Approximate Mean Value Analysis based on Markov Chain Aggregation by Composition

Dorina C. Petriu, C. Murray Woodside

Department of Systems and Computer Engineering
Carleton University, Ottawa, Canada, K1S 5B6
email: {petriu|cmw}@sce.carleton.ca

Abstract

Markovian performance models are impractical for large systems because their state space grows very rapidly with the system size. This paper derives an approximate Mean Value Analysis (AMVA) solution for Markov models that represent a composition of subsystems. The goal is robust scalable analytical approximation. The approach taken here is to create approximate aggregated Markov chain submodels, each representing a view of the Markov chain for the entire system from the perspective of a selected set D of tagged components, and to derive mean value equations from them. The analytic solutions of submodels are then combined using system-level relationships, which must be identified for each system; this is not automatic but is usually straightforward. The first point of novelty is the method used to create the aggregate submodels for different sets D, building up each submodel by composition of the components in D rather than by aggregating the entire state space. Another point of novelty is the use of partitioned Markov models to obtain analytic solutions.

Key words: Performance models, Software performance, Compositional modeling, Markov Chains, Aggregation by composition, Mean Value Analysis

1 Introduction

Markovian performance models based on system states and transitions are impractical for large systems because of the very rapid increase of the state space with system size, also known as state explosion. Different approaches have been identified to circumvent state explosion, such as:

- hierarchical decomposition into smaller submodels, linked by a high-level model, or by coordination relationships at their boundaries; the solution is iterated among the submodels

- analytic solutions, usually in some kind of "product form", developed for networks of queues and for some classes of Stochastic Petri Nets.

Analytic solutions are the most efficient but even so, large networks of queues require efficient numerical techniques such as Mean Value Analysis (MVA) (described for example by Bolch et al in chapters 8 and 9 of [2]). MVA computes mean performance values for subsystems (which are individual queues in a network) and combines them to give system performance measures. For systems with no exact analytic solution, approximate MVA (AMVA) has been developed based on the solutions of individual queues, coordination relationships with other queues, and iteration. These AMVA techniques are efficient and scalable and are often accurate, although important questions about the size of errors usually remain unanswered.

This work describes a new approach to obtain an approximate MVA solution for Markov models that represent a composition of components. The goal is a robust scalable analytic approximation. Similar to other AMVA solutions, it solves submodels for their performance measures, and uses system-level relationships among the performance measures to coordinate the submodels, through fixed point iterations. The composed system does not have to be a queueing system, although the examples studied here are special kinds of servers with queues.

Compositional modeling facilitates the definition of system models, since it allows practitioners to describe the behaviour of individual components by state-transition models, and to build the system by composing different components in different ways. This approach is simpler, more flexible and less error-prone than defining the entire system model at once. Components may be defined using *process algebras* (e.g. [8,7]), synchronized *stochastic automata networks* [15,16] or by applying the so-called *process view* from the simulation literature [1]. Formal methods and tools such as TIPP [7] or PEPA [5] can generate the global system model. However the problem of solving the global model is still serious due to state explosion. The present work began with a study of a particular family of components representing software tasks, and a particular approximate MVA solution technique called "Task-directed Aggregation" (TDA) [9–11]. We propose to generalize this to other component-based systems, and the particular MVA TDA solution is used as an example to expound the approach.

The solution proposed in this paper applies to systems that can be broken down into "processes", which, according to [1], describe the life cycle of entities that represent such things as customers, servers, resources, etc. The processes that form a system do not act independently, but need to interact with each other in different ways. In our approach, each process is described by an automaton, and the interaction between processes is realized by the means of synchronized transitions (i.e., executed simultaneously by more than one component) due to a set of shared events. In the domain of Markov processes, the term “process” is replaced by “stochastic process” and the term “automaton” by “stochastic automaton”.

To clarify the approach, let us consider a system (described in more detail in section 2), consisting of n customers $C_i, i = 1, 2, \ldots, n$, one queue Q and a server S. Hence, the system is composed of the following set of automata: $\{C_1, C_2, \ldots, C_n, Q, S\}$. Each automaton is in a certain state at any time. For instance, C_i can either be in state e_i (executing on its own), q_i (waiting for service) or r_i (in service). The states of Q are essentially given by the order in which different customers are in the queue, and the states of S indicate which customer is being served. By combining the states of the automata, one can derive the states of the entire system. In our example, a global state is a tuple $\boldsymbol{\sigma} = (\sigma_1\sigma_2\sigma_3\sigma_Q\sigma_S)$ containing the states of the three clients, the queue and the server. Naturally, not all state combinations are possible. Different systems may have different intended behaviours, which are represented by certain combinations of states. For instance, a desired behaviour for our system dictates that C_i be in state r_i (in service) if and only if the server S is in state s_i. Such constraints are enforced by the interaction between automata through synchronized transitions (when S is ready to move to state s_i, it forces C_i to move to state r_i).

In order to derive a MVA solution for this system, we note that the waiting time for a customer C_i is found from its interaction with each other customer C_j, which can be analyzed one at a time (see section 2.1 for a further discussion). This leads to the aggregation of states, and an aggregated Markov chain in which all customers, aside from C_i and C_j, are indistinguishable. There are $n(n-1)$ such aggregated submodels, and we consider all of them for the global solution. There is a difficulty, however: the interactions within each of the $n(n-1)$ aggregated submodels cannot be obtained exactly, so some approximations are proposed based on certain independence assumptions.

The first point of novelty is the method used to create the aggregated submodels, which builds up the submodel by composing the behaviour of the components we want to observe, rather than by generating and aggregating the entire state space. Our solution is applicable to systems with tightly interconnected components so we cannot use any time-scale decomposition (which only makes the problem harder). The complexity saving in the proposed method comes from ”hiding” inside the aggregated states the behaviour of the indistinguishable components, rather than by hiding internal component behaviour, as in other composition-based methods (e.g., [5,7]). The “Markov chain Aggregation by Composition” (MAC) proposed here uses composition techniques taken from other work in compositional modeling such as [8,6], but the construction of the aggregated MC is novel. Another point of novelty is the use of partitioned Markov models to obtain analytic solutions, as explained in section 3.1. Both of these are generalizations of basic ideas implied in [9–11]. The present paper may be seen as a combination of TDA with the more recent work on compositional modeling.

Compositional performance modeling is important, and MAC/MVA has the potential to solve a key practical problem in providing scalable performance calculations for these models. It gives an approach for deriving mean-value approximations for models based on stochastic process algebras, composed stochastic Petri nets, as well as layered queueing networks. The intention of this paper is to demonstrate feasibility rather than to present a watertight theory, for which further research is required. Some simplifying assumptions are satisfied by the example systems studied in the paper.

The accuracy of MAC/MVA is studied experimentally in section 6 by considering a special kind of multiclass server with "early replies" which is often used to model software processes. This server is also known as server with vacations or a "walking" server [14].

2 Component-based System Model

In this section we will consider in more detail the example system introduced in the first section. Fig. 1 shows how the five components representing three clients C_1, C_2, C_3, a FIFO queue Q and a server S are interconnected. The server has a different service time (exponentially distributed with rate μ_i) for each client C_i. We will call it a CMC (Closed Multi-Class) server.

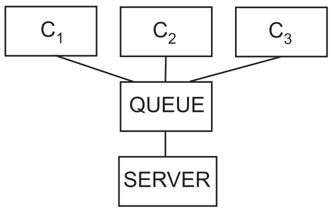


Fig. 1. Example of a component-based system

The stochastic automata of the components are shown in Fig. 2. An automaton can be either in an "active" state (the respective component is driving the system) or in a "passive" state (the component is waiting to be "synchronized" with an event generated by another component). An active state can be either "timed" (it takes a finite duration defined by the component) or "instantaneous" (it takes zero time). A transition triggered at the end of an active state, called "active transition", is labeled with a synchronizing event (shown in bold fonts in Fig. 2). The firing of an active transition may force a

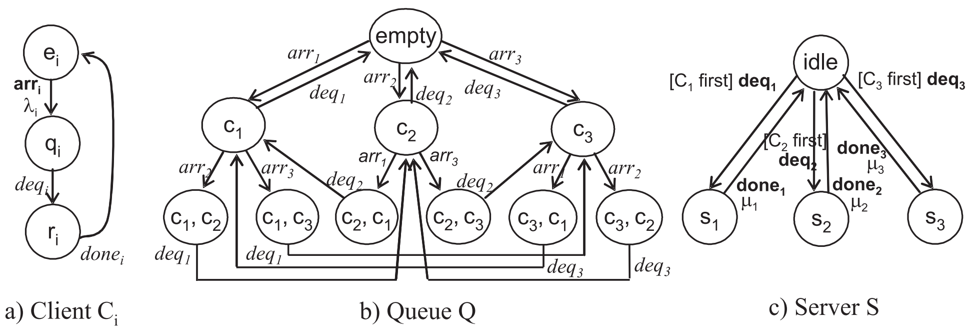
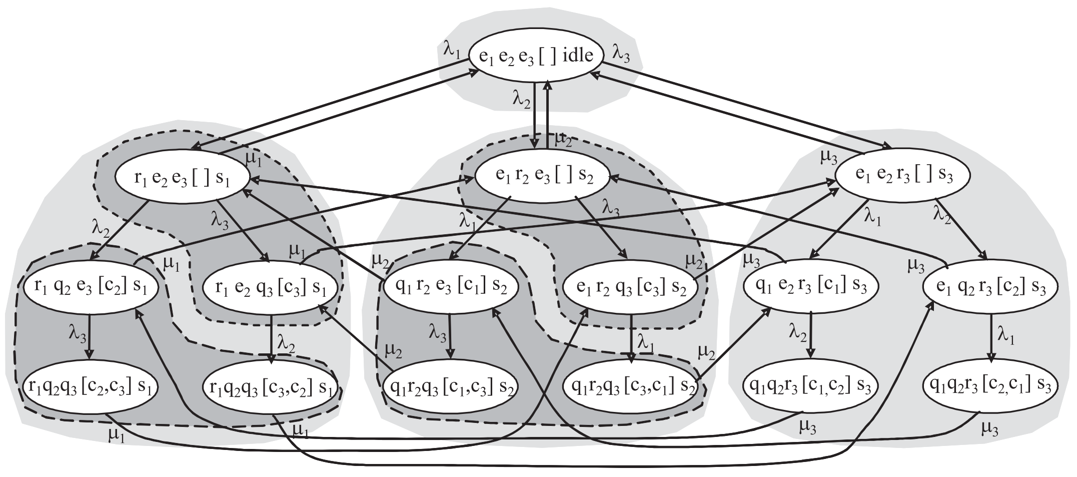


Fig. 2. Stochastic automata defining the behaviour of components in a CMC (Closed Multi-Class) server with three clients

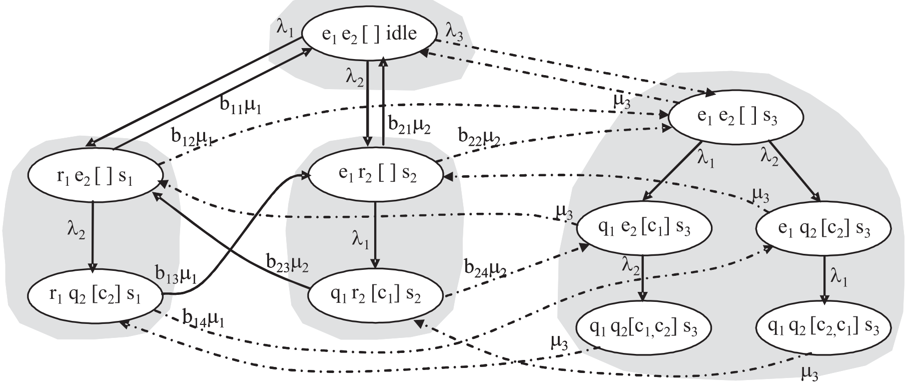
passive transition in another component that is labeled with the same synchronizing event (shown in italic fonts in Fig. 2). Some synchronizing events may be preceded by a "guard", which is a system-wide condition given in square brackets before the event. In our model, the duration of timed active states is assumed to be exponentially distributed. An active transition fired at the end of a timed active state has also a rate parameter governing the rate at which the transition is triggered from the source state (the inverse of the time in the source state). We have adopted some restrictions on the component automata, so that no state has more than one outgoing active timed transition (no timed conflicts inside a component), and when the same transition label appears in more than one component, only one of them is active.

The automaton of client C_i (see Fig. 2.a) can be in one of the following states: e_i (executing on its own or thinking), q_i (queued at the server), and r_i (in service). The transitions are labeled with synchronizing events $\mathbf{arr_i}$ (arrive to the queue), deq_i (dequeue to start service), and $done_i$ (service finished). State e_i is *active*; the client performs some exponentially distributed activity with rate λ_i. When the activity ends, the client produces a synchronizing event $\mathbf{arr_i}$ corresponding to an active transition, which in turn forces a passive transition labeled with the same event in component Q. The other two client states, q_i and r_i, are passive, because their outgoing transitions are driven by the server through the synchronizing events deq_i and $done_i$, rather than by the client itself.

The queue Q stores the clients waiting for service, but not the one in service. All states of Q, shown in Fig. 2.b are passive, being synchronized either with arrival events from the clients or with dequeueing events from the server. The automaton for the server S, shown in Fig. 2.c, has $n = 3$ active service states s_i for $i = 1, 2, 3$, exponentially distributed with rates μ_i. From state *idle*, the server will trigger one of the immediate dequeueing events, but only if the corresponding guard is true. For example, *[C_i first]* $\mathbf{deq}_i$ means that if C_i is the first in queue, then it will be dequeued instantaneously. This is an example



a) Markov Chain $\mathcal{M}$ for the closed multi-class system with a FIFO server and three clients



b) Aggregated Markov Chain $\mathcal{M}'(D)$ for the tagged set $D=\{C_1, C_2, Q, S\}$

Fig. 3. Markov model for the multiclass server system from Fig.1

of an active instantaneous transition.

The composition of components is governed by the sharing of events, as in [8,6] (or equivalently, by the synchronization of transitions with the same labels, as in [15,16]), where the shared events are defined in the interfaces between components. For simplicity in this presentation we shall assume that all events are shared between all components, however a more structured sharing can be defined, as described for example in [6]. When all the automata are composed together, a continuous time Markov chain $\mathcal{M}$ is obtained for the system. For completeness, we should mention that the process of building $\mathcal{M}$ directly by composition, may generate some instantaneous states, which are eliminated similarly to the elimination of “vanishing states” in GSPN [3]. This issue is not discussed in detail in the paper because our method avoids, in fact, the construction of $\mathcal{M}$.

Fig. 3.a shows the Markov model $\mathcal{M}$ for the CMC server with 3 clients from Fig. 1. The notation for a system state $\boldsymbol{\sigma}$ is a tuple containing the corresponding states for each component, with the state of the queue Q shown as an ordered list of queue contents in square brackets. For example, in state $(q_1 q_2 r_3 [c_2, c_1] s_3)$ the first two clients are in queue (the second before the first) and the third is in service. The entire state set of $\mathcal{M}$ is denoted by $\mathbf{\Omega}$.

The Markov chain model for three clients is quite small and can be solved directly. However, for an arbitrary number n of clients the state space size $|\mathcal{M}(n)| = \sum_{i=0}^{n} \frac{(n)!}{(n-i)!}$ grows combinatorially with the number of clients n [9], reaching for instance close to one million states for only nine clients. For this reason, we prefer to avoid building and solving $\mathcal{M}$ directly, and use aggregation instead.

2.1 System-level Performance Relationships

In order to do an MVA analysis, we need throughputs, waiting times and certain arrival instant probabilities. For example, in Fig. 1, the flow rate F_i of C_i arrivals equals the flow rate of C_i departures from the server. This flow rate can be found as:

$$F_i = 1/(\lambda_i^{-1} + w_i + \mu_i^{-1}) \tag{1}$$

Here w_i is the mean queueing delay of client C_i before being served. Since C_i must wait for the client in service and for all the other clients waiting in queue ahead of it, we get:

$$w_i = \sum_{j=1}^{n} (A_{ij}\mu_j^{-1} + B_{ij}\mu_j^{-1}) \tag{2}$$

where the first term gives the time until the client in service completes, and the second term provides the time needed to serve all other clients. Here the A_{ij} and B_{ij} are arrival-instant probabilities defined as follows:

$A_{ij} =$ the probability that C_i arriving to the server finds C_j in service
$B_{ij} =$ the probability that C_i arriving to the server finds C_j in queue.

In general, we will collect all such system-level relationships into a set that typically includes:

- flow identities (flow in = flow out),
- some applications of Little's result
- waiting times, in terms of system-level probabilities.

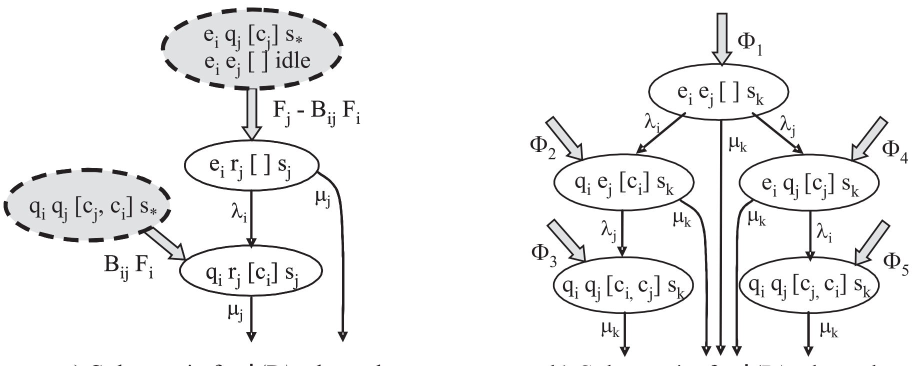
For example, the A and B probabilities can be computed from the aggregated Markov Chain models. At the modeler's discretion, some of these relationships may be approximate.

3 Markov chain Aggregation by Composition (MAC)

The form of the MVA equation for w_i (2) suggests that it is possible to derive the arrival instant probabilities A and B by analysing the interaction of two clients at a time with the server and its queue, ignoring the other clients. Therefore, we will build an aggregated Markov Chain that shows only the states of C_i, C_j, Q and S, which we name "tagged" components. The set of tagged components is denoted by D. For the complete analysis of the system, we shall build $n(n-1)$ such aggregated submodels, one for each pair of clients in the system.

As a general strategy, we propose to obtain aggregated $\mathcal{M}'(D)$ for various sets D, such that each aggregated view brings its contribution to the system mean values, and all the system components and their interactions are covered. A macrostate of the submodel $\mathcal{M}'(D)$ is a 4-tuple $\boldsymbol{\sigma}'$ containing the states of the tagged components from D; $\boldsymbol{\sigma}'$ hides the behaviour of the untagged components from $\bar{D}$. The state set of $\mathcal{M}'(D)$ is denoted by $\boldsymbol{\Omega}'$. Fig. 3.b illustrates the aggregated Markov chain $\mathcal{M}'(D)$ obtained from the model $\mathcal{M}$ in Fig. 3.a for $D = \{C_1, C_2, Q, S\}$. The darker shaded areas in Fig. 3.a define the states which are lumped together in Fig. 3.b. For instance, the states $(r_1 q_2 e_3 [c_2] s_1)$, $(r_1 q_2 q_3 [c_2, c_3] s_1)$ and $(r_1 q_2 q_3 [c_3, c_2] s_1)$, contained within a darker shading in Fig. 3.a, are lumped together to create the macrostate $(r_1 q_2 [c_2] s_1)$ in Fig. 3.b. The aggregation affects also the states of the queue, which will show only the relative position of the two tagged clients, whereas the position of the untagged client is hidden. Even though the state of the untagged client $C_3 \in \bar{D}$ is not shown directly in the aggregated state tuple, and its arrival events are hidden inside the macrostates, the effect of its behaviour is nonetheless represented in $\mathcal{M}'(D)$ indirectly. First, C_3 has an effect on the state space of $\mathcal{M}'(D)$, by the fact that in some macrostates the server is in state s_3 serving C_3 (see the rightmost shaded cluster in the figure). Second, some transitions in $\mathcal{M}'(D)$ (shown with dashed lines in Fig. 3.b) correspond to the beginning/ending of service for C_3 and are the effect of the interaction between the tagged components and C_3.

It is interesting to note that both $\mathcal{M}$ from Fig. 3.a and $\mathcal{M}'(D)$ from Fig. 3.b can be partitioned into $n+1$ subsets of states (enclosed in lightly shaded areas) based on the server state. The subset of $\boldsymbol{\Omega}$ grouping all the states in which the server is serving client C_i is denoted by $\mathcal{G}_i$, and the subset containing the idle state by $\mathcal{G}_0$. Note also that for each subset $\mathcal{G}_i$ in $\boldsymbol{\Omega}$ there is a corresponding



a) Subset $\mathcal{G}'_j$ of $\mathcal{M}'(D)$ where the tagged client C_j is in service

b) Subset $\mathcal{G}'_k$ of $\mathcal{M}'(D)$ where the non-tagged client C_k is in service

Fig. 4. Partitioning $\mathcal{M}'(D)$: subsets of macrostates

subset $\mathcal{G}'_i$ in $\mathbf{\Omega}'$.

Even though $\mathcal{M}'(D)$ can be obtained from $\mathcal{M}$ by aggregation, we intend to avoid the expensive process of constructing the entire Markov chain and lumping its states. We propose instead to build $\mathcal{M}'(D)$ by composing the tagged components, and to add the missing pieces due to the effect of the untagged component behaviour. The "aggregation by composition" proposed in this paper differs from the "compositional aggregation" from [6] in that we will consider all the states of the components in D, even if they are reached by interactions with untagged components. For example, if we were to consider a direct composition of the tagged components $D = \{C_1, C_2, Q, S\}$, we would obtain only the subsets $\mathcal{G}'_1$ and $\mathcal{G}'_2$ of $\mathcal{M}'(D)$, but would never reach the subset $\mathcal{G}'_3$ where C_3 is in service. We shall add this subset, and the transitions between it and the other subsets. In general, in a system with n clients, only 2 are tagged and $n-2$ are untagged. When building the aggregated submodels by composition, we shall add the subsets for all the untagged clients, in order to capture the relationship between the arrivals of the two tagged clients when S is serving other clients (needed for the derivation of the arrival instant probabilities).

The next section will describe how we deal with different transition rates in $\mathcal{M}'(D)$, some of which are known and others unknown.

3.1 Partitioned Submodels

Rather than determining individual unknown rates in the aggregated submodels, it is convenient to further simplify the submodels by partitioning them into subsets such that the aggregated transition rates inside each subset are known (but may be unknown between subsets). Then we will determine aggregated transition flows coming to/leaving from the states of a subset from/to outside the subset.

As mentioned before, for n clients, an aggregated submodel $\mathcal{M}'(D)$ is partitioned into n subsets $\mathcal{G}'_j$ for $j = 1, \ldots, n$ (where client C_j is in service) and one subset $\mathcal{G}'_0$ (where the server is idle).

The importance of this partitioning resides in the fact that a subset $\mathcal{G}'_k$ has the same structure and size independent of the number of clients n [9]. The form of a subset depends only on whether the client in service is tagged or untagged, as shown in Fig. 4.a and 4.b, respectively. Since $\mathcal{M}'(D)$ contains $n+1$ subsets for n clients, its size grows only linearly with n despite the fact that the original model $\mathcal{M}$ grows combinatorially with n. This is the basic reason for the proposed AMVA algorithm having a lower complexity (as shown in section 5.4) than the direct solution of $\mathcal{M}$.

The balance equations for the subsets $\mathcal{G}'_k$ are used to derive the solutions for the arrival instant probabilities A and B. It should be emphasized that partitioning is a convenience, which has been used in this and other cases, but this work does not present a general procedure for partitioning. In writing the balance equations for a subset, the transitions into a subset state from other subsets are represented as an aggregated in-flow rate, (shown in Fig. 4.a and 4.b by thick grey arrows). They indicate in-flows with rates in units of transitions/sec. There are also out-flows indicated by ordinary transitions terminating outside the subset. In the example system considered here, the subsets have the following properties (proved in [9]) that lead to a mean value solution.

Property 1. In any subset $\mathcal{G}'_k$, the transition rates corresponding to "arrival of C_i", and "arrival of C_j" events are given by λ_i and λ_j, respectively. Also, for each macrostate in which the server is serving a client C_k, the sum of rates over all outgoing transitions corresponding to "end of service" events is constant and equal to μ_k.

Property 2. The in-flows to a subset $\mathcal{G}'_j$ where a tagged client C_j is in service, shown in Fig. 4.a can be expressed exactly in terms of system-level mean values (i.e., arrival-instant probabilities and system throughputs) as follows:

$$Inflow(q_i r_j [c_i] s_j) = B_{ij}\ F_i \tag{3}$$

$$Inflow(e_i r_j [\,] s_j) = F_j - B_{ij}\ F_i \tag{4}$$

Symmetrical expressions exist for the in-flows to $\mathcal{G}'_i$.

A sketch of the proof for (3) is that the input flow to $(q_i\ r_j\ [c_i]\ s_j)$ comes from different macrostates of the form $(q_i\ q_j\ [c_j,\ c_i]\ s_k)$ for any k, in which

C_j was queued ahead of C_i. Retracing back to the moment when C_i arrived to the queue, this means that C_i found C_j already in the queue. Moreover, due to flow conservation, all C_i arrivals that have found C_j queued ahead of them, are still queued when the service for C_j begins by entering the state $(q_i\ r_j\ [c_i]\ s_j)$. From the arrival-instant probability definitions, the frequency of all possible C_i arrivals that find C_j in the queue equals $B_{ij}F_i$.

The proof for the second relationship (4) is based on the fact that the total input flow to the subset $\mathcal{G}'_j$ equals the frequency of F_j arrivals to the server.

Unfortunately, the in-flows of a subset $\mathcal{G}'_k$ where an untagged client C_k is in service, shown in Fig. 4.b, cannot be expressed exactly in terms of system mean values, so some approximations will be used instead.

3.2 Arrival instant probabilities equations from aggregated Markov submodels

The arrival-instant probabilities A and B used in the queueing delay equation (2) can be expressed in terms of steady-state solution of $\mathcal{M}'(D)$, for $D = \{C_i, C_j, Q, S\}$, as the ratio between the frequency of C_i arrivals occurring in some specific states over the total frequency F_i of C_i arrivals. More exactly, A_{ij} in (5) is the ratio of the frequency of C_i arrivals that find S serving another client C_j (computed as the occurrence rate of the transitions leaving all the states $\boldsymbol{\sigma'} \in \boldsymbol{\Omega}'_A$ with rate λ_i) over the frequency F_i:

$$A_{ij} = \sum_{\boldsymbol{\sigma}' \in \boldsymbol{\Omega}'_A} \lambda_i\, \mathcal{P}(\boldsymbol{\sigma'})/F_i \tag{5}$$

where $\boldsymbol{\Omega}'_A$ is the subset of $\mathcal{M}'(D)$ states of the form $(e_i \sigma_j \sigma_Q s_j)$ for any σ_j and σ_Q, and $\mathcal{P}(\boldsymbol{\sigma'})$ is the steady-state probability that $\mathcal{M}'(D)$ is in state $\boldsymbol{\sigma'}$.

Similarly, B_{ij} in (6) is the ratio of the frequency of C_i arrivals that find another client C_j in queue (computed as the occurrence rate of the transitions leaving all the states $\boldsymbol{\sigma'} \in \boldsymbol{\Omega}'_B$ with rate λ_i) and the frequency F_i.

$$B_{ij} = \sum_{\boldsymbol{\sigma}' \in \boldsymbol{\Omega}'_B} \lambda_i\, \mathcal{P}(\boldsymbol{\sigma'})/F_i \tag{6}$$

where $\boldsymbol{\Omega}'_B$ is the subset of $\mathcal{M}'(D)$ states of the form $(e_i q_j \sigma_Q \sigma_S)$ for any σ_Q and σ_S.

The probability A_{ij} defined in (5) can be obtained from the balance equations for the states of the subset $\mathcal{G}'_j$, where task $C_j \in D$ is in service. Due to *Property*

2, the two balance equations can be written as:

$$\mu_j \, \mathcal{P}(q_i \; r_j \; [c_i] \; s_j) = B_{ij} \; F_i + \lambda_i \; \mathcal{P}(e_i r_j \; [\,] \; s_j) \tag{7}$$

$$(\mu_j + \lambda_i) \; \mathcal{P}(e_i r_j \; [\,] \; s_j) = F_j - B_{ij} F_i \tag{8}$$

The first arrival probability equation (9) is obtained by a little algebraic manipulation from (5), (7) and (8):

$$(\mu_j/\lambda_i + 1) A_{ij} + B_{ij} = F_j/F_i \qquad \text{for } \; i,j = 1,\ldots,n; \;\; i \neq j \tag{9}$$

Since arrivals from C_i that find C_j in queue happen in all subsets $\mathcal{G}'_k$ for all $k \neq i,j$, the probability B_{ij} is computed by summing up its components $B_{ij,k}$:

$$B_{ij} = \sum_{k \neq i,j} B_{ij,k} \qquad \text{for } \; i,j,k = 1,\ldots,n; \;\; i \neq j; \;\; k \neq i,j \tag{10}$$

where $B_{ij,k}$ is the arrival-instant probability of C_i finding C_j in queue and C_k in service. By definition, similar to (5) and (6) we have:

$$B_{ij,k} = \sum_{\boldsymbol{\sigma}' \in \boldsymbol{\Omega}'_b} \lambda_i \; \mathcal{P}(\boldsymbol{\sigma}')/F_i \tag{11}$$

where $\boldsymbol{\Omega}'_b$ is the subset of $\mathcal{M}'(D)$ states of the form $(e_i q_j \sigma_Q s_k)$ for any σ_Q.

Similarly, we can define $\bar{B}_{ij,k}$ as the probability that a request from C_i arriving when S is serving C_k, does not find C_j either in queue or in service (i.e., C_j is executing in state e_j):

$$\bar{B}_{ij,k} = \lambda_i \mathcal{P}(e_i \; e_j \; [\;\;] \; s_k)/F_i \tag{12}$$

If we can compute $\bar{B}_{ij,k}$ and write A_{ik} by applying definition (5) to the states of $\mathcal{G}'_k$ in Fig. 4.b:

$$A_{ik} = \lambda_i(\mathcal{P}(e_i \; e_j \; [\;\;] \; s_k) + \mathcal{P}(e_i \; q_j \; [c_j] \; s_k))/F_i \tag{13}$$

then $B_{ij,k}$ can be found from the difference. To compute $\bar{B}_{ij,k}$ an approximation is needed, because the in-flows to the subset $\mathcal{G}'_k$ cannot be expressed exactly in terms of mean values. We make an independence assumption regarding the arrivals from C_i and C_j when C_k is in service. More exactly, we assume that when C_k is in service, the probability that C_i is in state e_i (when arrivals occur) is independent of the fact that C_j is executing or is in queue.

$$P(e_i | e_j s_k) = P(e_i | s_k) \tag{14}$$

By the general multiplication rule we can write:

$$P(e_i e_j s_k) = P(e_i | e_j s_k)\ P(e_j | s_k)\ \mathcal{P}(s_k) \tag{15}$$

By replacing (14) in (15) we obtain:

$$\begin{aligned} P(e_i e_j s_k) &= P(e_i | s_k)\ P(e_j | s_k)\ \mathcal{P}(s_k) = \\ &= \frac{\mathcal{P}(e_i s_k)\ \mathcal{P}(e_j s_k)}{\mathcal{P}(s_k)} \end{aligned} \tag{16}$$

$\mathcal{P}(e_i s_k)$ can be expressed, by using (9), as follows:

$$\mathcal{P}(e_i s_k) = \mathcal{P}(e_i e_j s_k) + \mathcal{P}(e_i q_j [c_j] s_k) = F_i \lambda_i^{-1} A_{ik} \tag{17}$$

There is a similar expression for $\mathcal{P}(e_j s_k)$:

$$\mathcal{P}(e_j s_k) = F_j \lambda_j^{-1} A_{jk} \tag{18}$$

$\mathcal{P}(s_k)$ represents the lumping of all macrostates from $\mathcal{G}'_k$, thus:

$$\mathcal{P}(s_k) = F_k \mu_k^{-1} \tag{19}$$

From (16), (17), (18), (19) and definition (12) for $\bar{B}_{ij,k1}$ we obtain:

$$B_{ij,k} = A_{ik} - A_{i,k} A_{j,k} (F_j \lambda_j^{-1}) / (F_k \mu_k^{-1}) \tag{20}$$

$$\text{for } \ i, j, k = 1, \ldots, n; \ \ i \neq j; \ \ k \neq i, j$$

The equations for the arrival-instant probabilities A and B of the CMC system are (9), (10) and (20). They will be solved iteratively, together with the mean value equations for F_i (1) and w_i (2) by the approach of simultaneous displacements (analogous to Jacobi's method), similar to the algorithm described in section 5.4.

4 Summary of Steps in MAC/MVA

The MAC/MVA approach has six major steps:

a) *Mean value breakdown at the system level.* At the level of system components and flows of tokens, representing customers, messages, requests etc., identify a set P of average performance measures and a set R of relationships between them. The relationships in R are (typically) flow and delay identities, and mean value delay equations.

b) *Define state-transition models for components.* For each component, create a model in which transitions are labeled with synchronizing events and (in some cases) with rate parameters.

c) *Create the MAC submodels.* Define sets $D_1, D_2, \ldots$ of components as a basis of aggregation, chosen to provide estimates of the measures in P. Form the composed state-transition model $ST(D)$ for each set D. From this, create the aggregated submodel $\mathcal{M}'(D)$, with "UNK" transition rates representing interactions with components not included in D.

d) *Partition the MAC submodels* so that the partitions (subsets) have known transition rates inside, but may have unknown transition rates between partitions. These rates are estimated by either exact or approximate mean-value considerations.

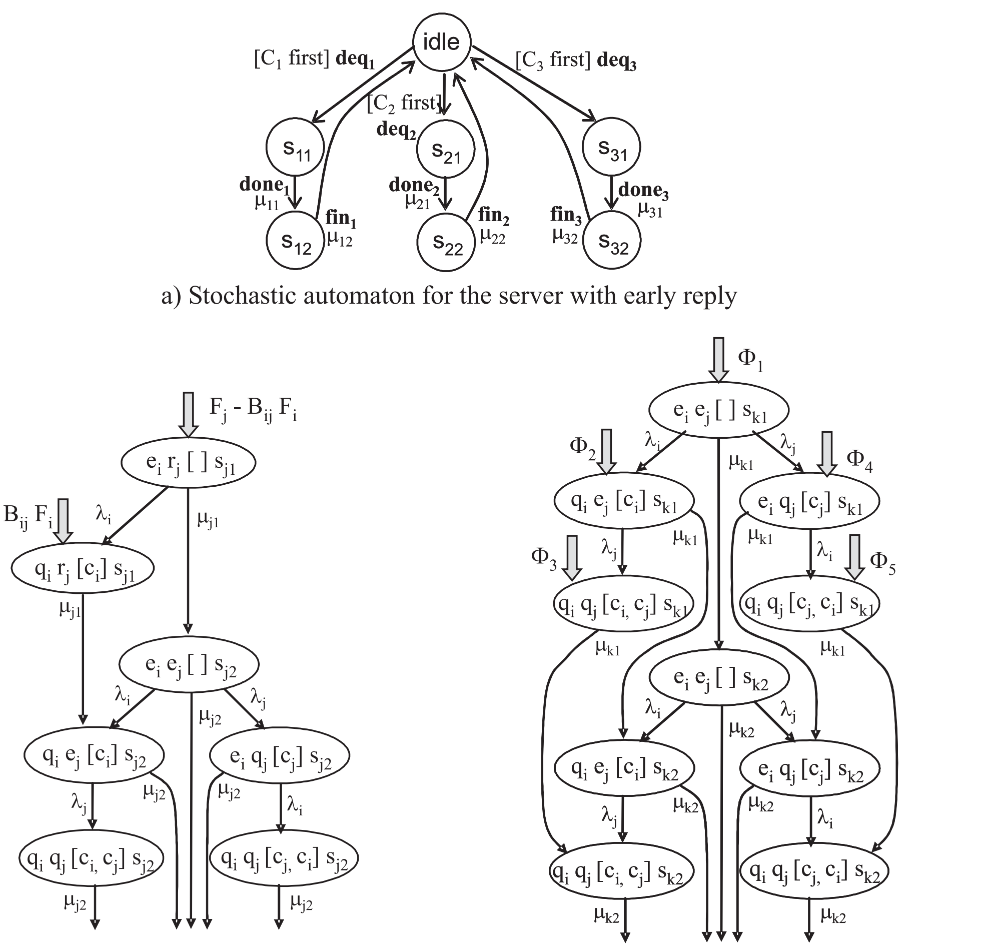
e) *Solve each partition or each submodel analytically* for its mean performance measures, in terms of its parameters. This gives a set of equations for each submodel for a vector of measures which may also depend on measures from other submodels.

f) *Collect together the MVA equations,* consisting in general of mean-value equations from each submodel, and system-level relationships and solve these equations, for instance by fixed point iteration.

5 MAC/MVA for a multi-class FIFO server with early reply (CMC-ER)

The MAC/MVA strategy will be applied to a more complex queueing system representing a kind of server found in software systems such as web services systems [18,4,13]. The clients are software tasks such as browsers running in workstations, or applications in network servers. The n clients send request messages to a certain server. The service offered to each client starts by executing a so-called *first phase of service*, after which the server replies to the client. The performance optimization of this type of server, however, has led to sending the reply *early*, before all the work of the server is completed; the remaining work is called the *second phase*, and must be performed before the next client request can be handled, as described in [4]. We will call this example a Closed Multi-Class server with Early Reply (CMC-ER). Servers with early replies are common in practice. The second phase work includes delayed writes to storage, logging, billing, buffer cleanup, and preparing the server for the next request.

The components of this system are the same as shown in Fig. 1, except that the server has two phases of service which are exponentially distributed with



a) Stochastic automaton for the server with early reply

b) Subset $\mathcal{G}'_j$ of $\mathcal{M}'(D)$ for the server with early reply where the tagged client C_j is in service

c) Subset $\mathcal{G}'_k$ of $\mathcal{M}'(D)$ for the server with early reply where the non-tagged client C_k is in service

Fig. 5. Models for the CMC-ER server with early reply

rates μ_{i1}, and μ_{i2}. The client automaton is unaffected, showing that the client returns to the state e_i after receiving its reply; the queue is also the same. The server model is now as shown in Fig. 5.a.

This server was studied first in [14] under the name of "walking server", and in other work is called a "server with vacations". In software systems the second phase is not due to gaps in operation, but it represents working time deliberately introduced to increase the concurrency in the system. There is no closed-form solution for the multiclass closed "walking server", so analysis of systems with these servers must use numerical approximations; some of these approximations were described in [18,4].

The performance measures F_i and w_i are similar to those defined for the previous example. As before, the throughput F_i of the client C_i is given by (1), but with μ_i replaced by μ_{i1}. The waiting time is modified to include the effects of the second phase as follows:

$$w_i = \sum_{j=1}^{n} [A_{i,j1}(\mu_{j1}^{-1} + \mu_{j2}^{-1}) + A_{i,j2}\ \mu_{j2}^{-1} + B_{ij}(\mu_{j1}^{-1} + \mu_{j2}^{-1})] \tag{21}$$

where A and B are arrival-instant probabilities; B_{ij} is defined as for CMC, and $A_{i,jp}$ is defined as:

$A_{i,jp} =$ the probability that a request from C_i arriving to S finds it busy in phase p serving a request from another client C_j (i.e., rate of arrivals of C_i when server S is in state s_{ip} over F_i)

Thus we can write a slightly different definition for A, including the phases, whereas (6) is still valid for B:

$$A_{i,jp} = \sum_{\boldsymbol{\sigma}' \in \boldsymbol{\Omega}'_A} (\lambda_i\ \mathcal{P}(\boldsymbol{\sigma}')/F_i \tag{22}$$

where $\boldsymbol{\Omega}'_A$ is the subset of $\mathcal{M}'(D)$ states of the form $(e_i \sigma_j \sigma_Q s_{jp})$ for any σ_j and σ_Q. The following relationship follows immediately from the definition of $A_{i,jp}$:

$$\sum_{j=1}^{n} \sum_{p=1}^{2} A_{i,jp} \leq 1 \tag{23}$$

The derivation of B_{ij} is similar to the CMC example, but modified to account for second phases. A C_i arrival may find S completing a second phase started earlier by itself (state s_{i2}). By the same arguments as used above we can derive:

$$B_{ij} = B_{ij,i2} + B_{ij,j2} + \sum_{k \neq i,j} \sum_{p} B_{ij,kp} \tag{24}$$

$$\text{for}\ \ i, j, k = 1, \ldots, n; \ \ i \neq j; \ \ k \neq i, j$$

where $B_{ij,kp}$ is the arrival-instant probability of finding C_j in the queue and C_k in service in phase p, defined as:

$$B_{ij,kp} = \sum_{\boldsymbol{\sigma}' \in \boldsymbol{\Omega}'_b} \lambda_i\ \mathcal{P}(\boldsymbol{\sigma}')/F_i \tag{25}$$

where $\mathbf{\Omega}'_b$ is the subset of $\mathcal{M}'(D)$ states of the form $(e_i q_j \sigma_Q s_{kp})$ for any σ_Q.

5.2 Markov model for CMC-ER: aggregation and partitioning

The aggregation of the Markov model may be performed similarly to the CMC system. For the general case with n clients, the aggregated submodel $\mathcal{M}'(D)$ for $D = \{C_i, C_j, Q, S\}$ has $n+1$ subsets, as follows:

- subset $\mathcal{G}'_0$ containing the state in which the server S is idle;
- subsets $\mathcal{G}'_i$ and $\mathcal{G}'_j$ containing the states in which S is serving the tagged client C_i and C_j, respectively; each such subset contains seven aggregated states, as shown in Fig. 5.b;
- subsets $\mathcal{G}'_k$, for $k = 1, \ldots, n$ and $k \neq i, j$ containing states in which S is serving an untagged client $C_k \in \bar{D}$; each subset $\mathcal{G}'_k$ contains ten states, as shown in Fig. 5.c.

The in-flows are defined as for the CMC system. The reasoning to determine A is exactly the same as in CMC, except that $A_{i,i2}$ is not zero, because C_i can overtake its own previously-initiated second phase of service. B_{ij} will be derived by summing its components over the various server states as in (24).

An auxiliary arrival-instant probability used in the derivation process is the probability $\bar{B}_{ij,kp}$ that a request from C_i arriving when S is serving C_k in phase p, does not find C_j either in queue or in service. The following relation is immediately obtained from the probability definitions:

$$A_{i,kp} = B_{ij,kp} + \bar{B}_{ij,kp} \tag{26}$$

5.3 Arrival-instant probability equations for CMC-ER

In this paper, the balance equations of the subsets $\mathcal{G}'$ from Fig 5.b and 5.c are solved analytically to derive the arrival-instant probability equations, by using the definitions (22), (24), (25) to eliminate the macrostate probabilities. The in-flow rates are calculated as for the CMC case. However, it is worth mentioning that another possible solution approach with the same complexity is to solve numerically the balance equations for the subsets $\mathcal{G}'_i$, $i = 1, \ldots, n$ of $\mathcal{M}'(D)$, then to apply the definitions (22), (24), (25) for computing the arrival-instant probabilities A and B.

The set of simultaneous equations for the arrival-instant probabilities are listed here. All equations are exact, except equation (32) that is the only approximation.

$$A_{i,i2} = \frac{\lambda_i}{\lambda_i + \mu_{i2}} \qquad \text{for } \; i = 1, \ldots, n \tag{27}$$

$$(\frac{\mu_{j1}}{\lambda_i} + 1)A_{i,j1} + B_{ij} = \frac{F_j}{F_i} \qquad \text{for } \; i, j = 1, \ldots, n; \;\; i \neq j \tag{28}$$

$$A_{i,j2} = \frac{\mu_{j1}}{\lambda_i + \mu_{j2}} A_{i,j1} \qquad \text{for } \; i, j = 1, \ldots, n; \;\; i \neq j \tag{29}$$

$$B_{ij,i2} = \frac{\lambda_i}{\lambda_i + \mu_{i2}} \cdot \frac{F_j}{F_i} \left[B_{ji} + (1 + \frac{\mu_{i1}}{\lambda_i + \lambda_j + \mu_{i2}})A_{j,i1}\right] \tag{30}$$

$$\text{for } \; i, j = 1, \ldots, n; \;\; i \neq j$$

$$B_{ij,j2} = \frac{\lambda_j}{(\lambda_i + \lambda_j + \mu_{j2})} A_{i,j2} \qquad \text{for } \; i, j = 1, \ldots, n; \;\; i \neq j \tag{31}$$

$$B_{ij,k1} = A_{i,k1} - \frac{F_j \lambda_j^{-1}}{F_k \mu_{k1}^{-1}} A_{i,k1} A_{j,k1} \tag{32}$$

$$\text{for } \; i, j, k = 1, \ldots, n; \;\; i \neq j; \;\; k \neq i, j$$

$$B_{ij,k2} = A_{i,k2} - \frac{\mu_{k1}}{\lambda_i + \lambda_j + \mu_{k2}} (A_{i,k1} - B_{ij,k1}) \tag{33}$$

$$\text{for } \; i, j = 1, \ldots, n; \;\; i \neq j; \;\; k \neq i, j$$

$$B_{ij} = B_{ij,i2} + B_{ij,j2} + \sum_{k \neq i,j} \sum_{p} B_{ij,kp} \tag{34}$$

$$\text{for } \; i, j, k = 1, \ldots, n; \;\; i \neq j; \;\; k \neq i, j$$

Equations (32) and (33) are used only when $n > 2$. Thus, for two clients the solution is exact.

5.4 MAC/MVA algorithm for CMC-ER

Equations (1) for F_i, (2) for w_i and (27)–(34) for the arrival-instant probabilities represent a set of nonlinear equations that are solved iteratively by the approach of simultaneous displacements (analogous to Jacobi's method) as follows:

a) *Initialize all F_i, B_{ij} with some feasible values;*
b) *Compute new values for $A_{i,i2}$ (eq. 27), $A_{i,j1}$ (eq. 28), $A_{i,j2}$ (eq. 29), $B_{ij,kp}$ (eqs. 30–33), B_{ij} (eq. 34);*

c) *Update the arrival-instant probabilities using an under-relaxation strategy, i.e.* $prob_{new} = \gamma(prob_{old} + prob_{computed})$
d) *Determine new values* w_i *(eq. 2) and* F_i *(eq. 1);*
e) *Repeat steps b), c) and d) until the total change in the arrival-instant probabilities values is less than a given tolerance.*

In our experience, we found that a value of 0.5 for the under-relaxation coefficient γ worked the best.

Complexity. The number of equations for the arrival-instant probabilities used in *step (b)* of the algorithm depends on the number of clients n as follows:

- n equations of form (27)
- $n(n-1)$ equations of each of the forms (29)–(31), (34)
- $n(n-1)(n-2)$ equations of each of the forms (32), (33)

The reduction in the order of computational complexity of the MAC/MVA algorithm compared to the complexity of the exact solution is a consequence of the approximation used by the algorithm, which reduces the problem of building and solving a Markov chain with $\mathbf{O}(n!)$ states to the problem of solving iteratively a system of $\mathbf{O}(n^3)$ nonlinear equations.

Complexity when clients are grouped into classes. The previous algorithm, derived for the case where each client is different from the others, can be easily generalized for the case where the clients can be grouped into classes, such that each class contains statistically identical clients, and in total there are c client classes. By symmetry, the number of arrival-instant probability equations is reduced to $\mathbf{O}(c^3)$ from $\mathbf{O}(n^3)$, depending on the number of classes instead of the number of clients.

No theoretical proof has been found for the uniqueness of the solution or the convergence of the MAC/MVA algorithm. However, the algorithm was applied to several hundred models in order to assess its accuracy and convergence, as presented in section 6.

6 MAC/MVA Experimental results for CMC-ER

The accuracy of the MAC/MVA algorithm introduced in section 5.4 was investigated by comparing its results with exact results for smaller models (up to seven clients) and with simulation results for larger models. The exact solutions were obtained with the GreatSPN package for Generalized Stochastic Petri Nets [3]. The simulations results were obtained with 99% confidence interval of less than $\pm 0.5\%$.

The following factors were found to affect the accuracy of MAC/MVA algorithm: the achieved server utilization, the imbalance between server entries, the imbalance between clients, and the number of clients. Two imbalance factors are defined:

R_s = *server imbalance ratio*, the ratio between the longest and the shortest service time among all server entries (taken as the sum over the two phases)
R_c = *client imbalance ratio*, the ratio between the longest and the shortest client execution time.

Several test suites were designed to study the impact of different factors on accuracy. Each test suite generates a curve plotted in one of Figures 6–9, and contains a set of cases (models) with the same imbalance R_s and R_c, as follows:

- the server is the same for all cases in a suite (the same number of entries and the same service times)
- the client service times are varied from case to case with the same proportionality factor, giving the same R_c for all cases.

The following percentage errors were plotted in function of the achieved server utilizations for various test suites:

- *Average Throughput Error (ATE):* the average relative error in absolute value of all clients throughputs $ATE = \sum_{i=1}^{n} e_i/n$, where the "relative error in absolute value" e_i of client i throughput was computed as the difference between the approximate and exact throughput in absolute value, expressed as a percentage of the exact throughput $e_i = 100\ |(approx_i - exact_i)|/exact_i$.
- *Maximum Throughput Error (MTE)* the highest relative error in absolute value among the client throughputs $MTE = max(e_1, e_2, \ldots, e_n)$.

In all cases, the errors are small when the server is lightly utilized, grow with the utilization to a peak between 0.85 and 0.95 %, and then become smaller when approaching saturation. This implies that the independence assumption made in section 3.2 that the arrivals of C_i are independent on whether another client is executing or is in queue works better at a lighter load. However, when the server is highly utilized, the clients spend more time in the queue and some dependencies between client behaviour seem to manifest themselves in the system. As the system has a FIFO queue, it is not surprising that the shorter clients are more affected than the longer ones (the former are more dependent on the later).

The first experiment studies the impact on accuracy of various imbalance ratios for test suites with 7 clients, as shown in Figures 6 and 7. The worst case was found to be a combination of server and client imbalance, where the service times of clients and server entries grow linearly from the shortest to the longest, and the shorter client calls the shorter entry, etc. (The service time

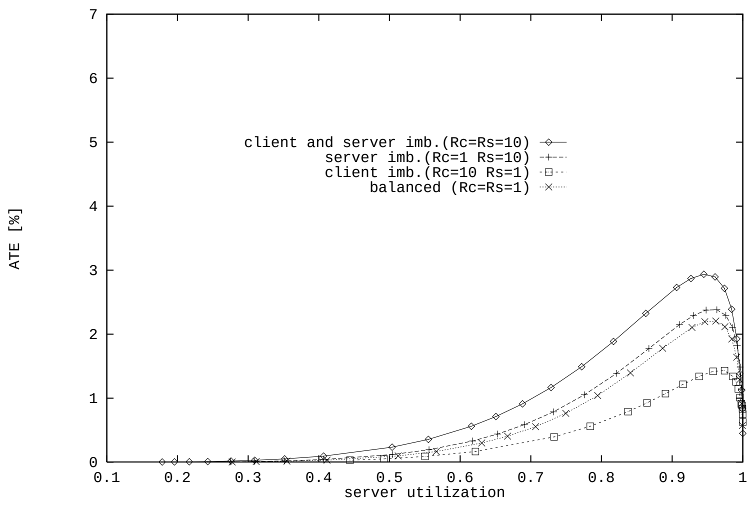


Fig. 6. ATE in function of server utilization for unbalanced and balanced test cases with 7 clients

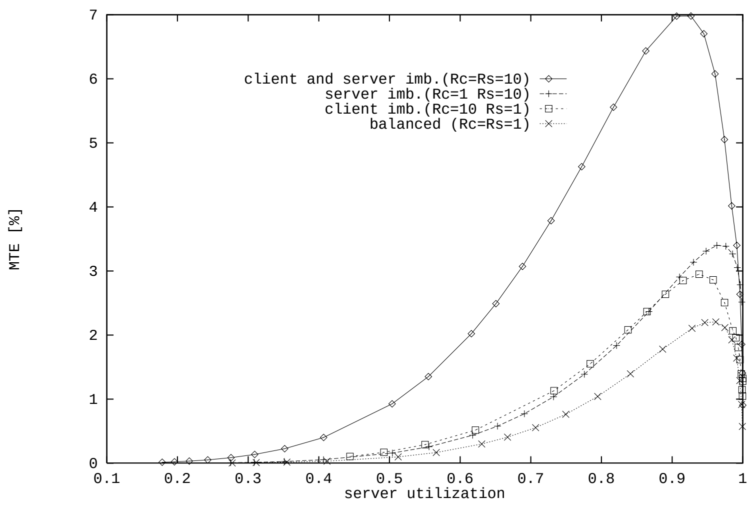


Fig. 7. MTE in function of server utilization for unbalanced and balanced test cases with 7 clients

of each entry is equally split between phases). The largest error corresponds to the shortest client. The balanced suite fares best in terms of average error ATE, and, with a single exception in terms of maximum error MTE. When comparing various degrees of client and server imbalance, where both R_c and R_s are simultaneously varied from 1 to 10, we observed (as expected) that a larger imbalance produces a larger error.

The number of clients has also a strong impact on accuracy. Figures 8 and 9 show the ATE and MTE in function of server utilization for unbalanced test suites with 3, 5, 7, 14, 21 and 28 clients, respectively (all with imbalance $R_c = R_s = 10$). The errors grow with n, but at a decreasing rate, so a given increase in the number of clients has a stronger impact for cases with a few clients than with many clients. This is to be expected since, in general, independence approximations such as the one used for equation (32) work better for large numbers of clients. We can conclude that the MAC/MVA algorithm works reasonably well even for a large number of clients.

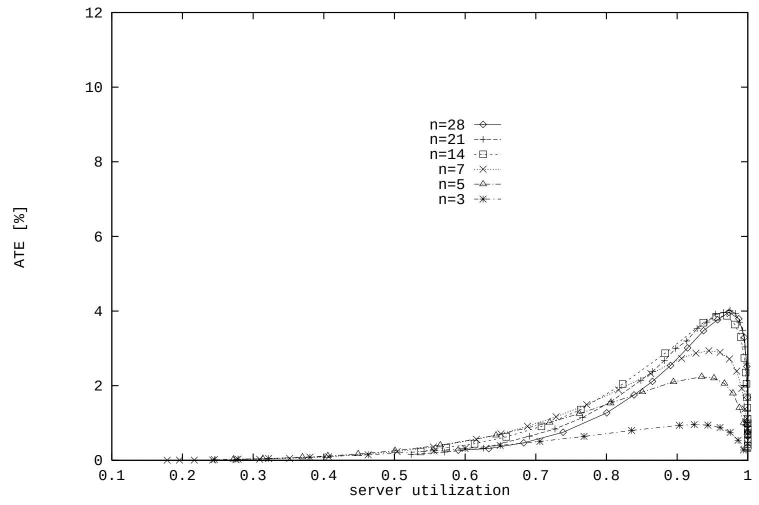


Fig. 8. ATE in function of server utilization for unbalanced test cases with different numbers of clients

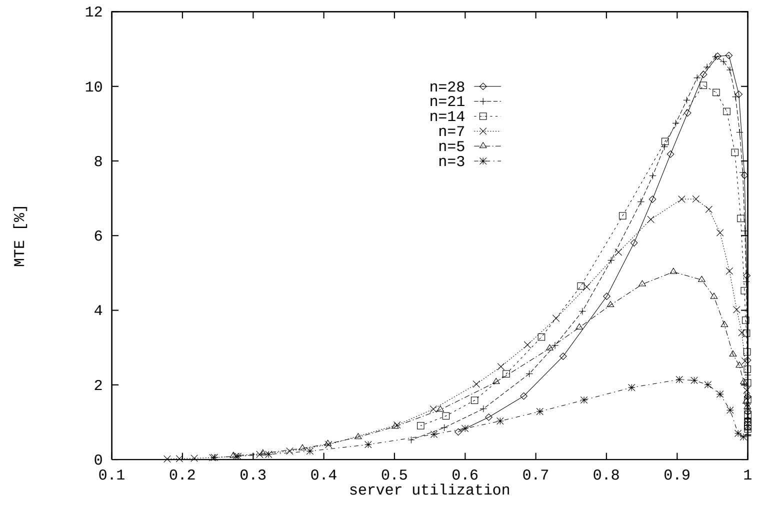


Fig. 9. MTE in function of server utilization for unbalanced test cases with different numbers of clients

The overall complexity of the MAC/MVA algorithm is dependent not only on the computational complexity of each iteration (which is $\mathbf{O}(n^3)$, as shown in section 5.4), but also on the number of iterations required. It is quite difficult to predict the number of iterations necessary for the solution of a given model. Experiments have shown that the feasible values chosen for initialization in the first step of the algorithm do not have an important impact on the final results. As the iteration progresses, the intermediate values for throughputs and arrival-instant probabilities are not guaranteed to remain feasible at any time. However, the experiments have shown that they do converge toward feasible values. We have observed that enforcing the relationship (23) when the sum from the first term becomes greater than 1 has a positive effect on the convergence, especially when the server is close to saturation. Some cases of oscillatory convergence have been observed for test cases with a very saturated server, when the nonlinear system of equations becomes ill-conditioned. This is not surprising, since the linear system of balance equations of the MC model becomes ill-conditioned itself at very high levels of saturation. From experience, the convergence of the MAC/MVA algorithm is obtained quickly (15 to 30 iterations) for the cases where the server is not saturated, but the number of iterations grows when the server approaches saturation. This phenomenon is stronger in unbalanced systems.

For seven clients (the largest model solved exactly with the GreatSPN package), the solution time was about two orders of magnitudes faster for the MAC/MVA algorithm, than for the numerical solution of the system Markov model $\mathcal{M}$ by the GreatSPN package.

7 Conclusions

The compositional approach described here for creating aggregated submodels avoids on one hand the effort of building the whole Markov model of the system, and on the other hand, the effort of aggregating from a very large state space. This has been applied earlier to exact aggregation for lumpable systems, essentially corresponding to systems with symmetries (references are given in [6]). The innovation here is to:

- create ad hoc approximate aggregated models for different component based systems,
- give a systematic approach to making the analysis simpler and more scalable, by partitioning the submodels even after aggregation,
- give a systematic approach to generating the approximation as a Mean Value Analysis, by combining solutions of the submodels with system level mean value relationships.

Because the solution is found through mean values, equilibrium state probabilities do not need to be computed. Some modeling judgement is required to complete the aggregated submodels and find the mean value relationships. More than one approximation can undoubtedly be found. Clearly it is easier to find the mean value equations if the submodel partitions are small and repetitive.

Accuracy is adequate, as shown in Section 6. For the CMC server in Section 3, without a second phase, results not included here showed somewhat better accuracy.

The examples shown here can easily be generalized to include classes with more than a single client, and to servers with priorities and other kinds of queueing discipline. The model for a single server has also been embedded in a network of servers, with iteration among the servers, to solve layered queueing problems [11,12]. The approach has been applied to systems with collections of similar components (the clients here), and this makes the solutions simpler, but in principle the approach applies to heterogeneous systems as well.

There is promise in this work for a general scalable technique for approximate numerical analysis of all kinds of systems defined by composition of components, using process algebras, stochastic automata or composable Petri Nets.

Acknowledgments

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC). The authors would like to thank the anonymous reviewers of various versions of the paper and the Special Editor for their helpful comments and suggestions.

References

[1] J.Banks, J.S. CarsonII, B.L. Nelson, D.M. Nicol, *Discrete-Event System Simulation*, 3rd Edition, p.72, Prentice Hall, 2000.

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

[3] G. Chiola, M. A. Marsan, G. Balbo, and G. Conte, “Generalized Stochastic Petri Nets: A Definition at the Net Level and Its Implications”, *IEEE Transactions on Software Engineering*, vol. 19, no. 2 pp. 89-107, February 1993.

[4] G. Franks, M. Woodside, ”Effectiveness of early replies in client-server systems,” *Performance Evaluation*, vol. 36–37, pp. 165-183, August 1999.

[5] S. Gilmore, J. Hillston, ”The PEPA Workbench: A Tool to Support a Process Algebra Based Approach to Performance Modelling,” in Proc. Seventh

Intern. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, Vienna, 1994.

[6] H. Hermanns, *Interactive Markov Chains*, LNCS. Vol. 2428, Springer, Berlin, 2002.

[7] H. Hermanns, U. Herzog, U. Klehmet, M. Seigle, and V. Mertsiotakis, "Compositional Performance Modeling with the TIPPtool", *Performance Evaluation*, vol. 39, no. 1-4 pp. 5 - 35, 2000.

[8] J. Hillston, *A Compositional Approach to Performance Modelling* Cambridge University Press, Cambridge,1996.

[9] D. C. Petriu, *Approximate Solution for Stochastic Rendezvous Networks by Markov Chain Task-Directed Aggregation.* PhD thesis, Dept. of Systems and Comp. Engineering, Carleton University, Ottawa, Canada K1S 5B6, May 1991.

[10] D. C. Petriu and C. M. Woodside, "Approximate MVA from Markov model of software client/server systems," in *Proc. of The Third IEEE Symposium on Parallel and Distributed Processing*, (Dallas, Texas), December 1991.

[11] D. C. Petriu, *Approximate Mean Value Analysis of Client–Server Systems with Multi-Class Requests*, Proc. of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, *Performance Evaluation Review*, Vol.22, nb.1, pp. 77-86, May 1994.

[12] D. C. Petriu, S. Chen, "Approximate MVA for Client–Server Systems with Nonpreemptive Priority", Proc. of Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'95), pp.155-162, Jan. 1995.

[13] C. Shousha, D. C. Petriu, A. Jalnapurkar, and K. Ngo, "Applying Performance Modelling to a Telecommunication System," in Proc. of First International Workshop on Software and Performance (WOSP98), October 1998, pp. 1-6.

[14] C. H. Skinner, "A priority queueing model with a server walking time" *0perations Research*, Vol.15, Nb.2, pp. 278-285, 1967.

[15] W. Stewart, *Introduction to the Numerical solution of Markov Chains*, Princeton University Press, Princeton, New Jersey, 1994.

[16] W. Stewart, K. Atif, B. Plateau, "The Numerical Solution of Stochastic Automata Network", *European Journal of Operational Research*, Vol. 86, pp. 503–525, 1995.

[17] C. M. Woodside, "Throughput calculation for basic Stochastic Rendezvous Networks," *Performance Evaluation*, vol. 9, pp. 143–160, 1989.

[18] C. M. Woodside, J. E. Neilson, D. C. Petriu, and S. Majumdar, "The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software," *IEEE Transactions on Computers*, vol. 44, no. 1 pp. 20-34, January 1995.