CoMoM: Efficient Class-Oriented Evaluation of Multiclass Performance Models

Giuliano Casale, *Member, IEEE*

***Abstract*— We introduce the Class-oriented Method of Moments (CoMoM), a new exact algorithm to compute performance indexes in closed multiclass queueing networks. Closed models are important for performance evaluation of multi-tier applications, but when the number of service classes is large they become too expensive to solve with exact methods such as Mean Value Analysis (MVA). CoMoM addresses this limitation by a new recursion that scales efficiently with the number of classes.**

Compared to the MVA algorithm, which recursively computes mean queue-lengths, CoMoM carries on in the recursion also information on higher-order moments of queue-lengths. We show that this additional information greatly reduces the number of operations needed to solve the model and makes CoMoM the best-available algorithm for networks with several classes.

We conclude the paper by generalizing CoMoM to the efficient computation of marginal queue-length probabilities, which finds application in the evaluation of state-dependent attributes such as energy consumption or quality-of-service metrics.

***Index Terms*— Multiclass applications, queueing networks, performance modeling, exact analysis, state-dependent attributes**

I. Introduction

Closed product-form queueing networks are popular quantitative models for performance evaluation and capacity planning of multiclass systems, such as multi-tier applications processing several types of transactions [2], [7], [13], [18], [22]. These systems are best modeled as closed due to the presence in real Web servers, application servers, and database servers of limits on the maximum number of concurrent user sessions [22]. We here focus on the probabilistic evaluation of models with several service classes, a case that cannot be handled efficiently with algorithms such as the widely-used Mean Value Analysis (MVA) [20], but which is often found in models of real applications. The main contribution of this paper is a new solution technique, called the Class-oriented Method of Moments (CoMoM), which can solve efficiently queueing networks with several classes.

The probabilistic evaluation of multiclass models is hard because the closed-form expressions of the state probabilities found in [2] include a normalizing constant that is very expensive to evaluate [3], [6], [9], [11], [19]. In particular, models with several classes are often infeasible since both MVA and multiclass Convolution [19] have requirements that grow exponentially in the number of classes. The RECAL class recursion [9] is tailored to models with several classes, but the limited scalability of this technique with respect to the population size makes it applicable only to models with few tens of requests. Recent studies on the analytical inversion of the generating function of the normalizing constant have led to the RGF class recursion [11], [12], which is always more efficient than RECAL, but still remains too expensive for evaluating models with hundreds or thousands of requests.

In this paper, we dramatically improve over the requirements of the above algorithms by a new exact approach called the Class-oriented Method of Moments (CoMoM). Compared to the MVA, which recursively computes mean queue-lengths, CoMoM is based on a recursive evaluation of higher-order moments of queue-lengths. At each recursive step, CoMoM considers a new model with increased population and solves a linear system of equations to update the value of the higher-order moments of queue-lengths. This approach minimizes the number of steps required to complete execution since they grow only linearly with the total population size. Finally, CoMoM returns the normalizing constant from the computed queue-length moments. By comparison with existing methods, we find that *CoMoM is the best-available algorithm for evaluating queueing networks with several classes*. For example, we show a model of a real J2EE application where CoMoM is several orders of magnitude faster and more memory-efficient than MVA. We also compare CoMoM with the Method of Moments (MoM) [3], [4], which solves the model by recursively computing binomial moments of queue-lengths [4]. The analysis reveals that the combinatorial characteristics of the data structures used by CoMoM and MoM are similar, but CoMoM scales much better on models with several classes, while MoM is preferable on models with many queues.

We conclude the paper by generalizing CoMoM to the efficient computation of marginal queue-lengths, which are needed to evaluate state-dependent indexes such as energy consumption or quality-of-service metrics. This analysis has never been performed efficiently by analytical methods, but we find that a generalization of CoMoM, called Pro-CoMoM, can perform the computation by an efficient recursive scheme.

This paper is organized as follows. After giving required definitions in Section II, we introduce the basic ideas behind CoMoM in Section III using an illustrative example. A general definition of the algorithm is given in Section IV, followed in Section V by a detailed analysis of its numerical and linear algebraic properties. Computational costs and case studies validating the efficiency of CoMoM are given in Section VI. In Section VII, we derive the Pro-CoMoM algorithm for computing marginal queue-length probabilities. A comparison of CoMoM with MoM is given in Section VIII. Conclusive remarks are reported in Section IX. Appendix A describes a hybrid algorithm for singular models. We point the interested reader to [5] and to the author's homepage for additional material on CoMoM.

II. Background

A summary of the notation defined in this section is given in Table I. We focus on the class of product-form models introduced in [2] and we consider a closed multiclass queueing network with

Giuliano Casale (*casale@cs.wm.edu*) is with the College of William & Mary, Department of Computer Science, 23187-8795, Williamsburg VA, USA. Homepage: `http://www.cs.wm.edu/∼casale`

constant population $\vec{N} \equiv (N_1, \ldots, N_r, \ldots, N_R)$, $N = \sum_{r=1}^{R} N_r$, where R is the number of service classes and N_r is the number of jobs (henceforth also called requests) of class r. The N jobs visit M constant-rate queues according to a state-independent routing scheme. The mean service demand of a class-r job at queue k is denoted by $D_{k,r}$, $1 \leq k \leq M$, $1 \leq r \leq R$, which is computed as a product between a mean service time and a mean visit ratio [17]. Z_r is the think time of class-r jobs before re-entering the network after completion; jobs in think state wait in a delay station indexed by $k = 0$. Scheduling at the queues can be first-come first-served, processor sharing or last-come first-served preemptive resume [2].

The equilibrium probability of the network being in state $\vec{n} = (n_{0,1}, \ldots, n_{k,r}, \ldots, n_{M,R})$, $\sum_{k=0}^{M} n_{k,r} = N_r$, $1 \leq r \leq R$, is [2]

$$P(\vec{n}|\vec{N}) = \frac{1}{G(\vec{N})} \left(\prod_{r=1}^{R} \frac{Z_r^{n_{0,r}}}{n_{0,r}!} \right) \left(\prod_{k=1}^{M} n_k! \prod_{r=1}^{R} \frac{D_{k,r}^{n_{k,r}}}{n_{k,r}!} \right), \quad (1)$$

where $n_{k,r}$ is the number of class-r jobs in station k, $n_k = \sum_{r=1}^{R} n_{k,r}$, and $G(\vec{N})$ is a normalizing constant. *Henceforth, we focus on the problem of computing $G(\vec{N})$ and related quantities such as marginal queue-lengths obtained by summation of (1).*

Among the recurrence equations for computing normalizing constants presented in the literature, we use the *convolution expression* (CE) [6], [19]

$$G(\vec{1}_k, \vec{N}) = G(\vec{N}) + \textstyle\sum_{r=1}^{R} D_{k,r} G(\vec{1}_k, \vec{N} - \vec{1}_r), \quad (2)$$

for arbitrary $1 \leq k \leq M$, where the notation $\vec{1}_t$ stands for a vector composed of all zeros except for a one in the t-th position. Here, $G(\vec{1}_k, \vec{N} - \vec{1}_r)$ is the normalizing constant of a model with a class-r job less and augmented with a *replica* of queue k. A replica is an additional queue having service demands $D_{k,r}$, $1 \leq r \leq R$, identical to the demands of queue k in the original model. For consistency we define $G(\vec{N}) = G(\vec{0}, \vec{N})$, where $\vec{0} = (0, 0, \ldots, 0)$. Equation (2) can be seen as a recursive formula for computing mean queue-lengths because the constants $G(\vec{1}_k, \vec{N})$ and $G(\vec{1}_k, \vec{N} - \vec{1}_r)$ may be interpreted as un-normalized queue-length values, see [16] and (4) given later.

We also use the *population constraint* (PC)

$$N_r G(\vec{N}) = Z_r G(\vec{N} - \vec{1}_r) + \textstyle\sum_{k=1}^{M} D_{k,r} G(\vec{1}_k, \vec{N} - \vec{1}_r), \quad (3)$$

for $1 \leq r \leq R$. Equation (3) is equivalent to the RECAL recurrence equation[1] [3], [9].

We conclude observing that the mean class-r throughput $X_r(\vec{N})$ and the mean class-r queue-length $Q_{k,r}$ of queue k are computed from normalizing constants respectively as [19]

$$X_r(\vec{N}) = \frac{G(\vec{N} - \vec{1}_r)}{G(\vec{N})}, \quad Q_{k,r}(\vec{N}) = D_{k,r} \frac{G(\vec{1}_k, \vec{N} - \vec{1}_r)}{G(\vec{N})}. \quad (4)$$

Response times and utilizations are easily obtained from throughputs and mean queue-lengths using Little's Law [17]. We point to comprehensive material in [17] for additional background on multiclass product-form queueing network models.

[1]Equation (3) can also be regarded as an un-normalized version of Little's Law [3] when used to compute the number of jobs N_Z in the delay server. This gives the expression $N_Z = N - \sum_{k=1}^{M} Q_{k,r}(\vec{N}) = Z_r X_r(\vec{N})$. By (4) it is easy to verify that the last equivalence reduces to (3).

TABLE I
SUMMARY OF QUEUEING NETWORK NOTATION

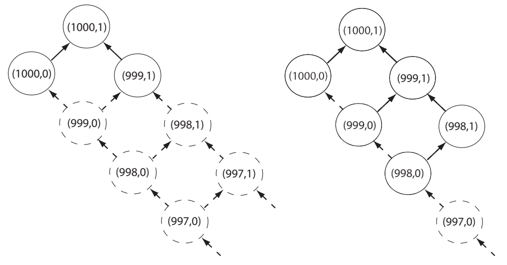
Symbol	*Description*
k, j	Queue indexes (range $1 \ldots M$)
r, s	Class indexes (range $1 \ldots R$)
$\vec{1}_t$	Vector of zeros with a one in the t-th position
$D_{k,r}$	Mean service demand of a class-r request at queue k
$G(\vec{N})$	Normalizing constant
$G(\vec{1}_k, \vec{N})$	Norm. constant of model with a replica of queue k more
M	Number of queues
$\vec{N}$	Population vector
N_r	Number of jobs (also called requests) of class r
N	Total number of jobs in the network ($N = \sum_r N_r$)
$P(\vec{n}\|\vec{N})$	Equilibrium probability of state $\vec{n} = (n_0, n_1, \ldots, n_M)$
R	Number of service classes
Z_r	Think time of class-r requests

III. CoMoM Illustrative Example

To build intuition on the CoMoM approach and compare it with the standard MVA recursion[2], we first show how to solve by CoMoM a small network with $R = 2$ classes and $M = 2$ queues. We focus on the computation of the normalizing constant for the population $(1000, 1000)$ and we initially discuss the critical intermediate recursive step for $\vec{N} = (1000, 1)$, which is the first population that cannot be evaluated by basic single class algorithms. A graphical representation of the recursive structure of MVA in a neighborhood of $\vec{N} = (1000, 1)$ is given in Figure 1(a). MVA reaches $(1000, 1)$ in a bottom-up approach, evaluating step-by-step models with populations summing to $N = N_1 + N_2 = 1, \ldots, 999, 1000, 1001$. The recursive step of MVA for $N = 1001$ is depicted in Figure 1(a), see the caption for a description. Figure 1(b) illustrates the approach of the CoMoM recursion. CoMoM computes $\vec{N} = (1000, 1)$ directly from the solutions of the single class models with populations $(1000, 0)$, $(999, 0)$, and $(998, 0)$, which avoids evaluating almost half of the populations, i.e., the states $(1, 1), (2, 1), \ldots, (997, 1)$ are all *skipped* by CoMoM. On larger models, while the number of states evaluated by MVA grows *combinatorially* as the set of all possible populations component-wise less than or equal to $\vec{N}$, the set of states spanned by CoMoM grows *linearly* with the total population size N, because the algorithm can recursively reapply the same step depicted in Figure 1(b) for all populations. This efficient schema is made possible by simultaneously evaluating the CEs and PCs (2)-(3) in a linear system of equations.

In order to understand how the CoMoM technique performs a recursive step, let us consider the state $\vec{N} = (1000, 1)$ and assume that we want to compute the set of normalizing constants $\lambda(1000, 1) = \{G(\vec{1}_1, 1000, 1), G(\vec{1}_2, 1000, 1), G(1000, 1)\}$, i.e., the normalizing constant $G(1000, 1)$ of the model and, according to (4), the un-normalized mean queue-lengths of stations 1 and 2. Let us also assume to know from the previous recursive steps the similarly-defined vectors $\lambda(1000, 0)$, $\lambda(999, 0)$, $\lambda(998, 0)$, which refer to models with a single class of jobs. These vectors are easily computed by efficient single-class algorithms. The CoMoM recursive step consists in the following recursive computation of the vectors λ.

[2]In order to perform a comparison using the normalizing constant approach, here we implicitly refer to the LBANC algorithm, i.e., the *un-normalized* version of MVA which computes normalizing constants instead of mean indexes (see [6], [16]).



(a) MVA (b) CoMoM

Fig. 1. **Illustrative Example**. *Recursive structure of MVA (Figure (a)) and CoMoM (Figure (b)) in a neighborhood of $\vec{N} = (1000, 1)$. Dashed states indicate populations of models solved previously by the recursion and not involved in the current recursive step; solid arrows indicate relations (e.g., (2)-(3)) that are exploited by the algorithm in the current step of the recursion.*

Observation 1: Using the PC in equation (3) for $r = 2$, we can immediately compute $G(\vec{N}) = G(1000, 1)$ from the normalizing constants in $\lambda(1000, 0)$ which are assumed known. Thus, the unknowns that we still need to determine $\lambda(1000, 1)$ are

$$\{G(\vec{1}_1, 1000, 1), G(\vec{1}_2, 1000, 1)\} \subset \lambda(1000, 1). \quad (5)$$

Observation 2: We compute the constants (5) by the CEs (2). In fact, applying the PCs to compute $G(\vec{1}_1, 1000, 1)$ or $G(\vec{1}_2, 1000, 1)$ would require the availability of constants with two queue replicas that are not included[3] in the vectors λ. Instead, the CEs can be used to evaluate $\lambda(1000, 1)$ by including in the analysis also the constants in the vector $\lambda(999, 1)$ which appear in the summation of (2) for $r = 1$. Note that also $\lambda(999, 1)$ is not known in advance and thus we have to expand the set of unknowns as

$$\{G(\vec{1}_1, 1000, 1), G(\vec{1}_2, 1000, 1)\} \subset \lambda(1000, 1), \quad (6)$$

$$\{G(\vec{1}_1, 999, 1), G(\vec{1}_2, 999, 1)\} \subset \lambda(999, 1), \quad (7)$$

where $G(999, 1)$ is omitted being easily computed by the PC for $r = 2$ from the known constants in $\lambda(999, 0)$.

Observation 3: The unknowns (6) are related to the unknowns (7) by two CEs for $k = 1, 2$, and the other constants which appear in these CEs are all known. The unknowns (7) are related to each other by the PC for $r = 1$. Therefore, by extending the set of unknowns also to the $\lambda(999, 1)$ vector we have moved from the initial problem (5) with $p = 2$ unrelated unknowns to a new problem with $p' = 4$ unknowns (6)-(7), but which are related by $q' = 3$ equations (2)-(3). This reduces the degrees of freedom df of the analysis from $df = p = 2$ of (5) to $df = p' - q' = 4 - 3 = 1$ of (6)-(7). In other words, *the extension of the analysis to the vector* $\lambda(999, 1)$ *in (7) has reduced the degrees of freedom in determining the normalizing constants*. This is a counter-intuitive property of multiclass models, since we are stating that increasing the difficulty of the analysis by adding to the unknowns also the constants in $\lambda(999, 1)$ has the opposite effect of increasing our information about the system.

Observation 4: Using the last observation, we can extend the analysis to include as unknown also the vector $\lambda(998, 1)$ in addition to $\lambda(1000, 1)$ and $\lambda(999, 1)$. Using the same arguments given above, we conclude that the new problem has $p'' = 6$ unknowns related by $q'' = 6$ linear equations (2)-(3) and the degrees of freedom become $df = p'' - q'' = 0$, which allows the exact computation of all unknowns by inverting a linear system of q'' equations (2)-(3). Note that the degrees of freedom are reduced under the *implicit assumption* that the q'' equations are linearly independent, a fact that in general depends on the specific values of the service demands $D_{k,r}$. We discuss cases of linear dependence between CEs and PCs in Section V-C, while in the rest of the paper we focus on non-degenerate cases.

Summarizing, starting from the vectors $\lambda(1000, 0)$, $\lambda(999, 0)$, and $\lambda(998, 0)$, CoMoM has computed $\lambda(1000, 1)$, $\lambda(999, 1)$, and $\lambda(998, 1)$. These vectors provide the new initial conditions for recursively applying the same scheme to models with more than one job in class 2. With CoMoM we can easily reach the final model solution for the population $(1000, 1000)$ after 1000 recursive steps on class 2. Conversely, MVA in Figure 1 still requires 990000 evaluations to reach $(1000, 1000)$. Thus, CoMoM can solve problems that are much harder than the ones that can be solved efficiently by the MVA algorithm. A general definition of the CoMoM algorithm for models with arbitrary populations, number of queues and classes is given below.

IV. The Class-Oriented Method of Moments

Following the illustrative example, we give a general definition of CoMoM. The theorems and corollaries of this section are organized as follows. We first define the *basis* of CoMoM, which is a set of normalizing constants that summarizes the results of the past recursive steps performed by the algorithm and from which we can computer performance indexes by (4). We show later in the section that the basis of CoMoM can be seen as a set of higher-order moments of queue-lengths. In Theorem 1, we explain how a basis may be computed recursively. Finally, in Corollary 1 we give evidence that our definition of basis is parsimonious, i.e., CoMoM uses the minimum possible information needed to perform its simple recursion.

Definition 1: Consider a model with M queues, R classes, and a population of $\vec{N}$ jobs, $N_r \geq 1$, $1 \leq r \leq R$. The basis $\Lambda(\vec{N})$ *of CoMoM is defined as the set*

$$\Lambda(\vec{N}) = \lambda(\vec{0}) \cup \lambda(\vec{1}_1) \cup \ldots \cup \lambda(\vec{1}_M), \quad (8)$$

where $\lambda(\cdot) = \{G(\cdot, \vec{N} - \vec{n}) \mid \sum_{r=1}^{R-1} n_r \leq M \wedge n_R = 0\}$ includes the normalizing constants of all models with population vectors having up to M jobs less than $\vec{N}$ and where these jobs are chosen among the first $R - 1$ service classes[4].

The above definition of $\Lambda(\vec{N})$ always makes available enough CEs and PCs to reduce the degrees of freedom df of the analysis to zero for any value of M, R and $\vec{N}$. This statement is proved in the next theorem, which is the main result of this paper.

Theorem 1: The basis (8) satisfies the linear matrix recurrence

$$\bar{\mathbf{A}}(\vec{N})\Lambda(\vec{N}) = \bar{\mathbf{B}}(\vec{N})\Lambda(\vec{N} - 1_R), \quad (9)$$

where $\bar{\mathbf{A}}(\vec{N})$ and $\bar{\mathbf{B}}(\vec{N})$ are square matrices of order $card(\Lambda(\vec{N})) = (M+1)\binom{M+R-1}{M}$, defined by all equations (2)-(3) that relate only constants in $\Lambda(\vec{N})$ and $\Lambda(\vec{N} - 1_R)$.

[3]It is interesting to note that the MoM recursive step in [3] differs from the CoMoM step described here because the λ vector is modified to include also constants with two or more queue replicas. This makes the MoM recursion significantly different from the CoMoM recursion. We point to [3], [4] and Section VIII for additional details.

[4]By definition, for populations $\vec{N} - \vec{n}$ with one or more negative components, the normalizing constant is equal to zero; normalizing constants of models with population $\vec{0}$ are instead equal to one. Thus, in models where some populations have $N_r \leq M$, some constants of (8) are immediately computed as one or zero.

Proof: The normalizing constants in $\lambda(\vec{0}) \subseteq \Lambda(\vec{N})$ are immediately computed from $\Lambda(\vec{N}-1_R)$ using the PC for $r = R$. The remaining normalizing constants $\lambda(\vec{1}_1) \cup \ldots \cup \lambda(\vec{1}_M)$ are $M\sum_{m=0}^{M}\binom{m+(R-1)-1}{m} = M\binom{M+R-1}{M}$, where we have observed that the sets $\lambda(\cdot)$ are uniquely defined by combinations with repetition of $m \in \{0,1,2,\ldots,M\}$ jobs chosen among $R-1$ classes. Observing that each PC requires to add a queue replica and remove a job with respect to the model in the left-hand side of (3), we can define $R-1$ PCs for each constant in $\{G(\vec{0},\vec{N}-\vec{n}) \mid \sum_{r=1}^{R-1} n_r \leq M-1 \wedge n_R = 0\} \subset \Lambda(\vec{N})$, which are thus in number $(R-1)\binom{M+R-2}{M-1}$. For each normalizing constant in this subset, we can also define M CEs, one for each possible queue k replica to be added to the network, bringing the total number of available equations to $(M+R-1)\binom{M+R-2}{M-1} = M\binom{M+R-1}{M}$, which is exactly the cardinality of $\lambda(\vec{1}_1)\cup\ldots\cup\lambda(\vec{1}_M)$. Therefore, the matrices $\bar{\mathbf{A}}(\vec{N})$ and $\bar{\mathbf{B}}(\vec{N})$ defined by all the PCs and CEs relating only the constants in $\Lambda(\vec{N})$ and $\Lambda(\vec{N}-1_R)$ are both square of order $(M+1)\binom{M+R-1}{M}$, and this also includes the PCs used to compute $\lambda(\vec{0}) \subseteq \Lambda(\vec{N})$. ■

We now examine in the next corollary the minimality of the definition of the basis $\Lambda(\vec{N})$.

Corollary 1: The basis $\Lambda(\vec{N})$ has the minimal size to let $\bar{\mathbf{A}}(\vec{N})$ and $\bar{\mathbf{B}}(\vec{N})$ be always square on models with arbitrary number of queues M, number of classes R, and populations $\vec{N}$.

Proof: Suppose to reduce the size of $\Lambda(\vec{N})$ by defining $\lambda(\cdot) = \{G(\cdot,\vec{N}-\vec{n}) \mid \sum_{r=1}^{R-1} n_r \leq C \wedge n_R = 0\}$ for some integer $0 \leq C < M$, then the size of the basis would be $(M+1)\binom{C+R-1}{C}$, and the number of available equations would drop to $(M+R-1)\binom{C+R-2}{C-1}$. Equating the number of unknowns and equations to have both $\bar{\mathbf{A}}(\vec{N})$ and $\bar{\mathbf{B}}(\vec{N})$ at least square, we get $(M+R-1)\binom{C+R-2}{C-1} = (M+1)\binom{C+R-1}{C}$, which simplifies to $(M+R-1)C = M(C+R-1)$ that is linear in C and thus has the unique solution $C = M$ that proves the minimality of (8). ■

The Class-oriented Method of Moments (CoMoM) algorithm follows from Theorem 1 by noting that, if $\bar{\mathbf{A}}(\vec{N})$ is an invertible matrix, then $\Lambda(\vec{N})$ is computed recursively as[5]

$$\Lambda(\vec{N}) = \bar{\mathbf{A}}^{-1}(\vec{N})\bar{\mathbf{B}}(\vec{N})\Lambda(\vec{N}-1_R). \qquad (10)$$

Note that (10) can be implemented in several ways, typically avoiding inversion of $\bar{\mathbf{A}}(\vec{N})$, see Section V.B for a discussion. Singularity of $\bar{\mathbf{A}}(\vec{N})$ is also possible depending on the specific values of the demands $D_{k,r}$, we discuss the implications for CoMoM in Section V-C. CoMoM pseudo-code and a computational procedure for generating the matrices $\bar{\mathbf{A}}(\vec{N})$ and $\bar{\mathbf{B}}(\vec{N})$ are reported in the technical report [5]. We also remark that an effective initialization of the recursion (10) can be done by an hybrid algorithm. That is, focusing on the un-normalized version of MVA which makes use of normalizing constants instead of mean-values, i.e., the LBANC algorithm [6], [16], we note that the set of normalizing constants in $\Lambda(N_1,0,\ldots,0)$ is exactly the set of constants computed by LBANC when evaluating the single-class populations $N_1, N_1-1,\ldots,N_1-M$. Thus, $\Lambda(N_1,0,\ldots,0)$ can be efficiently initialized by LBANC.

[5]Equation (10) holds also when $N_R = 0$ and thus the right-hand side recursion must be performed on a class $r < R$. The only difference in this case is that $\bar{\mathbf{A}}(\vec{N})$ and $\bar{\mathbf{B}}(\vec{N})$ are defined without the PCs of class R and some normalizing constants in the basis have negative populations (and thus these constants are set by definition equal to zero).

A. Probabilistic Interpretation

We interpret CoMoM as a recursion on a set of higher-order moments of queue-lengths. We show in Theorem 2 that these higher-order moments are conditioned on a group of jobs residing at a tagged station, which is consistent with previous work on the characterization of product-form models [24].

Theorem 2: The basis $\Lambda(\vec{N})$ is equivalent to the following set of conditional higher-order moments of queue-lengths:

$$\sum_{\vec{n}:\vec{n}_k\geq\vec{c}_k}\left(\prod_{r=1}^{R}\frac{(n_{k,r})_{c_{k,r}}}{(n_k)_{c_k}}\right)P(\vec{n};\vec{N}) = \prod_{r=1}^{R} D_{k,r}^{c_{k,r}}\frac{G(\vec{0},\vec{N}-\vec{c}_k)}{G(\vec{N})},$$

$$\sum_{\vec{n}:\vec{n}_k\geq\vec{c}_k}\left(\prod_{r=1}^{R}\frac{(n_{k,r})_{c_{k,r}}}{(n_k)_{c_k-1}}\right)P(\vec{n};\vec{N}) = \prod_{r=1}^{R} D_{k,r}^{c_{k,r}}\frac{G(1_k,\vec{N}-\vec{c}_k)}{G(\vec{N})},$$

where $(x)_c = x(x-1)\cdots(x-c+1)$, $(x)_c = 0$ if $x < c$, and $\vec{c}_k = (c_{k,1},\ldots,c_{k,R})$, $\sum_{r=1}^{R-1} c_{k,r} \leq M$, is a vector of jobs conditioned to reside in queue k, $1 \leq k \leq M$.

Due to limited space, we point to [5] for a proof of Theorem 2. In the above expressions, the left-hand sides are conditional higher-order moments, while the normalizing constants in the right-hand side together form the basis of CoMoM. The intuition behind this finding is that, differently from the MVA that only evaluates mean values, CoMoM carries on in the recursion also higher-order information on the performance behavior of each server. The importance of higher-order moments is that CoMoM can use this extra information to decrease the degrees of freedom of the analysis *df* and define by (10) a recursion that is scalable with the number of classes. *This is achieved by using the extra information of the higher-order moments to prevent CoMoM from stepping into the recursive branches of the classes* $r = 1,\ldots,R-1$, as we have shown in the illustrating example in Section III.

V. Numerical and Linear Algebraic Properties

In this section, we discuss extensively the solution of the linear system (9). We first describe in Section V-A a permutation to block triangular form of the coefficient matrix $\bar{\mathbf{A}}(\vec{N})$ which improves scalability of CoMoM. Numerical properties of the recursive evaluation of (9) are discussed in Section V-B where we also give recommendations on the best linear system solvers for CoMoM. Finally, in Section V-C we discuss degenerate models where (9) is singular and define strategies to address these cases.

A. Block Triangular Form and Fine-Grain Decomposition

We define a block triangular form (BTF) for $\bar{\mathbf{A}}(\vec{N})$ in (9) by observing that, while recurring on class R, the only entries of $\bar{\mathbf{A}}(\vec{N})$ that change from $\bar{\mathbf{A}}(\vec{N}-\vec{1}_R)$ are the N_R coefficients in the PCs of class R. These PCs are needed only to compute the constants in $\lambda(\vec{0})$, which suggests the following improvement.

Theorem 3: The matrix $\bar{\mathbf{A}}(\vec{N})$ can be permuted to the BTF

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

where $\bar{\mathbf{A}}_{1,1}$ is square of order $q = \binom{M+R-1}{M}M$, the blocks $\bar{\mathbf{A}}_{1,1}$ and $\bar{\mathbf{A}}_{1,2}$ are independent of N_R, and $\mathbf{I}_p$ is the identity matrix of order $p = \binom{M+R-1}{M}$ associated to the columns of $\lambda(\vec{0}) \subset \Lambda(\vec{N})$.

Proof: The proof of the statement follows by permutation of rows and columns of $\bar{\mathbf{A}}(\vec{N})$ according to the previous observation that only the PCs used to compute the unknowns in $\lambda(\vec{0})$ depend on N_R. The formulas for the cardinalities p and q follow immediately by the intermediate results in the proof of Theorem 1. ∎

Denote by $\Lambda_p(\vec{N})$ the partition of $\Lambda(\vec{N})$ associated to the unknowns in $\lambda(\vec{0})$, by $\Lambda_q(\vec{N})$ the partition for the remaining constants, and by $\bar{\mathbf{B}}_p$ and $\bar{\mathbf{B}}_q$ the related sub-matrices of $\bar{\mathbf{B}}(\vec{N})$ which are independent of N_R as well. We rewrite the recursive step of CoMoM as

$$\Lambda_q(\vec{N}) = \bar{\mathbf{A}}_{1,1}^{-1}(\bar{\mathbf{B}}_q\Lambda(\vec{N}-1_R) - \bar{\mathbf{A}}_{1,2}\Lambda_p(\vec{N}-1_R)), \quad (12)$$

$$\Lambda_p(\vec{N}) = N_R^{-1}\bar{\mathbf{B}}_p\Lambda(\vec{N}-1_R). \quad (13)$$

The main advantage of this form is that $\bar{\mathbf{A}}_{1,1}$ is smaller than $\bar{\mathbf{A}}(\vec{N})$ and, more importantly, all matrices are independent of N_R, thus they can be computed and possibly factorized only once at the beginning of the iteration on class R.

Another optimization can be obtained by fine decomposition of $\bar{\mathbf{A}}_{1,1}$. Figure 9, reported at the end of the paper, depicts $\bar{\mathbf{A}}_{1,1}$ for models with different number of service classes. We can see that also $\bar{\mathbf{A}}_{1,1}$ admits a permutation to a fine-grain BTF.

Theorem 4: The matrix $\bar{\mathbf{A}}_{1,1}$ admits the BTF

$$\bar{\mathbf{A}}_{1,1} = \begin{bmatrix} \bar{\mathbf{C}}_0 & \bar{\mathbf{C}}_{01} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{C}}_1 & \bar{\mathbf{C}}_{12} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \bar{\mathbf{C}}_2 & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \bar{\mathbf{C}}_{\bar{H}-1} & \bar{\mathbf{C}}_{\bar{H}-1,\bar{H}} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \bar{\mathbf{C}}_{\bar{H}} \end{bmatrix}, \quad (14)$$

where $\bar{H} = \min(M, R-1)$ and the macro-block $\bar{\mathbf{C}}_h \equiv \operatorname{diag}(\bar{\mathbf{C}}_h^{(1)}, \bar{\mathbf{C}}_h^{(2)}, \dots, \bar{\mathbf{C}}_h^{(t)}, \dots, \bar{\mathbf{C}}_h^{(T_h)})$, $0 \le h \le \bar{H}$, has $T_h = \binom{R-1}{h}$ micro-blocks $\bar{\mathbf{C}}_h^{(t)}$ of order $\binom{M}{h}M$.

Proof: The basis $\Lambda(\vec{N})$ is uniquely defined by the set of vectors $\vec{n}$ with $\sum_{s=1}^{R-1} n_s = M$, $n_s \ge 0$, which we partition according to the number and position of non-zeros in $\vec{n}$. We associate to each partition a value h equal to the number of non-zeros in $\vec{n}$ associated to its elements; it is easy to verify that the range of h is $0 \le h \le \bar{H}$, $\bar{H} = \min\{M, R-1\}$. The macro-blocks $\bar{\mathbf{C}}_h$ are obtained by selecting the columns of $\bar{\mathbf{A}}_{1,1}$ and related rows associated to the normalizing constants with population $\vec{N} - \vec{n}$, where $\vec{n}$ has h non-zeros. The matrix $\bar{\mathbf{C}}_h$ includes all micro-blocks $\bar{\mathbf{C}}_h^{(t)}$, which in number are equal to the way of assigning the positions of the h non-zeros to the first $R-1$ classes of $\vec{n}$, i.e., $\binom{R-1}{h}$. It is easy to verify that the cardinality $\binom{M}{h}M$ counts the number of feasible CEs and PCs in each micro-block and satisfies Vandermonde convolution [8] which states $\binom{M+R-1}{R}R = \sum_{h=0}^{\bar{H}}\binom{R-1}{h}\binom{M}{h}M$. where the left-hand side is exactly the order of $\bar{\mathbf{A}}_{1,1}$. ∎

The form (14) implies new computational savings, since (12) can be computed by solution of a sequence of small linear systems with coefficient matrices $\bar{\mathbf{C}}_h^{(t)}$ that are much smaller than $\bar{\mathbf{A}}_{1,1}$. Table II reports order and number of non-zeros in $\bar{\mathbf{A}}(\vec{N})$ and in the micro-block of (14) with largest order denoted by $\bar{\mathbf{C}}_{\bar{H}}^{max}$. The great majority of micro-blocks $\bar{\mathbf{C}}_h^{(t)}$ has much smaller order than $\bar{\mathbf{C}}_{\bar{H}}^{max}$, hence the properties of $\bar{\mathbf{C}}_{\bar{H}}^{max}$ characterize the hardest linear system solved by CoMoM. We also report the number of micro-block-level linear systems evaluated in (12)-(14). Table II indicates that the improvements of the BTF are dramatic. Prohibitive linear systems with hundreds of thousands of equations are decomposed into a long sequence of much smaller, feasible, linear systems. We also observe that the BTF is more effective when $R > M$, e.g., the BTF is better for $M = 8$ and $R = 10$ than for $M = 10$ and $R = 8$. This is a desirable feature of the BTF because the focus of CoMoM is on models with large number of classes R, see Section VIII for additional comments.

TABLE II
SCALABILITY OF THE BLOCK TRIANGULAR FORM (14) OF COMOM

MODEL		ORIGINAL FORM (9)		BLOCK TRIANGULAR FORM (14)		
M	R	$\bar{\mathbf{A}}(\vec{N})$ *order*	$\bar{\mathbf{A}}(\vec{N})$ *non-zeros*	*number of micro-blocks*	$\bar{\mathbf{C}}_{\bar{H}}^{max}$ *order*	$\bar{\mathbf{C}}_{\bar{H}}^{max}$ *non-zeros*
2	2	9	22	7	4	8
2	4	30	95	19	4	8
2	8	108	397	67	4	8
2	10	165	626	103	4	8
4	2	25	76	11	16	40
4	4	175	777	47	24	84
4	8	1650	9494	433	24	84
4	10	3575	21870	975	24	84
8	2	81	280	19	64	176
8	4	1485	8481	181	448	2464
8	8	57915	487311	6571	560	3640
8	10	218790	2033603	24829	560	3640
10	2	121	430	23	100	280
10	4	3146	19071	304	1200	6960
10	8	213928	1988987	19586	2520	20160
10	10	1016158	10575708	92900	2520	20160

B. Numerical Solution of CoMoM Linear Systems

We discuss the numerical evaluation of (9) in CoMoM with BTFs (12)-(14), focusing on the properties of *correctness* and *efficiency* of the recursive solution.

Correctness of CoMoM. Regarding correctness, we argue that the solution of (9), similarly to the MoM algorithm in [3], should be performed using exact linear algebra. This consideration arises from the fact that the solution of a *sequence* of linear systems of CEs and PCs using standard (inexact) linear algebra is subject to two numerical issues: floating-point range exceptions of normalizing constants [15] and potential round-off error accumulations while recurring on the sequence of linear systems [3]. While floating-point range exceptions may be addressed using extended numerical precision (e.g, quad-precision arithmetic) or dynamic scaling [15], we have observed that, similarly to MoM [3], the error accumulation in CoMoM becomes uncontrollable with inexact algebra when the populations exceed more than a few tens of requests. The instability is visible also if each individual linear system (9) is numerically well-conditioned (e.g., condition number $\kappa_{cond} < 10^2$) and the error accumulation can only be slowed, but not avoided, if extended precision arithmetic is used. For instance, Table III shows an error accumulation example for a small model with $M = 2$ queues and $R = 2$ classes, see Section V-C for $\bar{\mathbf{A}}(\vec{N})$ and $\Lambda(\vec{N})$ structure. Here we have set $D_{1,1} = 10s$, $D_{2,1} = 5s$, $D_{1,2} = 5s$, $D_{2,2} = 9s$, $Z_1 = Z_2 = 0s$, $N_1 = 150$ jobs and we have studied the error propagation in CoMoM solution as the class-2 population N_2 grows. We have tested with linear system solution methods based on Gaussian elimination and iterative algorithms, including CG, BiCG, GMRES, and QMR implemented from the templates defined in [1]. Table III shows error accumulation for Gaussian elimination (implemented with LU back-substitution) and for the iterative algorithm that performed in the most stable fashion, i.e., QMR with a maximum of $10,000$

TABLE III
ROUND-OFF ERROR PROPAGATION WITH INEXACT LINEAR ALGEBRA

	Relative Error Function: $\Delta = \lvert 1 - G^{comom}(\vec{N})/G^{exact}(\vec{N})\rvert$			
	Gaussian Elimination		Iterative Approximation	
N_2	LU 32-bit	LU 128-bit	QMR 32-bit	QMR 128-bit
1	$1.33 \cdot 10^{-15}$	$2.83 \cdot 10^{-16}$	$2.17 \cdot 10^{-14}$	$2.83 \cdot 10^{-16}$
25	$1.73 \cdot 10^{-11}$	$6.46 \cdot 10^{-16}$	$7.84 \cdot 10^{-8}$	$9.86 \cdot 10^{-16}$
50	$2.98 \cdot 10^{-5}$	$1.94 \cdot 10^{-16}$	0.15	$4.75 \cdot 10^{-10}$
75	1.02	$2.56 \cdot 10^{-15}$	1.00	$6.40 \cdot 10^{-4}$
100	1.00	$2.06 \cdot 10^{-9}$	1.00	0.99
125	1.00	$2.80 \cdot 10^{-4}$	1.00	1.00
150	1.00	1.00	1.00	1.00

iterations and convergence tolerance 10^{-13}. In all four cases the numerical solution of (9) becomes unstable when the population grows too large[6]. Extended precision arithmetic helps in slowing down the error accumulation, but its effectiveness depends on the values of the service demands $D_{k,r}$. As a further remark, even if not immediately visible from the most significant digits of $G(\vec{N})$, error accumulation starts as soon as the normalizing constant value exceeds the arithmetic range. This strongly suggests to use exact algebra to avoid introducing round-off errors in $G(\vec{N})$.

Following the last observation, we recommend for CoMoM a definitive solution to all correctness problems based on solving (9) with exact linear algebra, which clearly imposes a trade-off with computational efficiency. Numerical stabilization using inexact linear algebra is a difficult open problem which leaves room for increasing the efficiency of CoMoM, see Section IX. The use of exact algebra is motivated in the context of queueing network models by a fundamental observation: the computational overhead introduced by exact linear algebra can be upper bounded theoretically and it is found in Section VI to be much less than the gains resulting from the increased recursion efficiency of CoMoM compared to existing methods. Therefore, accepting exact linear algebra (9) as a method to stabilize CoMoM grants the correctness of the results while leaving a consistent margin of computational gain with respect to the MVA and other methods. An upper bound on the computational overheads of exact algebra is given in the next result obtained in [3].

Theorem 5 (Exact algebra overhead, [3]): The computational overhead of adopting exact linear algebra in normalizing constant computations is upper bounded by the cost of adopting an arithmetic with operands having up to $n = N\lceil\log(D_{max}(N + M + R))\rceil$ digits, where $D_{max} = \max_{k,r}\{D_{k,r}, Z_r\}$ and $\lceil\cdot\rceil$ is the ceiling operator. For instance, when multi-precision arithmetic is used, the computational overheads of arithmetic operations grow as $O(n \log n \log\log n)$ [21].

Proof: (Sketch of the proof, see [3].) The proof follows by observing that the number of digits required by the stabilization is upper bounded by $n = \lceil\log G\rceil$, where G is the largest normalizing constant used in computations. The value of n can be bounded as $n \leq n_{max}$, where $n_{max} = \lceil\log G^{max}\rceil$ is the number of digits of the normalizing constant of a model where we replace all $D_{k,r}$'s by $D_{max} = \max_{k,r}\{D_{k,r}, Z_r\}$. Noting that G^{max} refers to a model with balanced demands, it is easy to determine a closed-form expression for G^{max} which returns $n_{max} = N\lceil\log(D_{max}(N + M + R))\rceil$. ■

[6]Note that in Table III the errors converge to $\Delta = 1.00$ because CoMoM returns a normalizing constant that is several order of magnitude smaller than the exact one.

Efficiency of CoMoM. From now on, we assume that CoMoM recursive evaluation of (9) is implemented using exact algebra. Among existing exact solution techniques for linear systems of equations, we recommend to use sparse finite-field linear algebra (FFLA) for the evaluation of (9), see [23] and references therein for background. Compared to other exact methods, FFLA has minimal memory requirements thanks to the use of modulo arithmetic[7] and time requirements grow only quadratically in the linear system order.

The best algorithm for solving (9) using FFLA depends on the linear system size and sparsity. To achieve maximum scalability, one may use the Wiedemann algorithm [14], which is the standard solution method for large-scale systems in the high-performance Linbox library (*http://www.linalg.org*), a C++ library resulting from a large collaborative research effort for the implementation of state-of-the-art FFLA algorithms. The Wiedemann method is a probabilistic algorithm for sparse linear systems over finite fields which may be seen as a FFLA counterpart of iterative algorithms such as conjugate gradient and Krylov subspace methods [14]. The Wiedemann algorithm preserves the sparsity of the matrix $\bar{\mathbf{A}}(\vec{N})$, which is crucial for CoMoM efficiency on large-scale models. For a linear system of order n with ω non-zeros, the computational costs of Wiedemann algorithm grow as $O(n^2)$ in time and $O(n+\omega)$ in space [14], resulting in a much more efficient linear system solution than Gaussian elimination which requires $O(n^3)$ time and $O(n^2)$ space. It should however be noted that for models that are not too large, e.g., $M < 8 \wedge R < 8$, the BTF (14) results in a sequence of linear systems that have small order (few tens of elements) and where the coefficient matrix is not very sparse, see Table II. In these small linear systems, the cubic costs of LU factorization are often acceptable, the fill-ins do not grow too quickly, and LU back-substitution has quadratic costs which are generally much smaller than those of the Wiedemann algorithm. This can be explained by noting that the Wiedemann algorithm and iterative methods usually require some tens of iterations[8] before finding the exact solution of (9). Based on these observations, the Wiedemann algorithm remains the method of choice for CoMoM on large-scale models, while Gaussian elimination is the best choice in all other cases. We show later in Case Study 3 of Section VI an example of model where the Wiedemann algorithm is much more scalable than Gaussian elimination in the evaluation of (9).

Computational Complexity. The complexity formulas of the Wiedemann algorithm allow to estimate the computational costs of CoMoM under asymptotic growths of M, R, and N. In particular, from the intermediate results in theorems 1 and 4 which describe the number of CEs, PCs, and micro-blocks in $\bar{\mathbf{A}}(\vec{N})$, it is not difficult to show that for (9) with BTF (14) the time requirements grow approximately as

$$N^2 \log(D_{max}(N + M + R)) \sum_{h=1}^{\bar{H}} \binom{R-1}{h}\binom{M}{h}^2 M^2,$$

which is the cost of solving N micro-block level linear systems

[7]We recall that in modulo arithmetic a linear system is decomposed into a set of finite-field linear systems over different modulos. These can be solved sequentially or in parallel using standard floating-point arithmetic and with memory costs similar to inexact linear algebra. The computed solutions are later assembled into the final result by the Chinese remainder theorem.

[8]This is consistent also with the recommendations of the Linbox library tutorials, where it is observed that Gaussian elimination should be used as the method of choice for solving small-scale or medium-scale sparse FFLA systems, whereas the Wiedemann algorithm is best for large systems.

using the Wiedemann algorithm multiplied by the maximum numerical overhead of exact algebra[9]. Similarly, it is not difficult to show that space requirements grow approximately as

$$(\max_{1\leq h\leq \bar{H}} \tbinom{M}{h} M) + MN\log(D_{max}(N+M+R))\tbinom{M+R-1}{M} + (M(R+1)+R(M+1))\tbinom{M+R-2}{M-1},$$

where the first term is the maximum Wiedemann algorithm space overhead, the middle term is a bound on the size of the basis $\Lambda(\vec{N})$, and the last term is the storage cost of $\bar{\mathbf{A}}(\vec{N})$ and $\bar{\mathbf{B}}(\vec{N})$. Since existing methods such as the MVA, RECAL, and RGF grow in time and space requirements as $O(N^R)$ or $O(N^M)$, respectively, it is simple to conclude that for large enough populations CoMoM will be always more efficient than these methods since its complexity with respect to the population is $O(N^2\log N)$ in time and $O(N\log N)$ in space. In practice, the value of N after which CoMoM becomes preferable is of the order of tens if the number of classes is not too small (e.g., $R > 3$). We point to Section VI for experiments confirming the efficiency of CoMoM. We conclude by remarking that the expressions for computational complexity of Gaussian elimination, implemented with LU factorization and back-substitution, are worse than the ones of the Wiedemann algorithm: the time complexity requires to add a cubic term $\sum_{h=1}^{\bar{H}}\tbinom{R-1}{h}\tbinom{M}{h}^3 M^3$ for LU factorization; the storage complexity requires to replace the memory overhead $(\max_{1\leq h\leq \bar{H}}\tbinom{M}{h}M)$ by a quadratic term $\sum_{h=1}^{\bar{H}}\tbinom{R-1}{h}\tbinom{M}{h}^2 M^2$ that accounts for the fill-in resulting from the LU factorization. Nevertheless, the hidden constants of the asymptotic complexity notation make in practice Gaussian elimination more efficient than the Wiedemann algorithm for small- and medium-scale models.

C. Analysis of Degenerate Models

CoMoM can solve problems that are much harder than the ones that can be solved efficiently by the MVA, but the method requires independence between CEs and PCs. Occasionally, CEs and PCs are linearly dependent and this implies that $\bar{\mathbf{A}}(\vec{N})$ is singular and the linear system (9) does not have a unique solution. In this subsection, we discuss singularity conditions of $\bar{\mathbf{A}}(\vec{N})$ and outline technical remedies to solve the queueing network model in this case. To simplify exposition, we illustrate singularity conditions using a small case study. We also explain how to generalize the techniques to models with larger number of queues or classes. Figure 2 summarizes the analysis of this section.

Singularity Case Study. Let us consider a small model composed by $M = 2$ queues and $R = 2$ classes. While recurring on class 2, the left hand side $\bar{\mathbf{A}}(\vec{N})\Lambda(\vec{N})$ of the linear system (9) is

$$\begin{pmatrix} 1 & 0 & 0 & 0 & -D_{1,1} & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & -D_{2,1} & 0 & 0 & -1 \\ 0 & 0 & -D_{1,1} & -D_{2,1} & 0 & 0 & -Z_1 & N_1-1 & 0 \\ 0 & 0 & 0 & -D_{2,1} & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & -D_{1,1} & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -D_{1,1} & -D_{2,1} & 0 & -Z_1 & N_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & N_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & N_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & N_2 \end{pmatrix} \begin{pmatrix} G(1_1,\vec{N}) \\ G(1_2,\vec{N}) \\ G(1_1,\vec{N}-\vec{2}_1) \\ G(1_2,\vec{N}-\vec{2}_1) \\ G(1_1,\vec{N}-\vec{1}_1) \\ G(1_2,\vec{N}-\vec{1}_1) \\ G(\vec{N}-\vec{2}_1) \\ G(\vec{N}-\vec{1}_1) \\ G(\vec{N}) \end{pmatrix}$$

[9]We refer to the worst-case scenario where the prime number q defining the finite field is of the order of the normalizing constant, thus the overheads of exact arithmetic are $O(n\log n\log\log n)$ [21] with $n = N\lceil\log(D_{max}(N+M+R))\rceil$. To simplify expressions, we do not write explicitly the polylogarithmic factor $\log\log n$ which has a very small impact on the complexity.

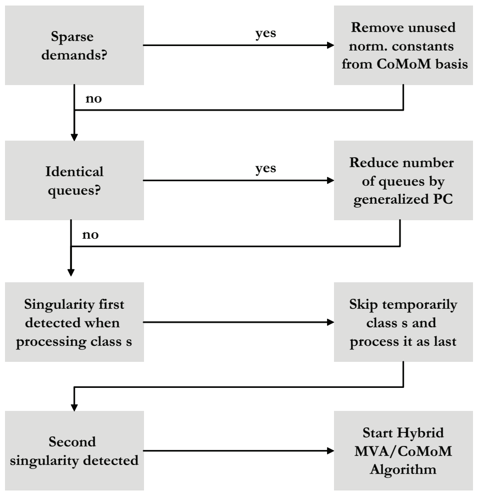

Sparse demands?
yes
Remove unused norm. constants from CoMoM basis
no
Identical queues?
yes
Reduce number of queues by generalized PC
no
Singularity first detected when processing class s
Skip temporarily class s and process it as last
Second singularity detected
Start Hybrid MVA/CoMoM Algorithm

Fig. 2. **Addressing Singularity in CoMoM**. *Summary of techniques proposed in Section V-C to address degenerate queueing network models.*

where $\vec{2}_1 = 2\cdot\vec{1}_1$, rows 1 and 5 are CEs (2) for $k = 1$, rows 2 and 4 are CEs for $k = 2$, rows 3 and 6 are PCs (3) for $r = 1$, and rows 7–9 are PCs (3) for $r = 2$. The coefficients of (2)-(3) not included above appear in the right hand side of (9). In the above example, the coefficient matrix has determinant $\det(\bar{\mathbf{A}}(\vec{N})) = D_{1,1}D_{2,1}(D_{2,1}-D_{1,1})N_2^3$, and, since $N_2 \geq 1$, singularity arises if and only if $D_{k,1} = 0$, for any $k = 1, 2$, or if $D_{1,1} = D_{2,1}$. We consider the two cases separately and show how we can solve these degeneracies.

1) Singularity Condition: Sparsity of Service Demands. Let us assume without loss of generality that it is $D_{1,1} = 0$. Then, column 3 and column 5 in $\bar{\mathbf{A}}(\vec{N})$ are zero which implies $\det(\bar{\mathbf{A}}(\vec{N})) = 0$. This singularity can be prevented by removing from the basis the unknowns of columns 3 and 5, i.e. $G(1_1,\vec{N}-\vec{1}_1)$ and $G(1_1,\vec{N}-\vec{2}_1)$, which leaves an overdetermined linear system. The other unknowns can still be computed because they are independent of these two constants. In the general case, singularity due to sparsity of demands in larger models is addressed similarly by reducing the basis size.

2.a) Singularity Condition: Identical Queues. The case $D_{1,1} = D_{2,1}$ is an example of degenerate demands belonging to a same service class. In this specific case, we can further distinguish two possibilities: (a) $D_{1,2} = D_{2,2}$ and queue 1 and 2 are identical for all demands; (b) $D_{1,2} \neq D_{2,2}$ which is discussed in the next singularity condition.

The singularity in subcase (a) can be handled easily because identical queues can be treated as one thanks to the following generalization of the PC [3]

$$N_r G(\vec{N}) = Z_r G(\vec{N}-\vec{1}_r) + \sum_{k=1}^{M} m_k D_{k,r} G(\vec{1}_k, \vec{N}-\vec{1}_r), \quad (15)$$

for $1 \leq r \leq R$, where $m_k \geq 1$ is the number of queues identical to queue k (including queue k itself). We point to [3] for a proof of this equation. For instance, if $D_{1,1} = D_{2,1}$, we consider the model as composed by $M = 1$ *distinct* queues and $R = 2$ classes. Thus, $\bar{\mathbf{A}}(\vec{N})$ reduces to the non-singular matrix

$$\begin{pmatrix} 1 & -D_{1,1} & 0 & -1 \\ 0 & -m_1 D_{1,1} & -Z_1 & N_1 \\ 0 & 0 & N_2 & 0 \\ 0 & 0 & 0 & N_2 \end{pmatrix} \begin{pmatrix} G(1_1,\vec{N}) \\ G(1_1,\vec{N}-\vec{1}_1) \\ G(\vec{N}-\vec{1}_1) \\ G(\vec{N}) \end{pmatrix},$$

where $m_1 = 2$ because there are two identical queues, row 1 is the CE for $k = 1$, row 2 is the generalized PC (15) for $r = 1$, and rows 3 and 4 are the generalized PCs for $r = 2$. Note that the linear system order decreases because for $M = 1$ the basis $\Lambda(\vec{N})$ is smaller than for $M = 2$. We conclude this case by remarking that this solution approach based on (15) generalizes to models of larger size by replacing sub-networks of identical queues by a single station and setting in (15) the multiplicities m_k accordingly.

2.b) Singularity Condition: Degenerate Demands for a Single Service Class. The singularity condition of subcase (b), in which $D_{1,2} \neq D_{2,2}$, is instead more difficult, because the two queues are not identical and (15) does not apply. However, this case can be handled effectively by changing the order in which we evaluate the service classes. Instead of processing first the class-1 populations $(1,0),(2,0),\ldots,(N_1,0)$, we first solve the model on the class-2 populations $(0,1),(0,2),\ldots,(0,N_2)$, which can be done by single-class algorithms without incurring into singularity problems. After this step, CoMoM processes the class-1 populations $(1,N_2),(2,N_2),\ldots,(N_1,N_2)$. Since we have inverted the order in which we process the two classes, the coefficient matrix $\bar{\mathbf{A}}(\vec{N})$ is now expressed as a function of $D_{k,2}$, Z_2, and N_2 instead of the corresponding parameters of class 1 which were responsible for singularity of $\bar{\mathbf{A}}(\vec{N})$. That is, while recurring on class 1 to reach (N_1,N_2), the parameters $D_{k,1}$, Z_1, and N_1 are now always included only in $\bar{\mathbf{B}}(\vec{N})$ and hence cannot affect the singularity of $\bar{\mathbf{A}}(\vec{N})$. This is sufficient to eliminate the singularity condition of identical demands in the linear system. Summarizing, this result is made possible by the fact that the demands of the currently processed class are *never* used as coefficients in $\bar{\mathbf{A}}(\vec{N})$ because, from the structure of (2)-(3), they appear only in $\bar{\mathbf{B}}(\vec{N})$. This means that on larger models we can address *all* cases where some demands of a specific class are responsible for singularity by processing the population of this class as last.

3) Singularity Condition: Multiple Degeneracies. Besides the conditions discussed for the case $M = 2$ and $R = 2$, other degenerate values of service demands arise in models with several queues and classes. For example, it is possible to show that in a model with $M = 3$ queues and $R = 3$ classes the coefficient matrix $\bar{\mathbf{A}}(\vec{N})$ can become singular due to several degenerate demands, e.g., for $D_{2,2} = D_{3,2}D_{2,1}/D_{3,1}$. Since the number of singularity conditions increases with the model complexity it is possible to find cases where there are simultaneous degenerate demands belonging to different classes. These cases may not be addressable by changing the order of processing of service classes. Instead, we define in Appendix A a hybrid MVA/CoMoM algorithm that can be used for models with multiple degeneracies. This is a general solution strategy where we focus only on the independent equations in $\bar{\mathbf{A}}(\vec{N})$. We stress that it is not possible to evaluate the singular linear system by approximate solvers because of the error accumulation issue, see Section V-B.

VI. Efficiency Analysis

In this section we illustrate the increased efficiency of CoMoM with respect to MVA [20], multiclass Convolution [19], RECAL [9], and RGF[10] [11]. We begin by two case studies of real multi-tier applications. In Case Study 1, we examine a model with six classes of requests and provide evidence that CoMoM is much more scalable than existing methods and can evaluate networks that are prohibitive with existing solution techniques. In Case Study 2, instead, we focus on a smaller case of model with three classes where MVA performs well and we prove that CoMoM can be valuable also on these small models. In Case Study 3, we show an example of model where the Wiedemann algorithm is much more scalable than Gaussian elimination. Finally, in Section VI-D, we explore the general scalability of CoMoM and compare it with previous work as the model parameters increase.

[10]On models with $Z_r > 0$ for some class r, RGF is run with $Z_r = 0$ since no theoretical formulas are available to cope with think times in RGF.

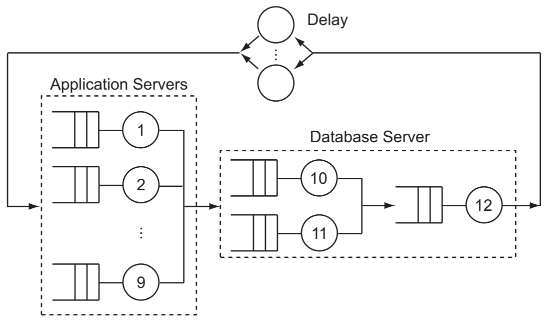


Fig. 3. **Multi-tier Application Case Study**. Queueing network model of a real J2EE e-business system [13].

A. Case Study 1: J2EE e-Business Model

We illustrate CoMoM on the queueing network model presented in [13] of a real J2EE e-business system, see Figure 3. The model captures the performance of a real software system in execution on a multi-tier architecture composed by a cluster of nine application servers (AS) and a dual-processor database server (DB). Queues indexed $1-9$ are application servers, queues $10-11$ represent database processors, queue 12 models HTTP communication overheads. The network does not include communication loops between the AS and the DB because their effect on performance is already accounted within the mean number of visits that define the service demands $D_{k,r}$ [13], [17]. All queues use either processor sharing (queues $1-11$) or first-come first-served scheduling (queue 12), with the exception of the delay server where jobs wait in think state (no queueing) before re-entering the network. The system processes $R = 5$ classes of business workloads:

- NewOrder, enters a new order in the system (class C1);
- ChangeOrder, changes an existing order (class C2);
- OrderStatus, reads the status of an existing order (class C3);
- CustStatus, lists orders of a given customer (class C4);
- WorkOrder, enters a new manufacturing order (class C5);

The service demands found in [13] for the five classes are given in Table IV. We set the mean think times to $Z_1 = Z_2 = Z_3 = Z_4 = 2s$ and $Z_5 = 3s$, respectively.

TABLE IV
Service demands $D_{k,r}$ [ms] of the J2EE application model

Queues vs Classes	*C1*	*C2*	*C3*	*C4*	*C5*
1–9	12.98	13.64	2.64	2.54	24.22
10–11	5.32	5.18	1.24	1.04	17.07
12	1.12	1.27	0.58	0.03	1.68

All population are kept of identical size while increasing N, i.e., $N_1 = N_2 = \ldots = N_R = N/R$. Because some queues

TABLE V
J2EE SYSTEM CASE STUDY

N *vs Time*	*MVA*	*Conv.*	*RECAL*	*RGF*	*CoMoM*/LU
10	1s	1s	1s	1s	1s
50	1s	1s	limit	1s	1s
100	2s	1s	limit	5s	2s
500	n/a	n/a	limit	limit	52s
1000	n/a	n/a	limit	limit	327s
N *vs Space*	*MVA*	*Conv.*	*RECAL*	*RGF*	*CoMoM*/LU
10	1MB	1MB	1MB	1MB	1MB
50	7MB	2MB	n/a	1MB	2MB
100	140MB	43MB	n/a	1MB	5MB
500	limit	limit	n/a	n/a	7MB
1000	limit	limit	n/a	n/a	9MB

are replicated, we use in CoMoM the generalized PC (15). The comparison of CoMoM with the computational costs of MVA [20], multiclass Convolution [19], RECAL [9] and RGF [12] is given in Table V. The table reports time and space requirements for the different methods. Results are rounded up to the larger integer. The notation “limit” indicates that the algorithm has exceed either space or time requirements which are set to 1GB of RAM and 10 minutes on a Intel Core Duo 2×1.60 GHz, respectively; “n/a” indicates that the value cannot be measured because of the infeasible requirement of another physical resource (time or memory). Numerical stabilization of the normalizing constant for existing methods is done by scaling of the service demands [9], [15]. We have experimented both with Gaussian elimination based on LU back-substitution and with the Wiedemann algorithm. On this example, we report results from the Gaussian elimination implementation which has been found at least one order of magnitude faster than the Wiedemann algorithm in all experiments, e.g., for $N = 100$ the Wiedemann algorithm takes 40s instead of the 1s of Gaussian elimination. The results in Table V indicate that CoMoM is the only method capable of solving models for all ranges of populations considered in the experiments. For larger populations values that the ones considered here, the gains of CoMoM are even better, e.g., for $N = 5000$ theoretical formulas indicate that the MVA time costs would be about five orders of magnitude larger than CoMoM and the memory occupation would be eight orders of magnitude larger. We remark that the much larger requirements of RECAL with respect to the other methods derive from its inefficiency in processing networks with queue replicas [12].

B. Case Study 2: Two-Tier PeopleSoft Application Model

We now illustrate a case study of multi-tier architecture where CoMoM shows increased effectiveness with respect to the MVA algorithm also on models with a small number of classes ($R = 3$). Differently from Case Study 1, here we consider the basic implementation of (9) without BTFs. Our aim is to explore a small model where the simplest implementation of CoMoM starts to be more effective than MVA.

We consider a queueing network similar to the two-tier application model in [7] of a PeopleSoft client-server system. Client applications access a remote database server and are modeled in the queueing network as a delay server. The database server is instead modeled by two queues representing its CPU and disk drive. Because we focus on constant-rate servers, we represent the network bandwidth contention by a processor-sharing queue

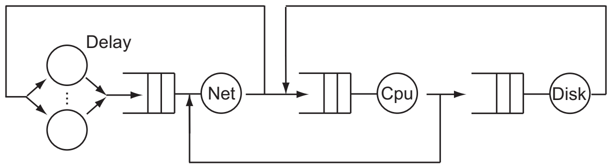


Fig. 4. **PeopleSoft Application Case Study**. Queueing network model of a two-tier PeopleSoft client-server system [7].

instead of the load-dependent station in [7]. The related queueing network model is shown in Figure 4. We use the benchmarking characterization and measurements in [7] to parameterize the queueing network. The benchmark has three classes of workloads, Stress Insert, Stress Update, and Stress Delete which differ for the type of SQL operations in the workload [7]. The service demands for the different classes and resources given in [7] are reported in Table VI. We set the think times at the delay server to $Z_1 = Z_2 = Z_3 = 5s$.

TABLE VI
SERVICE DEMANDS $D_{k,r}$ [MS] OF THE PEOPLESOFT MODEL

Queues vs Classes	C1	C2	C3
Cpu	12.21	21.73	21.96
Disk	26.98	31.94	29.24
Net	16.12	26.46	28.51

We have performed a comparison of CoMoM and MVA using an approach similar to Case Study 1, but without the use of BTFs. Also in this case, the implementation based on LU back-substitution was significantly faster than the Wiedemann algorithm (at least two orders of magnitude) because of the small size of the model under consideration. The results, shown in Table VII, further confirm the increased scalability of CoMoM with respect to MVA. MVA becomes memory inefficient as soon as the total population grows beyond 900 jobs, whereas the memory occupation of CoMoM in all experiments is always less than 14MB. The comparison of computational times is instead favorable to the MVA, thanks to the small number of classes which make the MVA recursion not too expensive. Nevertheless, on this small model, the requirements of CoMoM remain good also without BTFs. This indicates that, even using the simplest possible implementation of (9), one may obtain a solution algorithm that is much more memory efficient than MVA and with similar running times.

C. Case Study 3: Applicability of the Wiedemann Algorithm

In the previous case studies, we have reported that Gaussian elimination based on LU factorization and back-substitution performed better than the Wiedemann Algorithm recommended in Section V for the evaluation of large-scale models. In this section, we show cases of linear systems (9) where the Wiedemann algorithm outperforms Gaussian elimination.

We consider a model with $M = 4$ queues and $R = 8$ classes and service demands $D_{k,r} = k^r$. We compare the solution of a single recursive step of (9) with Gaussian elimination and with the Wiedemann algorithm. From Table 2, it is possible to see that $\mathbf{A}(\vec{N})$ in this case has order 1650 with 9494 non-zeros. We have performed experiments where Gaussian elimination requires 17s of CPU time and 244MB of memory to complete a single recursive step when processing class 8. On the same machine, the Wiedemann algorithm performs the same recursive step in 9s

TABLE VII
PEOPLESOFT APPLICATION CASE STUDY

	Time [s] - no BTF		*Space [MB] - no BTF*	
N	*MVA*	*CoMoM*/LU	*MVA*	*CoMoM*/LU
100	1s	1s	2MB	3MB
300	1s	5s	38MB	5MB
600	3s	27s	280MB	8MB
900	11s	82s	938MB	10MB
1200	n/a	191s	limit	14MB

and the memory occupation is dramatically lowered to 4MB. This is a consequence of the fact that the Wiedemann algorithm is an iterative methods which preserves the sparsity of $\mathbf{A}(\vec{N})$.

We have also performed the same experiment for $M = 4$ and $R = 10$ and we have observed that Gaussian elimination becomes infeasible due to the high computational requirements that would require approximately 220s and 1.2GB for the solution. The Wiedemann algorithm is instead able to solve the recursive step in $69s$ with only 7MB of memory occupation. Note that these values are relative to the evaluation of (9) without BTF; whenever BTF is used, the advantages of the Wiedemann algorithm become evident only when the size of the micro-blocks is of the order of thousands, see Table II.

D. Parametric Analysis of Computational Costs

We conclude the section by showing with a parametric analysis that the computational requirements of CoMoM scale efficiently with the number of service classes. The requirements of CoMoM are compared to those of MVA and RGF in Figure 5 for increasing values of the number of classes R. In the experiments, we vary the number of classes in $R = 1, 2, \ldots, 12$ and set $M = 3$, $N = 1000$, $N_r = N/R$, $D_{max} = 1$ and $Z_r = 0$, $1 \leq r \leq R$. Experiments with different populations or number of queues give results that are even more favorable to CoMoM. In the evaluation of computational requirements, we do not consider the Wiedemann algorithm because this is a probabilistic algorithm and hence its computational requirements vary significantly depending on the actual value of the service demands $D_{k,r}$. We instead focus on the CoMoM implementation based on Gaussian elimination (LU factorization and back-substitution), which provides a worst-case estimate of CoMoM efficiency. In order to show the expect behavior of the methods also on models that are infeasible, estimates are obtained from theoretical formulas.

The figures show that CoMoM has typically the lowest requirements among the considered methods as R grows beyond 3 or 4 classes. MVA and RGF can be better only for models with a very small number of classes. In general, CoMoM becomes inefficient only if both M and R grow simultaneously, because the size of the blocks (14) depends on $\bar{H} = \min\{M, R-1\}$, but this inefficiency is present also in all existing algorithms and remains an open issue of multiclass models.

The analysis of storage requirements indicates that RGF has minimal memory costs. However, CoMoM is much faster than RGF while keeping good memory requirements that are also dramatically lower than the MVA memory occupation. We have found with additional analyses that, as the number of queue M grows, the memory requirements of CoMoM and RGF become similar, while the time costs remain much more favorable to CoMoM. We also remark that, for the models where CoMoM

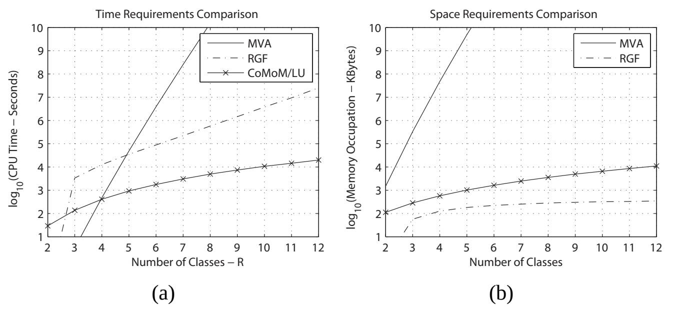


Fig. 5. **Parametric Analysis of CoMoM Requirements**. Parametric comparison of computational requirements for different values of the number of classes. On the considered model: $M = 3$, $N = 1000$, $N_r = N/R$, $1 \leq r \leq R$. Results for models with different number of queues or populations are qualitatively similar.

has the largest memory requirements, one may simply use the implementation based on Wiedemann algorithm that has even negligible storage costs, see the formulas in Section V-B for memory occupation estimates in this case.

VII. EVALUATION OF STATE-DEPENDENT COST INDEXES

In this section, we define a generalization of CoMoM, called Probabilistic CoMoM (Pro-CoMoM), that can efficiently compute marginal probabilities in multiclass models. The marginal probability of observing n jobs in queue k is defined as [19]

$$P_k(n \,|\, \vec{N}) = \textstyle\sum_{\vec{n}_k : n_k = n} F_k(\vec{n}_k) G(-\vec{1}_k, \vec{N} - \vec{n}_k)/G(\vec{N}), \quad (16)$$

where $F_k(\vec{n}_k) = n_k! \prod_{r=1}^{R} D_{k,r}^{n_{k,r}}/n_{k,r}!$. Pro-CoMoM computes marginal probabilities by first obtaining the un-normalized values

$$\widetilde{P}_k(n \,|\, \vec{N}) = P_k(n \,|\, \vec{N}) G(\vec{N})$$

and then scaling back these terms into probabilities by imposing

$$G(\vec{N}) = \textstyle\sum_{n=0}^{N} \widetilde{P}_k(n \,|\, \vec{N})$$

that follows from the condition $\sum_{n=0}^{N} P_k(n \,|\, \vec{N}) = 1$.

We obtain Pro-CoMoM in two steps. First, we generalize CEs and PCs to the computation of marginal queue-length probabilities. Then, we combine these equations into linear systems to obtain a recursive computational scheme similar to CoMoM. *Generalization of CEs and PCs*. The fundamental prerequisite for the development of a recursive scheme for marginal queue-lengths similar to CoMoM is the existence of equations similar to the CEs and PCs and which relate marginal probabilities instead of normalizing constants. Theorem 6 investigates this generalization.

Theorem 6: The un-normalized probabilities $\widetilde{P}_k(n \,|\, \vec{N})$, $1 \leq n \leq N$, satisfy the probabilistic convolution expressions (PCE)

$$\widetilde{P}_k(n \,|\, \vec{1}_j, \vec{N}) = \widetilde{P}_k(n \,|\, \vec{N}) + \textstyle\sum_{r=1}^{R} D_{j,r} \widetilde{P}_k(n \,|\, \vec{1}_j, \vec{N} - \vec{1}_r), \quad (17)$$

for all $1 \leq n \leq N$, $1 \leq k \leq M$, $1 \leq j \leq M$, $j \neq k$, and the probabilistic population constraints (PPC)

$$\begin{aligned} N_r \widetilde{P}_k(n \,|\, \vec{N}) = Z_r \widetilde{P}_k(n \,|\, \vec{N} - \vec{1}_r) + n D_{k,r} \widetilde{P}_k(n-1 \,|\, \vec{N} - \vec{1}_r) \\ + \textstyle\sum_{j \neq k} D_{j,r} \widetilde{P}_k(n \,|\, \vec{1}_j, \vec{N} - \vec{1}_r), \quad (18) \end{aligned}$$

for arbitrary $1 \leq r \leq R$, where $\widetilde{P}_k(n \,|\, \vec{1}_j, \vec{N})$ and $\widetilde{P}_k(n - 1 \,|\, \vec{1}_j, \vec{N} - \vec{1}_r)$ refer to models with a replica of queue j more.

Proof: We use the notation, e.g., $G(\vec{1}_j - \vec{1}_k, \vec{N})$, to describe the normalizing constant of a model obtained from the original network by adding a replica of queue j and removing queue k. Formula (17) follows by writing

$$\widetilde{P}_k(n \,|\, \vec{1}_j, \vec{N}) = \textstyle\sum_{\vec{n}_k : n_k = n} F_k(\vec{n}_k) G(\vec{1}_j - \vec{1}_k, \vec{N} - \vec{n}_k),$$

and applying to $G(\vec{1}_j - \vec{1}_k, \vec{N} - \vec{n}_k)$ the CE for queue j, i.e.,

$$\begin{aligned} G(\vec{1}_j - \vec{1}_k, \vec{N} - \vec{n}_k) &= G(-\vec{1}_k, \vec{N} - \vec{n}_k) \\ &\quad + \textstyle\sum_{r=1}^{R} D_{j,r} G(\vec{1}_j - \vec{1}_k, \vec{N} - \vec{1}_r - \vec{n}_k). \end{aligned}$$

The resulting convolution with F_k gives immediately (17).

Instead, the derivation of (18) starts by observing that

$$\textstyle\sum_{\vec{n}_k : n_k = n} n_{k,r} F_k(\vec{n}_k) G(-\vec{1}_k, \vec{N} - \vec{n}_k) = n D_{k,r} \widetilde{P}(n-1 \,|\, \vec{N} - \vec{1}_r).$$

which implies the following equivalence

$$\begin{aligned} &\textstyle\sum_{\vec{n}_k : n_k = n} (N_r - n_{k,r}) F_k(\vec{n}_k) G(-\vec{1}_k, \vec{N} - \vec{n}_k) \\ &\quad = N_r \widetilde{P}_k(n \,|\, \vec{N}) - n D_{k,r} \widetilde{P}_k(n-1 \,|\, \vec{N} - \vec{1}_r). \end{aligned} \tag{19}$$

Noting that $G(-\vec{1}_k, \vec{N} - \vec{n}_k)$ in the left-hand side of (19) can be expanded by (3) as

$$Z_r G(-\vec{1}_k, \vec{N} - \vec{1}_r - \vec{n}_k) + \textstyle\sum_{j \neq k} D_{j,r} G(\vec{1}_j - \vec{1}_k, \vec{N} - \vec{1}_r - \vec{n}_k),$$

which accounts for the fact that queue k is not anymore in the network, the proof of (18) follows by applying the last expansion to the left-hand side of (19) and observing that the resulting convolutions over $\vec{n}_k$ immediately give (18). ∎

The PCEs and PPCs equations provide generalizations of the CEs and PCs that hold also outside their application in the Pro-CoMoM algorithm. In particular, one may solve any of the two equations in isolation by a recursive scheme similar to MVA/LBANC for the PCEs, or to RECAL for the PPCs, and obtain the value of the marginal probabilities $P_k(n|\vec{N})$. While the computation of marginal probabilities with a recursion similar to the MVA has been already explored in the literature (load-dependent MVA, [20]), a recursive evaluation of the PPCs would produce an algorithm with different computational trade-offs with respect to state-of-the-art methods, i.e., which grows efficiently with the number of classes R, but exponentially with the number of queues M. However, this appears of limited practical applicability due to the high cost of recursions similar to RECAL on models with more than a few tens of requests[11].

Pro-CoMoM Algorithm. We now define the Pro-CoMoM algorithm by solving PCEs and PPCs into a recursive linear system of equations similar to (9). We proceed similarly to the derivation of CoMoM by first defining a basis of marginal queue-length probabilities, and then describing a matrix equation for its recursive computation.

Definition 2: The probabilistic basis $\Pi_k(n \,|\, \vec{N})$ *for queue k is the set of un-normalized marginal probabilities*

$$\Pi_k(n \,|\, \vec{N}) = \pi_k(n, \vec{0}) \bigcup_{1 \le j \le M \,\wedge\, j \neq k} \pi_k(n, \vec{1}_j), \tag{20}$$

where $\pi_k(n, \vec{v}) = \{\widetilde{P}_k(n \,|\, \vec{v}, \vec{N} - \vec{n}) \,|\, \vec{v} \neq \vec{1}_k \wedge \sum_{r=1}^{R-1} n_r \le M \wedge n_R = 0\}$ *is the set of un-normalized marginal probabilities associated to models with $\vec{v}$ queues replicas.* We remark that the values $\widetilde{P}_k(n-1 \,|\, \vec{1}_k, \vec{N} - \vec{1}_r)$, $1 \le n \le N$, $1 \le k \le M$, are not used in the PPCs and thus are omitted from the basis $\Pi_k(n \,|\, \vec{N})$. The Pro-CoMoM recursion is introduced below. Pseudo-code of the recursions and algorithms for generating the matrices used in the recursive step are reported in [5]. We also note that, since (17)-(18) involve only marginal probabilities of the same queue k, the analysis of a queue marginals can be performed independently from the other queues. A complete evaluation of all marginal probabilities is then performed by M successive runs of Pro-CoMoM for $k = 1, 2, \ldots, M$.

Theorem 7: The probabilistic basis $\Pi_k(n \,|\, \vec{N})$ satisfies

$$\begin{aligned} &\bar{\mathbf{A}}_k(n \,|\, \vec{N}) \Pi_k(n \,|\, \vec{N}) = \bar{\mathbf{B}}_k(n \,|\, \vec{N}) \Pi_k(n \,|\, \vec{N} - \vec{1}_r) \\ &\quad + \bar{\mathbf{C}}_k(n \,|\, \vec{N}) \Pi_k(n-1 \,|\, \vec{N}) + \bar{\mathbf{D}}_k(n \,|\, \vec{N}) \Pi_k(n-1 \,|\, \vec{N} - \vec{1}_r), \end{aligned}$$

for all $0 \le n \le N$ and $1 \le k \le M$, where the four matrices have all order $M\binom{M+R-1}{M}$ and are defined by the coefficients of all equations (17)-(18) relating $\Pi_k(n \,|\, \vec{N})$, $\Pi_k(n-1 \,|\, \vec{N})$, $\Pi_k(n \,|\, \vec{N} - \vec{1}_r)$, and $\Pi_k(n-1 \,|\, \vec{N} - \vec{1}_r)$.

Proof: Similarly to the approach in Theorem 1, $\pi_k(0)$ is first computed using the PPC of class R. We prove the rest of the statement by induction on the set of known marginal probabilities. *Case* $n = 0$. In this case (17)-(18) have the same identical structure of (2)-(3) except for the missing marginal probabilities $P_k(0 \,|\, \vec{1}_k, \vec{N} - \vec{N})$ which reduce the number of unknowns in the set difference $\Pi_k(\vec{N})/\pi_k(0)$ to $(M-1)\binom{M+R-1}{M}$. Similarly to the proof of Theorem 1, we can therefore formulate $(M+R-2)\binom{M+R-2}{M-1}$, where the first coefficient is now $(M+R-2)$ instead of $(M+R-1)$ because we can formulate only $M-1$ PCEs due to the missing marginal probabilities. By comparison of the number of unknowns with the number of rows, we find that the degrees of freedom are $df = (M-1)\binom{M+R-1}{M} - (M+R-2)\binom{M+R-2}{M-1} = \binom{M+R-2}{M-1} - \binom{M+R-1}{M} \le 0$ equations, which make the linear system always square or over-determined.

Inductive step. We assume that, when processing the queue-length size n, the solution for $n-1$ is already available. In this case (17)-(18) have again the same structure of (2)-(3) since the terms depending on the marginal probabilities for queue-length $n-1$ are known. Therefore, the same considerations for $n = 0$ apply readily to this case and complete the proof. ∎

The Pro-CoMoM recursive solution is qualitatively similar to the CoMoM recursion. The only significant differences are two: (a) the slightly different structure of (17)-(18) can occasionally result in an over-determined linear system, but this can still be solved, e.g., by the Wiedemann algorithm; (b) for each population vector $\vec{N}$, Pro-CoMoM needs to evaluate a set of $N+1$ marginal probabilities $P_k(n|\vec{N})$ using a different linear system for each of them. This implies that the computational costs of Pro-CoMoM grow as $N+1$ times the computational costs of CoMoM. Since also the MVA algorithm for marginal queue-length probabilities (i.e., the load-dependent MVA) has computational requirements that are $N+1$ times those of the original MVA, it is immediate to conclude that the computational gains of CoMoM over MVA apply also to the Pro-CoMoM algorithm when compared to the load-dependent MVA. For illustration purposes, we show below an example of model that is too computationally expensive to solve with the load-dependent MVA and which can instead be solved exactly and efficiently by the Pro-CoMoM algorithm.

[11]Although we focus on the extension of CoMoM, the development of PCEs and PPCs suggests that it may also be possible to derive also a generalization of the MoM algorithm in [3] to compute marginal queue-length probabilities. However, this generalization of MoM lies outside the scope of this paper.

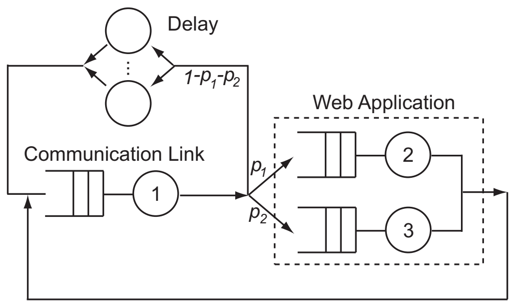


Fig. 6. **Energy Consumption Case Study**. The application servers $2-3$ can reconfigure dynamically their frequency to minimize energy consumption.

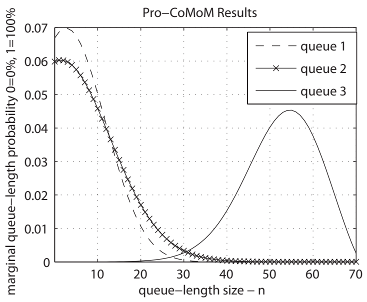
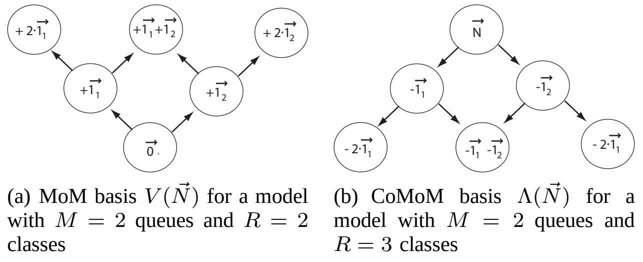


Fig. 7. **Energy Management Case Study: Marginal Queue-Length Probabilities**. The three curves represent the marginal probabilities $P_k(n)$ computed by Pro-CoMoM for the model in Figure 6.



(a) MoM basis $V(\vec{N})$ for a model with $M=2$ queues and $R=2$ classes

(b) CoMoM basis $\Lambda(\vec{N})$ for a model with $M=2$ queues and $R=3$ classes

Fig. 8. **MoM and CoMoM basis comparison**. The MoM basis $V(\vec{N})$ combinatorially increases the number of queue replicas, the CoMoM basis $\Lambda(\vec{N})$ combinatorially removes customers from the network.

A. Energy Consumption Analysis Example

We apply the Pro-CoMoM recursive technique to a simple case study of energy management in Web applications. We evaluate a Web application running on two heterogeneous application servers and serving a population of requests arriving from a shared communication channel, see Figure 6. The servers are able to increase or decrease their frequency without significant overhead. We assume that the frequency scaling in first approximation does not alter the mean service time perceived by the requests. We also assume that both servers $2-3$ in Figure 6 implement frequency scaling and we define an energy management policy $f_k(n)$ as a function of the number of jobs n at queue k which specifies the CPU frequency of that server associated to that particular state. In this example, we consider

$$f_k(n)=\begin{cases}1.00f_k^{max} & n\geq 15,\\ 0.50f_k^{max} & 5\leq n<15,\\ 0.25f_k^{max} & n<5,\end{cases}$$

where f_k^{max} is the maximum operational frequency of the application server k's CPU. Table VIII gives the service demands for the five service classes considered in the example. The mean think times are $Z_1=10ms$, $Z_2=4ms$, $Z_3=2ms$, $Z_4=5ms$, and $Z_5=3ms$. The total request population is $N=70$ distributed across the classes with a mix $\vec{\beta}=(0.36,0.21,0.2,0.14,0.09)$, $\beta_r=N_r/N$, $1\leq r\leq 5$. Throughout the example we assume that on average requests pay a single visit to the servers before completing, a condition that follows by imposing $p_1=p_2=0.25$.

TABLE VIII
SERVICE DEMANDS $D_{k,r}$ [MS] FOR THE MODEL IN FIGURE 6

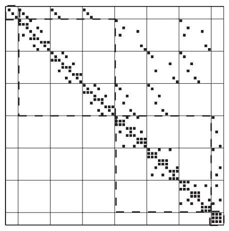
Queues vs Classes	C1	C2	C3	C4	C5
queue 1	13	35	75	70	55
queue 2	50	59	26	89	15
queue 3	96	23	51	96	16

We begin by showing that the MVA approach cannot evaluate accurately power consumption levels in this example. The relative power saving due to energy management during the periods where the servers is non-idle is approximately $\Delta P=(1-f^3)$, where the cubic dependency on frequency follows from [10] and f is the ratio of the mean CPU frequency under the energy management policy to the maximal CPU frequency f_k^{max}. The marginal probabilities computed by Pro-CoMoM for this example are plotted in Figure 7. Using the energy management policy $f_2(n)$ for server 2, the relative savings due to frequency scaling are $\Delta P=70.13\%$. Instead, starting from the MVA results, one may assume that the marginal probabilities are uniformly distributed around the mean. According to this approximation, the power saving of the application server 2 predicted by MVA is $\Delta P=52.64\%$, with a large error with respect to the exact Pro-CoMoM estimate. Similarly, using frequency scaling for server 3, Pro-CoMoM computes a negligible $\Delta P=0.01\%$ saving, while the MVA prediction is $\Delta P=12.42\%$ which is three orders of magnitude different. This shows that both for low utilized and heavily utilized servers, the predictions of MVA are affected by consistent errors that make the estimates unreliable. The Pro-CoMoM algorithm, instead, can provide *exact* estimates of the energy savings thus supporting with correct information the energy management decisions.

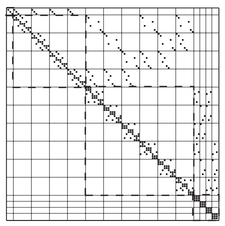
As an additional experiment, we show the scalability of Pro-CoMoM in evaluating marginal probabilities compared to the load-dependent MVA. For the same experiment discussed above, we were able to solve with CoMoM models where the total population is $N=2\times 70=140$, $N=4\times 70=280$, and $N=7\times 70=490$. In these evaluations, the total memory requirement of Pro-CoMoM was always less than 200MB, whereas the load-dependent MVA is infeasible in all three cases, having a computational requirement ranging from 20GB to 40TB. This confirms that Pro-CoMoM is the best algorithm for the exact evaluation of marginal probabilities.

VIII. METHODS OF MOMENTS: COST COMPARISON

In the previous sections we have focused on the comparison of CoMoM with established solution algorithms. In this section, instead, we compare CoMoM with the linear algebraic computational algorithm recently presented in [3], henceforth referred to as the Method of Moments (MoM). In both MoM and CoMoM a basis of normalizing constants is recursively computed using

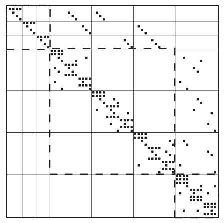


(a) $M = 3$, $R = 4$, order $= 60$, 4 macro-blocks, 8 micro-blocks, maximum inverted block size$= 9$

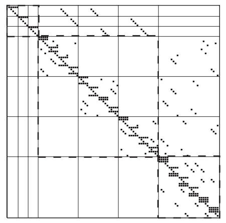


(b) $M = 3$, $R = 5$, order $= 105$, 4 macro-blocks, 15 micro-blocks, maximum inverted block size $= 9$

Fig. 9. **CoMoM Block Triangular Form**. Sparsity structure of the CoMoM coefficient matrix $\bar{\mathbf{A}}_{1,1}$ for models with different number of service classes. Points indicate nonzero entries; on the main diagonal, dashed lines indicate macro-blocks, continuous lines identify micro-blocks.



(a) $M = 3$, $R = 4$, order $= 60$, 3 macro-blocks, 7 micro-blocks, maximum inverted block size $= 12$



(b) $M = 3$, $R = 5$, order $= 105$, 3 macro-blocks, 7 micro-blocks, maximum inverted block size $= 30$

Fig. 10. **MoM Block Triangular Form**. Sparsity structure of $\bar{\mathbf{A}}_{1,1}$ in MoM on the models used in Figure 9. As the number of classes R grows, the CoMoM BTF in Figure 9 generates smaller diagonal blocks that the MoM BTF shown above.

a linear system of CEs and PCs. However, the two methods greatly differ for the choice of the basis of unknowns computed at each recursive step. In the simplest implementation based on an equation similar to (9), MoM defines its basis $V(\vec{N})$ by combinatorially increasing the number of queue replicas in the model up to R queues more. Then, MoM evaluates at each recursive step the related normalizing constants on the populations $\vec{N}$, $\vec{N}-\vec{1}_1$, …, $\vec{N}-\vec{1}_{R-1}$. Instead, CoMoM evaluates normalizing constants on a much larger set of populations (up to M jobs less), but without adding more than one queue replica. Both strategies of MoM and CoMoM increase the number of PCs and CEs so to reduce the degrees of freedom df of the analysis to zero.

Figure 8 compares the basis of MoM for a model with $M = 2$ queues and $R = 2$ classes with the basis of CoMoM in a model with $M = 2$ and $R = 3$. Even though the CoMoM model has one class more, the combinatorial structure of $V(\vec{N})$ and $\Lambda(\vec{N})$ is symmetric. This is a consequence of the fact that, while CoMoM uses conditional higher-order moments of queue-lengths, MoM is based on binomial moments of queue-lengths, and it can be shown that the two sets of moments grow with similar combinatorial structure. We also remark that the basis of CoMoM is quite different from the information carried on by other methods. Focusing on class recursions such as RECAL and RGF, which have a linear recursion on the population that is comparable with that of CoMoM, it is possible to show that RECAL implicitly uses a basis that is similar to that of MoM, but where the number of elements is not fixed with R, and grows combinatorially with the current population size N. That is, at each recursive step, a new "level" is added on top of a basis similar to the one in Figure 8(a). Also RGF has a basis similar to RECAL: RGF performs R recursive steps, one for each service class, in which the basis of normalizing constants is increased by N_r levels after the step for class r. These comparison clarifies that the structure of the basis of CoMoM is different with respect to methods in the literature.

In general, $V(\vec{N})$ and $\Lambda(\vec{N})$ have different cardinalities. We prove the following statement.

Theorem 8: The basis $\Lambda(\vec{N})$ of CoMoM has $\binom{M+R-1}{R}(R-1)$ normalizing constants less then the basis $V(\vec{N})$ of MoM.

Proof: MoM evaluates R constants for each models with up to R queue replicas more. The cardinality of its basis $V(\vec{N})$ is thus $\sum_{l=0}^{R}\binom{M+l-1}{l}R = \binom{M+R-1}{M}(M+R)$, which by comparison with the size of $V(\vec{N})$ given in [3] proves the theorem. ■

The result in (8) proves that the simplest implementation of CoMoM is always much more efficient than the simplest implementation of MoM. However, when considering the BTFs of the two methods, the coefficient matrix $\bar{\mathbf{A}}_{1,1}$ of CoMoM has the same identical size of the coefficient matrix $\mathbf{A}_{1,1}$ of MoM. Table IX compares the properties of the most efficient MoM and CoMoM implementations based on BTFs.

TABLE IX
COMPARISON OF MOM AND COMOM

Algorithm Property	*CoMoM*	*MoM*
Linear system order	$\bar{\mathbf{A}}_{1,1} : \binom{M+R-1}{M}M$	$\mathbf{A}_{1,1} : \binom{M+R-1}{R}R$
H/H coefficients	$H : \min\{M, R-1\}$	$H : \min\{M, R\}$
macro-blocks	$\bar{\mathbf{C}}_h : \bar{H}+1$	$\mathbf{C}_h : H$
Number of micro-blocks	$\bar{\mathbf{C}}_h^{(t)} : \binom{R-1}{h}$	$\mathbf{C}_h^{(t)} : \binom{M}{h}$
micro-block order	$\bar{\mathbf{C}}_h^{(t)} : \binom{M}{h}M$	$\mathbf{C}_h^{(t)} : \binom{R-1}{R-h}R$

The formulas for the micro-blocks order indicate the following rule of thumb for determining which between MoM and CoMoM is more efficient on a model with M queues and R classes: if $M \geq R$, then the $\mathbf{C}_h^{(t)}$ micro-blocks are the smallest and MoM should be adopted, otherwise the $\bar{\mathbf{C}}_h^{(t)}$ micro-blocks are smaller and CoMoM is the method of choice. These observations are confirmed by Figures 9-10 which show the fine-grain decomposition of MoM and CoMoM on models with increasing number of service classes. While CoMoM is able to effectively reduce the order of the micro-blocks and minimize the computational requirements, the coefficient matrix of MoM remains composed of a few large diagonal blocks which impose larger costs in the linear system solution. Therefore, MoM does not scale as effectively as CoMoM on models with many service classes. This makes CoMoM the best exact algorithm for models with several classes.

IX. CONCLUSION

We have introduced CoMoM, a new efficient algorithm for the solution of multiclass queueing network models widely used in capacity planning and performance evaluation of multiclass systems such as multi-tier architectures. CoMoM solves multi-class performance models several orders of magnitude faster and less memory consuming than MVA and existing methods. We have also shown that the algorithm generalizes to the recursive computation of marginal queue-length probabilities, which are important to estimate state-dependent performance attributes.

A possible line of future research for CoMoM is the numerical stabilization of the linear recursion (9). This would open the possibility of solving quickly queueing network models with tens of classes and thousands of requests, which would make exact methods computationally competitive with approximate methods, but without the drawback of returning inaccurate solutions. It would be also interesting to study if new approximate methods can be derived directly from (9).

Acknowledgement

The author wishes to thank Ningfang Mi and Evgenia Smirni for comments on a early version of this manuscript presented at the MAMA Workshop 2008, Annapolis, MD. The author thanks Vittoria Miceli and Giorgio Casale for their invaluable support during the preparation of this work.

References

[1] R. Barrett, M. Berry, et al., *Templates for the Solution of Linear Systems*. SIAM, 1994.
[2] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *JACM*, 22(2):248–260, 1975.
[3] G. Casale. An efficient algorithm for the exact analysis of multiclass queueing networks with large population sizes. In *Proc. of ACM SIGMETRICS/IFIP Performance 2006*, pages 169–180, ACM, 2006.
[4] G. Casale. Exact Analysis of Performance Models by the Method of Moments. (under submission, journal version of [3]), 2008.
[5] G. Casale. CoMoM: A Class-Oriented Algorithm for Probabilistic Evaluation of Multiclass Queueing Networks. Tech. Rep. WM-CS-2008-04, Dept. of Computer Science, College of William and Mary, 2008.
[6] K. M. Chandy and C. H. Sauer. Computational algorithms for product-form queueing networks models of computing systems. *Comm. ACM*, 23(10):573–583, 1980.
[7] Y. Chu, C. J. Antonelli, and T. J. Teorey. Performance Modeling of the PeopleSoft Multi-Tier Remote Computing Architecture. University of Michigan, Technical Report CITI 975, Dec 1997.
[8] D. J. A. Cohen. *Basic Techniques of Combinatorial Theory*. John Wiley and Sons, 1978.
[9] A. E. Conway and N. D. Georganas. RECAL - A new efficient algorithm for the exact analysis of multiple-chain closed queueing networks. *JACM*, 33(4):768–791, 1986.
[10] E. N. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proc. of 2nd Workshop on Power-Aware Computer Systems* – LNCS 2325, pages 179–196, Springer, 2002.
[11] P. G. Harrison and S. Coury. On the asymptotic behaviour of closed multiclass queueing networks. *Perf. Eval.*, 47(2):131–138, 2002.
[12] P. G. Harrison and T. T. Lee. A new recursive algorithm for computing generating functions in closed queueing networks. In *Proc. of IEEE MASCOTS*, pages 223–230, IEEE Press, 2004.
[13] S. Kounev and A. Buchmann. Performance modeling and evaluation of large-scale J2EE applications. In *Proc. of the 29th Int. Conference of the Computer Measurement Group (CMG)*, 273–283, 2003.
[14] B. A. LaMacchia and A. M. Odlyzko. Solving Large Sparse Linear Systems over Finite Fields. In *Proc. of CRYPTO Conf.*, 109–133, 1990.
[15] S. Lam. Dynamic scaling and growth behavior of queueing network normalization constants. *JACM*, 29(2):492–513, 1982.
[16] S. Lam. A simple derivation of the MVA and LBANC algorithms from the convolution algorithm. *IEEE T. Computers*, 32:1062–1064, 1983.
[17] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance*. Prentice-Hall, 1984.
[18] D. Menasce and V. A. F. Almeida. *Capacity Planning for Web Performance: Metrics, Models, and Methods*. Prentice Hall, 1998.
[19] M. Reiser and H. Kobayashi. Queueing networks with multiple closed chains: Theory and computational algorithms. *IBM J. Res. Dev.*, 19(3):283–294, 1975.
[20] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queueing networks. *JACM*, 27(2):312–322, 1980.
[21] A. Schonhage and V. Strassen. Schnelle multiplikation groer zahlen. *Computing*, 7:281–292, Springer Wien, 1971.
[22] 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, 2005.
[23] G. Villard. Some recent progress in exact linear algebra and related questions In *Proc. of ISSAC*, pages 391–392. ACM, 2007.
[24] J. Zahorjan. The distribution of network states during residence times in product form queueing networks. *Perf. Eval.*, 4:99–104, 1984.

Appendix A: Hybrid MVA/CoMoM Algorithm

In this appendix, we sketch the hybrid MVA/CoMoM algorithm to be used in degenerate models with singular matrix $\bar{\mathbf{A}}(\vec{N})$. After identifying and removing the set of linearly dependent rows in $\bar{\mathbf{A}}(\vec{N})$ with numerical methods, we are left with an under-determined linear system with d rows less than the original and without linear dependencies, but we still have to compute all normalizing constants in $\Lambda(\vec{N})$ because otherwise we would miss the term $\mathbf{B}(\vec{N})\Lambda(\vec{N}-\vec{1}_R)$ at the next recursive step. We propose to recursively compute these d constants by a hybrid MVA/CoMoM algorithm. In this hybrid algorithm, we replace (9) by

$$\bar{\mathbf{A}}'(\vec{N})\Lambda'(\vec{N}) = \sum_{r\in\mathcal{R}'} \bar{\mathbf{B}}'_r(\vec{N})\Lambda'(\vec{N}-1_r) + \bar{\mathbf{B}}'_R(\vec{N})\Lambda'(\vec{N}-1_R),$$

where $\Lambda'(\vec{N}) \subset \Lambda(\vec{N})$ does not longer include the d normalizing constants, $\det(\bar{\mathbf{A}}'(\vec{N})) \neq 0$, and $\mathcal{R}'$ is the set of classes on which we perform the additional recursions. Similarly to (9), the choice of the normalizing constants in $\Lambda'(\vec{N})$ immediately defines from (2)-(3) the structure of the matrices $\bar{\mathbf{A}}'(\vec{N})$, $\bar{\mathbf{B}}'_r(\vec{N})$, and $\bar{\mathbf{B}}'_R(\vec{N})$. For example, suppose that in a degenerate model with $R = 4$ classes we remove the normalizing constant $G(\vec{N} - 1_2)$ from $\Lambda(\vec{N})$. Then, the recursion on class 4 would be

$$\bar{\mathbf{A}}'(\vec{N})\Lambda'(\vec{N}) = \bar{\mathbf{B}}'_2(\vec{N})\Lambda'(\vec{N} - 1_2) + \bar{\mathbf{B}}'_4(\vec{N})\Lambda'(\vec{N} - 1_4),$$

where we are now obtaining the value of $G(\vec{N} - 1_2)$ from $\Lambda'(\vec{N} - 1_2)$ instead than from $\Lambda'(\vec{N})$. It is also possible to note that $\Lambda'(\vec{N} - 1_2)$ includes other elements of $\Lambda'(\vec{N})$, e.g., $G(\vec{1}_k, \vec{N} - 1_2)$, $1 \leq k \leq M$. This redundancy can be exploited to minimize computational costs of the hybrid method: we first select the d normalizing constants to remove from $\Lambda'(\vec{N})$ as a set of unknowns such that $\det(\bar{\mathbf{A}}'(\vec{N})) \neq 0$ and which minimizes the number of elements in $\mathcal{R}'$. Then, we remove all redundant constants by defining $\Lambda'(\vec{N})$ as the basis of a model with classes $\{1, 2, \ldots, R\}/\mathcal{R}'$. Note that $\bar{\mathbf{A}}'(\vec{N})$ becomes $\bar{\mathbf{A}}(\vec{N})$ for a model with $R - card(\mathcal{R}')$ classes and thus we can still apply the BTFs defined in Section V-A. The computational cost of the hybrid algorithm can be computed in advance as the cost of solving with CoMoM a model with $R - card(\mathcal{R}')$ classes multiplied by the number of the additional populations spanned by the MVA-like recursion, which is $\prod_{r\in\mathcal{R}'}(N_r + 1)$.

Giuliano Casale received the MS and the PhD degrees in computer engineering from the Politecnico di Milano, Milan, Italy, in 2002 and 2006, respectively. From January 2007 he is a postdoctoral research associate at the College of William and Mary, Williamsburg, Virginia, where he studies the performance impact of burstiness in systems. In Fall 2004, he was a visiting scholar at UCLA studying bounds for queueing networks. His research interests include performance evaluation, modeling, optimization, and simulation. He is member of the ACM, IEEE, and officer of the IEEE Hampton Roads Section.