CALCULATING JOINT QUEUE LENGTH DISTRIBUTIONS IN PRODUCT FORM QUEUEING NETWORKS

E.DE SOUZA E SILVA
S.S. LAVENBERG *

NCE-05/88

Abril/88

Universidade Federal do Rio de Janeiro
Núcleo de Computação Eletrônica
Caixa Postal 2324
20001 - Rio de Janeiro - RJ
BRASIL

* IBM T.J. Watson Research Center

Esta pesquisa contou com o apoio da IBM-Brasil.
Este trabalho foi publicado como relatório técnico do Centro de Pesquisas da IBM T.J. Watson, RC 13235, Outubro 1987.

Resumo

Um novo algorítmo computacional chamado "Análise de Distribuições por Cadeia" (DAC) é desenvolvido. O algorítmo calcula distribuições conjuntas de tamanhos de filas para redes de filas com solução em forma de produto com centros de serviço do tipo servidor único com taxa fixa, servidores infinitos e servidores dependentes do tamanho da fila. Distribuições conjuntas são essenciais em problemas tais como o cálculo de medidas de confiabilidade usando-se modelos de redes de filas. O algorítmo é eficiente pois o custo para se avaliar probabilidades conjuntas de tamanho de fila é da mesma ordem que o número destas distribuições. Este fato contrasta com o custo de se avaliar estas probabilidades usando-se outros algorítmos existentes na literatura. O algorítmo DAC também computa tamanho médio de filas e *throughputs* de uma maneira mais eficiente que os algorítmos RECAL e MVAC, recentemente propostos. Além do mais, o algorítmo é numericamente estável e a sua forma é muito simples.

Abstract

We develop a new computational algorithm called Distribution Analysis by Chain (DAC) which computes joint queue length distributions for product form queueing networks with single server fixed rate, infinite server and queue dependent service centers. Joint distributions are essential in problems such as the calculation of availability measures using queueing network models. The algorithm is efficient since the cost to evaluate joint queue length probabilities is of the same order as the number of these probabilities. This contrasts with the cost of evaluating these probabilities using previous algorithms. The DAC algorithm also computes mean queue lengths and throughputs more efficiently than the recently proposed RECAL and MVAC algorithms. Furthermore, the algorithm is numerically stable and its recursion is surprisingly simple.

1 Introduction

In the past few years, several algorithms have been developed to solve product form queueing network models with multiple closed chains [LAVE83]. These algorithms can compute marginal queue length distributions, mean queue lengths and throughputs. Among the algorithms developed during the 1975-1980 period, the Convolution algorithm [REIS75] and the Mean Value Analysis (MVA) algorithm [REIS80] are the most popular. The Convolution algorithm requires the computation of the normalization constant. From this constant, mean queue lengths, throughputs and marginal queue length distributions can be efficiently calculated. (Mean response times are immediately obtained using Little's result.) However, this algorithm suffers from floating point range problems, although some methods have been proposed which partially alleviate these problems. The MVA algorithm allows the computation of mean queue lengths and throughputs directly, without having to evaluate the normalization constant. One of the main advantages of MVA is the simplicity of the equations used and the numerical stability of the algorithm. Both Convolution and MVA require approximately the same amount of computation if only mean queue lengths and throughputs are calculated and if the network has single server fixed rate (SSFR) and infinite servers (IS) service centers only. This cost grows roughly exponentially with the number of closed chains in the network. MVA has been extended to solve for networks with queue dependent service centers and to compute marginal queue length distributions. However, this extension is more costly [REIS81].

Recently, Conway and Georganas [CONW86a] developed a computational algorithm called RECAL (Recursion by Chain Algorithm) which has significant computational advantages over the Convolution algorithm when the network to be solved has many closed chains and few service centers. This is true since its computational requirements grows as a polynomial function of the number of closed chains. Like Convolution, RECAL requires the computation of the normalization constant and suffers from the same kind of underflow/overflow problems which may occur in the implementation of the Convolution algorithm. Subsequently, Conway, de Souza e Silva and Lavenberg [CONW86b] developed an algorithm called MVAC (Mean Value Analysis by Chain) that is similar in form and computational requirements to RECAL but does not require the computation of the normalization constant. Like MVA, for networks consisting only of SSFR and IS service centers the equations used involve only the mean performance measures of interest.

As we will discuss, when calculating availability measures using queueing network models and in certain performance modeling applications, it may be required to calculate joint distributions of queue lengths at some or all service centers. (We are refering to the total queue lengths at the service centers, not the queue lengths for each chain at the service centers.) MVA and MVAC cannot be used to compute these joint probabilities. Once the normalization constant is obtained, using Convolution or RECAL, it is possible to directly compute these probabilities. This would require computing the steady state probabilities for the entire state space of the network, and then adding the appropriate probabilities to obtain the joint queue length probabilities.

In this paper we develop a new equation which can be used to compute joint distributions of queue lengths at service centers for product form queueing networks with SSFR, IS and queue dependent service centers. The resulting algorithm will be referred to as DAC (Distribution Analysis by Chain). It will be shown that the computational cost to evaluate joint queue length probabilities is of the same order as the number of these probabilities and is much less than the cost of computing these probabilities from the normalization constant as discussed above. Another important characteristic of the algorithm is that it can compute marginal queue length distributions, mean queue lengths and throughputs more efficiently than RECAL or MVAC. Also, one drawback of RECAL and MVAC is that both algorithms involve the use of modified networks which contain special chains which visit only one center in the network. The use of these chains obscures the description of the algorithms. The DAC algorithm does not involve such modifications and is thus easier to describe and understand.

The paper is organized as follows. In section 2 we derive the main equations used by DAC and present the details of the algorithm. In section 3 we present examples which illustrate the applicability of the algorithm. In section 4 we consider the computational requirements for DAC. Section 5 presents extensions to the basic algorithm which can reduce computational costs when not all joint queue length probabilities are required to be calculated. It is also shown that the equations obtained can be used to compute joint queue length probabilities if an algorithm such as MVA is first used to compute throughputs. Finally, in section 6 we present our conclusions.

2 The Distribution by Chain Algorithm.

We consider closed product form queueing networks with multiple chains and queue dependent service centers. The main result of this section is an algorithm, called the Distribution Analysis by Chain (DAC) algorithm, that allows recursive computation of the joint distribution of queue lengths at all service centers, where we consider the total queue length at a service center and not the queue length for each chain. (We will consider the joint distribution of queue lengths for each chain in section 5.) As was done when deriving the MVAC and RECAL algorithms, we convert every chain that has more than one customer into identical single customer chains. Joint queue length distributions are not affected by this transformation. Therefore, in this section we will restrict our consideration to networks that have a single customer per chain. The following notation is used throughout the paper:

J	$=$	number of service centers.
K	$=$	total number of chains in the network.
$j(k)$	$=$	a specified service center visited by chain k.
θ_{jk}	$=$	visit ratio of a chain k customer to center j, scaled so that $\theta_{j(k)k} = 1$.
T_{jk}	$=$	mean service time of a chain k customer at center j.
a_{jk}	$=$	$\theta_{jk}.T_{jk}$.
$\mu_j(n)$	$=$	service rate of service center j when there are n customers present, $\mu_j(1) = 1$.
n_{jk}	$=$	number of chain k customers at center j.
n_j	$=$	$\sum_{k=1}^{K} n_{jk}$ = number of customers at center j.
$\vec{n}_j$	$=$	$(n_{j1}, \ldots, n_{jK})$ = state of center j.
$\underline{\vec{n}}$	$=$	$(\vec{n}_1, \ldots, \vec{n}_J)$ = state of the network.
$\vec{n}$	$=$	$(n_1, \ldots, n_J)$ = joint queue length.
$\vec{1}_j$	$=$	J-dimensional vector whose j-th element is one and whose other elements are zero.

In the following additional notation a superscript k denotes a quantity for a network when only chains $1, \ldots, k$ are present. Thus, the superscript K denotes the network with full population.

$\vec{1}^k$	$=$	population vector = k-dimensional vector all of whose elements are one.
$\underline{S}^k$	$=$	$\{\underline{\vec{n}} : \sum_{j=1}^{J} \vec{n}_j = \vec{1}^k\}$ = state space of the network.
$P^k(\underline{\vec{n}})$	$=$	steady state probability that the network is in state $\underline{\vec{n}} \in \underline{S}^k$.
S^k	$=$	$\{\vec{n} : \sum_{j=1}^{J} n_j = k,\ n_j$ is a nonnegative integer for all $j\ \}$ = set of possible queue lengths.

$P^k(\vec{n})$	$=$	steady state probability that the joint queue length is $\vec{n} \in S^k$.
$P_j^k(n)$	$=$	steady state probability that there are n customers at service center j.
λ_{jt}^k	$=$	$\theta_{jt}.\lambda_t^k =$ throughput of chain t customer at center j, where $\lambda_t^k = \lambda_{j(t)t}^k$.
L_{jt}^k	$=$	mean number of chain t customers at center j.
W_{jt}^k	$=$	mean waiting time (queueing time + service time) of a chain t customer at center j.

We now derive a recursion relating the joint queue length probabilities for a network to those for a network with one chain removed.

Theorem 1 *For a product form network with queue dependent service centers:*

$$P^k(\vec{n}) = \lambda_k^k \sum_{j=1}^{J} a_{jk} \frac{n_j}{\mu_j(n_j)} P^{k-1}(\vec{n} - \vec{1}_j) \tag{1}$$

Proof:

For a product form network, the steady state probability that the network is in state $\underline{\vec{n}} \in \underline{S}^k$ is given by [LAVE83]

$$P^k(\underline{\vec{n}}) = \frac{1}{G^k} \prod_{l=1}^{J} \frac{n_l!}{\prod_{v=1}^{n_l} \mu_l(v)} \prod_{t=1}^{k} \frac{a_{lt}^{n_{lt}}}{n_{lt}!} \tag{2}$$

where G^k is the normalization constant of the network with single customer chains $1, \ldots, k$ present.

Multiplying and dividing equation (2) by G^{k-1}, we obtain

$$P^k(\underline{\vec{n}}) = \lambda_k^k \frac{1}{G^{k-1}} \prod_{l=1}^{J} \frac{n_l!}{\prod_{v=1}^{n_l} \mu_l(v)} \prod_{t=1}^{k} \frac{a_{lt}^{n_{lt}}}{n_{lt}!}$$

where λ_k^k is obtained from [LAVE83]:

$$\lambda_k^k = \frac{G^{k-1}}{G^k}$$

Since $\underline{\vec{n}}$ is a specified state of $\underline{S}^k$ and there is only one customer in chain k, from all the values $n_{1k}, n_{2k}, \ldots, n_{Jk}$ only one is different than zero, indicating that customer k is in that center. Assume that $n_{jk} = 1$ and $n_{lk} = 0 \ \forall l \neq j$. Then,

$$\begin{aligned} P^k(\underline{\vec{n}}) &= \lambda_k^k \frac{1}{G^{k-1}} \prod_{\substack{l=1 \\ l\neq j}}^{J} \frac{n_l!}{\prod_{v=1}^{n_l} \mu_l(v)} \prod_{t=1}^{k-1} \frac{a_{lt}^{n_{lt}}}{n_{lt}!} \times \frac{n_j!}{\prod_{v=1}^{n_j} \mu_j(v)} \prod_{t=1}^{k-1} \frac{a_{jt}^{n_{jt}}}{n_{jt}!} a_{jk} \\ &= \lambda_k^k a_{jk} \frac{n_j}{\mu_j(n_j)} \left[\frac{1}{G^{k-1}} \prod_{\substack{l=1 \\ l\neq j}}^{J} \frac{n_l!}{\prod_{v=1}^{n_l} \mu_l(v)} \prod_{t=1}^{k-1} \frac{a_{lt}^{n_{lt}}}{n_{lt}!} \frac{(n_j-1)!}{\prod_{v=1}^{n_j-1} \mu_j(v)} \prod_{t=1}^{k-1} \frac{a_{jt}^{n_{jt}}}{n_{jt}!} \right] \end{aligned}$$

The term in brackets in the above equation is $P^{k-1}(\underline{\vec{n}} - \underline{\vec{1}}_{jk})$ (where $\underline{\vec{1}}_{jk}$ is the JK dimensional vector in which all elements are zero except one element at position jk which is one). Therefore,

$$P^k(\underline{\vec{n}}) = \lambda_k^k a_{jk} \frac{n_j}{\mu_j(n_j)} P^{k-1}(\underline{\vec{n}} - \underline{\vec{1}}_{jk}) \tag{3}$$

Now note that

$$P^k(\vec{n}) = \sum_{\underline{\vec{n}} \in \underline{S}^k : \sum_{t=1}^{k} n_{lt} = n_l, \forall l} P^k(\underline{\vec{n}}) \tag{4}$$

Since chain k has only one customer, the sum over all states in equation (4) can be written as the double sum:

$$P^k(\vec{n}) = \sum_{j=1}^{J} \sum_{\substack{\underline{\vec{n}} \in \underline{S}^k : \\ n_{jk}=1, \sum_{t=1}^{k} n_{lt} = n_l, \forall l}} P^k(\underline{\vec{n}}) \tag{5}$$

where the second sum in (5) is over all the states where customer k is in center j. Substituting (3) into (5) yields,

$$P^k(\vec{n}) = \lambda_k^k \sum_{j=1}^{J} a_{jk} \frac{n_j}{\mu_j(n_j)} \sum_{\substack{\underline{\vec{n}} \in \underline{S}^k : \\ n_{jk}=1, \sum_{t=1}^{k} n_{lt} = n_l, \forall l}} P^{k-1}(\underline{\vec{n}} - \underline{\vec{1}}_{jk}) \tag{6}$$

Note that

$$\sum_{\substack{\underline{\vec{n}} : \sum_{t=1}^{k} n_{lt} = n_l, \forall l, \\ n_{jk}=1}} = \sum_{\substack{\underline{\vec{n}} : \sum_{t=1}^{k-1} n_{lt} = n_l, \forall l \neq j, \\ \sum_{t=1}^{k-1} n_{jt} = n_j - 1}}$$

Now, let $\underline{\vec{m}} = \underline{\vec{n}} - \vec{\underline{1}}_{jk}$. Then, $m_{lt} = n_{lt}\ \forall l,\ \forall t \neq k$ and $m_{lk} = 0\ \forall l$ so that, $m_l = n_l\ \forall l \neq j$ and $m_j = n_j - 1$. Therefore, equation (6) can be rewritten as:

$$P^k(\vec{n}) = \lambda_k^k \sum_{j=1}^{J} a_{jk} \frac{n_j}{\mu_j(n_j)} \sum_{\underline{\vec{m}} \in \underline{S}^{k-1} : \sum_{t=1}^{k} m_{lt} = m_l,\ \forall l} P^{k-1}(\underline{\vec{m}}) \tag{7}$$

Since the right most sum in equation (7) is equal to $P^{k-1}(\vec{m})$, (1) is proved. □

We now show that the throughput λ_k^k in equation (1) can be computed from the marginal queue length probabilities for the network with the customer of chain k removed as follows:

$$\lambda_k^k = \frac{1}{\sum_{j=1}^{J} a_{jk} \sum_{n=1}^{k} \frac{n}{\mu_j(n)} P_j^{k-1}(n-1)} \tag{8}$$

To show this we use equation (2.15) in [REIS80] which in our notation becomes

$$L_{jk}^k = \lambda_k^k a_{jk} \sum_{n=1}^{k} \frac{n}{\mu_j(n)} P_j^{k-1}(n-1) \tag{9}$$

Since there is only one customer in chain k, equation (8) follows from summing equation (9) over all j. The marginal queue length probabilities $P_j^{k-1}(n)$ are obtained by summing the joint queue length probabilities, i.e.,

$$P_j^{k-1}(n) = \sum_{\substack{i=1 \\ i \neq j}}^{J} \sum_{n_i=0}^{k-1} P^{k-1}(n_1, n_2, \ldots, n_j = n, \ldots, n_J) \tag{10}$$

Equations (1), (8) and (10), allow the joint queue length probabilities to be computed recursively starting with the marginal queue length probabilities for a network with only chain 1 present. Since there is only one customer in chain 1, it follows that

$$P_j^1(1) = L_{j1}^1 = \frac{a_{j1}}{\sum_{l=1}^{J} a_{l1}} \quad \forall j \tag{11}$$

where the second equality follows from equation (9). We now state the DAC algorithm.

Step 1: Set $k = 1$.
Use equation (11) to calculate $P_j^1(1)$ for all centers j in the network.

Step 2: For $k = 2, \ldots, K$:

(a) Use equation (8) to calculate λ_k^k.

(b) Use equation (1) to calculate $P^k(\vec{n})$ for all $\vec{n} \in S^k$.

(c) Use equation (10) to calculate $P_j^k(n)$ for all j and $n = 0, \ldots, k$.

The DAC algorithm as stated above yields not only the joint queue length probabilities $P^K(\vec{n})$, $\vec{n} \in S^K$, for the network with full population but also yields the full population throughput for chain K, λ_K^K. If equation (9) is also used the full population mean queue lengths for chain K, L_{jK}^K, are also obtained. The algorithm does not directly yield full population throughputs and mean queue lengths for the other chains. We will show below how these quantities can be calculated, if needed. Furthermore, one can easily verify that if a center, say center i, is visited only by a single chain, say chain t, then the throughput for this chain is given by:

$$\lambda_t^k = \frac{P_i^k(1)}{a_{it}} \tag{12}$$

The full population throughputs and mean queue lengths for chains other than chain K can be computed as follows. Let $S^{K-1,d}$ denote the set of possible joint queue lengths when chain d only is omitted from the network and let $P^{K-1,d}(\vec{n})$ be the steady state probability that the joint queue length is $\vec{n} \in S^{K-1,d}$. It is easy to show in the same manner used to prove equation (1) that

$$P^K(\vec{n}) = \lambda_d^K \sum_{j=1}^{J} a_{jd} \frac{n_j}{\mu_j(n_j)} P^{K-1,d}(\vec{n} - \vec{1}_j) \tag{13}$$

Once the $P^K(\vec{n})$ are determined using the DAC algorithm, we compute λ_d^K as follows:

(1) Assume $\lambda_d^K = 1$.

(2) Compute $P^{K-1,d}(\vec{n} - \vec{1}_j)$ from (13) for $\vec{n} = (K, 0, \ldots, 0)$, $(K-1, 1, 0, \ldots, 0)$, $(K-2, 2, 0, \ldots, 0)$, ..., $(K-1, 0, 1, \ldots, 0)$, ..., $(0, \ldots, 0, K)$, in this lexicographic order. Note that $P^{K-1,d}(K-1, 0, \ldots, 0)$ is determined from $P^K(K, 0, \ldots, 0)$; $P^{K-1,d}(K-2, 1, 0, \ldots, 0)$ is determined from $P^K(K-1, 1, 0, \ldots, 0)$ and the recently calculated value of $P^{K-1,d}(K-1, 0, \ldots, 0)$, etc. Therefore, $P^{K-1,d}(\vec{n} - \vec{1}_j)$ for a given value of $(\vec{n} - \vec{1}_j)$ is evaluated from $P^K(\vec{n})$ and previously calculated values of $P^{K-1,d}(\vec{n} - \vec{1}_l), l \neq j$.

(3) $\lambda_d^K = \frac{1}{\sum_{\vec{n} \in S^{K-1,d}} P^{K-1,d}(\vec{n})}$

From (9) and the values calculated above, L_{jd}^K can be evaluated $\forall\, j$.

We emphasize that Steps 1 and 2 of the DAC algorithm compute all the joint queue length probabilities for the full population network, without the need for the additional computation just given. This may be sufficient for a particular application as we will see in the examples of the following section.

3 Examples.

Example 1:

An availability model of a computer system can often be represented by a queueing network and , if certain assumptions are made, by a product form network, e.g. [GOYA87]. In this example we consider an availability model of a computer system with two types of components, say memory and CPU. There are three memory modules, one of which is a spare module and two are active modules. There is only one CPU. We assume that the spare memory cannot fail if it is not being used. Furthermore, when a memory unit fails the spare unit is immediately switched to full operation. We assume independent exponential failure times. Once a component fails it goes to a repair facility. There is only one repair man. He chooses a component to repair at random from the repair queue and a new failure preempts the repair of an old one. We assume independent general repair times that can be different for different types of components. Figure 1 shows the queueing network model for this system.

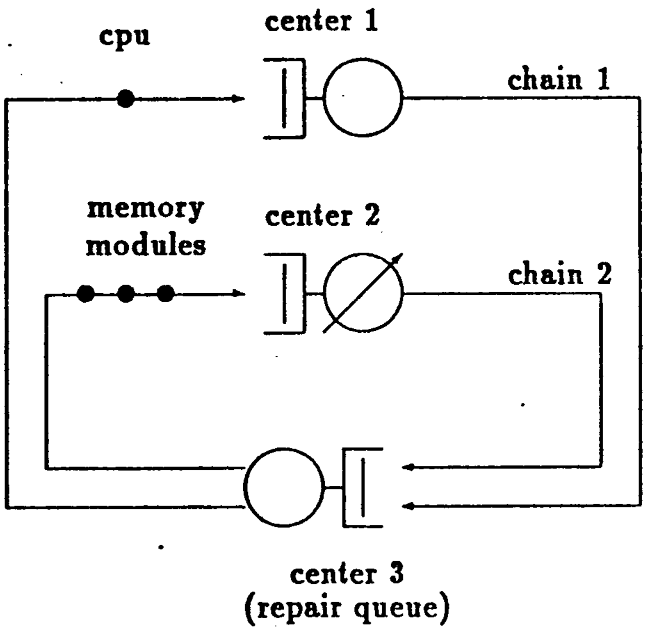

Figure 1: An availability model of a computer system.

In that figure chain 2 has three customers and represents the behavior of the memory modules and chain 1 has one customer and represents the behavior of the CPU. Center 2 models the failure behavior of the memory modules. Note that the queue length dependent center 2 is equivalent to a two server service center which models the fact that only 2 memory modules can fail if 3 modules are in good condition. Center 1 models the failure behavior of the CPU and center 3 models the repair facility. With the assumptions we have made, e.g. exponential failure times and random order of repair service with preemptions, the queueing network has a product form solution [GOYA87]. Appendix A presents the parameter values for this network and the calculations at each step of the algorithm to obtain the joint queue length distribution.

Now we assume that the spare memory is being powered even if it is not in use and thus can also fail (hot stand by). This behavior can be modeled by the same network of Figure 1 if we alter the service rate of center 2 to include the failure rate of the spare unit. In other words, $\mu_2(3)/a_{22} = 2\lambda_{mem} + \lambda_{spare}$ where λ_{mem} and λ_{spare} are the failure rates of the working units and the spare unit, respectively. Appendix A also presents the final results when a hot stand by unit is used. Note that only step $k = 4$ has to be recomputed in this case.

From the example we verify the need to compute the joint queue length probabilities. For instance, the probability that all four components are working simultaneously is simply $P^4(1,3,0)$. If we assume that the system is operational when at least the CPU and one memory module is operational, then the availability of the system is $AV = P^4(1,3,0) + P^4(1,2,1) + P^4(1,1,2)$.

Since the algorithm requires the recursive calculation of the joint queue length probabilities with one less "customer", using the example we can determine, for instance, the degradation of availability when no spare memory is used. This can be done by comparing the results of the original network with a new network with one less memory module (which correspond to the results of step $k = 3$ in appendix A). This contrasts with MVAC for queue dependent centers or RECAL, in which only the last step gives meaningful results. Previous steps produce results for networks with the so called self looping chains.

Example 2:

Consider a queueing model of a distributed system where there are K personal workstations and M computer servers. Each customer uses his/her workstation and once in a while submits a job for processing in one of the computer servers. A queueing model for the system is shown in Figure 2. In this model, the computer servers are represented by queue dependent service centers, since we may expect job delays to increase with increasing number of jobs being processed. Each workstation is used by a single customer whose behavior is represented by a single customer closed chain. If we want to calculate the probability that one (or more) computer servers are processing significantly more jobs than the other servers, then we need to calculate joint queue length distributions.

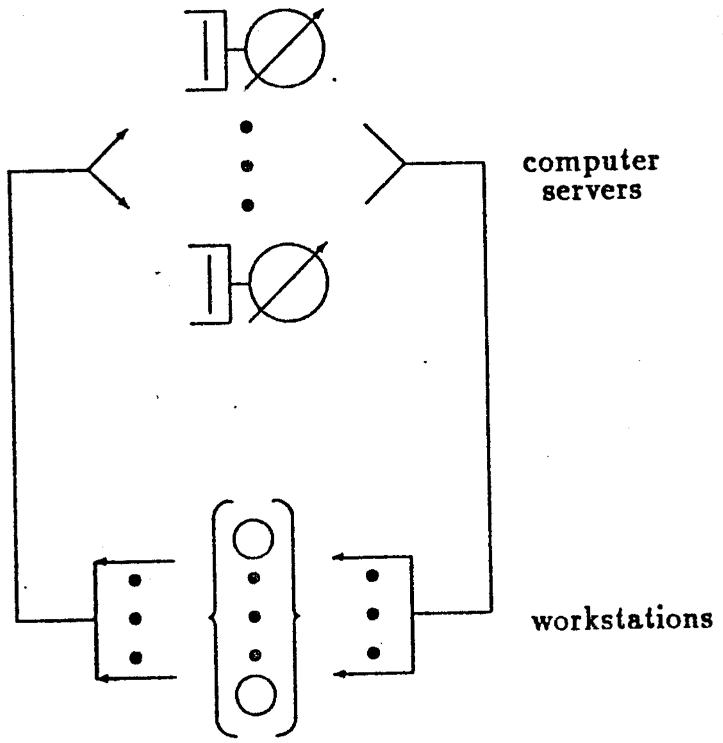

Figure 2: A queueing model of a distributed system.

4 Computational Requirements.

In this section we derive the computational cost of the DAC algorithm. We assume for simplicity that all chains visit all service centers. The cost will be less if this is not the case. We first determine the cost to compute the steady state joint queue length probabilities. We compare this cost with the cost of computing steady state joint queue length probabilities using the Convolution and RECAL algorithms. Next we derive the cost when mean queue lengths and throughputs for all chains are computed. Comparisons with MVAC will also be made.

We point out that significant savings can be obtained if only a subset of these probabilities needs to be calculated, as we will demonstrate in section 5. In the expressions for computational costs, we count only the operations of multiplication and division, each having a unitary cost. The cost of additions will be ignored. We also assume that all centers in the network are queue dependent. Savings can be obtained if we have single server fixed rate and/or infinite server centers.

Step 1 of the algorithm requires J divisions. For a subsequent step k we need to compute:

(a) λ_k^k using equation (8). This requires $(2k+1)$ operations for a single value of j. Therefore, for calculating (8) we need a total of $(2Jk+J+1)$ operations.

(b) $P^k(\vec{n})$ for $\vec{n} \in S(\vec{N})$. From equation (1) it is easy to see that we require $(3J+1)$ operations at most to compute $P^k(\vec{n})$ for a given value of $\vec{n}$. Since there are $\binom{J-1+k}{J-1}$ values of $\vec{n}$, the total cost is:

$$(3J+1)\binom{J-1+k}{J-1} \quad .$$

Finally, the total cost σ_{DAC}^1 of the algorithm is [1]:

$$\begin{aligned}\sigma_{DAC}^1 &= J + \sum_{k=2}^{K}\left[2Jk + J + 1 + (3J+1)\binom{J-1+k}{J-1}\right] \\ &= (3J+1)\binom{J+K}{J} + JK(K+2) + K - 3J^2 - 6J - 2\end{aligned}$$

[1]The cost σ_{DAC}^1 ignores the cost of additions to calculate $P_j^k(n)\ \forall\, j,\ \forall\, n$, since we are taking into account the cost of multiplications and divisions only. However, for a step k, it is easy to see that the cost to calculate $P_j^k(n)\ \forall\, j\ \forall\, n$ is less than $J\binom{J-1+k}{J-1}$ additions.

The first term is the dominant factor. Therefore,

$$\sigma^1_{DAC} \approx (3J+1)\binom{J+K}{J}$$

Both the Convolution and RECAL algorithms require the computation of the normalization constant before computing the joint queue length probabilities. If we assume that the network has K distinct single customer chains and all centers are queue dependent, then the Convolution algorithm calculates the normalization constant in approximately $J3^K$ operations. The RECAL algorithm requires approximately $J\binom{J+K-1}{J}$ operations to calculate the normalization constant (assuming single customer chains). However, once the normalization constant is calculated, we need to evaluate $P(\underline{\vec{n}})$ for all $\underline{\vec{n}} \in \underline{S}^K$, and then sum the appropriate probabilities to compute $P(\vec{n})$ for all $\vec{n} \in S^K$. Each value of $P(\underline{\vec{n}})$ requires roughly JK operations, and there are J^K such probabilities. Therefore, it is considerably more efficient to compute joint length distributions using DAC than using either Convolution or RECAL.

Now we compare the cost of DAC with the cost of MVAC. The basic step of the MVAC algorithm has cost σ^{BAS}_{MVAC}:

$$\sigma^{BAS}_{MVAC} = J(J+2)\frac{K}{J+1}\binom{J+K}{J} > JK\binom{J+K}{J}$$

which is comparable to the DAC algorithm. However, the basic step of the MVAC algorithm only computes mean queue lengths and throughputs for chain K. The DAC algorithm besides computing these measures for chain K computes all joint queue length probabilities.

Next, suppose that we use the DAC algorithm to compute mean queue lengths and throughputs of all chains plus the joint queue length probabilities. Assume that the K chains are partitioned into D subsets of chains, where the chains in any subset are identical and any two chains in different subsets are not identical. From the results of section 2 we see that, after computing joint queue length probabilities, mean queue lengths and throughputs for chain K, we need to use equation (13) for one chain from each of the remaining $D-1$ subsets of chains and for each such chain d for each $\vec{n} \in S^{K-1,d}$. Since (13) requires $(3J+1)$ operations at most, the subsequent steps that need to be performed have cost:

$$\sigma^2_{DAC} < (D-1)(3J+1)\binom{J+K-1}{J-1}$$

and so the total cost will be $\sigma_{DAC} = \sigma^1_{DAC} + \sigma^2_{DAC}$.

So far we have considered networks that have a single customer per chain. Now consider a network that has D distinct chains each having N customers. The equivalent single customer per chain network then has $K = ND$ chains and D subsets of N identical chains each. If we fix J and N and let $D \rightarrow \infty$ it follows that σ_{DAC} is at most $O(D^J)$. The cost of MVAC to compute all throughputs and mean queue lengths is at least $O(D^{J+1})$ and can be as large as $O(D^{J+2})$. (See [CONW86b] for details.) The cost of RECAL is $O(D^{J+1})$.

5 Extensions to the Algorithm.

In this section we extend the DAC algorithm to the case when we only need to calculate the joint queue length probabilities at a subset of the service centers and show the potential savings. Next we make a few observations concerning the application of the algorithm when the goal is to calculate the state probabilities $P^K(\vec{\underline{n}})$ for $\vec{\underline{n}} \in \underline{S}^k$, i.e. when we differentiate customers from different chains at different service centers. Finally we show how the equations obtained in this paper can be applied to calculate joint queue length probabilities in conjunction with the MVA algorithm.

5.1 Calculating Joint Queue Length Probabilities at a Subset of the Service Centers.

We present results that can be used to reduce the cost of the DAC algorithm when the goal is to calculate the joint queue length distributions at a subset of the service centers. Assume that in step k of the algorithm we need to calculate $P^k(n_{l_1}, \ldots, n_{l_v})$ where the set $\{l_i\}$ represent indices of the service centers in the network. We need the following corollary.

Corollary 1 *For a product form queueing network*

$$
\begin{aligned}
P^k(n_{l_1}, \ldots, n_{l_v}) &= \\
&= \lambda_k^k \sum_{\substack{j=1 \\ j \in \{l_i\}}}^{J} a_{jk} \frac{n_j}{\mu_j(n_j)} P^{k-1}(n_{l_1}, \ldots, n_j - 1, \ldots, n_{l_v}) \\
&+ \lambda_k^k \sum_{\substack{j=1 \\ j \notin \{l_i\}}}^{J} a_{jk} \sum_{n_j=1}^{k-(n_{l_1}+\ldots+n_{l_v})} \frac{n_j}{\mu_j(n_j)} P^{k-1}(n_{l_1}, \ldots, n_{l_v}, n_j - 1)
\end{aligned}
\tag{14}
$$

The proof is presented in appendix B.

Note that the joint queue length probabilities at v of the service centers with k single customer chains present depend on the joint queue length probabilities at $v+1$ of the centers with one chain removed. For example consider a queueing network with three service centers and four single customer chains and assume that we wish to calculate the probability that the customer of chain 4 is in center 1, i.e. $P_1^4(1)$. Using (14) to calculate

$P_1^4(1)$ we have:

$$P_1^4(1) = \lambda_4^4 \left[a_{12} P_1^3(0) + a_{22} \sum_{n_2=1}^{3} \frac{n_2}{\mu_2(n_2)} P^3(1, n_2 - 1) + a_{32} \sum_{n_3=1}^{3} n_3 P^3(1, n_3 - 1) \right]$$

To illustrate the computational savings that can be obtained using equation (14) suppose that the goal is to calculate $P_j^K(n_j)$ $1 \le j \le J$ and assume that $K > J$. In this case the computation remains the same up to step $k = K - J + 1$, i.e. the joint queue length probabilities at all centers need to be calculated. However, from this step up to $k = K$, the joint probabilities need not be calculated at all centers. At step $k = K - J + 1 + v$, $1 \le v \le J - 1$ the number of computations is:

$$\sigma^L(k) \approx (3J + 1) \left[\sum_{i=0}^{k} \binom{J - v - 1 + i}{J - v - 1} \binom{J}{J - v} \right] \tag{15}$$

This is true since, from equation (14), at step $k = K - J + 1 + v$ we need to calculate joint probabilities of the type $P^k(n_{l_1}, \ldots, n_{l_{J-v}})$ for all combinations of $J - v$ centers from the J original centers. There are $\binom{J}{J-v}$ combinations of J centers (one such combination is $(n_{l_1}, \ldots, n_{l_{J-v}})$). For each combination of centers, if the total population in these centers is i $(0 \le i \le k)$, the number of possible joint probabilities is $\binom{J-v-1+i}{J-v-1}$. The population in centers $l_1, \ldots, l_{J-v}$ may vary from 0 to k. Therefore the number of joint probabilities of the type $P^k(n_{l_1}, \ldots, n_{l_{J-v}})$ is given by the term in brackets in (15). Since each joint probability requires at most $(3J + 1)$ operations to be evaluated, (15) is obtained. Simplifying (15):

$$\sigma^L(k) \approx (3J + 1) \binom{J - v + k}{J - v} \binom{J}{J - v} = (3J + 1) \binom{K + 1}{K - k + 1} \binom{J}{K - k + 1} \tag{16}$$

Comparing the above cost with the cost obtained in section 4 for step k (the cost is for calculating, at step k, all the joint queue length probabilities), we see that we use the previous algorithm up to step $k_r - 1$ such that

$$\binom{J - 1 + k_r}{J - 1} \ge \binom{K + 1}{K - k_r + 1} \binom{J}{K - k_r + 1}$$

Then, for steps $k_r \le k \le K$ we use the steps outlined above, i.e. equation (14). For example, if $K = 22$ and $J = 8$ then $k_r = 19$. Therefore in the last four steps we use equation (14).

5.2 Calculating State Probabilities.

Equation (3) in section 2 gives a recursion relating the state probabilities to those with one chain removed for a network that has one customer per chain. Suppose a network has D distinct chains, each which can have more than one customer. We will consider both this network which we denote $\mathcal{N}^*$ and its single customer per chain equivalent which we denote $\mathcal{N}$. We let $\underline{\vec{n}}^*$ denote a state for $\mathcal{N}^*$. For each such state $\underline{\vec{n}}^*$ there is a corresponding subset of states for $\mathcal{N}$. Let $S^k(\underline{\vec{n}}^*)$ denote this subset where the superscript k denotes a quantity when only chains $1, \ldots, k$ are present in $\mathcal{N}$. Let $d(k)$ denote the chain in $\mathcal{N}^*$ corresponding to chain k in $\mathcal{N}$. Then, it follows from summing equation (3) over $\underline{\vec{n}} \in S^k(\underline{\vec{n}}^*)$ that

$$P^k(\underline{\vec{n}}^*) = \lambda_k^k \sum_{j=1}^{J} a_{jd(k)} \frac{n_j}{\mu_j(n_j)} P^{k-1}(\underline{\vec{n}}^* - \vec{1}_{jd(k)}) \qquad (17)$$

However, there is a more efficient way to calculate $P(\underline{\vec{n}}^*)$, as shown in the following corollary. We need the following additional notation. Let N_d^k denote the number of chains in $\mathcal{N}$ that are present when only chains $1, \ldots, k$ are present and that correspond to chain $d \in \mathcal{N}^*$.

Corollary 2 *For a product form queueing network with queue dependent centers:*

$$P^k(\underline{\vec{n}}^*) = \lambda_k^k a_{jd(k)} \frac{n_j}{\mu_j(n_j)} \frac{N_{d(k)}^k}{n_{jd(k)}} P^{k-1}(\underline{\vec{n}}^* - \vec{1}_{jd(k)}) \;\; \forall\, j \textit{ such that } n_{jd(k)} > 0 \qquad (18)$$

Proof:

Since the N_d^k $(1 \leq d \leq D)$ chains in $\mathcal{N}$ corresponding to chain d in $\mathcal{N}^*$ have the same probabilistic behavior, $P^k(\underline{\vec{n}}) = P^k(\underline{\vec{n}}')$ if $\underline{\vec{n}} \in S^k(\underline{\vec{n}}^*)$ and $\underline{\vec{n}}' \in S^k(\underline{\vec{n}}^*)$. Now we note that (a) in the set $S^k(\underline{\vec{n}}^*)$ there is at least one state $\underline{\vec{n}}(j)$ such that single customer chain k is in center j, if $n^*_{jd(k)} > 0$, (b) there are $\Gamma(\underline{\vec{n}}^*)$ states in the set $S^k(\underline{\vec{n}}^*)$, where

$$\Gamma(\underline{\vec{n}}^*) = \frac{N_1^k!}{n^*_{11}! \ldots n^*_{J1}!} \cdots \frac{N_D^k!}{n^*_{1D}! \ldots n^*_{JD}!}$$

and each combinatorial fraction above corresponds to a distinct chain d, $1 \leq d \leq D$, and represents all the possible combinations of customers from that chain in each service center in $\mathcal{N}$. Therefore, since all states in the set $S^k(\underline{\vec{n}}^*)$ have the same probability:

$$P^k(\underline{\vec{n}}^*) = \Gamma(\underline{\vec{n}}^*) P^k(\underline{\vec{n}}(j)) \;\; \forall\, j \text{ such that } n^*_{jd(k)} > 0 \qquad (19)$$

Substituting (3) in (19) we have:

$$P^k(\underline{\vec{n}}^*) = \lambda_k^k a_{jk} \frac{n_j}{\mu_j(n_j)} P^{k-1}(\underline{\vec{n}} - \underline{\vec{1}}_{jk}) \prod_{d=1}^{D} \frac{N_d^k!}{\prod_{l=1}^{J} n_{ld}^*!} \quad \forall\, j \text{ such that } n_{jd(k)}^* > 0 \tag{20}$$

Now, from (19):

$$P^{k-1}(\underline{\vec{n}} - \underline{\vec{1}}_{jk}) = \frac{1}{\prod_{d=1}^{D} \frac{(N_d^k - \delta_d(k))!}{(n_{jd(k)}^* - \delta_d(k))!}} P_*^{k-1}(\underline{\vec{n}}^* - \underline{\vec{1}}_{jd(k)}) \quad \forall\, j \text{ such that } n_{jd(k)}^* > 0 \tag{21}$$

where

$$\delta_d(k) = \begin{cases} 1 & \text{if } d = d(k) \\ 0 & \text{otherwise} \end{cases}$$

Substituting (21) in (20), (18) is obtained. □

The computational cost depends on the number of states $\underline{\vec{n}}^*$ which need to be calculated. Note that if the values of λ_k^k $1 \le k \le K$ are known then the probability for any state $\underline{\vec{n}}^*$ can be easily calculated from equation (18) in K steps, each step requiring at most six operations. In summary, to calculate $P^k(\underline{\vec{n}}^*)$ for any state $\underline{\vec{n}}^*$, we:

1. Use the DAC algorithm to calculate λ_k^k $\forall\, k$ and store these values.

2. Use equation (18) recursively.

Example 3:

Consider a computer system with redundancy as illustrated in Figure 3. In this system, one of the CPU's is spare and critical data ($data_1$ and $data_2$) are replicated. The availability model for this system is shown in Figure 4 where the chains visiting the IS center represent the bus, controller 1, controller 2 and disks 1, 2 and 3. We assume that all data is available if there is a path between the operational CPU and both data items $data_1$ and $data_2$. Therefore, to determine the probability that the data is available we need to calculate a subset of $\{P^8(n_1, \vec{n}_2, n_3)\}$, which requires equation (18).

Consider now a more general case. Assume that we have C distinct types of components, C_1 of them have spare units and C_2 of them do not. For simplicity, assume that all types of components have N units (including spares if any). Therefore, the availability model has $C_1 + 2$ centers where C_1 centers are queue dependent, one is an IS center and

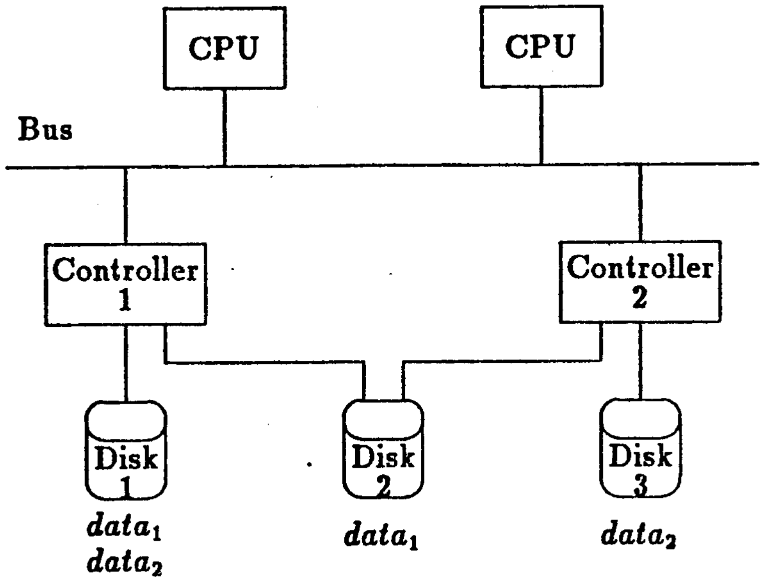

Figure 3: A computer system with redundancy.

the other one represents the repair man. In this model each chain visits only two centers. It is easy to show that the total cost to calculate all the throughputs using DAC is:

$$\sigma_{DAC}^{tot} \cong 7(NC_2+1)\left[\frac{NC_2+2}{2}+\frac{N+2}{2}\left(\sum_{v=1}^{C_1}(N+1)^v\right)+(C_2-1)(N+1)^{C_1}\right]$$

Once the throughputs are calculated, availability measures can be easily obtained, using equation (18).

5.3 Calculating Joint Probabilities Using the MVA Algorithm.

The recursive equations derived in this paper can be used to obtain joint queue length probabilities and/or state probabilities not only when the DAC algorithm is used, but also when the MVA algorithm is used.

To see that, assume that MVA was used to obtain mean performance measures. Therefore, the throughput of each chain is calculated for all population vectors in the network. Consider one such population vector $(N_1^k, \ldots, N_D^k)$ such that $\sum_{i=1}^{D} N_i^k = k$. This network is equivalent to a single customer per chain network where chains $1, \ldots, k$ are present.

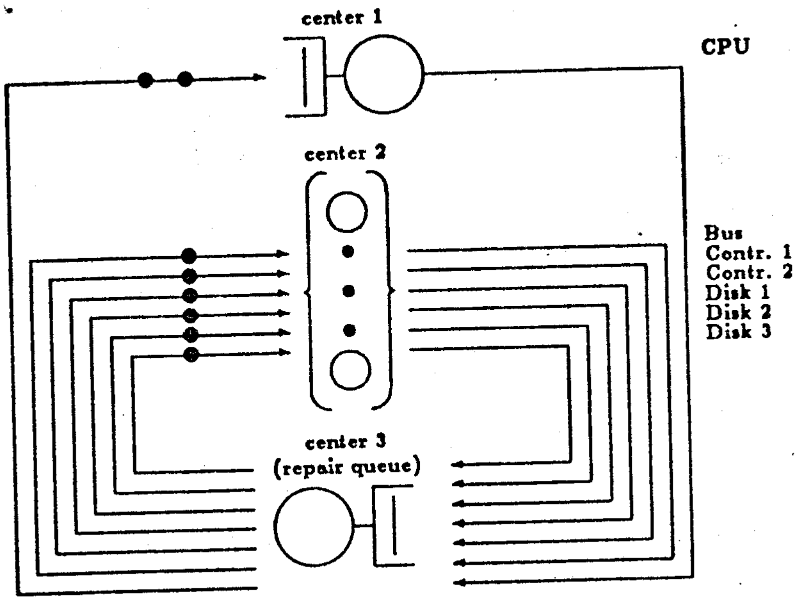

Figure 4: The availability model of the system of Figure 3.

Therefore,

$$\lambda_k^k = \frac{\lambda_{d(k)}(N_1^k, \ldots, N_D^k)}{N_{d(k)}^k} \qquad \text{if } N_{d(k)}^k > 0$$

For example, consider a network with three chains (labeled d_1, d_2 and d_3) and two customers per chain. Then, $\lambda_{d_i}(\vec{N})$ is calculated, $1 \leq i \leq 3 \; \forall \; \vec{N} \in \{(100), (010), \ldots, (222)\}$. Assume that we store the values of $\lambda_{d_i}(\vec{M})$, $1 \leq i \leq 3$ $\vec{M} \in \{(100), (200), (210), (220), (221), (222)\}$. It is easy to see that if we consider an equivalent single customer per chain network and label the single customer chains as $1, \ldots, 6$, where chains 1 and 2 correspond to chain d_1, etc, then the values of $\lambda_{d_1}(100)$, $\lambda_{d_1}(200)$, $\lambda_{d_2}(210)$, $\ldots$, $\lambda_{d_3}(222)$ would be the values of $\lambda_1^1, 2\lambda_2^2, \lambda_3^3 \ldots, 2\lambda_6^6$, respectively. In general, we would need to store $K = \sum_k N_k$ throughput values only. With these throughput values available, we can use equation (18) or (14) to calculate the desired probabilities.

6 Conclusions.

We developed a new algorithm, called DAC, for calculating joint queue length distributions at service centers in product form queueing networks with single server fixed rate, infinite servers and queue dependent service centers. These distributions are needed for availability modeling applications and certain performance modeling applications. The algorithm is simple and efficient. The cost for evaluating joint probabilities of queue lengths is of the same order as the number of these probabilities. If the measures of interest are mean queue lengths and throughputs, we have shown that DAC is more efficient than the recently proposed RECAL and MVAC algorithms. Therefore we believe that DAC is the algorithm of choice when (a) joint queue length distributions need to be computed or (b) only mean performance measures need to be computed and there are few service centers and many chains, i.e. when MVAC or RECAL would previously have been the algorithm of choice.

We also introduced extensions to the basic algorithm. These extensions are reducing computational costs when joint queue length distributions are only required at a subset of the service centers, calculating state probabilities and combining the algorithm with any algorithm, such as MVA, that can be used to compute throughputs.

APPENDIX

A Numerical Examples from Section 3.

The parameter values for the network of Figure 1 are (assuming the spare memory unit cannot fail): $J = 3$, $a_{11} = 5$, $a_{12} = 0$, $a_{21} = 0$, $a_{22} = 10$, $a_{31} = 2$, $a_{32} = 1$. Since we are assuming chains with single customers, we subdivide chain 2 into 3 chains: 2, 3, and 4. Therefore, $K = 4$.

- Computations for $k = 1$.

  $P_1^1(1) = L_{11}^1 = \frac{a_{11}}{a_{11}+a_{31}} = 0.7143$
  $P_2^1(1) = L_{21}^1 = 0$
  $P_3^1(1) = L_{31}^1 = 0.2857$

- $k = 2$.

  $\lambda_2^2 == 0.08861$

  $P^2(2,0,0) = 0$; $P^2(1,1,0) = 0.6329$; $P^2(1,0,1) = 0.06329$
  $P^2(0,2,0) = 0$; $P^2(0,1,1) = 0.2532$; $P^2(0,0,2) = 0.05063$

  $P_1^2(0) = 0.3038$, $P_1^2(1) = 0.6962$, $P_1^2(2) = 0$
  $P_2^2(0) = 0.1139$, $P_2^2(1) = 0.8861$, $P_2^2(2) = 0$
  $P_3^2(0) = 0.6329$, $P_3^2(1) = 0.3165$, $P_3^2(2) = 0.05063$

- $k = 3$.

  $\lambda_3^3 = 0.08758$

  $P^3(3,0,0) = P^3(2,1,0) = P^3(2,0,1) = P^3(0,3,0) = 0$
  $P^3(1,2,0) = 0.5543$, $P^3(1,1,1) = 0.1109$, $P^3(1,0,2) = 0.01109$
  $P^3(0,2,1) = 0.2217$, $P^3(0,1,2) = 0.08869$, $P^3(0,0,3) = 0.0133$

  $P_1^3(0) = 0.3237$, $P_1^3(1) = 0.6763$, $P_1^3(2) = 0$, $P_1^3(3) = 0$
  $P_2^3(0) = 0.02439$, $P_2^3(1) = 0.1996$, $P_2^3(2) = 0.7761$, $P_2^3(3) = 0$
  $P_3^3(0) = 0.5543$, $P_3^3(1) = 0.3326$, $P_3^3(2) = 0.09978$, $P_3^3(3) = 0.0133$

- $k = 4$.

  $\lambda_4^4 = 0.06472$

$P^4(4,0,0) = P^4(3,1,0) = P^4(3,0,1) = P^4(2,2,0) = P^4(2,1,1) = P^4(2,0,2) = P^4(0,4,0) = 0$

$P^4(1,3,0) = 0.5381$, $P^4(1,2,1) = 0.1076$, $P^4(1,1,2) = 0.02152$
$P^4(1,0,3) = 0.002152$, $P^4(0,3,1) = 0.2153$, $P^4(0,2,2) = 0.08609$
$P^4(0,1,3) = 0.02582$, $P^4(0,0,4) = 0.003442$

The final results when the spare memory unit can fail (assume $\lambda_{spare} = 1/20$) are:

- $k = 4$

  $\lambda_4^4 = 0.0762$

  $P^4(4,0,0) = P^4(3,1,0) = P^4(3,0,1) = P^4(2,2,0) = P^4(2,1,1) = P^4(2,0,2) = P^4(0,4,0) = 0$

  $P^4(1,3,0) = 0.5068$, $P^4(1,2,1) = 0.1267$, $P^4(1,1,2) = 0.02535$
  $P^4(1,0,3) = 0.002535$, $P^4(0,3,1) = 0.2027$, $P^4(0,2,2) = 0.1014$
  $P^4(0,1,3) = 0.03041$, $P^4(0,0,4) = 0.004054$

B Proof of Corollary 1.

The proof follows directly from Theorem 1. First we note that:

$$P^k(\vec{n}_{l_1},\ldots,\vec{n}_{l_v}) = \sum_{\substack{\vec{n}_m=\vec{0}\\ \forall m\notin\{l_i\}}}^{\vec{1}^k} P^k(\underline{\vec{n}})$$

where $\vec{0}$ is the k-dimensional vector in which all elements are zero, $\sum^{\vec{1}^k}_{\substack{\vec{n}_m=\vec{0}\\ \forall m\notin\{l_i\}}} = \sum_{\vec{n}_{u_1}}\sum_{\vec{n}_{u_2}}$ $\ldots\sum_{\vec{n}_{u_m}}$ and the set $\{u_r\}$ contains indices of service centers not contained in the set $\{l_i\}$. Using equation (3):

$$P^k(\vec{n}_{l_1},\ldots,\vec{n}_{l_v}) = \lambda^k_k \sum_{\substack{\vec{n}_m=\vec{0}\\ \forall m\notin\{l_i\}}}^{\vec{1}^k} \left[a_{jk}\frac{n_j}{\mu_j(n_j)}\right] P^k(\underline{\vec{n}}-\vec{1}_{jk}) \tag{22}$$

We have to consider two cases:

(a) $j\in\{l_i\}\ 1\le i\le v$.
In this case the term in brackets in equation (22) can be shifted outside the two sums:

$$P^k(\vec{n}_{l_1},\ldots,\vec{n}_{l_v}) = \lambda^k_k a_{jk}\frac{n_j}{\mu_j(n_j)} \sum_{\substack{\vec{n}_m=\vec{0}\\ \forall m\notin\{l_i\}}}^{\vec{1}^k} P^{k-1}(\underline{\vec{n}}-\vec{1}_{jk})$$

Since the sum above is for all states $\vec{n}_m$ such that $m\notin\{l_i\}$ and, by definition, $P^k(\underline{\vec{n}})=0$ for $n_{jk}=0$,

$$P^k(\vec{n}_{l_1},\ldots,\vec{n}_{l_v}) = \lambda^k_k a_{jk}\frac{n_j}{\mu_j(n_j)} P^{k-1}(\vec{n}_{l_1},\ldots,\vec{n}_j-\vec{1}_k,\ldots\vec{n}_{l_v}) \tag{23}$$

(b) $j\notin\{l_i\}\ 1\le i\le v$.
In this case the term in brackets in equation (22) can be shifted outside the sum when $m\neq j$ only. Therefore:

$$\begin{aligned}
P^k(\vec{n}_{l_1},\ldots,\vec{n}_{l_v}) &= \\
&= \lambda^k_k a_{jk} \sum_{\vec{n}_j=\vec{0}}^{\vec{1}^k} \frac{n_j}{\mu_j(n_j)} \sum_{\substack{\vec{n}_m=\vec{0}\\ \forall m\notin\{l_i\},\, m\neq j}}^{\vec{1}^k} P^{k-1}(\underline{\vec{n}}-\vec{1}_{jk}) = \\
&= \lambda^k_k a_{jk} \sum_{n_j=0}^{k} \frac{n_j}{\mu_j(n_j)} \sum_{|\vec{n}_j|=n_j} P^{k-1}(\vec{n}_{l_1},\ldots,\vec{n}_{l_v},\vec{n}_j-\vec{1}_k)
\end{aligned} \tag{24}$$

where $|\vec{n}_j| = \sum_{t=1}^{k} n_{jt}$, and we wrote $\sum_{\vec{n}_j=\vec{0}}^{\vec{1}^k}$ as the double sum: $\sum_{n_j=0}^{k} \sum_{|\vec{n}_j|=n_j}$.

Combining (23) and (24) and using the same arguments as in the proof of Theorem 1 we obtain (14). $\square$

References

[CONW86a] A.E. Conway & N.D. Georganas, "RECAL - A New Efficient Algorithm for the Exact Analysis of Multiple-Chain Closed Queueing Networks", *JACM*, vol. 33, no. 4, pp. 768-791, October 1986.

[CONW86b] A.E. Conway, E. de Souza e Silva & S.S. Lavenberg "Mean Value Analysis by Chain of Product Form Queueing Networks", IBM Research Report, RC 11641, April 1986, to appear in IEEE Transactions on Computers.

[GOYA87] A. Goyal, S.S. Lavenberg & K.S. Trivedi "Probabilistic Modeling of Computer Systems Availability", Annals of Oper. Res., vol. 8, pp. 285-306, 1987.

[LAVE83] S.S. Lavenberg (Editor), "Computer Performance Modeling Handbook", *Academic Press, New York*, 1983.

[REIS75] M. Reiser & H. Kobayashi "Queueing Networks with Multiple Closed Chains: Theory and Computational Algorithms", *IBM J. Res. Development* vol. 19, pp. 283-294, 1976.

[REIS80] M. Reiser & S.S. Lavenberg, "Mean Value Analysis of Closed Multichain Queueing Networks" *JACM*, no. 27, pp. 313-322, 1980.

[REIS81] M. Reiser, "Mean-Value Analysis and Convolution Method for Queue-Dependent Servers in Closed Queueing Networks", *Performance Evaluation*, no. 1, pp. 7-18, 1981.