive manner using a system of linear equations involving normalizing constants. The equations of this linear system correspond to the fundamental recursions used in algorithms such as RECAL [14] and LBANC [10], which are called in this paper the *population constraint* (PC) and the *convolution expression* (CE). The main result is that the simultaneous evaluation of such equations into a linear system allows MoM to avoid the combinatorial growth of the recursion on the population sizes that affects all known exact methods, noticeably the MVA algorithm. In particular, for a queueing network with a multiclass population of N jobs, one needs to evaluate just N linear systems to solve the model.

The present work summarizes and extends [5] as follows. First, we introduce a compact formulation of MoM; the algorithm presented in [5] is instead discussed in Section 5.3. The new MoM formulation presented in this paper is much easier to implement compared to the one proposed in [5]. Next, we derive an extended analysis of computational costs and we report experimental results on several models and case study that help in understanding the benefits and limitations of MoM.

We also remark that in a recent paper we have proposed CoMoM [7], a new algorithm for closed queueing networks that specializes in the computation of normalizing constants for models with large number of classes. CoMoM stems from the MoM theory presented in this paper to derive a novel approach, where the basis is no longer defined in terms of queue replica addition, but a larger number of population vectors is considered in the basis definition. We point to [7] for an investigation of the mutual relationships between MoM and CoMoM, proving that MoM is more efficient on models with more queues than classes; in the opposite case CoMoM is more efficient. The result presented in [7] also shows that the MoM approach is not limited to the applications discussed here, but may be used as a starting point for the definition of new methods for the exact evaluation of product-form models. Additionally, in a very recent paper [8] we have shown how the MoM theory leads to hybrid approaches for the solution of closed queueing network models that further increase the efficiency of the solution.

The remainder of this paper is organized as follows. In Section 2 we give background and preliminary definitions. We give motivation in Section 3 by reviewing the computational limitations of existing exact algorithms. Section 4 defines the basic MoM algorithm, which is later improved in Section 5 for increased efficiency. Computational costs of the proposed algorithms are studied in Section 6. Section 7 gives experiments and case studies. Finally, Section 8 gives final remarks.

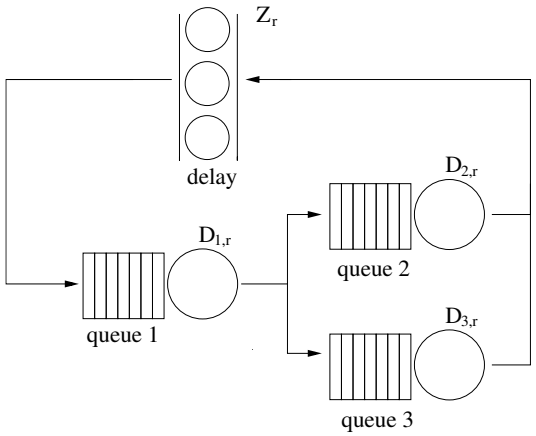


Fig. 1. *A closed product-form network with $M = 3$ queues. In the proposed notation, $D_{1,r}$ is the service demand of a job of class r at queue 1 and Z_r is the delay of class r spent in the infinite server station depicted in the figure. Note that routing probabilities are implicitly accounted for by the number of visits that define the values $D_{2,r}$ and $D_{3,r}$.*

2 Performance Model

Thanks to the large availability of material on product-form theory, we refer the reader to books and surveys such as [3,14,18,27–29,33] for comprehensive background and we limit here to definitions and required concepts.

Notation. We consider a computer system that may be modeled as a queueing network processing jobs belonging to R workload classes (i.e., transaction types). The non-negative integer N_r is the number of jobs of class r, the network population is $\vec{N} = (N_1, N_2, \ldots, N_R)$, and the total number of jobs in the model is $N = \sum_{r=1}^{R} N_r$. Without loss of generality, we consider a network of M load-independent queues serving jobs according to a first-come-first-served, processor-sharing or last-come-first-served-preemptive-resume scheduling discipline [1]. Queues with load-dependent service times are not considered in this paper. We denote by $D_{k,r}$ the *service demand* of class r at queue k, that is the product of the mean service time and the mean number of visits of class r jobs at station k [18]. Finally, we indicate with Z_r the mean *delay* of class r, i.e., the mean time spent by job at infinite-server stations. If not otherwise stated, in examples we assume $Z_r = 0$ for all classes. The main notation introduced above is summarized by Figure 1.

Model Evaluation. Recall that a queueing network model is a high-level abstraction of a continuous-time Markov chain where each state describes a distribution of jobs across the network [3]. For a model with M queues and R classes, a state of the chain is uniquely identified by a tuple

$$\vec{S} = (n_{0,1}, n_{0,2}, \ldots, n_{0,R}, \ldots, n_{k,1}, n_{k,2}, \ldots, n_{k,r}, \ldots, n_{k,R}, \ldots, n_{M,R}),$$

where $n_{0,r}$ is the total number of class r jobs in infinite-server stations and $n_{k,r}$, $1 \leq k \leq M$, is the number of class r jobs in queue k, either waiting or receiving service. The state space of the Markov chain is thus

$$\mathcal{S}(\vec{N}) = \left\{ \vec{S} \,|\, \sum_{k=0}^{M} n_{k,r} = N_r, 1 \leq r \leq R;\ n_{k,r} \geq 0, 0 \leq k \leq M, 1 \leq r \leq R \right\},$$

which defines a process that cannot be solved by the global balance equations because of its prohibitive dimension [28]. A direct evaluation of the state space can be avoided by the BCMP theorem [1], which asserts that the equilibrium state probabilities of the model are

$$\Pr(\vec{S}) = \frac{1}{G(\vec{N})} \left(\prod_{r=1}^{R} \frac{Z_r^{n_{0,r}}}{n_{0,r}!} \right) \left[\prod_{k=1}^{M} \left(n_k! \prod_{r=1}^{R} \frac{D_{k,r}^{n_{k,r}}}{n_{k,r}!} \right) \right],$$

where $\vec{S} \in \mathcal{S}(\vec{N})$, $n_k = \sum_{r=1}^{R} n_{k,r}$, and $G(\vec{N})$ is the *normalizing constant* which assures that the state probabilities sum to one, i.e.,

$$G(\vec{N}) = \sum_{\vec{S} \in \mathcal{S}(\vec{N})} \left(\prod_{r=1}^{R} \frac{Z_r^{n_{0,r}}}{n_{0,r}!} \right) \left[\prod_{k=1}^{M} \left(n_k! \prod_{r=1}^{R} \frac{D_{k,r}^{n_{k,r}}}{n_{k,r}!} \right) \right].$$

Given an efficient computational scheme for normalizing constants, performance indices such as server utilizations or mean response times can be readily computed [34]. For instance, the mean server utilization of class r jobs at queue k is given by

$$U_{k,r}(\vec{N}) = D_{k,r} \frac{G(\vec{N} - \vec{1}_r)}{G(\vec{N})}, \tag{1}$$

where $G(\vec{N} - \vec{1}_r)$ is the normalizing constant of a model having population $\vec{N} - \vec{1}_r$, i.e., $\vec{N}$ with a job of class r removed. Similarly, the mean queue-length of class r jobs at queue k is

$$Q_{k,r}(\vec{N}) = D_{k,r} \frac{G(\vec{1}_k, \vec{N} - \vec{1}_r)}{G(\vec{N})}, \tag{2}$$

where $G(\vec{1}_k, \vec{N} - \vec{1}_r)$ denotes the normalizing constant of a model where we artificially add a *replica* of queue k to the network, i.e., a new station k' with mean service demand $D_{k',r} = D_{k,r}$ for all classes r. Thus the solution of the model under study is $G(\vec{N}) = G(\vec{0}, \vec{N})$, $\vec{0} = (0, 0, \ldots, 0)$. Note that other mean indexes such as the mean response time $R_{k,r}(\vec{N})$ of class r jobs at queue k, the mean class r throughput of the system [1] $X_r(\vec{N})$, and the mean class r

[1] We assume system throughput to be observed at a reference station j for which the mean number of visits is $V_j = 1$. We point to the force flow law in [29, Chap. 3] for further details on the relation between the local throughput at each station and the system throughput.

end-to-end response time $R_r(N)$ are readily obtained from utilizations and response times as $R_{k,r}(\vec{N}) = Q_{k,r}(\vec{N})/X_r(\vec{N})$, $X_r(\vec{N}) = U_{k,r}(\vec{N})/D_{k,r}$ for any queue k, and $R_r(\vec{N}) = N/X_r(\vec{N})$, see [29] for further details.

3 Previous Work

The analysis of product-form models can be done either by the MVA approach [35], in which one recursively computes mean queue-lengths and mean throughputs, or by the normalizing constant approach [4], where one focuses on the computation of $G(\vec{N})$ and related constants and then obtains performance indexes by (1)-(2) and similar formulas. In this section, we review previous work in computational algorithms for the normalizing constant $G(\vec{N})$. Note that the MVA algorithm can be equivalently seen as a scaled version of the LBANC normalizing constant algorithm [10]. Therefore, we refer in this section to MVA and LBANC indifferently, since the two algorithms differ significantly only from a numerical standpoint, but not for their computational costs or for the fundamental structure of their recursion [10, 35]. Other exact approaches in the literature not considered in this section have essentially the same asymptotic computational costs of the methods presented below.

Convolution algorithm. In the Convolution approach [4, 34], the normalizing constant is computed using the following expression

$$G(\Delta m, \vec{N}) = G(\Delta m - \vec{1}_k, \vec{N}) + \sum_{r=1}^{R} D_{k,r} G(\Delta m, \vec{N} - \vec{1}_r), \tag{3}$$

for $1 \leq k \leq M$, where $\Delta m - \vec{1}_k$ indicates the removal of a replica of queue k from the network if $\Delta m_k > 0$ and the full removal [2] of queue k if $\Delta m_k = 0$. The above equation holds for any choice of the queue k to be removed and therefore can be solved recursively up to the termination conditions which are $G(\vec{N}) = \prod_{r=1}^{R} Z_r^{N_r}/N_r!$ for a model without queues and $G(\cdot, \vec{0}) = 1$ for a model without jobs. However, it is well-known that the computational costs of this recursion grow exponentially with the number of classes in the model and polynomially with the total population size, that is, as $O(N^R)$ in both time and space. As a result, the Convolution algorithm is not scalable in practice on models with more than 3 to 4 classes and with population of the order of a few tens of requests.

[2] This is valid if all stations in the model under study are distinct. We point to [8] for a discuss of applications of this formula involving the full removal of one or more stations.

Class recursion algorithms. The RECAL algorithm [14] and the RGF algorithm [20,21] require $O(N^M)$ time and storage and are best-suited for models with a small number of queues. In particular, the recurrence equation underlying RECAL is the *population constraint* (PC)

$$N_r G(\Delta\vec{m}, \vec{N}) = Z_r G(\Delta\vec{m}, \vec{N} - \vec{1}_r) + \sum_{k=1}^{M} \Big[(m_k + \Delta m_k) D_{k,r} G(\Delta\vec{m} + \vec{1}_k, \vec{N} - \vec{1}_r) \Big], \tag{4}$$

for arbitrary choice of the class r, $1 \leq r \leq R$, and where $m_k \geq 1$ is the number of queues with same demands $D_{k,r}$, $1 \leq r \leq R$, in the model under study. For simplicity, we assume throughout the rest of the paper that in the model under study all queues are distinct [3], i.e., $m_k = 1$. Using (2) it can be proved that (4) implicitly states the conservation law for the population in closed networks, i.e., $N_r = \sum_{k=1}^{M} Q_{k,r}(\vec{N}) + X_r(\vec{N}) Z_r$, where by Little's law [29] $X_r(\vec{N}) Z_r$ is the total number of class r requests waiting at the delay servers.

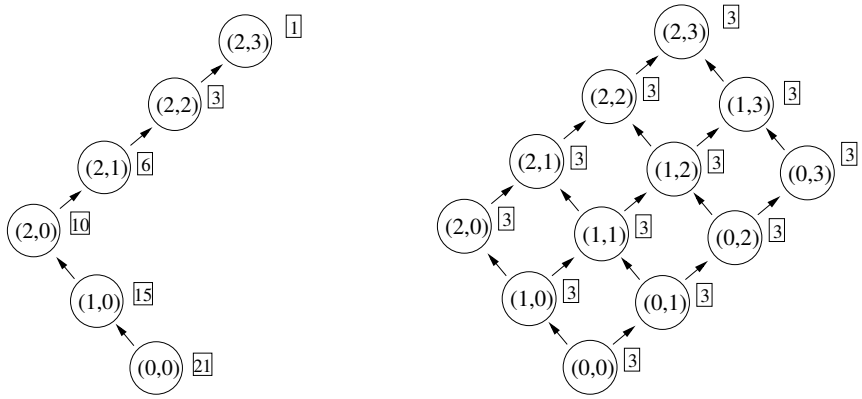
The fundamental property of methods such as RECAL and RGF is that the recursion is performed by changing the population of a single class at a time, thus the recursion tree is substantially different from that of Convolution, as shown in Figure 2(a). The diagram illustrates that, as the recursion on the different population proceeds, (4) considers an increasingly large number of normalizing constants at each step and it can be shown that this number grows as $O(N^M)$ with the total population [14]. This increase is due to the summation in the right-hand side of (4), which requires evaluating M models with different number of queue replicas. Thus, similarly to the fundamental equation of the Convolution algorithm, the PC cannot be used alone for the efficient evaluation of large multiclass models.

LBANC and MVA algorithms. The LBANC approach is based on a recursive evaluation of a reformulation of (3), i.e.,

$$G(\Delta\vec{m} + \vec{1}_k, \vec{N}) = G(\Delta\vec{m}, \vec{N}) + \sum_{r=1}^{R} D_{k,r} G(\Delta\vec{m} + \vec{1}_k, \vec{N} - \vec{1}_r), \tag{5}$$

for arbitrary choice of the queue k, $1 \leq k \leq M$. Because of the fundamental role played by the above equation in the MoM algorithm, we henceforth refer to (5) as the *convolution expression* (CE). In the CE, differently from (3), queues are added to the model instead of being removed, and the value of $G(\Delta\vec{m}, \vec{N})$ appearing in (5) is computed from the intermediate results of the previous recursive step based on a formula that is very similar to the PC (4). It can be shown by (1)-(2) that if $\Delta\vec{m} = \vec{0}$ then the above recursion reduces to

[3] However, the MoM pseudo-code given in the appendix is valid also for $m_k > 1$. We observe that in some models the multiplicity m_k may change throughout the recursion, we point to [5,8] for further details.



(a) CE (b) PC

Fig. 2. *Recursion tree for the model in Figure 1 with* $\vec{N} = (2, 3)$*,* $R = 2$ *classes,* $M = 3$ *queues, and* $Z_1 = Z_2 = 0$*. Each state is a different population vector spanned by the recursion. The values in the boxes indicate the number of constants evaluated by the recursion for that population vector.*

a computation of un-normalized throughputs and mean queue-lengths which has the same properties of the MVA algorithm.

Figure 2 compares the population recursion trees of the PC (4) and of the CE (5) for a small model with $R = 2$ classes, $M = 3$ queues, and $\vec{N} = (2, 3)$ jobs. Through the recursion, the constants $G(\vec{0}, \cdot)$ can be obtained immediately from the normalizing constants of the populations with a job less with no effort, see the LBANC algorithm for details [10]. It is interesting to observe that the recursion tree of the CE grows combinatorially with the population as $N_1 \times N_2$, but the number of evaluated constants remains fixed to 3 at each recursive step. Conversely, the PC evaluates a minimal set of populations, but during the recursion the number of evaluated constants grows significantly from 1 to 21 constants, which is very large for a model of this simplicity. These observations extend to the algorithms of previous works which employ relations similar to (4)-(5) and explain the high computational requirements of these methods on models with large populations where the number of evaluated constants can be even of the order of millions or more.

Clearly, it would be ideal to have an algorithm which performs a recursion on a small population such as for the PC in Figure 2(a), but in which the number of evaluated constants stays fixed for each population as for the CE in Figure 2(b). The main result of this paper, given in the next section, is to show that (4) and (5) can be used simultaneously to obtain this result, i.e., MoM has a recursion tree that grows linearly with the total population and where the number of terms evaluated at each step remains constant.

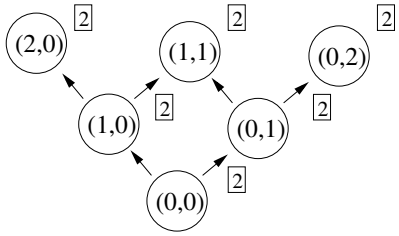


Fig. 3. *The basis $V(\vec{N})$ for a model with $M = 2$ queues and $R = 2$ classes. Each state represents a different value of $\Delta\vec{m}$; the boxed numbers indicate the number of constants $G(\Delta\vec{m}, \cdot)$ in $V(\vec{N})$.*

4 Method of Moments

In this section we introduce the MoM algorithm. We begin by considering a set of higher-order moments such that CE and PC can be used simultaneously and efficiently in computing the normalizing constant.

Definition 1 (Basis of Moments) *For a model with R classes, a* basis *of higher-order binomial moments is the set*

$$V(\vec{N}) = \nu(\vec{N}) \cup \nu(\vec{N} - \vec{1}_1) \cup \ldots \cup \nu(\vec{N} - \vec{1}_{R-1}),$$

where $\nu(\vec{n}) = \{G(\Delta\vec{m}, \vec{n}) \mid \sum_{k=1}^{M} \Delta m_k \leq R, \Delta m_k \geq 0\}$ is the set of normalizing constants of all possible models with population $\vec{n} \in \{\vec{N}, \vec{N} - \vec{1}_1, \ldots, \vec{N} - \vec{1}_{R-1}\}$ and $l = \sum_k \Delta m_k$ added queues for all $0 \leq l \leq R$.

An example of basis $V(\vec{N})$ for a model with $M = 2$ queues is shown in Figure 3. We remark that the moments in $\nu(\vec{N} - \vec{1}_R)$ are not included in the basis $V(\vec{N})$ because these moments are immediately available from the knowledge of $V(\vec{N} - \vec{1}_R)$ which is an intermediate result of the MoM recursive scheme.

We now give the main theoretical result of the paper.

Theorem 1 *There exist square matrices $\mathbf{A}(\vec{N})$ and $\mathbf{B}(\vec{N})$, defined by the coefficients of (4)-(5), which relate in a linear expression all and only the normalizing constants in the bases $V(\vec{N})$ and $V(\vec{N} - \vec{1}_R)$. That is, $V(\vec{N})$ satisfies the matrix difference equation*

$$\mathbf{A}(\vec{N})V(\vec{N}) = \mathbf{B}(\vec{N})V(\vec{N} - \vec{1}_R), \tag{6}$$

where both $\mathbf{A}(\vec{N})$ and $\mathbf{B}(\vec{N})$ have order independent of the total population N.

Proof 1 We first note that all constants in $V(\vec{N})$ augmented with l, $0 \leq l \leq R-1$, queue replicas can be immediately computed from $V(\vec{N} - \vec{1}_R)$ using the PC of class R. It is therefore only left to prove that we can compute recursively also the remaining elements of $V(\vec{N})$, which are normalizing constants

of models having R additional replicas. Note that in $V(\vec{N})$ there are

$$\binom{M+R-2}{R-1}R$$

constants, each having $R-1$ additional replicas, For each of this normalizing constants, we can write $R-1$ PCs of class $s \neq R$ and M CEs, therefore the number of equations obtained to estimate such unknowns is

$$\binom{M+R-2}{R-1}(M+R-1) = \binom{M+R-1}{R}R$$

However, the number of unknown constants with $l = R$ additional replicas is exactly

$$\binom{M+R-1}{R}R.$$

This proves that $\mathbf{A}(\vec{N})$ and $\mathbf{B}(\vec{N})$ are square matrices. $\square$

A pseudo-code that generates the matrices $\mathbf{A}(\vec{N})$ and $\mathbf{B}(\vec{N})$ using the derivations in the proof of Theorem 1 is given in the appendix. We remark that the structure of the matrices can be updated from population $\vec{N}$ to $\vec{N}+\vec{1}_R$ by changing the values N_R in $\mathbf{A}(\vec{N})$ to $N_R+\vec{1}_R$. This is discussed in details in Section 5.1.

We also observe that, from the given definition of basis, the knowledge of $V(\vec{N})$ and $V(\vec{N}-\vec{1}_R)$ immediately implies the knowledge of $G(\vec{N})$, $G(\vec{N}-\vec{1}_r)$, $1 \leq r \leq R$ and $G(\vec{1}_k, \vec{N}-\vec{1}_r)$, $1 \leq k \leq M$, $1 \leq r \leq R$, which are all elements of the two vectors. These normalizing constants are sufficient to compute the mean performance indices of interest using (1)-(2) and related formulas. The remaining normalizing constants included in the vectors $V(\vec{N})$ and $V(\vec{N}-\vec{1}_R)$ are instead higher-order quantities that, for instance, may be used to estimate queue-length variances and covariances [22].

The recursive computation of the basis $V(\vec{N})$ can be performed directly from (6) by interpreting the matrix difference equation as a system of linear equations where the right hand side vector of known terms is the result of a recursive step on a model with population $\vec{N}-\vec{1}_R$. Consistently, we define the Method of Moments (MoM) algorithm as follows.

Definition 2 (Method of Moments) *If $\mathbf{A}(\vec{N})$ is non-singular, then the basis $V(\vec{N})$ is computed recursively from $V(\vec{N}-\vec{1}_R)$ by*

$$V(\vec{N}) = \mathbf{A}^{-1}(\vec{N})\mathbf{B}(\vec{N})V(\vec{N}-\vec{1}_R). \tag{7}$$

Upon reaching the population $N_R = 0$, the recursion proceeds in a similar way on class $R-1$. Finally, the recursion terminates when $V(\vec{0})$ is reached,

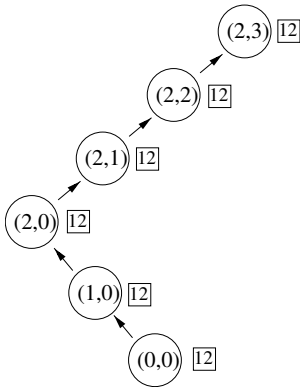


Fig. 4. *Recursion tree of the MoM algorithm for a model with $M = 2$ queues and $R = 2$ classes. Note that the number of normalizing constants evaluated at each step remains fixed throughout the recursion.*

since this vector contains only normalizing constants that take known values $G(\Delta\vec{m}, \vec{0}) = 1$ or $G(\Delta\vec{m}, \vec{0} - \vec{1}_r) = 0$ for all classes r.

We remark that, if $\mathbf{A}(\vec{N})$ is singular, then the MoM approach fails because a part of the normalizing constants in the basis $V(\vec{N})$ cannot be computed from $V(\vec{N} - \vec{1}_R)$ due to the linear dependencies between CEs and PCs. In this degenerate case, additional independent information is required to complete the evaluation of the queueing network model. We point to Appendix B for an overview of how the MoM approach changes under these degenerate cases and to [7] for further discussion of the problem. The following pseudo-code summarizes the general structure of the MoM recursion.

Method of Moments (MoM)

1: Compute $V(N_1, 0, \ldots, 0)$ using an efficient single class method, e.g., convolution [4] or LBANC [10].
2: **for** $r = 2$ **to** R **do**
3: Initialize elements of $V(N_1, \ldots, N_{r-1}, 0, \ldots, 0)$ based on the previous class results and the termination conditions $G(\Delta\vec{m}, \vec{0}) = 1$ and $G(\Delta\vec{m}, \vec{0} - \vec{1}_r) = 0$.
4: **for** $n_r = 1$ **to** N_r **do**
5: Setup and evaluate (7) obtaining $V = (N_1, \ldots, N_{r-1}, n_r, 0, \ldots, 0)$
6: **end for**
7: **end for**
8: Return mean performance indexes using (1)-(2) and the normalizing constants in $V(N_1, \ldots, N_R)$ and $V(N_1, \ldots, N_R - 1)$.

Remark 1: MoM Features. The key feature of MoM with respect to established methods such as MVA, Convolution, or LBANC, is that MoM requires just N steps to solve the queueing network model as illustrated in Figure 4. Compared to RECAL, instead, the number of normalizing constants evaluated by MoM remains fixed throughout the recursion and equal to the cardinality of $V(\vec{N})$, whereas throughout the RECAL recursion the number of normalizing constants grows combinatorially. We show in Section 6 that the complexity

expressions for the size of $V(\vec{N})$ are independent of the total population size N, thus the requirements of MoM in both time and space are always better than existing methods at least for a sufficiently large population N (typically a few tens of jobs in a model with more than three or four classes, even less for models with many classes).

Remark 2: Higher-Order Moments of Queue-Lengths. One of the main technical ideas behind the MoM algorithm is to add a certain number of queue replicas to the network in order to recursively compute higher-order moments of queue-lengths for the model under study. To provide intuition, we here introduce a probabilistic interpretation of the queue replica addition operation used in formulas such as (2).

Theorem 2 *Let $\Delta\vec{m} = (\Delta m_1, \Delta m_2, \ldots, \Delta m_M)$ be a vector of nonnegative integers and let the normalizing constant $G(\Delta\vec{m}, \vec{N})$ be for a model which differs from the original queueing network only for the inclusion of $\Delta m_k \geq 0$ additional replicas of queue k, for all queues $1 \leq k \leq M$. Further, assume that all queues are distinct, i.e., $m_k = 1$. Then, the (joint) binomial moment of queue-length*

$$\mathbb{E}[\Delta\vec{m}, \vec{N}] = \sum_{\vec{S} \in \mathcal{S}(\vec{N})} \prod_{k=1}^{M} \binom{n_k + \Delta m_k}{n_k} \Pr(\vec{S}), \tag{8}$$

is computed in terms of normalizing constants as

$$\mathbb{E}[\Delta\vec{m}, \vec{N}] = \left(\frac{1}{\Delta m_1! \Delta m_2! \cdots \Delta m_M!}\right) \frac{G(\Delta\vec{m}, \vec{N})}{G(\vec{N})}, \tag{9}$$

where $G(\vec{0}, \vec{N}) = G(\vec{N})$, and $\vec{0} = (0, 0, \ldots, 0)$.

The proof is reported in Appendix A. The immediate conclusion following from the statement of Theorem 2 is that a normalizing constant with a number of additional queue replicas can be interpreted as an un-normalized joint binomial moment of queue-lengths. Note that the use of binomial moments in place of the most popular power moments $E[X^k]$ provides an equivalent representation. In fact well-known queue-length descriptors such as mean, variance, skewness are readily computed from binomial moments as well, we point to [22] for moment conversion formulas.

Remark 3: Probabilistic Interpretation of MoM. A high-level probabilistic interpretation of (7) is as follows. The normalizing constants in the bases are interpreted as moments of order $l \leq R$ of the queue-lengths. Higher-order moments in $V(\vec{N})$ and $V(\vec{N} - \vec{1}_R)$ are not independent of each other because of the linear relations (4) and (5). Compared to all existing approaches in the literature, MoM is able to *simultaneously* exploit all these dependencies

thanks to a linear system of equations. In particular, the existence of a sufficient number of equations to avoid $\mathbf{A}(\vec{N})$ being under-determined is the main result and it is also the main driver behind Definition 1. For instance, if we set in $V(\vec{N})$ a different maximum number of additional replicas, say $R-1$ instead of R, then the linear system (6) would be under-determined and the MoM recursion would not be possible.

Remark 4: Intuitive Justification. We now sketch a simple illustration of the linear dependencies exploited by the MoM approach; this can help in the definition of new efficient algorithms for queueing networks. Let us begin by observing that each $G(\Delta\vec{m}, \vec{N}) \in V(\vec{N})$ may be considered as the solution of a queueing network that is more complex than the model under study because of the additional queue replicas. Therefore, it is not immediately clear how this may lead to the simplifying result (7). We explain the intuition behind this by an example. Consider the recursive computation of the normalizing constant $G((1,0), \vec{N})$ using the PC of class 1

$$N_1 G((1,0), \vec{N}) = D_{1,1} G((2,0), \vec{N} - \vec{1}_1) + D_{2,1} G((1,1), \vec{N} - \vec{1}_1),$$

where we assume to avoid unnecessary complexity that $Z_1 = 0$. Assume now to evaluate also the constant $G((0,1), \vec{N})$, then we can write the PC

$$N_1 G((0,1), \vec{N}) = D_{1,1} G((1,1), \vec{N} - \vec{1}_1) + D_{2,1} G((0,2), \vec{N} - \vec{1}_1).$$

Noting that the two PCs share the same normalizing constant $G((1,1), \vec{N} - \vec{1}_1)$, it follows that the cost of adding $G((0,1), \vec{N})$ to the analysis is the new unknown $G((0,2), \vec{N} - \vec{1}_1)$, but this is compensated, in terms of the linear system rank in (7), by the extra equation added. If we now repeat the same reasoning on normalizing constants with two or more additional replicas, it is found that the degree of overlapping between the equations is much greater, e.g., it is possible to formulate several CEs in addition to the PCs. That is, we found that the number of shared normalizing constants between the equations grows faster than the number of equations itself. MoM exploits this remarkable property of closed models to perform a recursion with length that grows linearly with the population size.

Remark 5: Numerical Properties. Normalizing constant computations are known from long time to be numerically difficult [26], often due to floating-point range exceptions that arise in large models where the number of states can exceed many billions. Such numerical issues affect MoM as well, and they are even worsened by round-off errors that often accumulate in the solution of a long sequence of linear systems. Despite this difficult, MoM addresses effectively both problems using an exact linear system solver in implementations, such as those available in many popular computer algebra systems, e.g., MAPLE, Mathematica, MATLAB Symbolic Toolbox, Maxima. Essentially, exact solvers provide arbitrary solution accuracy at the expense of some overheads due to

multiprecision computations or modulo representations. Since this is a simple and definitive solution to all numerical issues of normalizing constant computations, we no longer focus on this problem except in the analysis of computational overheads in Section 6.

4.1 Illustrating Example

The structure of (6) is now illustrated on a small model with $M = 2$ queues, $R = 2$ classes, a population of users[4] $\vec{N} = (N_1, N_2)$, and where $m_k = 1$, $1 \leq k \leq M$. We start from an intermediate step of the recursion where we know $\vec{V}(N_1, 0)$ and we need to compute $\vec{V}(N_1, N_2)$ by progressively computing using (6) the vectors $\vec{V}(N_1, 1)$, $\vec{V}(N_1, 2)$, ..., $\vec{V}(N_1, n2)$, ..., $\vec{V}(N_1, N_2 - 1)$,

[4] Throughout the paper we refer to users and jobs indifferently.

$\vec{V}(N_1, N_2)$. In this case, we can write

$$\mathbf{A}(N_1, n_2) = \begin{bmatrix} n_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & n_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & n_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & n_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & n_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n_2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -L_{1,1} & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -L_{1,1} & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -L_{2,1} & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -L_{2,1} \\ 0 & N_1 & 0 & 0 & 0 & 0 & 0 & -Z_1 & 0 & -2L_{1,1} & -L_{2,1} & 0 \\ 0 & 0 & N_1 & 0 & 0 & 0 & 0 & 0 & -Z_1 & 0 & -L_{1,1} & -2L_{2,1} \end{bmatrix}$$

$$\mathbf{B}(N_1, n_2) = \begin{bmatrix} Z_2 & L_{1,2} & L_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & Z_2 & 0 & 2L_{1,2} & L_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_2 & 0 & L_{1,2} & 2L_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & Z_2 & L_{1,2} & L_{2,2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z_2 & 0 & 2L_{1,2} & L_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z_2 & 0 & L_{1,2} & 2L_{2,2} \\ 0 & 0 & 0 & L_{1,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & L_{1,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & L_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & L_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

where we use the following basis

$$\vec{V}(\vec{N}) = [G(\vec{0}, \vec{N}), G(\vec{1}_1, \vec{N}), G(\vec{1}_2, \vec{N}), G(2{\cdot}\vec{1}_1, \vec{N}), G(\vec{1}_1{+}\vec{1}_2, \vec{N}), G(2{\cdot}\vec{1}_1, \vec{N}), G(\vec{0}, \vec{N}{-}\vec{1}_1), G(\vec{1}_1, \vec{N}{-}\vec{1}_1), G(\vec{1}_2, \vec{N}{-}\vec{1}_1), G(2{\cdot}\vec{1}_1, \vec{N}{-}\vec{1}_1), G(\vec{1}_1{+}\vec{1}_2, \vec{N}{-}\vec{1}_1), G(2{\cdot}\vec{1}_1, \vec{N}{-}\vec{1}_1)]^T \tag{10}$$

Here, the variable n_2 is incremented throughout the MoM iterations starting from $n_2 = 1$. For a model where $D_{1,1} = 10$, $D_{1,2} = 5$, $D_{2,1} = 4$, $D_{2,2} = 9$, $Z_1 = 0$, $Z_2 = 0$, $N_1 = 3$, the intermediate vector we start from to apply (6) is

$$\vec{V}(3,0) = [1624, 5584, 2536, 12944, 7616, 3800, 156, 396, 228, 736, 508, 316]^T$$

Using (6) for $n_2 = 1$, we immediately compute

$$\begin{aligned}\vec{V}(3,1) =& \mathbf{A}^{-1}(3,1)\mathbf{B}(3,1)\\ =& [50744, 197984, 106480, 519024, 347840, 198760, 4032, 11932, 8228, 25632, 20328, 14520]^T\end{aligned}$$

and similarly for $n_2 = 2$ we have

$$\begin{aligned}\vec{V}(3,2) =& \mathbf{A}^{-1}(3,2)\mathbf{B}(3,2)\\ =& [974120, 4160400, 2658440, 12031680, 9219840, 6023840, 66856, 219636, 181500, 527616, 482\end{aligned}$$

which are the exact values of the normalizing constants in the $\vec{V}(3,2)$ vector, in particular $G(\vec{0}, \vec{N}) = 974120$.

5 Optimized Method of Moments

This section investigates structural properties of (6)-(7) and shows how to minimize the computational costs of the MoM recursion.

5.1 Block Triangular Form

A significant computational cost associated to (7) is that we need to solve the linear system for all the N populations evaluated during the recursion. In general, because of the numerical properties discussed in the previous section, one may only use exact or iterative methods that provide an exact solution to the linear system. We henceforth assume that Gaussian elimination is used in MoM together with LU decomposition [19] which provides an upper bound on the computational requirements of the linear system solution.

In order to optimize MoM, we begin by observing that if LU decomposition is used to compute (7), a new decomposition needs to be performed at each step of the recursion. We now show that the structure of $\mathbf{A}(\vec{N} - \vec{1}_R)$ can be largely reused for defining $\mathbf{A}(\vec{N})$.

Proposition 1 *The matrix $\mathbf{A}(\vec{N})$ can always be permuted to the block upper*

triangular form

$$\mathbf{A}(\vec{N}) = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{0} & N_R \mathbf{I}_p \end{bmatrix}, \tag{11}$$

where $\mathbf{A}_{1,1}$ *is square of order* $q = \binom{M+R-1}{R} R$*, the blocks* $\mathbf{A}_{1,1}$ *and* $\mathbf{A}_{1,2}$ *are independent of* N_R*, and* $\mathbf{I}_p$ *is the identity matrix of order* $p = \binom{M+R}{R} R - q$*.*

Proof 2 The block triangular form of $\mathbf{A}(\vec{N})$ follows by observing that, during the recursion on class R, the only coefficients that change are the N_R's of the PC of class R. The size of $\mathbf{A}_{1,1}$ is q, as observed in the proof of Theorem 2; thus, since the size of $\mathbf{A}(\vec{N})$ is

$$\sum_{l=0}^{R} \binom{M+l-1}{l} R = \binom{M+R}{R} R,$$

p follows immediately. □

The last result can be exploited to avoid unnecessary LU decompositions as follows. Let us partition $V(\vec{N})$ into vectors $V_q(\vec{N})$ and $V_p(\vec{N})$ with respectively the q and p rows considered in the above Theorem. Let $\mathbf{B}_q(\vec{N})$ and $\mathbf{B}_p(\vec{N})$ be analogous partitions of $\mathbf{B}(\vec{N})$. We can reformulate (7) as

$$V_q(\vec{N}) = \mathbf{A}_{1,1}^{-1}(\mathbf{B}_q(\vec{N})V(\vec{N} - \vec{1}_R) - \mathbf{A}_{1,2}V_p(\vec{N})), \tag{12}$$

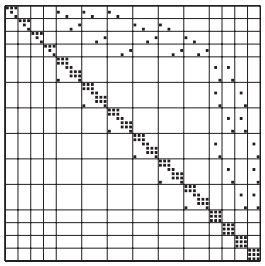
$$V_p(\vec{N}) = N_R^{-1}\mathbf{B}_p(\vec{N})V(\vec{N} - \vec{1}_R). \tag{13}$$

in which we reuse the same LU decomposition of $\mathbf{A}_{1,1}^{-1}$ while processing the population in class R[5]. That is, at each step of the recursion we can compute (12) by LU back-substitution [19] instead of the more expensive LU decomposition. This implies a significant computational saving, since back-substitution has quadratic complexity in the order of $\mathbf{A}_{1,1}$ instead of the cubic time of LU decomposition. With the proposed approach, LU decomposition only needs to be performed at the beginning of the recursion on a new class. For example, in the recursion tree in Figure 1(b) this would be performed when the recursion reaches populations $(1, 0)$ and $(2, 1)$.

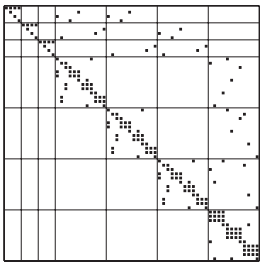
5.2 Fine-Grain Decomposition

A further optimization is obtained by decomposing the matrix $\mathbf{A}_{1,1}$ in (12). Figure 5 depicts $\mathbf{A}_{1,1}$ for models with different values of M and R. It is im-

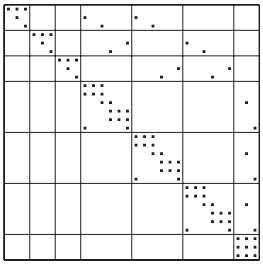
[5] Indeed, to optimize time and space requirements, inverse matrices are never explicitly computed in MoM and the inverse matrix notation used throughout the paper indicates the solution of a linear system of equations.



(a) $M = 4$, $R = 3$, order $= 60$



(b) $M = 3$, $R = 4$, order $= 60$



(c) $M = R = 3$, order $= 30$

Fig. 5. *Sparsity structure of* $\mathbf{A}_{1,1}$ *for different model sizes. Points indicate nonzero entries; lines indicate the fine-grain decomposition. The decomposition generates blocks with much smaller order than* $\mathbf{A}_{1,1}$ *and is more effective as* M *grows. Note that the order of* $\mathbf{A}_{1,1}$ *is independent of the populations in* $\vec{N}$*.*

mediate to see that also $\mathbf{A}_{1,1}$ is a structured matrix. In particular, we prove below that $\mathbf{A}_{1,1}$ admits a simple block triangular form.

Theorem 3 *The matrix* $\mathbf{A}_{1,1}$ *admits the block triangular form*

$$\mathbf{A}_{1,1} = \begin{bmatrix} \mathbf{C}_1 & \mathbf{C}_{12} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_2 & \mathbf{C}_{23} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{C}_3 & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{C}_{H-1} & \mathbf{C}_{H-1,H} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{C}_H \end{bmatrix}, \tag{14}$$

where $H = \min(M, R)$ *and*

$$\mathbf{C}_h \equiv \mathrm{diag}(\mathbf{C}_h^{(1)}, \mathbf{C}_h^{(2)}, \dots, \mathbf{C}_h^{(t)}, \dots, \mathbf{C}_h^{(T_h)}), \quad 1 \le h \le H,$$

is block diagonal composed by $T_h = \binom{M}{h}$ *blocks* $\mathbf{C}_h^{(t)}$*, each having order* $\binom{R-1}{R-h} R$*.*

Proof 3 Consider the set of normalizing constants $G(\Delta\vec{m}, \vec{N})$ in $V(\vec{N})$ with $l = R$ additional queues. This is uniquely defined by the set of vectors $\Delta\vec{m}$ with $\sum_{k=1}^{M} \Delta m_k = R$, $\Delta m_k \geq 0$, which we partition according to the number and position of non-zeros in $\Delta\vec{m}$. E.g., for $M = R = 3$ we have the partitions $v_{11} = \{(3,0,0)\}$, $v_{12} = \{(0,3,0)\}$, $v_{13} = \{(0,0,3)\}$, $v_{21} = \{(1,2,0),(2,1,0)\}$, $v_{22} = \{(1,0,2),(2,0,1)\}$, $v_{23} = \{(0,2,1),(0,1,2)\}$, and $v_{31} = \{(1,1,1)\}$, where we have used the notation v_{hp} to indicate a set of vectors $\Delta\vec{m}$ with h non-zeros placed according to the pth pattern. It is possible to verify that the range of h is $1 \leq h \leq H$, $H = \min\{M, R\}$. The block $\mathbf{C}_h^{(t)}$ are then obtained by selecting the columns associated to a set of normalizing constants with $\Delta\vec{m}$ vector equal to $v_{h,t}$, and the rows for the related CE's and PC's. The matrix C_h includes all blocks $c_{h,\cdot}$, which in number are equal to the way of choosing the positions of the h non-zeros in $\Delta\vec{m}$, i.e.,

$$\binom{h + (R-h) - 1}{R-h} = \binom{R-1}{R-h}.$$

Using Vandermonde's convolution [12], we have also

$$\binom{M+R-1}{R} R = \sum_{h=1}^{H} \binom{M}{h} \binom{R-1}{R-h} R,$$

where the left-hand side is exactly the order of $\mathbf{A}_{1,1}$. This implies that the cardinality of each C_h matrix is exactly $T_h = \binom{M}{h}$ blocks of order $\binom{R-1}{R-h} R$. □

The form (14) implies new computational savings, since (12) can be computed by operating on the small diagonal blocks $\mathbf{C}_h^{(t)}$. Since the costs of LU decomposition and back-substitution are respectively cubic and quadratic in the order of the coefficient matrix of the linear system this optimization immediately translates in large computational savings in both time and space.

5.3 Fine-Tuning the MoM algorithm

An optimized MoM algorithm is obtained by recursively solving (12)-(14) in place of (6). The block triangular forms provide substantial computational savings and they can be applied directly to the basis $V(\vec{N})$. However, for increased efficiency, one could further fine-tune the implementation by making the composition of the basis adaptive with respect to the number of non-empty classes. That is, for an intermediate population $\vec{n} = (n_1, n_2, \ldots, n_s, 0, \ldots, 0)$, $\vec{n} \leq \vec{N}$, in which only the first s classes are non-empty, the fine-tuned optimized MoM defines a basis of normalizing constants

$$V^s(\vec{n}) = \nu^s(\vec{n}) \cup \nu^s(\vec{n} - \vec{1}_1) \cup \ldots \cup \nu^s(\vec{n} - \vec{1}_s),$$

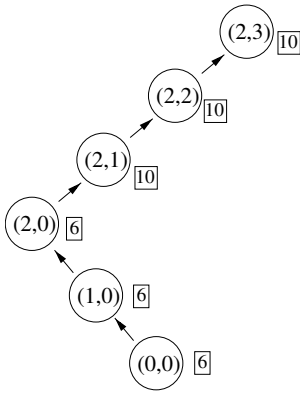


Fig. 6. *Recursion tree of the optimized MoM algorithm with fine-tuning. The model has $M = 2$ queues and $R = 2$ classes. Note that the adaptive base makes the number of normalizing constants evaluated at each step dependent on the currently processed class. Compared to Figure 4, it is also possible to see that the basis size is always smaller than in the basic MoM algorithm.*

where $\nu^s(\vec{n}) = \{G(\Delta\vec{m}, \vec{n}) \,|\, s - 1 \leq \sum_k \Delta m_k \leq s\}$. Such a basis can be seen a set of binomial moments of order s and $s-1$. It is simple to show with a proof similar to the one of Theorem 2 that the system of linear equations that relates the basis $V^s(\vec{n})$ with $V^s(\vec{n} - \vec{1}_s)$ has coefficient matrix $\mathbf{A}^s(\vec{n})$ that is square. Therefore, during the recursion on the intermediate classes $s < R$, such a basis can be used without making the MoM approach fail. This basis, originally proposed in [5], is much smaller than the one used in this paper since it ignores the normalizing constants for $\sum_k \Delta m_k < s - 1$. A comparative evaluation of the basis size in MoM and its optimized version is given in Section 6. Figure 6 shows the recursive tree of the optimized MoM in comparison with the one of the basic MoM is presented in Figure 4.

We remark that the reduced basis size requires an increased implementation effort. Besides the implementation of an adaptive basis, which requires generating different $\mathbf{A}^s(\vec{n})$ and $\mathbf{B}^s(\vec{n})$ matrices depending on the number of non-empty classes s. Furthermore, one needs to compute $V^{s+1}(n_1, n_2, \ldots, n_s, 0, \ldots, 0)$ from $V^s(n_1, n_2, \ldots, n_s, 0, \ldots, 0)$ when starting the iteration on class $s + 1$. This can be done by using directly the equations (4)-(5) which relate all and only the unknown constants in $V^{s+1}(\vec{n})$ with the constants in $V^s(\vec{n})$. The resulting linear system has the same $\mathbf{A}_{1,1}$ matrix used for class $s+1$ and therefore is always invertible. Computing performance indices also requires solving the inverse problem of determining $V^{s-1}(\vec{n})$ from $V^s(\vec{n})$, for all $1 \leq s \leq R$, which is similarly solved by direct application of (4)-(5).

6 Computational Costs

In this section we give an asymptotic complexity analysis of the basic and optimized versions of MoM as the population N grows.

6.1 MoM requirements

The recursive evaluation of (7) requires solving N linear systems with order equal to the cardinality of the basis $V(\vec{N})$ which consists of $\binom{M+R}{M}R$ normalizing constants. Assume that the linear system is solved by LU decomposition, then the cost of the solution is cubic in the order of the matrix, and the overall time requirement is approximately

$$\alpha(N)\left[\binom{M+R}{M}R\right]^3 N$$

operations, where $\alpha(N)$ accounts for the additional costs due to the use of exact algebra in linear system solution. The $\alpha(N)$ factor is implementation-dependent; assuming that multiprecision arithmetic is used to achieve an exact solution of the linear system, $\alpha(N)$ accounts for the number of significant digits in the normalizing constants. In order to determine the complexity of MoM we first study the asymptotic properties of $\alpha(N)$ [6].

Lemma 1 *The number of digits of the entries of* $\mathbf{A}^{-1}(\vec{N})$ *and* $\mathbf{B}(\vec{N})$ *grows asymptotically as* $O(\log N)$ *for increasing* N.

Proof 4 We focus on $\mathbf{A}^{-1}(\vec{N})$, but the proofing technique applies similarly to $\mathbf{B}(\vec{N})$ leading to the same conclusion. The digits of the entries of $\mathbf{A}(\vec{N})$ grow as $O(\log N)$ for the entries N_r, $1 \leq r \leq R$, and as $O(1)$ for all other elements. Note that the computation of the inverse involves at worst products of the type $N_1^{k_1}N_2^{k_2}\cdots N_R^{k_R}$ for some constants $k_1, k_2, \ldots, k_R$ depending on the order of $\mathbf{A}(\vec{N})$. But being the order of $\mathbf{A}(\vec{N})$ constant with respect to N, the digits in $\mathbf{A}^{-1}(\vec{N})$ grow as $O(\log(N_1^{k_1}N_2^{k_2}\cdots N_R^{k_R})) = O(k_1\log(N_1)+\cdots+k_R\log(N_R))) = O(\log N)$.

Lemma 2 *The number of digits of* $G(\vec{N})$ *grows asymptotically as* $O(N\log N)$.

[6] Throughout the section, we assume that all multiprecision numbers are rational numbers, and that the number of digits is the maximum between the digits of the numerator and of the denominator. This is consistent with the representation used in most multiprecision libraries and, as long as the service demands can be defined as rational numbers, it does not affect in any way the generality of MoM.

Proof 5 Let us first consider the case $Z_r = 0$, $1 \leq r \leq R$. By the definition of $G(\vec{N})$ given in Section 2, the maximum number of digits is for the case where all service demands are equal to $D = \max_{k,r} D_{k,r}$. Since the system is balanced, using an approach similar to the proof of Theorem 2 we have that

$$G(\vec{N}) = \frac{N!}{N_1!N_2!\cdots N_R!}\binom{M+N-1}{N}D^N = \frac{(M+N-1)!}{(M-1)!N_1!N_2!\cdots N_R!}D^N$$

and the asymptotic grow of digits is dominated by $(M+N-1)!$ which is $O(N \log N)$. The case where one or more classes are $Z_r > 0$ follows similarly by considering a model with $M+1$ queues with identical demands $D = \max_r\{Z_r, \max_k D_{k,r}\}$, and no delay. Replacing the delay with a queue introduces an additional factor in the normalizing constant summation, and hence the new normalizing constant has an increased number of digits but still grows as $O(N \log N)$.

Let $O^{\sim}(\cdot)$ be the standard "soft-Oh" notation which, for readability, ignores poly-logarithmic factors, e.g., $O(N \log N \log\log N)$ becomes $O^{\sim}(N)$. We can now bound the growth of $\alpha(N)$ as follows.

Theorem 4 *The asymptotic growth of $\alpha(N)$ is $O^{\sim}(N)$ and approximately $O(N \log N)$.*

Proof 6 The computation of (7) requires first the multiplication of normalizing constants with the coefficients in $\mathbf{A}^{-1}(\vec{N})$ and $\mathbf{B}(\vec{N})$, and then the summation of the results. The cost of these operations grows no more than $O(n \log n \log\log n)$ with the number of digits n of the operands [36]. According to the previous lemmas, n grows at most as $O(N \log N)$, and we finally obtain that the growth of $\alpha(N)$ is

$$O(N \log N \log(N \log N) \log\log(N \log N))$$

which is $O^{\sim}(N)$ and approximately $O(N \log N)$.

Since $\alpha(N)$ grows as $O^{\sim}(N)$, the asymptotic time complexity of MoM is $O^{\sim}(N^2)$ which largely improves the $O(N^R) = O^{\sim}(N^R)$ and $O(N^M) = O^{\sim}(N^M)$ of existing methods for multiclass models.

Given the order $\binom{M+R}{M}R$ of the linear system, the storage cost is approximately

$$\left[\binom{M+R}{M}R\right]^2 + \binom{M+R}{M}R$$

multiprecision values [7], where the first term accounts for the matrices $\mathbf{A}(\vec{N})$;

[7] Here and in the rest of the section we only account for the storage costs of $\mathbf{A}(\vec{N})$ and $\mathbf{A}_{1,1}(\vec{N})$, since all other matrices have few non-zeros.

Table 1
Comparison of Linear System Size in MoM

		Order		*Sparsity* $(0 \div 1)$	
M	R	*MoM**	*opt. MoM**	*MoM*	*opt. MoM*
2	2	12(12)	6(2)	0.181	0.333
2	3	30(30)	12(6)	0.073	0.208
2	4	60(60)	20(12)	0.036	0.140
2	5	105(105)	30(20)	0.020	0.100
3	2	20(20)	12(2)	0.125	0.188
3	3	60(60)	30(6)	0.045	0.100
3	4	140(140)	60(12)	0.019	0.058
3	5	280(280)	105(30)	0.010	0.037
4	2	30(30)	20(2)	0.091	0.120
4	3	105(105)	60(6)	0.029	0.056
4	4	280(280)	140(12)	0.011	0.029
4	5	630(630)	280(30)	0.005	0.016
5	2	42(42)	30(2)	0.069	0.083
5	3	168(168)	105(6)	0.020	0.034
5	4	504(504)	280(12)	0.007	0.016
5	5	1260(1260)	630(30)	0.003	0.008

**: the order of largest matrix or block solved in a linear system is given between brackets*

the latter is the storage size of $V(\vec{N})$ and $V(\vec{N} - \vec{1}_R)$. Hence, the asymptotic storage is $O^{\sim}(N)$ due to $\alpha(\vec{N})$ since the number of stored values for the linear system matrices is independent of the population size.

Indeed, in actual implementations the hidden constants in the $O^{\sim}(\cdot)$ notation may have a non-negligible computational impact. For this reason, we report in the following subsections experimental results on the actual time and storage requirements of the method on various instances.

6.2 Optimized MoM requirements

Despite the asymptotic complexity of the optimized MoM algorithm is theoretically the same of MoM, i.e., $O^{\sim}(N^2)$ time and $O^{\sim}(N)$ storage, the former performs much better than the latter. Consider, for instance, Table 1, which

compares the actual order of $\mathbf{A}(\vec{N})$ for MoM with the size of $\mathbf{A}_{1,1}(\vec{N})$ in the linear system (12) used by the optimized MoM. The latter is approximately half of the former. For instance, for a model with $M = R = 5$, MoM needs to invert a matrix $\mathbf{A}(\vec{N})$ of order 1260, while the optimized MoM considers a matrix $\mathbf{A}_{1,1}(\vec{N})$ of order 630; further, it only inverts a set of matrix blocks in the block triangular form of $\mathbf{A}_{1,1}(\vec{N})$ with order no greater than 30. Since the cost of inversion grows cubically with matrix order, the optimized MoM is far more efficient that the basic MoM.

Recalling that LU decomposition needs to be performed only once for each class, and later one only needs LU back-substitution [19], the time cost of the optimized MoM is approximately

$$\alpha(N)\left[\binom{M+R}{M}R\right]^3 R + \alpha(N)\left[\binom{M+R}{M}R\right]^2 N$$

operations, where the first term accounts for LU decompositions and the second for LU back-substitutions. Note that in the first term $\alpha(N)$ indicates the cost of the entries of $\mathbf{A}(\vec{N})$ only, which grows as $O(\log N)$, nevertheless the second term dominates asymptotically yielding a $O^{\sim}(N^2)$ time complexity.

Regarding storage, the cardinality of the basis $V^s(\vec{N})$ is

$$\binom{M+s-2}{s-1}s + \binom{M+s-1}{s}s$$

normalizing constants, and thus the maximum storage cost is for $s = R$ and is approximately

$$\left[\binom{M+R-2}{R-1}R + \binom{M+R-1}{R}R\right]^2 + \left[\binom{M+R-2}{R-1}R + \binom{M+R-1}{R}R\right],$$

multiprecision values. The first term accounts the store requirements of $\mathbf{A}(\vec{N})$; the second term is the memory occupation of the bases. Since the number of stored value is again independent of N, the optimized MoM storage requirements grow asymptotically as $O^{\sim}(N)$.

7 Experimental Evaluation and Case Studies

To provide intuition on the actual performance of MoM and of the optimized MoM, we performed an experimental campaign on a Intel Centrino 2 - 1.30GHz. During experiments, the available physical memory was approximately 4GB, and we interrupted execution whenever this limit was exceeded.

Table 2
Experimental Comparison of MoM, the optimized MoM, and MVA - CPU Time. Class populations are $N_r = \lceil N/R \rceil$, $1 \leq r \leq R$.

			CPU [s]		
M	R	N	MoM*	Opt.MoM*	MVA
3	3	100	2.45	0.02	0.02
3	3	350	5.99	0.06	0.56
3	3	600	10.96	0.12	2.84
3	3	850	17.53	0.22	8.10
3	3	1100	24.74	0.35	17.39
3	3	1350	32.87	0.93	limit
4	3	100	7.32	0.03	0.02
4	3	350	23.19	0.16	0.68
4	3	600	43.54	0.38	3.46
4	3	850	66.67	0.69	9.76
4	3	1100	85.62	0.69	21.07
4	3	1350	122.47	0.99	limit
3	4	100	11.17	0.03	0.16
3	4	350	39.30	0.16	25.63
3	4	600	71.86	0.37	45.13
3	4	850	108.22	0.65	limit
3	4	1100	145.58	0.75	limit
3	4	1350	174.05	1.07	limit

$*$: includes numerical stabilization overhead.
limit : memory exceeded (4GB).

The comparison was performed against the MVA algorithm [35] due to its wide diffusion, and also because for large populations it is typically the best performing among existing methods, albeit inefficient. For the optimized MoM and MVA we developed fine-tuned C implementations; for MoM, we have used a Java implementation. The multi-precision arithmetic has been implemented in the fine-tuned optimized MoM using the GNU GMP libraries, conversely for MoM we have used the standard Java BigNum class.

We consider in this section simple models with no more than $M = 4$ queues or $R = 4$ classes, for which existing methods can be feasible under large populations and can be compared with MoM and the optimized MoM. The experimental results for different values of the total population N are given in Table 2 and Table 3. For instance, the first experiments are done on a model with $M = R = 3$, $N = 100$, and each population N_r, $1 \leq r \leq R$, is set equal to the integer value closest to N/R, i.e., $N_r = 33$. For each experiment, we

Table 3
Experimental Comparison of MoM, the optimized MoM, and MVA - Memory. Class populations are $N_r = \lceil N/R \rceil$, $1 \leq r \leq R$.

			Memory [MB]		
M	R	N	MoM*	Opt.MoM*	MVA
3	3	100	0.41	0.22	2.15
3	3	350	0.44	0.39	61.74
3	3	600	0.46	0.64	310.43
3	3	850	0.49	0.87	874.45
3	3	1100	0.52	1.10	1886.29
3	3	1350	0.55	1.32	limit
4	3	100	0.68	0.31	2.75
4	3	350	0.72	0.57	86.19
4	3	600	0.77	0.90	434.33
4	3	850	0.82	1.20	1223.98
4	3	1100	0.87	1.50	2640.44
4	3	1350	0.92	1.82	limit
3	4	100	0.94	0.41	18.09
3	4	350	1.00	0.78	2288.31
3	4	600	1.05	1.11	3528.38
3	4	850	1.11	1.49	limit
3	4	1100	1.17	1.94	limit
3	4	1350	1.23	2.31	limit

$*$: includes numerical stabilization overhead.
limit : memory exceeded (4GB).

collected CPU time and peak memory occupation. The results in the tables indicate the improved applicability of MoM and its optimized version with respect to the MVA algorithm, as discussed in the following remarks.

Remark 1. The CPU requirements are always favorable to the optimized MoM algorithm, with the exception of the two cases with just $N = 100$ jobs. Indeed, for smaller population values existing methods like the MVA are very effective, and tend to be faster than the optimized MoM since they do not employ multi-precision algebra. However, as N increases, the degradation of performance of the MVA is much faster than for the optimized MoM, and this confirms the effectiveness of the MoM approach on models with large populations.

Remark 2. The most striking computational gain is however for the storage costs, where the optimized MoM largely outperforms MVA. Among the twelve experiments, the MVA failed several times due to its prohibitive storage costs

which greatly exceeded the available memory; on the same experiments, the optimized MoM occupied no more than 1.7MB. Thus, this feature is a fundamental advantage of MoM over MVA, which allows one to evaluate models with large populations. Note that, despite the theoretical worst-case storage complexity is linear in the population size, the storage occupation and its growth rate are both negligible on all examples.

Remark 3. MoM without optimization is slower than the other methods, yet it is able to solve in a few minutes models that are prohibitive for the MVA. However, the implementation of MoM in Java requires about 400 lines of code, whereas the optimized MoM needs more than 1000 lines of code due to the additional complexity in handling the block diagonal form decomposition and the change in basis size described in Section 5.3.

Remark 4. Figure 7 illustrates the practical limits of the MoM algorithm on commodity hardware (Intel Centrino 2 - 1.30GhZ). We have run a series of experiments considering models with random service demands drawn uniformly in $[1, 10000]$ with minimum possible population $\vec{N} = (1, 1, \ldots, 1)$ such that each class has a single job. Since the MoM recursion in the number of jobs is efficient, the objective of this experiments is to determine when either the storage requirements or the cost of the matrix factorizations using LU decomposition become too large for practical use. In this analysis, we have used a time limit of 5 hours and a memory limit of 3.5GB. The results in Figure 7 indicate that the number of queues that can be considered as the number of classes grows decreases quite quickly, although models with a few queues and more than 10 classes are still manageable. This is a result of the combinatorial growth of the size of the basis $\vec{V}(\vec{N})$ and consequently of the linear system order in (6). In comparison, the MVA algorithm becomes too expensive to apply already with $4-5$ classes. Conversely, MoM is applicable also to models with tens of classes, and experiments not reported in the figure indicate that with just $M = 2$ queues one may consider in the model up to $R = 50$ classes. For models with fewer than 4 classes, models with more than 100 queues can be solved efficiently by MoM, however models of this size are seldom used in practice. Summarizing, MoM is far more scalable than MVA but it may not be applicable to models with very large number of queues or classes. To address this issue we have proposed in [7, 8] related algorithms that explore in new ways the potentiality of the higher-order moment approach to solve closed queueing networks. In those works, it is found that models with many classes and many queues can be solved more efficiently with recursions that differ from the one proposed in the present paper. Still, MoM is much simpler and practical to implement than methods such as the ones in [8].

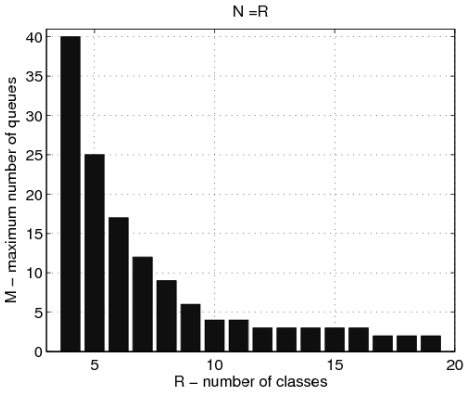


Fig. 7. *Applicability of the optimized MoM on commodity hardware*

Table 4
Service demands $D_{k,r}$ [ms] of the J2EE application model in [24]

Subnetwork	Queues	C1	C2	C3	C4	C5
AS	$m_{AS} = 9$	12.98	13.64	2.64	2.54	24.22
DB	$m_{DB} = 2$	5.32	5.18	1.24	1.04	17.07
CO	$m_{CO} = 1$	1.12	1.27	0.58	0.03	1.68

7.1 Case Study: a J2EE application

We continue the experimental evaluation considering a real application model which is prohibitive for current solution methods. The results provide intuition on how MoM improves the analysis of real-world computer systems. We consider the large-scale distributed J2EE service application recently modeled by Kounev et al. in [24]; we point to that paper for a description, and we limit here to give a synthetic overview.

The application runs on a two-tier architecture with a cluster of m_{AS} replicated application servers (AS) and a back-end composed of a dual-processor database server (DB). The workload is represented in terms of $R = 5$ classes, here labeled C1–C5, e.g., representing a new order placement or an order status query. We consider the most complex architecture among those evaluated in [24], which has $m_{AS} = 9$ clustered AS, and an overall number of $M = 12$ queues which also include a queue modeling communication overheads (CO). The delay varies with experiments, here we consider the case $Z_1 = \ldots = Z_4 = 2s$ for the first four classes and $Z_5 = 3s$ for the last workload class. The service demands for the five classes are given in Table 4. Following [24], we solve models corresponding to three workload profiles:

- Low load: $\vec{N} = (30, 10, 50, 40, 50)$
- Medium load: $\vec{N} = (50, 40, 100, 70, 100)$
- Heavy load: $\vec{N} = (100, 50, 150, 50, 200)$

Table 5
Service demands $D_{k,r}$ [ms] of the stress case with $N = 16660$ jobs

Queue	C1	C2	C3	C4	C5	C6	C7
J1	64	77	93	27	19	60	8
J2	68	70	14	22	32	2	64
J3	28	2	68	34	22	47	81
J4	35	97	69	15	82	83	10
J5	10	85	88	71	62	79	98

The storage requirements of the MVA exceeded the memory limit on all three models. Using the optimized MoM, we were able to solve the low load model in 0.9s using 1.6MB of memory. Similarly, the medium load case was solved in 4.7s with 2.3MB of memory, and the heavy load model in 14.1s using 3.4MB. These results further confirm the applicability of MoM on models practical interest.

7.2 Case Study: Stress Case 1

We now consider a model with 5 distinct queues labeled J1–J5, $R = 7$ workload classes labeled C1–C7, and 16660 jobs. The random demands are given in Table 5; the population vector is

$$\vec{N} = (10000, 5000, 1000, 500, 100, 50, 10).$$

On the same machine used before, the solution is obtained with the optimized MoM in $409s$. The computed normalizing constant is $G(\vec{N}) = 2.43756 \times 10^{37591}$. We also determined throughputs and queue-lengths for all classes and queues, e.g., the throughput of class 2 is $X_2(\vec{N}) = 3.89724 \times 10^{-3}$ jobs per ms, and the mean class 1 queue-length of $J1$ is $Q_{J1,1} = 9982.23$ jobs. During recursion, the memory occupation is monotonically increasing and reaches a maximal value of 407MB. The storage requirement of convolution, LBANC and MVA, would be approximately 10^{10}GB which is infeasible. All other existing methods, such as RECAL, have on this example larger storage requirements than MVA. As for the previous experiments, this example further stresses that the key computational advantage of MoM and the optimized MoM is the low space complexity.

8 Conclusion

We have introduced the Method of Moments (MoM), a new algorithm for the exact analysis of multiclass queueing networks. MoM addresses the classic problem of infeasibility of exact solution methods on large multiclass closed queueing networks. The proposed algorithm is based on the recursive evaluation of higher-order moments of queue-lengths instead of mean-values, which is the established approach follows by the MVA algorithm. We have shown that studying higher-order moments allows one to formulate a recursion with a number of steps that grows linearly in the total population. At each step, the number of evaluated constants is fixed and independent of the number of jobs in the model. Numerical experiments and case studies have shown that MoM remarkably outperforms MVA, especially for the storage requirements.

9 Acknowledgement

This work has been supported by the Imperial College Junior Research Fellowship. The author wishes to thank the anonymous referees for their very detailed comments that helped in improving both the presentation and the technical content of this paper. The author also thanks Michalis Makaronidis for useful comments, careful proof-reading of this manuscript, and for the prototype Java implementation of the MoM algorithm.

References

[1] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, 1975.

[2] A. Bertozzi and J. McKenna. Multidimensional residues, generating functions, and their application to queueing networks. *SIAM Review*, 35(2):239–268, 1993.

[3] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains*. John Wiley and Sons, 1998.

[4] J. P. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Comm. of the ACM*, 16(9):527–531, 1973.

[5] G. Casale. An Efficient Algorithm for the Exact Analysis of Multiclass Queueing Networks with Large Population Sizes. In *Proc. of ACM SIGMETRICS/IFIP Performance*, 169 – 180, Jun 2006.

[6] G. Casale. On single-class load-dependent normalizing constant equations. In *Proc. of the 3rd Conf. on Quantitative Evaluation of Systems (QEST)*, 333–342. IEEE Press, 2006.

[7] G. Casale. CoMoM: Efficient class oriented evaluation of multiclass performance models. *IEEE Trans. on Software Engineering*, 35(2):162-177, Mar/Apr 2009.

[8] G. Casale. A Generalized Method of Moments for Closed Queueing Networks. *Performance Evaluation*, 68(2), 180-200, Feb 2011.

[9] K. M. Chandy and D. Neuse. Linearizer: A heuristic algorithm for queuing network models of computing systems. *Comm. of the ACM*, 25(2):126–134, 1982.

[10] K. M. Chandy and C. H. Sauer. Computational algorithms for product-form queueing networks models of computing systems. *Comm. of the ACM*, 23(10):573–583, 1980.

[11] G. L. Choudhury, K. K. Leung, and W. Whitt. Calculating normalization constants of closed queuing networks by numerically inverting their generating functions. *Journal of the ACM*, 42(5):935–970, 1995.

[12] D. J. A. Cohen. *Basic Techniques of Combinatorial Theory.* John Wiley and Sons, 1978.

[13] A. E. Conway, E. de Sousa e Silva, and S. S. Lavenberg. Mean value analysis by chain of product form queueing networks. *IEEE Trans. on Computers*, 38(3):432–442, 1989.

[14] A. E. Conway and N. D. Georganas. RECAL - A new efficient algorithm for the exact analysis of multiple-chain closed queueing networks. *Journal of the ACM*, 33(4):768–791, 1986.

[15] P. Cremonesi, P. J. Schweitzer, and G. Serazzi. A unifying framework for the approximate solution of closed multiclass queuing networks. *IEEE Trans. on Computers*, 51:1423–1434, 2002.

[16] E. de Sousa e Silva and S. S. Lavenberg. Calculating joint queue-length distributions in product-form queueing networks. *Journal of the ACM*, 36(1):194–207, 1989.

[17] E. de Sousa e Silva and R. R. Muntz. Queueing networks: Solutions and applications. Technical Report CSD-890052, CS Dep. - UCLA, September 1989.

[18] P. J. Denning and J. P. Buzen. The operational analysis of queueing network models. *ACM Computing Surveys*, 10(3):225–261, 1978.

[19] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.).* Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[20] P. G. Harrison and S. Coury. On the asymptotic behaviour of closed multiclass queueing networks. *Performance Evaluation*, 47(2):131–138, 2002.

[21] P. G. Harrison and T. T. Lee. A new recursive algorithm for computing generating functions in closed queueing networks. In *Proc. of IEEE MASCOTS Symposium*, pages 223–230. IEEE Press, 2004.

[22] A. Heindl and A. van de Liefvoort. Moment conversions for discrete distributions. In *Proc. of PMCCS-6 workshop*, 2003.

[23] H. Kobayashi. Application of the Diffusion Approximation to Queueing Networks I: Equilibrium Queue Distributions. In *Journal of the ACM*, pages 316–328, 1974.

[24] S. Kounev and A. Buchmann. Performance modeling and evaluation of large-scale J2EE applications. In *Proc. of CMG Conference*, pages 273–283, 2003.

[25] K.W. Ross, D.H.K. Tsang and Jie Wang. Monte Carlo summation and integration applied to multiclass queueing networks. *Journal of the ACM*, 41(6):1110–1135, 1994.

[26] S. Lam. Dynamic scaling and growth behavior of queueing network normalization constants. *Journal of the ACM*, 29(2):492–513, 1982.

[27] S. S. Lavenberg. *Computer Performance Modeling Handbook*. Academic Press, 1983.

[28] S. S. Lavenberg. A perspective on queueing models of computer performance. *Performance Evaluation*, 10(1):53–76, 1989.

[29] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance*. Prentice-Hall, 1984.

[30] J. McKenna and D. Mitra. Asymptotic expansions and integral representations of moments of queue lengths in closed markovian networks. *Journal of the ACM*, 31(2):346–360, April 1984.

[31] D. Menasce and V. A. F. Almeida. *Capacity Planning for Web Performance: Metrics, Models, and Methods*. Prentice Hall, 1998.

[32] D. A. Menascé, H. Ruan, and H. Gomaa. QoS management in service-oriented architectures. *Performance Evaluation*, 64(7-8):646–663, 2007.

[33] R. D. Nelson. The mathematics of product form queuing networks. *ACM Computing Surveys*, 25(3):339–369, 1993.

[34] M. Reiser and H. Kobayashi. Queueing networks with multiple closed chains: Theory and computational algorithms. *IBM J. Res. Dev.*, 19(3):283–294, 1975.

[35] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queueing networks. *Journal of the ACM*, 27(2):312–322, 1980.

[36] A. Schonhage and V. Strassen. Schnelle multiplikation grosser zahlen. *Computing*, 7:281–292, 1971.

[37] P. J. Schweitzer. Approximate analysis of multiclass closed networks of queues. In *Proc. of the Int'l Conf. on Stoch. Control and Optim.*, pages 25–29, Amsterdam, 1979.

[38] R. Suri. Robustness of Queuing Networks Formulas. *Journal of the ACM*, 30(3):564–594, 1984.

[39] 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*, pages 291–302. ACM Press, 2005.

[40] J. Zahorjan, K. C. Sevcik, D. L. Eager, and B. Galler. Balanced job bound analysis of queueing networks. *Comm. of the ACM*, 25(2):134–141, 1982.

A Proof of Theorem 2

From (8) and from the BCMP formula (1) of the equilibrium probability $\Pr(\vec{S})$ we rewrite (8) as

$$\mathbb{E}[\Delta\vec{m}, \vec{N}] = G^{-1}(\vec{N}) \sum_{\vec{S}\in\mathcal{S}(\vec{N})} \left(\prod_{r=1}^{R} \frac{Z_r^{n_{0,r}}}{n_{0,r}!} \right) \prod_{k=1}^{M} G_k(\vec{n}_k), \tag{A.1}$$

where

$$G_k(\vec{n}_k) = \Delta m_k! \binom{n_k + \Delta m_k}{n_k} \left(n_k! \prod_{r=1}^{R} \frac{D_{k,r}^{n_{k,r}}}{n_{k,r}!} \right).$$

We prove that the summation in (A.1) is $G(\Delta\vec{m}, \vec{N})$ by showing that $G_k(\vec{n}_k)$ is the normalizing constant of a subnetwork composed by queue k together with its Δm_k replicas and with population vector $\vec{n}_k = (n_{k,1}, n_{k,2}, \ldots, n_{k,R})$. After proving this fact, the final result in (9) follows straightforwardly by convolution [4, 34]; in fact, the convolution of the G_k, $1 \leq k \leq M$, is equivalent to the convolution of the normalizing constants of M subnetworks.

Recalling that the class r throughput of a queueing network with population $\vec{N}$ may be written as [34]

$$X_r(\vec{n}_k) = \frac{G(\vec{n}_k - \vec{1}_r)}{G(\vec{n}_k)}, \tag{A.2}$$

if this formula is applied to the individual subnetworks with normalizing constants $G_k(\vec{n}_k)$ then we have

$$G_k(\vec{n}_k) = X_r^{-1}(\vec{n}_k) G_k(\vec{n}_k - \vec{1}_r). \tag{A.3}$$

However, from the analysis of balanced systems [40] it follows that for a model composed of $\Delta m_k + 1$ identical queues we have

$$X_r(\vec{n}_k) = \frac{n_{k,r}}{D_{k,r}\,(\Delta m_k + n_k)}.$$

The formula for $G_k(\vec{n}_k)$ thus follows by unfolding the recurrence (A.3) for all classes up to the obvious termination conditions $G_k(\vec{0}) = 1$.

B Analysis of Degenerate Models

As mentioned in Section 4, degenerate models exists for which $\det(\mathbf{A}(\vec{N})) = 0$ and thus (7) does not apply. We categorize these models in groups and we propose strategies to address the problem in each case.

Sparse routing matrices

A second important class of degeneracies is that of networks with one or more $D_{k,r} = 0$. The non-singularity of $\mathbf{A}(\vec{N})$ can be readily restored by removing all columns composed of zero elements and the related variables in $V(\vec{N})$. Note that this can significantly reduce computational costs for models with sparse routing where several demands are set to zero.

Generic singularity

The third and last class includes all other singular models that arise due to particular choices of the demands $D_{k,r}$. It is quite difficult to encounter such problem if the $D_{k,r}$ are measured from real applications, where the measurement errors hardly allow demands that are perfectly linear dependent, and thus singularity often can be resolved by considering additional significant digits [8] in $D_{k,r}$. However, to achieve maximum generality we propose three possible solutions; the first two are exact methods, the last one is an approximation where *arbitrarily small* error can be achieved by increasing computational costs.

The first approach applies when a single class s has some demands $D_{k,s}$ that are responsible for singularity, and consists in re-labeling class indices such that class s becomes class R. This indeed solves the problem, since the entries of $\mathbf{A}(\vec{N})$ never depend on the demands of class R which appear in $\mathbf{B}(\vec{N})$ only.

In the second approach, if demands of several classes are responsible for singularity it is possible to compute recursively part of the basis $V(\vec{N})$ using traditional, more expensive, recursive schemes, such as LBANC or RECAL. We point to [7] for a description on how an hybrid algorithm of this type could be defined for degenerate models.

Finally, a practical technique which does not require additional implementations consists in perturbing the service demands $D_{k,r}$ responsible for singularity by arbitrary small quantities $\epsilon_{k,r}$. In traditional inexact linear solvers, this approach

[8] We recall that, since MoM and the optimized MoM have to be implemented in multi-precision algebra (see Sect. 4), there is virtually no limit to the number of significant digits that can be considered in computations.

would not be viable, since perturbations are lower bounded by machine accuracy and thus may not always be small enough; more importantly, they may produce ill-conditioned linear systems. However, these limitations do not apply to exact multi-precision algebra, where the main drawback is that the additional digits required to represent the perturbation just add computational overhead. By definition of $D_{k,r}$, it is also intuitive that the perturbed network with servers having speeds $D_{k,r} + \epsilon_{k,r}$ performs arbitrarily close to the reference model for sufficiently small $\epsilon_{k,r}$'s, which is not difficult to prove isolating the components of $G(\vec{N})$ affected by the perturbation. For instance, recall that the utilization $U_j(\vec{N}) = \sum_{r=1}^{R} U_{j,r}(\vec{N})$ of queue j may be computed as [34]

$$U_j(\vec{N}) = 1 - \frac{G^{-j}(\vec{N})}{G(\vec{N})},$$

where $G^{-j}(\vec{N})$ is the normalizing constant of a model with queue j less. Setting $D_{j,1} + \epsilon_{j,1}$, then the perturbed utilization is

$$\widehat{U}_j(\vec{N}) = 1 - \frac{\widehat{G}^{-j}(\vec{N})}{\widehat{G}(\vec{N})} = 1 - \frac{G^{-j}(\vec{N})}{\widehat{G}(\vec{N})},$$

where $\widehat{G}^{-j}(\vec{N}) = G^{-j}(\vec{N})$ being independent of queue j service demands, and

$$\widehat{G}(\vec{N}) = \sum_{\vec{S} \in \mathcal{S}(\vec{N})} \left(\prod_{r=1}^{R} \frac{Z_r^{n_{0,r}}}{n_{0,r}!} \right) \left[n_j! \left(\frac{(D_{j,1} + \epsilon_{j,1})^{n_{j,1}}}{n_{j,1}!} \prod_{r=2}^{R} \frac{D_{j,r}^{n_{j,r}}}{n_{j,r}!} \right) \prod_{k \neq j} n_k! \prod_{r=1}^{R} \frac{D_{k,r}^{n_{k,r}}}{n_{k,r}!} \right].$$

Note that, by definition of normalising constant, $|\widehat{G}(\vec{N}) - G(\vec{N})|$ is a continuous function that is monotonically decreasing with $|\epsilon_{j,1}|$, we have

$$\lim_{|\epsilon_{j,1}| \to 0} \widehat{G}(\vec{N}) = G(\vec{N}) \Rightarrow \lim_{|\epsilon_{j,1}| \to 0} \widehat{U}_j(\vec{N}) = 1 - \lim_{|\epsilon_{j,1}| \to 0} \frac{G^{-j}(\vec{N})}{\widehat{G}(\vec{N})} = U_j(\vec{N}),$$

which justifies the perturbation approach.

To provide intuition on the accuracy of the perturbations, we report a simple numerical example.

Example 1 *Consider a model with $M = 2$ queues, $R = 3$ classes, and demands $D_{1,1} = 80ms$, $D_{1,2} = 40ms$, $D_{1,3} = 7ms$ at queue 1, and $D_{2,1} = 20ms$, $D_{2,2} = 10ms$, $D_{2,3} = 13ms$ at queue 2. Let also $Z_1 = Z_2 = Z_3 = 1s$, and $N_1 = N_2 = N_3 = 15$. During the recursion on the second class, since*

$$\det\left(\begin{bmatrix} D_{1,1} & D_{1,2} \\ D_{2,1} & D_{2,2} \end{bmatrix} \right) = 0,$$

it is found that $\mathbf{A}_{1,1}$ includes singular blocks, and we cannot solve the model. The exact value, e.g., of the utilization of queue 2 would be $U_2(\vec{N}) = 0.7433$. Using the perturbation approach, with a perturbation $\epsilon_{1,2} = 0.1$ such that $D_{1,2} + \epsilon_{1,2} = 40.1$ we

restore singularity and obtain the estimate $\widehat{U}_1(\vec{N}) = 0.7431$*; similarly, for* $\epsilon_{1,2} = 0.01$ *we have* $D_{1,2} + \epsilon_{1,2} = 40.01$ *and* $\widehat{U}_1(\vec{N}) = 0.7433$ *which is the same as the exact value. In larger models a few tests are sufficient to size the perturbation, however these may be defined to a reasonable value by considering the relative difference with respect to the service demands.*

C. Algorithm 1 - *Generation of the matrices* $\mathbf{A}(\vec{N})$ *and* $\mathbf{B}(\vec{N})$ *in MoM.*

Let $\mathbf{A}(\vec{N})$ be square of order $\binom{M+R}{R}R$ with element a_{ij} on row $i \geq 1$ and column $j \geq 1$
Let $\mathbf{B}(\vec{N})$ be square of order $\binom{M+R}{R}R$ with element b_{ij} on row $i \geq 1$ and column $j \geq 1$
Initialize all a_{ij} and b_{ij} to zero
$i := 0$
for l **from** 0 **to** $R-1$
  if $l = R-1$
    forall $\Delta\vec{m}$ with M non-negative elements and $\sum_{k=1}^{M} \Delta m_k = R-1$
      for k **from** 1 **to** M */* Add CE for queue k */*
        $i := i+1$
        $j :=$ index of $G(\Delta\vec{m} + \vec{1}_k, \vec{N})$ in $V(\vec{N})$
        $a_{ij} := 1$
        $j :=$ index of $G(\Delta\vec{m}, \vec{N})$ in $V(\vec{N})$
        $a_{ij} := -1$
        for r **from** 1 **to** $R-1$
          $j :=$ index of $G(\Delta\vec{m} + \vec{1}_k, \vec{N} - \vec{1}_r)$ in $V(\vec{N})$
          $a_{ij} := -D_{kr}$
        end for
        $j :=$ index of $G(\Delta\vec{m} + \vec{1}_k, \vec{N} - \vec{1}_R)$ in $V(\vec{N} - \vec{1}_R)$
        $b_{ij} := D_{kR}$
      end for
    end forall
    forall $\Delta\vec{m}$ with M non-negative elements and $\sum_{k=1}^{M} \Delta m_k = R-1$
      for r **from** 1 **to** $R-1$ */* Add PC of class r */*
        $i := i+1$
        $j :=$ index of $G(\Delta\vec{m}, \vec{N})$ in $V(\vec{N})$
        $a_{ij} := N_r$
        $j :=$ index of $G(\Delta\vec{m}, \vec{N} - \vec{1}_r)$ in $V(\vec{N})$
        $a_{ij} := -Z_r$
        for k **from** 1 **to** M
          $j :=$ index of $G(\Delta\vec{m} + \vec{1}_k, \vec{N} - \vec{1}_r)$ in $V(\vec{N})$
          $a_{ij} := -(m_k + \Delta m_k)D_{kr}$
        end for
      end for
    end forall
  end if
  forall $\Delta\vec{m}$ with M non-negative elements and $\sum_{k=1}^{M} \Delta m_k = l$
    for s **from** 0 **to** $R-1$ */* Add PC of class R */*
      $i := i+1$
      $j :=$ index of $G(\Delta\vec{m}, \vec{N} - \vec{1}_s)$ in $V(\vec{N})$, where $\vec{N} - \vec{1}_0 = \vec{N}$
      $a_{ij} := N_R$
      $b_{ij} := Z_R$
      for k **from** 1 **to** M
        $j :=$ index of $G(\Delta\vec{m} + \vec{1}_k, \vec{N} - \vec{1}_s - \vec{1}_R)$ in $\vec{V}(\vec{N} - \vec{1}_R)$
        $b_{ij} := (m_k + \Delta m_k)D_{kR}$
      end for
    end for
  end forall
end for