Approximate Mean Value Analysis based on Markov Chain Aggregation by Composition

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

March 31, 2003

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.

Index Terms: *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 [1]). 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) have 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 obtaining 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 is an important capability for those who define system models in practice, since it allows them to define behaviour by a state-transition model of a component (a smaller task than a system model, and less error-prone), and to compose a set of components in different ways. Components may be defined using process algebras (e.g. [7, 6]) or syncrhonized stochastic automata networks [14, 15]. Formal methods and tools such as TIPP [6] or PEPA [4] can generate the global model. However the problem of solving the global model is still serious. The present work began with a study of a particular family of components representing software tasks, and a particular approximate solution technique called "Task-directed Aggregation" (TDA) [8, 9, 10]. This is generalized here to any set of components, and the particular TDA solution is used as an example to expound the approach.

The approach taken here is to create approximate aggregated Markov chain submodels, each representing a view of the Markov model for the entire system from the perspective of a selected set D of "tagged" components, and to derive mean value equations from them. (A different set D is considered for each aggregated submodel). 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 sometimes straightforward. In our work the submodels have included parameters which come from solutions of other submodels, so a fixed-point iteration has been used. The first point of novelty is the method used to create the aggregate submodels for different sets D, which builds up the submodel by composing the behaviour of the components in D, rather than by aggregating the entire state space. However, such a submodel does not represent the aggregated behaviour of the whole system, unless its transitions are modified to incorporate some interactions with the components from the complementary set $\bar{D}$ (which contains the "untagged" components). This "Markov chain Aggregation by Composition" (MAC) uses composition techniques taken from other work in compositional modeling such as [7, 5], but the augmentation is novel. Another point of novelty is the use of partitioned Markov models to obtain analytic solutions, as explained below. Both of these are generalizations of basic ideas implied in [8, 9, 10]. 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 satified 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 [13].

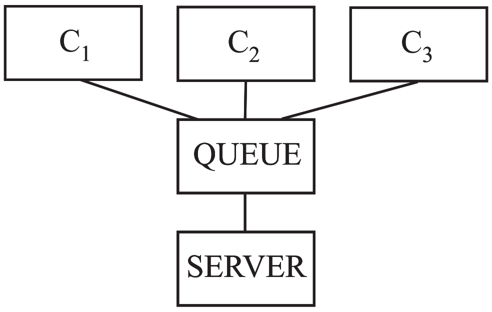


Figure 1: Example of a component-based system

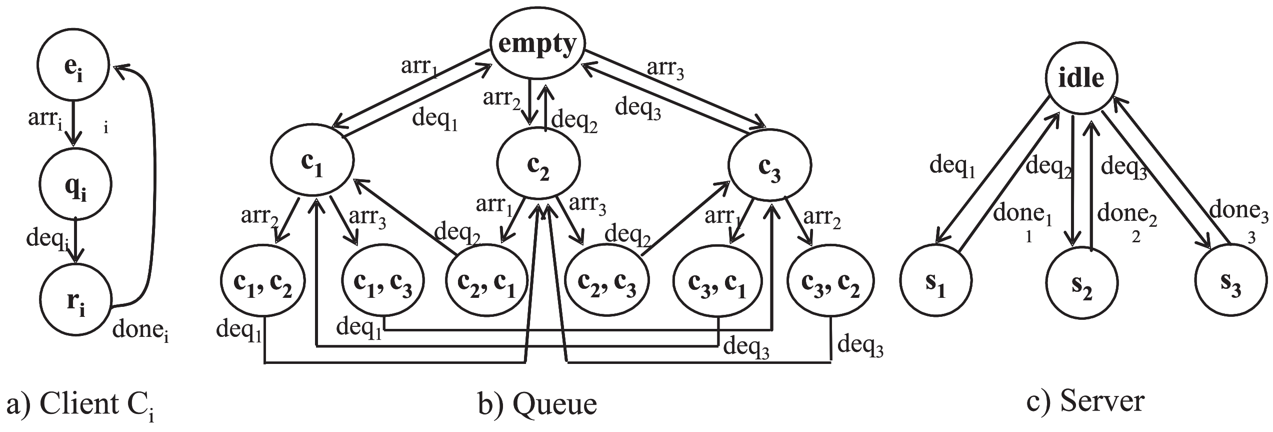


Figure 2: Stochastic automata defining the behaviour of components in a CMC (Closed Multi-Class) server with three clients

2 Component-based System Model

We will consider a system composed from a set of components which has a set of mean performance measures, such as throughputs and delays. The behaviour of each component is described by a finite stochastic automata, which is synchronized with the other components by the means of some common events associated to different transitions.

Fig. 1 shows an example system with five components representing three clients $C_1, \dots, C_3$, a FIFO queue Q and a server S, which 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. The stochastic automaton for client C_i, shown in Fig. 2.(a), has the states e_i (executing on its own, or "thinking"), q_i (queued), and r_i (in service at the server). The transitions are labeled with synchronizing events arr_i (arrive to the queue), deq_i (dequeue to start service), and $done_i$ (service finished). Some transitions also have 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; these times are exponentially distributed). These are called "active" transitions and their source states are called "active" states. Transitions with no rate parameter are called "passive".

We have adopted some restrictions on the component automata, so that no state has more than one active outgoing transition (no timed conflicts), and when the same transition label appears in more than one component, only one of them is active.

State e_i is an *active state*, in which the client performs some activity. When the activity ends, the client produces a synchronizing event arr_i, which forces a 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 $done_i$, and not by the client. 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 active service states $s_i, i = 1, n$ and one passive state *idle*.

The composition of the components is governed by the sharing of events (or equivalently, by the synchronization of transitions with the same labels), 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 [5]. When all the components are composed together, the Markov model $\mathcal{M}$ is obtained. For example, Fig. 3.(a) shows the continuous time Markov chain obtained for the system given in Fig. 1. The notation for a system state 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 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 [8], reaching for example 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

From the interactions of the components, overall performance relationships can be deduced involving rates of flow between components, and delays imposed by one component on another. While a formal derivation may be possible, these performance relationships are often obvious to the analyst. The MVA equations for the example system from Fig. 1 have been defined using throughputs, waiting times and arrival-instant probabilities, in line with the usual MVA analysis of queues [1]. For example, in Fig. 1 the flow rate F_i of client C_i arrivals equals the flow rate of C_i departures from the server and it can be computed as in (1). Also, the mean delay in the queue for C_i can be expressed in terms of arrival instant probabilities as in (2). Both MVA equations are exact.

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

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

where A and B are arrival-instant probabilities defined as follows:

$A_{ij} =$ the probability that a C_i request arriving to the server finds it serving client C_j
$B_{ij} =$ the probability that a C_i request arriving to the server finds C_j in queue.

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

- flow identities (flow in = flow out),
- some applications of Little's result (mean tokens = mean arrival rate times mean delay, where "tokens" are any measure of occupancy of some abstract state in a system),

- delay equations, expressing a delay in terms of component delays,
- waiting times, in terms of system state 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.

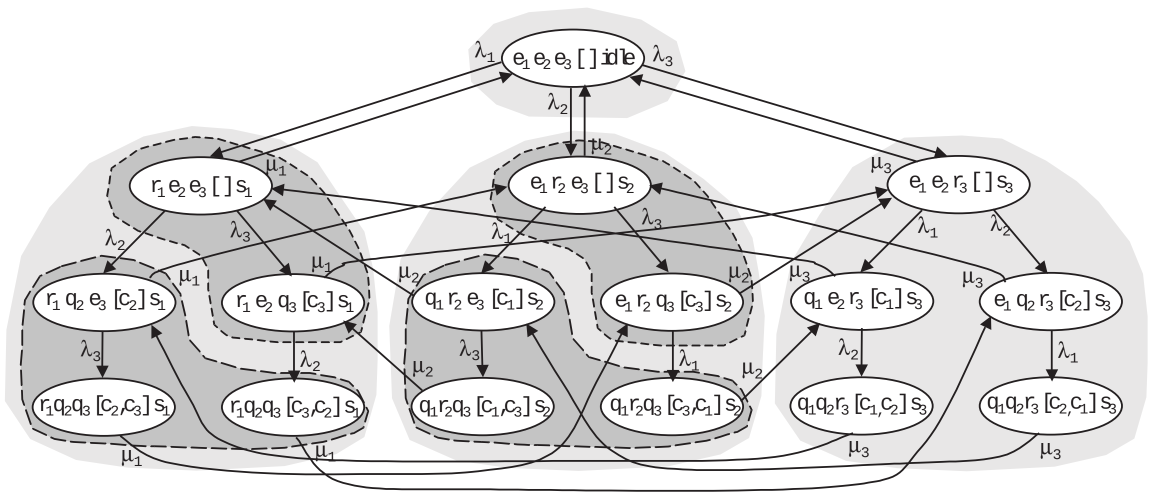
3 Markov chain Aggregation by Composition (MAC)

The arrival probabilities used in the MVA equation (2) suggest using an aggregation that shows the state of two clients C_i and C_j, which we will call "tagged clients", together with the queue and the server. For a complete analysis of the system we shall build such an aggregated submodel for each pair of clients.

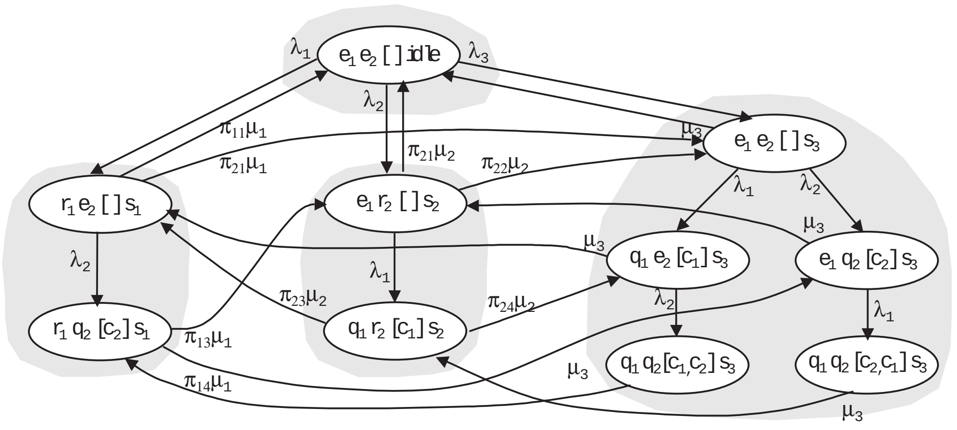
As a general strategy, we propose aggregating with respect to various sets D of tagged components, to obtain for each set D an aggregated submodel $\mathcal{M}'(D)$. The macrostates of the submodel are tuples of the states of the tagged components from D, which hide the states of the untagged components from $\bar{D}$. Fig. 3.(b) illustrates the aggregated Markov chain $\mathcal{M}'(D)$ obtained from the model $\mathcal{M}$ in Fig. 3.(a) for the tagged set $D = \{C_1, C_2, Q, S\}$. The darker shaded areas in part (a) define the states which are aggregated in part (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 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 the server state is s_3, serving C_3, in some of the macrostates (enclosed in the rightmost shaded cluster in the figure). Second, some transitions in $\mathcal{M}'(D)$ (shown with dotted lines) 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 at this point that both $\mathcal{M}$ from Fig. 3.(a) and $\mathcal{M}'(D)$ from Fig. 3.(b) can be partitioned into $n+1$ clusters based on the states of the server (the clusters are enclosed in lightly shaded areas). Each one of these clusters groups all the states in which a certain client C_i is in service, except for one cluster that contains the idle state. Note also that each cluster in $\mathcal{M}$ has a corresponding cluster in $\mathcal{M}'(D)$.

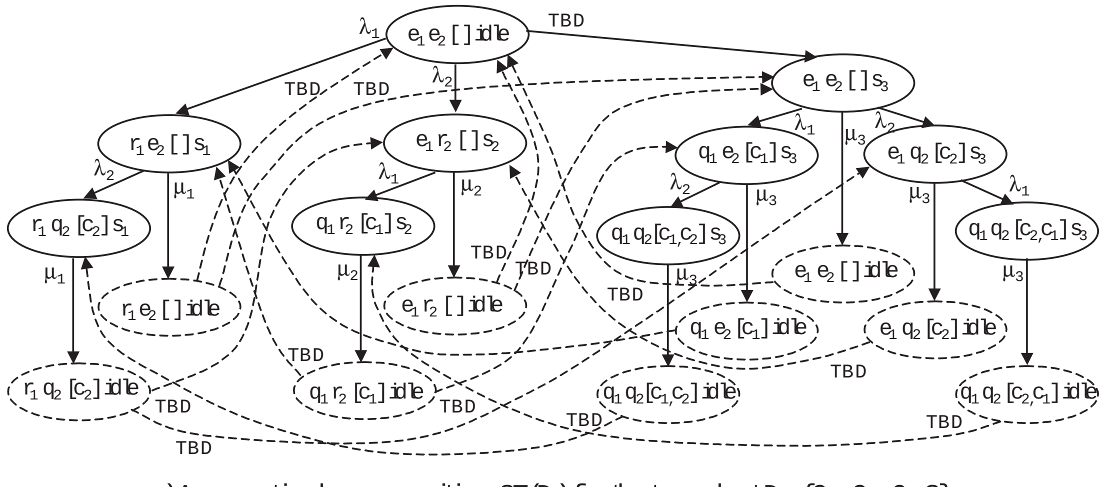
Even though $\mathcal{M}'(D)$ can be obtained from $\mathcal{M}$ by aggregation, we intend to avoid the expensive process of constructing the entire Markov chain and lumping its states. We propose instead to build $\mathcal{M}'(D)$ by composing the tagged components, and to add the missing pieces due to the effect of the untagged component behaviour. The "aggregation by composition" proposed in this paper differs from the "compositional aggregation" from [5] in that we must 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 those $\mathcal{M}'(D)$ clusters of states where C_1 and C_2 are in service, but would never reach the cluster where C_3 is in service. We must add this cluster, and the transitions between it and ther other clusters. In general, in a system with n clients, only 2 are tagged and $n-2$ are unttaged. When building the aggregated submodels by composition, we must add the clusters for all the untagged clients, in order to capture the relationship between arrivals from the two tagged clients when S is serving other clients (needed for the derivation of the arrival instant probabilities).



a) Markov Chain M for the closed multi-class system with a FIFO server and three clients



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



c) Aggregation by composition: $ST(D)$ for the tagged set $D = \{C_1, C_2, Q, S\}$

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

In this general strategy, the tagged components in D are composed to give a "submodel" automaton $ST(D)$. Fig. 3.(c) shows the $ST(D)$ model for the CMC example system with 3 clients. It contains both timed states (drawn with continuous lines in the figure) and instantaneous or vanishing states (drawn with dashed lines). In our example, instantaneous states are reached immediately after the end of a service, which is marked by a server event $done_i$. A dequeueing event follows immediately, generated by the server and synchronized with the queue. Depending on the queue state, either the server will begin a new service for the first client in the queues, or the the system will move to the idle state. The instantaneous transitions leaving instantaneous states are represented with dashed lines in the figure.

Transitions between two states in $ST(D)$ correspond to either a single transition in one component, in which case it has the rate of that transition if it is active, or to a synchronized combination of transitions with the same label, in which case it has the rate of the single instance that is active. (It was assumed that only one transition with a given label is active). Any of these transitions may be passive, as well.

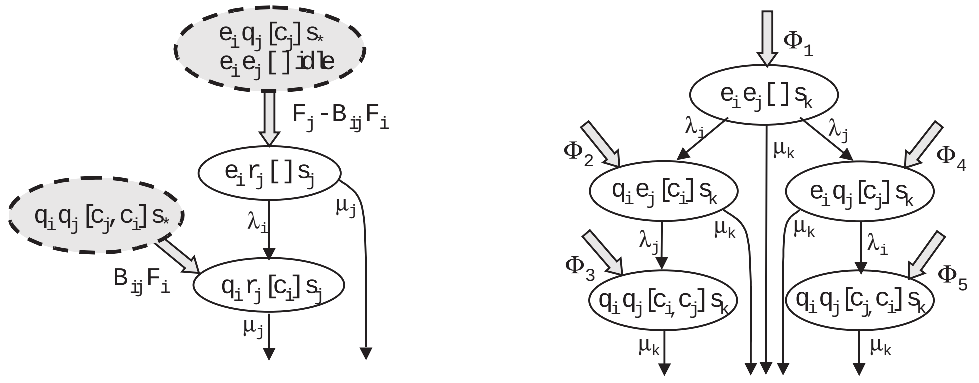
By composing the tagged components $D = \{C_1, C_2, Q, S\}$ directly, we obtain at first only the states where C_1 and C_2 are in service. We will augment the state space of $ST(D)$ to contain all the feasible combinations of tagged component states, as explained above. Now the submodel $ST(D)$ has a complete state space but it does not have the correct transition rates for an aggregated system, and some transitions are missing (marked in Fig 3.(c) with "TBD" for "to be determined"). These are transitions which enter/leave states of $ST(D)$ that represent the effect of tagged component interactions with untagged componets (such as serving untagged clients). In Fig 3.(c), transitions into states with $\sigma_s = s_3$, in which the server is serving C_3 are added and labelled "TBD". Where to add these transitions must be discovered in each case by understanding the whole system behaviour. Here, such "TBD" transitions can occur after an end of service for a tagged client (representing hidden cases in which C_3 is in the queue), or after an idle period (when C_3 arrives to an idle server). Also, missing "TBD" transitions are added for the end of service of an untagged client.

The next step is to eliminate the instantaneous transitions (considered to have an infinite rate) and to remove their source states from the model (similar to the elimination of "vanishing states" in GSPN [2]). If there are conflicts between two instantaneous transitions, they must be resolved by means not described here, for example by assigning them priorities or probabilities. In the CMC and CMC-ER examples studied here, there are no conflicts. Thus a Markov chain $\mathcal{M}'(D)$ is obtained for the submodel, which may have some unknown transition rates (In our example in Fig 3.(b), the unknown rates contain a factor π).

3.1 Partitioned Submodels

Rather than determining the individual unknown rates in the aggregated submodels, it is convenient to further simplify the submodels by partitioning them, and then determine some corresponding transition flows. The aggregated submodel $\mathcal{M}'(D)$ is partitioned into clusters of states (or partitions) based on the server states, as mentined before. In a case with n clients, $n+1$ clusters can been identified based on the server state, denoted by $\mathcal{G}'_j(D)$ (where client C_j is in service) or $\mathcal{G}'_0(D)$ (where the server is idle).

The importance of this partitioning resides in the fact that a cluster $\mathcal{G}'_k(D)$ has the same structure and size independent of the number of clients n [8]. The form of a cluster depends only on whether the client in service is tagged or untagged, as shown in Fig. 4.(a) and (b), respectively. Since $\mathcal{M}'(D)$ contains $n+1$ clusters for n clients, its size grows only linearly with n despite the



a) Partition $G'_j(D)$ of the aggregated MC where a tagged client C_j is in service

b) Partition $G'_k(D)$ of the aggregated MC where a non-tagged client C_k is in service

Figure 4: Partitioning $\mathcal{M}'(D)$: clusters of macrostates

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 clusters are much simpler than those for the entire submodel, and will be used to derive the solutions for the arrival probabilities. It should be emphasized that partitioning is a convenience, which has been used in some cases, but this work does not present a general procedure for partitioning. In writing the balance equations for a cluster, the transitions into a cluster state from other partitions are represented as an aggregated in-flow rate, (shown in Fig. 4.(a) and (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 partition. In the example system considered here, the partitions have the following properties (proved in [8]) that lead to a mean value solution.

Property 1. In any partition $\mathcal{G}'_k(D)$, the transition rates corresponding to “arrival from C_i”, and “arrival from 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 partition $\mathcal{G}'_j(D)$ where a tagged client C_j is in service, given in Fig. 4.(a) can be expressed exactly in term of system-level mean values (i.e., arrival-instant probabilities and system throughputs) as follows:

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

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

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

A sketch for 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 of the C_i arrival, this means that C_j was 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 $(q_i\ r_j\ [c_i]\ s_j)$. From the arrival-instant probability definitions, the frequency of the 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 cluster $\mathcal{G}'_j(D)$ equals the frequency of F_j arrivals to the server.

Unfortunately, the in-flows of a partition $\mathcal{G}'_k(D)$ 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 $\mathcal{M}_n$ as the ratio between the frequency of C_i arrivals occuring in some specific states over the total frequency F_i of C_i arrivals. More exactly, A_{ij} in (5) is the ratio between the frequency of C_i arrivals that find S serving another client C_j (computed as the occurrence rate of all transitions leaving a state $\boldsymbol{\sigma}$ with rate λ_i, in which C_i is executing and S in serving C_j) and the frequency F_i.

$$A_{ij} = \sum_{\boldsymbol{\sigma}\in\boldsymbol{\Omega}} (\lambda_i \, \mathcal{P}(\boldsymbol{\sigma}|\sigma_i = e_i, \sigma_s = s_j)/F_i \tag{5}$$

Similarly, B_{ij} in (6) is the ratio between the frequency of C_i arrivals that find another client C_j in queue (computed as the occurrence rate of all transitions leaving a state $\boldsymbol{\sigma}$ with rate λ_i, in which C_i is executing and C_j in queue) and the frequency F_{is}.

$$B_{ij} = \sum_{\boldsymbol{\sigma}\in\boldsymbol{\Omega}} (\lambda_i \, \mathcal{P}(\boldsymbol{\sigma}|\sigma_i = e_i, \sigma_j = q_j)/F_i \tag{6}$$

The probability A_{ij} defined in (5) is obtained from the balance equations for the states of the partition $\mathcal{G}'_j(D)$, where task $j \in D$ is in service. Due to *Property 2*, the two balance equations can be written as:

$$\mu_{j1} \, \mathcal{P}(q_i \; r_j \; [c_i] \; s_j) = B_{ij} \; F_i + \lambda_i \; \mathcal{P}(e_i \; r_j \; [\;\;] \; s_j) \tag{7}$$

$$(\mu_{j1} + \lambda_i) \; \mathcal{P}(e_i r_j s_{j1}) = F_j - B_{ij} F_i \tag{8}$$

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 partitions $\mathcal{G}'_k(D)$ for all $k \neq i,j$, the probability B_{ij} is computed by summing up its components $B_{ij,k}$:

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

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:

$$B_{ij,k} = \sum_{\boldsymbol{\sigma}\in\boldsymbol{\Omega}} \lambda_i \; \mathcal{P}(\boldsymbol{\sigma}|\sigma_i = e_i, \sigma_j = q_j, \sigma_s = s_k)/F_i \tag{11}$$

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 $\mathcal{G}'_k(D)$ 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 cluster $\mathcal{G}'_k(D)$ 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 $\mathcal{G}'_k(D)$, 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:

$$B_{ij,k} = A_{ik} - A_{i,k} A_{j,k} (F_j \lambda_j^{-1})/(F_{ks} \mu_k^{-1}) \qquad \text{for } i,j,k = 1,\ldots,n;\ \ i \neq j;\ \ k \neq i,j \tag{20}$$

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

1. 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.

2. Define state-transition models for components.

For each component, create a model in which transitions have labels and (in some cases) rate parameters. Transitions with no rate parameter are termed "passive".

3. Create the MAC submodels.

Define sets D1, D2,... 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 "TBD" transition rates representing interactions with components not included in D. Rates for these transitions must be determined, probably by approximation.

4. Optionally, partition the MAC submodels so the partitions have known transition rates inside, but may have unknown transition rates between partitions. These rates are given by either exact or approximate mean-value considerations.

5. 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.

6. 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 [17, 3, 12]. 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 [3]. 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 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 [13] under the name of "walking server", and in other work is called a "server with vacations". In software systems the use of second phases is not due to

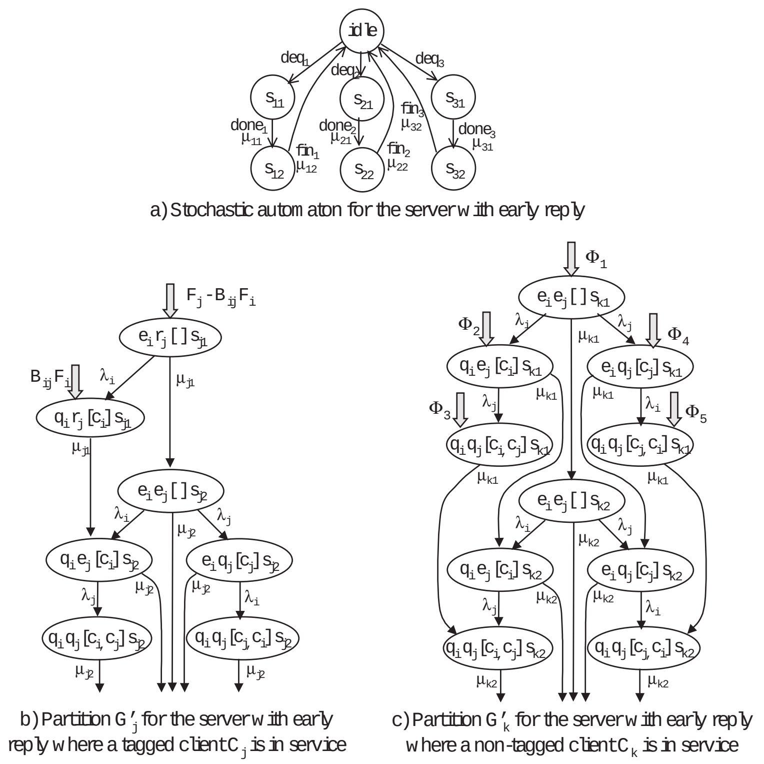


Figure 5: Models for the CMC-ER server with early reply

gaps in operation, but is 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 [17, 3].

5.1 System Level Performance Measures for CMC-ER

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:

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

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:

$$A_{ij} = \sum_{\boldsymbol{\sigma} \in \boldsymbol{\Omega}} (\lambda_i\, \mathcal{P}(\boldsymbol{\sigma} | \sigma_i = e_i, \sigma_s = s_{jp})/F_i \tag{22}$$

where $\mathcal{P}(\boldsymbol{\sigma})$ is the steady-state probability of a global state $\boldsymbol{\sigma} \in \boldsymbol{\Omega}$.

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:

$$B_{ij} = B_{ij,i2} + B_{ij,j2} + \sum_{k \neq i,j} \sum_{p} B_{ij,kp} \quad \text{for } i,j,k = 1,\ldots,n;\ \ i \neq j;\ \ k \neq i,j \tag{23}$$

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:

$$B_{ij,kp} = \sum_{\boldsymbol{\sigma} \in \boldsymbol{\Omega}} \lambda_i\, \mathcal{P}(\boldsymbol{\sigma} | \sigma_i = e_i, \sigma_j = q_j, \sigma_s = s_{kp})/F_i \tag{24}$$

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, there are two kinds of partitions in an aggregated submodel. The first, shown in Fig. 5.(b), includes states in which one of the tagged clients C_i and C_j is in service. The second, shown in Fig. 5.(c), includes states in which a third client (designated as C_k, with k running over all the other clients) is in service. The in-flows are defined as before.

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 (23).

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:

$$A_{i,kp} = B_{ij,kp} + \bar{B}_{ij,kp} \tag{25}$$

5.3 Arrival-instant probability equations for CMC-ER

The balance equations of the partitions $\mathcal{G}'$ from Fig 5.(b) and (c) are solved analyticaly to derive the arrival-instant probability equations, by using the definitions (5)–(24) to eliminate the macrostate probabilities. The in-flow rates are calculated as for the CMC case.

The set of simultaneous equations for the arrival-instant probabilities are listed here. All equations are exact, except equation (31) that is the only approximation.

$$A_{i,j2} = \frac{\mu_{j1}}{\lambda_i + \mu_{j2}} A_{i,j1} \quad \text{for } i,j = 1,\ldots,n;\ \ i \neq j \tag{26}$$

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

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

$$B_{ij,j2} = \frac{\lambda_j}{(\lambda_i + \lambda_j + \mu_{j2})} A_{i,j2} \quad \text{for } i,j = 1,\ldots,n;\ \ i \neq j \tag{29}$$

$$B_{ij,i2} = \frac{\lambda_i}{\lambda_i + \mu_{i2}} \cdot \frac{F_j}{F_i} [B_{ji} + (1 + \frac{\mu_{i1}}{\lambda_i + \lambda_j + \mu_{i2}}) A_{j,i1}] \qquad \text{for } i, j = 1, \ldots, n; \; i \neq j \tag{30}$$

$$B_{ij,k1} = A_{i,k1} - \frac{F_j \lambda_j^{-1}}{F_k \mu_{k1}^{-1}} A_{i,k1} A_{j,k1} \qquad \text{for } i, j, k = 1, \ldots, n; \; i \neq j; \; k \neq i, j \tag{31}$$

$$B_{ij,k2} = A_{i,k2} - \frac{\mu_{k1}}{\lambda_i + \lambda_j + \mu_{k2}} (A_{i,k1} - B_{ij,k1}) \qquad \text{for } i, j = 1, \ldots, n; \; i \neq j; \; k \neq i, j \tag{32}$$

$$B_{ij} = B_{ij,i2} + B_{ij,j2} + \sum_{k \neq i,j} \sum_{p} B_{ij,kp} \qquad \text{for } i, j, k = 1, \ldots, n; \; i \neq j; \; k \neq i, j \tag{33}$$

Equations (31) and (32) 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 (26)–(32) 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:
Algorithm 1 : *MAC/MVA algorithm for a CMC-ER server*

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. 26), $B_{ij,kp}$ (eqs. 30–32), B_{ij} (eq. 33);*
c) *Update the arrival-instant probabilities using an under-relaxation strategy, by applying only half of the change for each probability, i.e. $prob_{new} = 0.5(prob_{old} + prob_{computed})$;*
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.*

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 (26)–(29), (33)
- $n(n-1)(n-2)$ equations of each of the forms (31), (32)

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 [2]. 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.

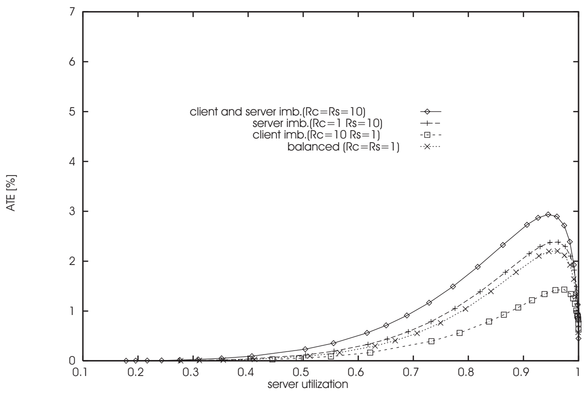
Several test suites were designed to study the impact of different factors on accuracy. Each test suite generates a curve plotted in one of Figures 6(a)–8(b), and contains a set of cases (models) with the same imbalance R_s and R_c, as follows:

- 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.

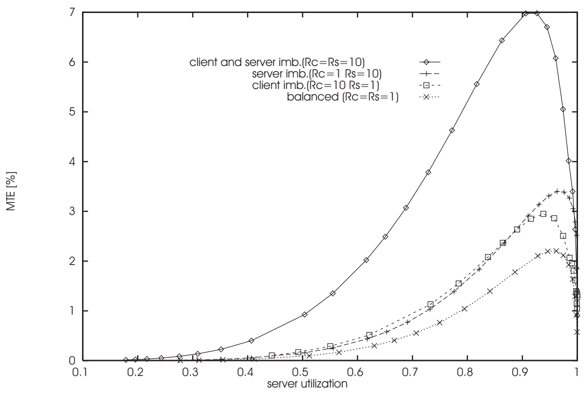
The following percentage errors were plotted in function of the achieved server utilizations for various test suites: *average throughput error* ATE (the average relative error, in absolute value, of all clients throughputs), and *maximum throughput error* MTE (the highest relative error, in absolute value, among the client throughputs). 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.

The first experiment studies the impact on accuracy of various imbalance ratios for test suites with 7 clients, as shown in Figures 6(a) and 6(b). 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 of each entry is equally split between phases). The balanced suite fares best in terms of average error ATE, and, with a single exception in terms of maximum error MTE.

Figures 7(a) and 7(b) show the ATE and MTE errors for suites with various degree of client and server imbalance, where both R_c and R_s are simultaneously varied from 1 to 10. As expected, a larger imbalance produces a worse accuracy.

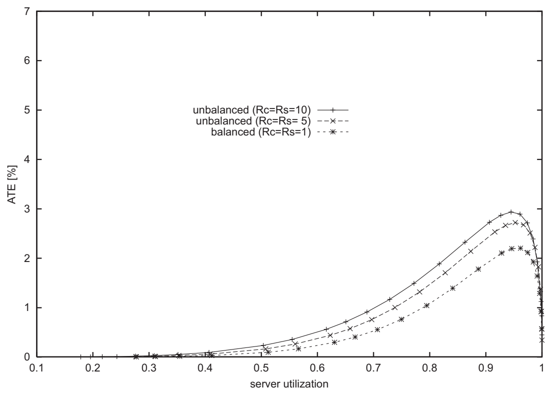


(a) Average Throughput Error

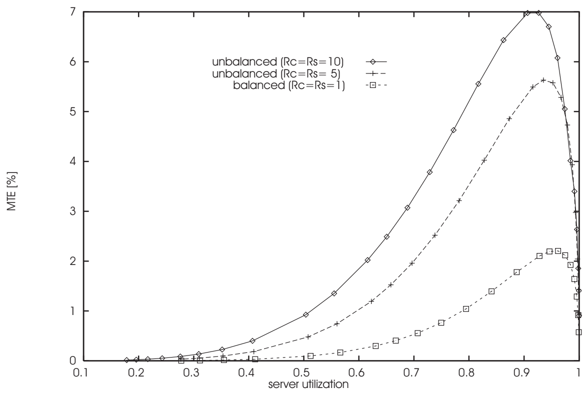


(b) Maximum Throughput Error

Figure 6: ATE and MTE for different unbalanced and balanced test cases with 7 clients

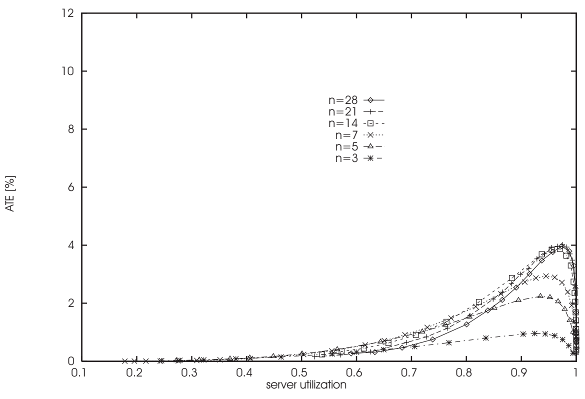


(a) Average Throughput Error

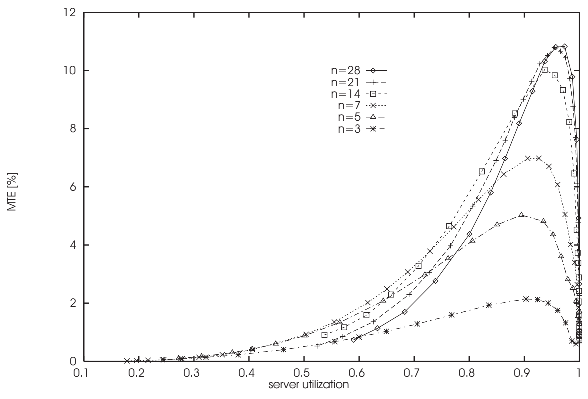


(b) Maximum Throughput Error

Figure 7: ATE and MTE for test cases with 7 clients, where R_c, R_s range from 1 to 10



(a) Average Throughput Error



(b) Maximum Throughput Error

Figure 8: ATE and MTE for unbalanced test cases with different numbers of clie nts

The number of clients has also a strong impact on accuracy. Figures 8(a) and 8(b) 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 (31) 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.

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 any 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. 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 (a few dozen 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 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 [5]). The innovation here is to

- create ad hoc approximate aggregated models for any kind of component based system,
- 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 solution of the submodels with system level mean value relationships.

Some modeling judgement is required to complete the aggregated submodels with parameters described above as "TBD", and to find the mean values from the submodels and the system level 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 [10, 11]. 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, stochatic automata or composable Petri Nets.

Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada, through its Discovery Grants program.

References

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

[2] 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.

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

[4] 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.

[5] H. Hermanns, *Interactive Markov Chains*, LNCS. Vol. 2428, Springer, berlin, 2002.

[6] 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.

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

[8] D. 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

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

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

[17] 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.