CHAPTER 7

STABILIZATION TECHNIQUES FOR LOAD-DEPENDENT QUEUEING NETWORKS ALGORITHMS

Giuliano Casale
Giuseppe Serazzi
Politecnico di Milano, Dip. di Elettronica e Informazione
Via Ponzio 34/5, I-20133 Milano, Italy
{*giuliano.casale, giuseppe.serazzi*}*@polimi.it*

Product-Form (PF) queueing networks are one of the most popular modelling techniques for evaluating the performances of computing and telecommunication infrastructures. Several computational algorithms have been developed for their exact solution. Unlike the algorithms for load-independent models, the ones for models with queue-dependent servers, either with single class or multiclass workloads, are numerically unstable. Furthermore, existing numerical stabilization technique for multiclass load-dependent models are not always efficient. The search for such result is motivated by the complexity of nowadays systems, which often require the use of load-dependent models.

In this work we review numerical instabilities of computational algorithms for product-form queueing networks. Then, we propose a general solution to the problem based on arbitrary precision arithmetics. Finally, we discuss a new specialized stabilization technique for two class load-dependent models.

1. Introduction

Closed queueing network models[13] have played an important role in the performance evaluation of complex computer and telecommunication systems[20]. Their applications[21,4,16] span over several types of performance evaluation studies, such as system tuning, optimization and capacity planning.

Among the most important reasons of this success is the availability of a simple product-form expression of the exact steady-state probability distribution of network states[3]. The queueing networks that have such

closed form expression are referred to as Product-Form Queueing Networks (PFQN). This property has led to the definition of efficient exact algorithms, including the Convolution Algorithm[7], the Mean Value Analysis[24] (MVA) and algorithms that work by recursion on the number of customer classes, e.g. RECAL[9]. An important feature of the MVA algorithm, compared to the others, is that it grants the numerical stability of the computation, meaning that: (i) the quantities computed at each step always lie within the floating-point range allowed by the computer architecture; (ii) no significant precision loss due to numerical round-offs is suffered. Unfortunately, this interesting property holds only for *load-independent* models, i.e. models where the rate at which the jobs are served are processed at each server is assumed to be constant. Indeed, exact solution algorithms for load-dependent models, i.e., models comprising servers with queue-dependent service rates, either with a single class or multiclass customers, frequently exhibit numerical instabilities.

Nevertheless, models of Web architectures contain a large number of heterogenous components that often require the use of load-dependent servers. For example, approximations of subsystems including population constraints (e.g. maximum number of HTTP connections) may be obtained with load-dependent PF flow-equivalent servers[21]. In this work we review numerical instabilities of computational algorithms for PFQN. Then, we propose a general solution to the problem based on arbitrary precision arithmetics. Finally, we consider a stabilization technique for two class load-dependent models. The paper is organized as follows: Section 2 gives preliminary concepts on numerical instabilities and queueing networks; Section 3 explores instabilities in the Convolution Algorithm and in the load-dependent MVA. Section 4 discusses two new stabilization techniques. Finally, Section 5 concludes the chapter.

2. Preliminaries

2.1. *Numerical Exceptions*

In this section we give a brief overview of numerical instabilities in floating-point computations. In actual computer architectures a floating point number[1] is represented using three elements: a sign, an exponent e and a mantissa m. For instance, the number $x = -1.04 \cdot 10^{-12}$ has negative sign, mantissa $m = 1.04$ and exponent $e = -12$. Looking at the binary representation, the sign takes a single bit while the remaining bits, whose number depends on the data type, are split between the mantissa and the exponent.

For instance, the double precision arithmetics is implemented with 53 bits (i.e. 16 decimal digits) in the mantissa and 11 bits in the exponent, with the representable numbers ranging approximately in $[10^{-308}, 10^{+308}]$. Despite this floating-point range may appear sufficient for most algorithms, numerical exceptions frequently arise. In general, an algorithm $A(\mathbf{x})$ for computing a function $f(\mathbf{x})$ is said to be *numerically stable* if

$$A(\mathbf{x}) \approx f(\mathbf{x} + \epsilon),$$

where ϵ is a small perturbation of the input vector. Informally, this definition wishes to capture that arbitrary small perturbations of the input parameters, like those implied by limited machine accuracy, should not affect macroscopically the output of $A(\mathbf{x})$. Unfortunately, several numerical exceptions may prevent $A(\mathbf{x})$ from returning a correct result:

- *round-off errors*: errors associated with the existence of a limited number of digits for representing the mantissa. For instance, some irrational numbers, like $\pi = 3.1415\ldots$, cannot be represented with a finite number of digits. Thus, rounding schemes (like toward zero or nearest-value rounding[1]) are required to achieve the best possible approximation. As an example of the problems that may arise with double precision arithmetics, let $y = 10^{+23}$ and $x = 1$, define $w = (y - y) + x$ and $z = (y + x) - y$, then due to round-off errors we get $w = 1$ and $z = 0$! Note that even if rounding schemes may result in small errors for a single operation, e.g. of the order of 10^{-16} for double precision arithmetics, they can accumulate during execution and yield macroscopic errors on $A(\mathbf{x})$;
- *underflow/overflow errors*: underflow and overflow errors occur when the number to be represented is outside the allowed floating-point range. The limiting factor in this case is the number of digits of the exponent. For instance, let $x = 10^{+257}$ and $y = 10^{+52}$, it is straightforward to see that the product $z = xy = 10^{+309}$ exceeds the IEEE double precision range. Thus it is internally represented as an infinite (`Inf`) that cannot be used in other arithmetics operations.

The effect of the above exceptions on computational algorithms for product-form queueing networks are discussed in the following sections.

2.2. *Closed Product-form Queueing Networks*

From now on, we consider closed PFQN composed of M servers that process the request of N customers. We assume that $M_Q \geq 0$ queue-dependent servers are present in the network. Customers are partitioned in R classes, each with a constant population N_r. Network population is therefore described by a population vector $\mathbf{N} = (N_1, \ldots, N_R)$ and we define $N = \sum_{r=1}^{R} N_r$. The vector $\mathbf{0} = (0, \ldots, 0)$ denotes an empty population; $\mathbf{1}_r$ is, instead, the r-dimensional unit vector. We denote L_{ir} the class r loading at server i, i.e., the average time spent by class r customers at server i if no queueing is experienced. Note that this term also accounts for network topology[10]. For queue-dependent servers, the rate at which request are processed when n customers queue at server i is expressed through the capacity function[6] $c_i(\mathbf{n})$. Later, we refer to the following mean performance indices:

$$\begin{aligned} X_r(\mathbf{N}) &= \text{mean throughput of class } r \\ Q_{ir}(\mathbf{N}) &= \text{mean queue length of class } r \text{ at server } i \end{aligned}$$

Finally, if not otherwise specified, $i, m = 1, .., M$ and $r, s = 1, .., R$ should be intended as servers and customer classes indices, respectively.

3. Numerical Instabilities in PFQN Algorithms

3.1. *Convolution Algorithm*

The Convolution Algorithm[7,22] allows to recursively compute the normalization constant $G(\mathbf{N})$ for load-dependent and load-independent multiclass PFQN. Simple analytical expression allow to derive any $X_r(\mathbf{N})$ or $Q_{ir}(\mathbf{N})$ from the knowledge of a set of normalization constants. For instance, the throughput can be computed using the following formula:

$$X_r(\mathbf{N}) = \frac{G(\mathbf{N} - \mathbf{1}_r)}{G(\mathbf{N})} \tag{1}$$

The Convolution Algorithm for load-dependent models is based on the following recursion:

$$G(\mathbf{N}) = G(M, \mathbf{N}) = \sum_{\mathbf{n} \leq \mathbf{N}} F_M(\mathbf{n}) G(M-1, \mathbf{N} - \mathbf{n}) \tag{2}$$

with initial condition $G(1, \mathbf{N}) = F_1(\mathbf{N})$ and where $\mathbf{n} = (n_1, \ldots, n_R)$ is a vector of positive integers. In the above formula, the term $G(M-1, \mathbf{N}-\mathbf{n})$ refers to the normalization constant of the *M-complement system*, i.e. the network obtained from the original one by removing the server indexed with M. Instead, the *product-form factor* (*PF factor*) $F_M(\mathbf{n})$ for queue

M accounts for the probability that server M contains a population $\mathbf{n}$[3]. Alternative forms of load-dependence are possible[21,6]. Let $|\mathbf{N}| \triangleq \prod_r (N_r + 1)$ be the number of populations vectors $\mathbf{n}$ such that $\mathbf{0} \leq \mathbf{n} \leq \mathbf{N}$. Let $M \geq 2$, then the time complexity of a recursive computation of (2) for a network consisting of queue-dependent stations ($M = M_Q$) is bounded by $O(M_Q \, |\mathbf{N}|^2)$, while the space complexity is $O(2 \, |\mathbf{N}|)$. Multiclass models that contain both open and closed classes can be solved with a slightly modified version of (2) and with a similar computational effort. Note that, for load-independent models, (2) can be simplified to

$$G(\mathbf{N}) = G(M, \mathbf{N}) = G(M-1, \mathbf{N}) + \sum_r L_{mr} G(M, \mathbf{N} - \mathbf{1}_r) \qquad (3)$$

and its solution requires $O(M \prod_r (N_r+1))$ time and $O(2 \prod_r (N_r+1))$ space. As discussed in several works[18], a major problem connected to the use of normalization constants is that their growth can easily led to buffer overflows or underflows. For example, for $M = R = 3$, $N_1 = N_2 = N_3 = 100$ and all loadings L_{ir} equal to 100, the normalization constant has value $G = 1.7 \cdot 10^{445}$, which exceeds the IEEE double precision range. Scaling techniques, described in the next section, are effective in addressing the problem.

3.1.1. *Static and Dynamic Scaling Techniques*

Several solutions have been proposed in literature for range extension problems on the basis of the following observation: assume that, for each class r, the loadings L_{ir} of all servers are initially scaled by some common factor β_r, then the normalization constant can be rescaled through the following relation:

$$G(\alpha, \mathbf{N}) = r(\beta, \mathbf{N}) G(\beta_1, .., \beta_R, \mathbf{N}) \qquad (4)$$

where $\alpha = (\alpha_1, .., \alpha_R)$ and $\beta = (\beta_1, .., \beta_R)$ are two *scaling vectors* and

$$r(\beta, \mathbf{N}) = \prod_r^R \left(\frac{\alpha_r}{\beta_r} \right)$$

is a conversion function between normalization constants employing different scaling vectors. For instance, for the three servers and three classes example considered before, assume that the class-1 loadings L_{k1} are all multiplied by $\alpha_1 = 0.001$. Let $\alpha = (0.001, 1, 1)$ and $\beta = (1, 1, 1)$, then the resulting normalization constant $G(\alpha, \mathbf{N})$ is computed as

$$G(\alpha, \mathbf{N}) = 10^{-300} G(\beta, \mathbf{N}) = 1.7 \cdot 10^{145}$$

that now lies in the range $[10^{-308}, 10^{+308}]$. Mean performance measures are easily obtained from the computed normalization constants and the associated scaling vectors. Thus, a non-negligible memory overhead may be associated with the use of scaling techniques. However, this may be limited by employing for all classes a same scaling factor $\alpha = \alpha_1 = \ldots = \alpha_R$ that can be computed as a function of N and of the floating-point range[18].

A problem that may arise in the implementation of scaling techniques is the choice of an initial $\alpha = \alpha_1 = \ldots = \alpha_R$ everytime the normalization constant of a new population is evaluated. Empirically, if in (3) the scaling factor α is initially assumed to be $\alpha = 1$, then the resulting computation remains numerically unstable because $r(\alpha, \mathbf{N})$ may produce buffer overflows/underflows for large populations. Thus, α should be evaluated as a function of the scaling factors of the normalization constants involved in (3). We observed that if α is set to the maximum of the scaling factors β, then no stability problems arise.

Following a simpler approach, it also possible to scale the loadings L_{ir} before algorithm execution. This form of scaling is commonly referred to as *static scaling*, while the one described above is known as *dynamic scaling*. Previous works[23,18] have shown the effectiveness of static scalings for load-independent models. In particular, the following scaling can often limit the growth of $G(\mathbf{N})$ in multiclass models:

$$L_{ir} \leftarrow \left(\frac{\sum_m \sum_s L_{ms}}{\sum_k \sum_c L_{kc}^2} \right) L_{ir} \quad \forall i = 1, \ldots, M; \ \forall r = 1, \ldots, R;$$

For instance, applying the above transformation on the previous example, all loadings are scaled to $L_{ir} = 1$ and the output becomes again $G(\mathbf{N}) = 1.7 \cdot 10^{145}$ with a negligible time overhead. However, this form of *static scaling*, compared to the *dynamic scaling* technique of (4), cannot prevent the occurrence of range exceptions and, more seriously, is applicable only to load independent models. More complex scalings for load-dependent models that are have been proposed[19], but anyway do not represent a general solution to numerical instabilities.

Finally, we report another type of overflow problem that should be accounted when solving (2). Consider a PFQN with $R = 2$ classes and assume that a large population vector $\mathbf{n} = (600, 600)$ requires the evaluation of the product-form factor $F_M(\mathbf{n})$. Since $(600 + 600)! \gg (600 + 600)!/600! > 10^{+308}$, the computation of the factorial $F_M(\mathbf{n})$ yields a buffer overflow. A possible solution consists in recursively evaluate $F_M(\mathbf{n})$ from $F_M(\mathbf{n} - \mathbf{1}_s)$

for any class s such that $n_s \neq 0$ using the following relation[26]:

$$F_M(\mathbf{n}) = L_{Ms} c_i^{-1}(\mathbf{n}) F_M(\mathbf{n} - \mathbf{1}_s) \tag{5}$$

Nevertheless, (5) does not guarantee that $F_M(\mathbf{n})$ lies in the allowed floating point range. For instance, consider a model with $M = 2$, $R = 2$ and the loadings $L_{12} = L_{21} = 5$, $L_{11} = 10$ and $L_{22} = 9$. Let the station 1 be a queue-dependent station with $c(\mathbf{n}) = n$ (i.e. a delay station), and let station 2 be a load-independent station. For a population vector $\mathbf{N} = (150, 150)$ it is $F_2(\mathbf{N}) = 8.99 \cdot 10^{+336}$ which generates a buffer overflow.

Summing up, the exact solution of networks including queue-dependent servers with (2) may require also the scaling of possibly thousands of PF factors. Furthermore a study on the mutual interaction between the dynamic scaling of Lam and the numerical range of the PF factors is missing. These problems sensibly limit the ease of implementation and the applicability of (2) on complex models.

3.2. *Load Dependent Mean Value Analysis (MVA-LD)*

The Mean Value Analysis (MVA) algorithm[24] relates the computation of the average performance measures at population $\mathbf{N}$ with the ones computed for the populations $\mathbf{N} - \mathbf{1}_s$. The following MVA formulas allow the exact solution of a load-dependent model and will be referred to as the MVA-LD algorithm:

$$X_r(\mathbf{N}) = \frac{N_r}{\sum_i \sum_{\mathbf{n}=\mathbf{1_r}}^{\mathbf{N}} n_r L_{ir} c^{-1}(n) p_i(\mathbf{n} - \mathbf{1}_r | \mathbf{N} - \mathbf{1}_r)} \tag{6}$$

$$p_i(\mathbf{n}|\mathbf{N}) = \sum_r L_{ir} c^{-1}(n) X_r(\mathbf{N}) p_i(\mathbf{n} - \mathbf{1}_r | \mathbf{N} - \mathbf{1}_r) \tag{7}$$

$$p_i(\mathbf{0}|\mathbf{N}) = 1 - \sum_{\mathbf{0}<\mathbf{n}\leq\mathbf{N}} p_i(\mathbf{n}|\mathbf{N}) \tag{8}$$

where $p_i(\mathbf{n}|\mathbf{N})$ is the probability that server i contains a population $\mathbf{n}$ when $\mathbf{N}$ jobs are present in the network. Compared to the Convolution Algorithm, the MVA-LD shares a similar computational complexity, but it may requires less space since only the performance measures for the the populations $\mathbf{N} - \mathbf{1}_s$ must be kept in memory.

In the case of a load-independent model, equations (6)-(8) simplify to

$$X_r(\mathbf{N}) = \frac{N_r}{\sum_i L_{ir}[1 + \sum_s Q_{is}(\mathbf{N} - \mathbf{1}_r)]} \tag{9}$$

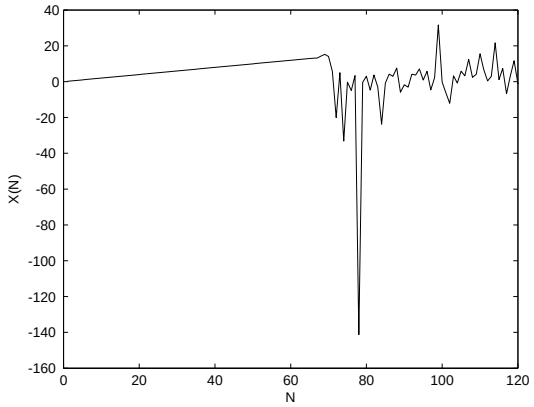

Fig. 1. Effects of round-off errors on a single-class throughput X($\mathbf{N}$) of a load-dependent model.

$$Q_{ir}(\mathbf{N}) = L_{ir}X_r(\mathbf{N})[1 + Q_{ir}(\mathbf{N} - \mathbf{1}_r)] \tag{10}$$

Note that the algorithm resulting from equations (9)-(10) is numerically robust. Instead, numerical exceptions may affect the MVA-LD algorithm, where (8) has been frequently addressed as a source of instability. In particular, if the bottleneck server is a load-dependent server, then the narrow difference between $p_i(\mathbf{0}|\mathbf{N})$ and zero may produce underflow errors and round-offs to negative numbers (note that overflows cannot occur being $p_i(\mathbf{n}|\mathbf{N}) \leq 1$ for all $\mathbf{n}$). These may result on a wrong evaluation of the performance indices as shown in Fig. 1 for a single-class throughput $X(\mathbf{N})$ of a load-dependent model. More importantly, round-off errors and underflows cannot be addressed by static or dynamic scaling since queue-lengths are unaffected by such transformations. Reiser[25] have proposed a general variable scaling technique to avoid underflow exceptions. The stabilization of round-off errors is instead discussed in the next section.

4. Improved Stabilization Techniques

4.1. *Software Stabilization*

As seen in the previous sections, numerical issues of various types may occur during the implementation of queueing-networks algorithms. Despite static and dynamic scaling may be prevent some overflows and underflows, they do not constitute a final solution to the problem. We now discuss a general solution to numerical exceptions in queueing networks algorithms.

Most of the existing algorithms for load-dependent models have been defined during the 70's and 80's. Since then, a number of software libraries for arbitrary precision arithmetics have been developed[2], e.g. the GNU GMP library[14]. These allow to prevent round-off and range exceptions at software level. The basic idea behind arbitrary precision arithmetics is providing library functions for storing and manipulating floating-point numbers with a user-customizable number of precision bits. In this way operations are performed through a software layer and precision is obtained at the expense of an increased computational effort.

Such operations can be handled either with straightforward or state-of-the-art algorithms. If the required precision increase during computation, the latter can limit the overheads. For example, a multiplication between numbers with N precision bits can be performed with the Karatsuba multiplication[15] in $O(N^{1.585})$, much less than the $O(N^2)$ of a traditional multiplication.

It is clear that the major advantage of arbitrary precision arithmetics over the techniques presented in queueing networks literature is the ease of implementation. Scalings and round-offs are automatically handled by the software library. Nevertheless, we can show that the overhead connected to the use of arbitrary arithmetics is almost equivalent to that of dynamic scaling. Table 1 and Table 2 report CPU times for the Normalization Constants algorithm for two groups of networks where $M = 2$, $M_Q = 0$ and $M = 2, M_Q = 2$. In all cases, we considered a set of 30 random matrices with $R = 2$, $N_r = N/R$ for all $r = 1, \ldots, R$. Three versions of the Convolution Algorithm have been tested:

- `NC` does not use any scaling technique for $G(\mathbf{N})$;
- `NC-DS` implements a dynamic scaling[18] of $G(\mathbf{N})$;
- `NC-GMP` employs arbitrary precision arithmetics for $G(\mathbf{N})$;

Equation (3) was used to solve the models in which $M_Q = 0$. Since we were interested in timing results, we measured the execution times of `NC` regardless of the correctness of the results. For the same reason the queue-dependent stations have been set to $c_i(\mathbf{n}) = n$. We considered a maximum timeout of $1500s$ for each algorithm. We point out that the implementations of `NC-DS` and `NC-GMP` were obtained with minor modifications from the code of `NC`, and no additional optimizations were employed.

As we can see from the reported CPU times, the overhead introduced in load independent models (see Table 1) by arbitrary precision arithmetics is comparable to that produced by dynamic scaling. However, the imple-

Table 1. Execution times for unstable and stabilized *load independent* ($M = 2$, $M_Q = 0$ and $N_1 = N_2 = N/2$) multiclass normalization constant algorithms.

N	1000	2000	3000	4000	5000	6000
`NC`	0.08s	0.33s	0.74s	1.32s	2.06s	2.95s
`NC-DS`	0.40s	1.97s	4.82s	8.89s	14.24s	20.75s
`NC-GMP`	0.50s	1.99s	4.47s	8.16s	12.80s	18.52s

Table 2. Execution times for unstable and stabilized *load dependent* ($M = M_Q = 2$ and $N_1 = N_2 = N/2$) multiclass normalization constant algorithms.

N	50	100	200	300	400	500
`NC`	0.02s	0.19s	2.58s	17.91s	63.07s	159.38s
`NC-DS`	0.11s	1.52s	24.71s	127.68s	415.99s	$>$ 1500s
`NC-GMP`	0.13s	0.41s	7.68s	47.90s	168.47s	459.47s

mentation of `NC-GMP` has been straightforward, compared to the time required by the implementation of `NC-DS`. Furthermore, if we look at the load-dependent models of Table 2, we see that the amount of overhead of the dynamic scaling is sensibly bigger than that of `NC-GMP`. This is easily explained by noting that while performing the convolution (2), an increasing number of rescalings (4) must be applied at each step in `NC-DS`. Thus, our experimental results show that the software stabilization approach is more efficient (and in our practice simpler to implement) than dynamic scaling.

4.2. *Stabilization of MVA-LD with Two Customer Classes*

Currently, the best approach for dealing with round-off errors consists in replacing (8) with the generalized Reiser's equation[25]:

$$p_i(\mathbf{0}|\mathbf{N}) = \frac{X_r(\mathbf{N})}{X_r^{-i}(\mathbf{N})} p_i(\mathbf{0}|\mathbf{N} - \mathbf{1}_r) \tag{11}$$

for each server $i = 1, \ldots, M$ and for any class r such that $N_r \geq 1$. In (11) $X_r^{-i}(\mathbf{N})$ denotes the class-r throughput of the i-complementary system. Note that since MVA-LD requires the evaluation of M_Q probabilities $p_i(\mathbf{0}|\mathbf{N})$, M_Q complementary systems have to be evaluated with (11). Since each of these complementary systems suffer the same instability problems of the original MVA-LD, this would lead to a repeated application of (11) with a combinatorial growth of the number of systems to be solved. Specifically, for a model consisting of $M = M_Q$ queue-dependent servers the number

$E(M)$ of complementary systems to be evaluated with (11) is

$$E(M) = \sum_{m=1}^{M} \binom{M}{m} = 2^M - 1$$

which grows exponentially with M. Thus, it is clear that if the number of queue-dependent servers is greater than two or three, then MVA-LD has probably no advantage over (2).

In order to avoid this difficulty, the following solution has recently been proposed[8] for load-dependent models with $R = 2$ customer classes. Define

$$L^*_{ir}(\mathbf{N}) = \sum_{\mathbf{n}} L_{ir} c_i^{-1}(\mathbf{n}) p_i(\mathbf{n} - \mathbf{1}_r | \mathbf{N} - \mathbf{1}_r)$$

for $N_r \geq 1$, and $L^*_{ir}(\mathbf{N}) = L_{ir} c_i^{-1}(\mathbf{n})$ otherwise. Let also

$$\pi_{kr}(\mathbf{N}) = \frac{p_k(\mathbf{0}|\mathbf{N} - \mathbf{1}_r)}{X_r^{-k}(\mathbf{N})}$$

Then, it is possible to show that if $N_1 > 0$ and $N_2 > 0$, then

$$X_1(\mathbf{N}) = \frac{\pi_{k2}(\mathbf{N})}{\pi_{k1}(\mathbf{N}) L^*_{k2}(\mathbf{N}) + \pi_{k2}(\mathbf{N}) L^*_{k1}(\mathbf{N}) + \pi_{k1}(\mathbf{N}) \pi_{k2}(\mathbf{N})} \tag{12}$$

$$\begin{aligned} X_2(\mathbf{N}) &= \frac{\pi_{k1}(\mathbf{N})}{\pi_{k1}(\mathbf{N}) L^*_{k2}(\mathbf{N}) + \pi_{k2}(\mathbf{N}) L^*_{k1}(\mathbf{N}) + \pi_{k1}(\mathbf{N}) \pi_{k2}(\mathbf{N})} \\ &= \frac{\pi_{k1}(\mathbf{N})}{\pi_{k2}(\mathbf{N})} X_1(\mathbf{N}) \end{aligned} \tag{13}$$

For the special case $N_1 = 0$ we have

$$X_1(\mathbf{N}) = 0, \quad X_2(\mathbf{N}) = \frac{1}{L^*_{k2}(\mathbf{N}) + \pi_{k2}(\mathbf{N})} \tag{14}$$

Similarly, when $N_2 = 0$ it is

$$X_2(\mathbf{N}) = 0, \quad X_1(\mathbf{N}) = \frac{1}{L^*_{k1}(\mathbf{N}) + \pi_{k1}(\mathbf{N})} \tag{15}$$

The computational requirements of the algorithm resulting from the above formulas are similar to those of MVA-LD. Nevertheless, the computation is stable with respect to round-off errors without resorting to arbitrary precision libraries. Thus, for reasonable population size, no underflows are generally experienced. The effect of the stabilization algorithm is shown in Fig. 2 for the example considered in the next section.

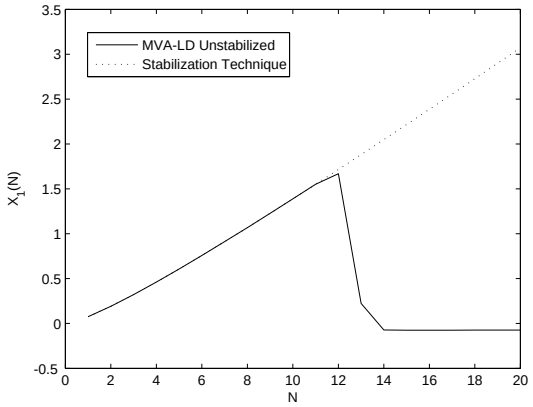

Fig. 2. Effect of stabilization of $X_1(\mathbf{N})$ as $N_1 = N_2 = N$ grow for the system of section 4.2.1 with two queue-dependent servers. After the population $\mathbf{N} = (12, 12)$ the MVA-LD algorithm exhibits fatal round-off errors.

4.2.1. *Numerical Example*

Let us consider a closed load-dependent queueing network model with $R = 2$ customer classes and with $M = M_Q = 2$ queue-dependent stations. Station loadings are:

$$L_{11} = 10,\ L_{12} = 5$$
$$L_{21} = 5,\ L_{22} = 9$$

while the capacity functions are $c_1(\mathbf{n}) = n^2$ and $c_2(\mathbf{n}) = n$. We show how to apply the algorithm defined in previous section for a simple population vector is $\mathbf{N} = (2, 1)$.

We begin the recursion on the number of stations $m = 1, \ldots, M$. We denote with $\mathcal{M} = \{i,k\}$ a model with two stations labelled by i and k.

Complementary system $\mathcal{M}_1 = \{1\}$.
Note that $\mathcal{M}_1$ is the 2-complementary system. Since only a single station is present, all jobs are in the same resource. Thus, the throughputs are easily computed as

$$X_r^{-2}(\mathbf{N}) = \frac{N_r}{L_{1r} c_1^{-1}(\mathbf{N}) N}$$

and for the populations under consideration we get

$$\begin{aligned} X_1^{-2}(1,0) &= 0.100, \quad X_2^{-2}(1,0) = 0.000 \\ X_1^{-2}(0,1) &= 0.000, \quad X_2^{-2}(0,1) = 0.200 \\ X_1^{-2}(1,1) &= 0.200, \quad X_2^{-2}(1,1) = 0.400 \\ X_1^{-2}(2,1) &= 0.600, \quad X_2^{-2}(2,1) = 0.600 \end{aligned}$$

No other information is kept in memory for $\mathcal{M}_1$.

Complete system $\mathcal{M}_2 = \{1, 2\}$.

Population $\mathbf{n}$=(0,0): initialize $p_2(0,0|0,0) = 1$.

Population $\mathbf{n}$=(1,0): from equations (15) and (7) we get $X_1(1,0) = 0.067$, $X_2(1,0) = 0$, $p_2(1,0|1,0) = 0.330$. Finally, using Reiser's equation (11) we get $p_2(0,0|1,0) = X_1(1,0)X_1^{-2}(1,0)p_2(0,0|0,0) = 0.670$.

Population $\mathbf{n}$=(0,1): similarly to $\mathbf{n}$=(1,0), we exploit (14) to compute $X_1(1,0) = 0$, $X_2(1,0) = 0.071$, $p_2(0,1|0,1) = 0.643$, $p_2(0,0|0,1) = X_2(0,1)X_2^{-2}(0,1)p_2(0,0|0,0) = 0.357$.

Population $\mathbf{n}$=(2,0): the computation is analogous to that of $\mathbf{n}$=(1,0) and we obtain $X_1(2,0) = 0.171$, $X_2(2,0) = 0$, $p_2(2,0|2,0) = 0.286$, $p_2(1,0|2,0) = 0.143$, $p_2(0,0|2,0) = 0.571$.

Population $\mathbf{n}$=(1,1): the computation is analogous to that of $\mathbf{n}$=(1,0) and we obtain $X_1(2,0) = 0.171$, $X_2(2,0) = 0$, $p_2(2,0|2,0) = 0.286$, $p_2(1,0|2,0) = 0.143$, $p_2(0,0|2,0) = 0.571$.

Population $\mathbf{n}$=(2,1): using (12) we compute $X_1(2,1) = 0.182$ and $X_2(2,1) = 0.086$. From (7) we compute $p_2(0,1|2,1) = 0.221$, $p_2(1,0|2,1) = 0.123$, $p_2(1,1|2,1) = 0.443$, $p_2(2,0|2,1) = 0.061$ and $p_2(2,1|2,1) = 0.111$. Finally, we can apply Reiser's equation (11) to derive $p_2(0,0|2,1)$ either from $p_2(0,0|1,1)$ or $p_2(0,0|2,0)$. In general, in order to let $p_2(0,0|\mathbf{n}) \to 0$ smoothly also after one between $p_2(0,0|\mathbf{n} - \mathbf{1}_r)$ or $p_2(0,0|\mathbf{n} - \mathbf{1}_s)$ as produced an underflow, it is sufficient to choose the maximum between the two obtained values of $p_2(0,0|\mathbf{n})$. In this case, since $p_2(0,0|1,1)$ and $p_2(0,0|2,0)$ are far from the bounds of the representation range, the choice is indifferent and we finally get $p_2(0,0|2,1) = 0.041$.

Output: The throughputs are $X_1(2,1) = 0.182$, $X_2(2,1) = 0.086$. The queue-length probabilities are available only for station 2. However, in the special case $M = 2$ the probabilities for station 1 are easily computed from the relation

$$p_1(\mathbf{n}|\mathbf{N}) = p_2(\mathbf{N} - \mathbf{n}|\mathbf{N})$$

If other queue-lengths are required, these must be computed using (7) and the computed throughputs of system $\mathcal{M}_2$.

5. Conclusions

In this chapter we considered numerical instabilities that arise in computational algorithms for product-form queueing networks. In particular we focused on load-dependent models, showing how the application of arbitrary precision arithmetics can solve the problem more efficiently than previously proposed techniques. Furthermore, we gave an illustrative example of a new technique for avoiding round-off exceptions in load-dependent MVA when only two customer classes are present.

References

1. ANSI/IEEE, "IEEE Standard for Binary Floating Point Arithmetic" Std 754-1985 edition (New York) 1985.
2. Bailey, D.H., "High-Precision Arithmetic in Scientific Computation, Computing in Science and Engineering" *AIP/IEEE Comp. Soc.*, (2005) (to appear).
3. Baskett, F., Chandy, K. M., Muntz, R. R., Palacios, F.G., "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers" *Journal of the ACM*, **22**(2), (1975) 248-260.
4. Berger, A.W., Kogan, Y., "Dimensioning Bandwidth for Elastic Traffic in High-Speed Data Networks" *IEEE Trans. on Networking*, **8**(5), (2000), 643-654.
5. Brent, R.P., "The Complexity of Multiple-Precision Arithmetic" The Complexity of Computational Problem Solving (Univ. of Queensland Press) 1976, 126-165.
6. Bruell, S.C., Balbo, G., Afshari, P.V., "Mean Value Analysis of Mixed, Multiple Class BCMP Networks with Load Dependent Service Stations" *Perform. Eval.*, **4**, (1984), 241-260.
7. Buzen, J. P., "Computational algorithms for closed queueing networks with exponential servers" *Commun. ACM*, **16**(9), (1973), 527-531.
8. Casale, G., *The Throughput Analysis of Product-form Queueing Networks* Ph.D. Thesis (Dip. di Elettronica ed Informazione, Politecnico di Milano, Milan, Italy) 2006 (to appear).

9. Conway, A. E., Georganas, N.D., "RECAL - A New Efficient Algorithm for the Exact Analysis of Multiple-Chain Closed Queueing Networks" *JACM*, **33**(4), (1986), 768-791.
10. Denning, P. J., Buzen, J. P., "The Operational Analysis of Queueing Network Models" *ACM Computing Surveys*,**10**(3), (1978), 225-261.
11. Eager, D. L., Sevcik, K.C., "Bound Hierarchies for Multiple-Class Queueing Networks" *Journal of the ACM*, **33**(1), (1986), 179-206.
12. Gelenbe, E., Mitrani, I., *Analysis and synthesis of computer systems* (Academic Press, London and New York) 1980.
13. Gordon, W.J., Newell, G.F., "Closed Queueing Systems with Exponential Servers" *Operations Research*, **14**(2), (1967), 254-265.
14. Granlund, T.,"GNU MP: The GNU multiple precision arithmetic library Version 4.1.2" 2003. `http://www.swox.com/gmp/`.
15. Karatsuba, A., Ofman, Yu, "Multiplication of Many-Digital Numbers by Automatic Computers" Doklady Akad. Nauk SSSR 145, (1962), 293-294. (*Translation in Physics-Doklady* **7**, (1963), 595-596)
16. Kobayashi, H., Gerla, M., "Optimal routing in closed queuing networks" *ACM Transactions on Computer Systems*, **1**(4), (1983), 294-310.
17. Lazowska, J., Zahorjan, J., Graham, G.S., Sevcik, K.C., *Quantitative System Performance:Computer System Analysis Using Queueing Network Models* (Prentice-Hall) 1984.
18. Lam, S. S.,"Dynamic scaling and growth behavior of queuing network normalization constants" *Journal of the ACM*, **29**(2), (1982), 492-513.
19. Lavenberg, S.S., *Computer Performance Modeling Handbook* (Academic Press, Inc) 1983.
20. Lavenberg, S.S., "A Perspective on Queueing Models of Computer Performance" *Perform. Eval.*, **10**(1), (1989), 53-76.
21. Lazowska, J., Zahorjan, J., Graham, G.S., Sevcik, K.C., *Quantitative System Performance:Computer System Analysis Using Queueing Network Models* (Prentice-Hall) 1984.
22. Reiser, M., Kobayashi, H., "Queueing networks with multiple closed chains: Theory and computational algorithms" *IBM J. Res. Dev.*, **19**, (1975), 283-294.
23. Reiser, M., Kobayashi, H., "Numerical methods in separable queueing networks" *Studies Manage Sci.*, **7**, (1977), 113-142.
24. Reiser, M., Lavenberg, S.S., "Mean-value analysis of closed multichain queueing networks" *Journal of the ACM*, **27**(2), (1980), 312-322.
25. Reiser, M., "Mean-value analysis and Convolution Method for Queue-Dependent Servers in Closed Queueing Networks" *Perform. Eval.*, **1**(1), (1981), 7-18.
26. Sauer, C.H., Chandy, K.M., *Computer System Performance Modeling* (Prentice-Hall, Englewoods Cliffs, N.J.) 1981.