MAXIMUM PROCESSING RATES OF MEMORY BOUND SYSTEMS

by

R. M. Bryant

Computer Science Technical Report #362

AUGUST 1979

MAXIMUM PROCESSING RATES OF MEMORY BOUND SYSTEMS

R. M. Bryant

University of Wisconsin-Madison

Madison, Wisconsin

ABSTRACT

Multiresource queueing systems are of particular importance in modelling computer systems because a job must have access to a processor and main memory simultaneously in order to proceed. Existing methods of determining maximum processing rates for multiresource queueing systems are limited to small memory sizes because problem complexity grows exponentially with increasing memory size. By restriciting our attention to a particular scheduling discipline (first come first loaded or FCFL) and treating memory as the limiting resource, methods of calculating maximum processing rates of memory bound systems for realistic main memory sizes are derived. The distribution of the number of jobs loaded under the FCFL policy is given in terms of a convolution of the memory request size distribution. Time averaged behavior of the number of loaded jobs is also found. Finally, the framework is

extended to allow multiple job classes in the input stream. Results of this approach allow one to estimate main memory size requirements from a workload characterization given in terms of arrival rate, memory size distribution and CPU service rate.

KEY WORDS AND PHRASES: computer system models, multiresource queue, computer system design, memory allocation, finite memory size models.

CR CATEGORIES: 4.32, 4.35, 4.6, 5.5, 6.29, 8.1.

MAXIMUM PROCESSING RATES OF MEMORY BOUND SYSTEMS

R. M. Bryant
University of Wisconsin-Madison
Madison, Wisconsin

1. INTRODUCTION

In a series of papers [23; 26; 28; 27], Omahen and Marathe have defined and explored some of the properties of multiresource queueing systems. In their terminology [27], a multiresource queueing system is a single congestion point associated with a number of resources. Each arriving job simultaneously requires some combination of those resources in order to be processed. The primary result from their approach is an algorithm [27] for calculating the capacity bound (or maximum processing rate) of a multiresource queueing system. Their solution is based on solving a linear programming problem in which the proportions of time spent by the system in each of its feasible states are the unknowns. Equality constraints for the problem are derived using Little's result [22] and the law of total probability.

Such queueing systems are particularly important in the modelling of computer systems, where a job must have access to both main memory and a processor in order to proceed.

One problem with using a multiresource queue to represent this dual resource queueing system is that the number of feasible states grows exponentially with increasing main memory size. For example, in [10] it is shown that for a two processor system with 100 blocks of memory, there are more than 190,000,000 feasible states. It is unlikely that a linear programming package will ever be written which can handle this number of independent variables. For this reason, the applications of Omahen and Marathe's theory have only been made to systems with very small main memory sizes. (In [27] an example is solved for a system with eight blocks of main memory.)

In this paper we are concerned with determining maximum processing rates of processor and memory dual resource queueing systems for realistic main memory sizes (hundreds of blocks of memory). We will treat memory as the limiting resource, hence we are concerned with "memory bound" systems. To do this, we will assume a specific scheduling discipline. Thus we lose the advantage of Omahen and Marathe's approach, which allows one to determine the optimal schedule with respect to throughput.

Related work. Papers concerned with analytically modelling the simultaneous use of processors and memory can be divided into two groups depending on whether the multiprogramming level (MPL) is constant or allowed to vary in relation to the memory sizes of loaded jobs. Models of the first type are discussed in [24; 20; 21] and apply to

paging systems with a fixed upper limit on the MPL and a saturated memory queue. Such models are often insensitive to small changes in main memory size or to any change in the memory size request distribution not reflected by a change in the mean. Models of the second type as applied to paging systems have been studied by Brandwajn [5; 6; 7] and Gelenbe [2; 15]. The effect of memory size in these models is modelled indirectly through the use of lifetime functions. Models of the variable MPL type in which the memory request size distribution is an explicit input have been studied by Bard [3] and by Brown et al [8]. In this paper we simplify some of the results of [3] and place some of the results of [8] on a rigorous foundation.

Another class of papers deals with memory allocation in swapping systems. Betteridge [4] considers a Markov Chain model of memory allocation with contiguous placement of each job's address space. Once again the exponential growth of the state space limits application of this approach to very small memory sizes. Buzen and Rubin [12] discuss the effect of compaction in swapping type computer systems. They derive a formula for calculating the distribution of residual (unused) core which will be present even when as many jobs as possible are loaded into memory. We will discuss the relation between these results and the results of this paper in Sections 2 and 3. For a more detailed survey of the literature see [10].

Notation and Assumptions. We consider a computer

system with M independently allocatable units (blocks) of main storage. We assume that the size of main memory is the limiting resource in the system; thus as many jobs as possible are loaded at any given time. The remaining jobs wait in a queue (the memory queue) and we will assume that the arrival rate of jobs is high enough that the memory queue is never allowed to become empty. We assume that no external storage fragmentation takes place. This is equivalent to assuming that memory is compacted at each job departure time [12] or that paging hardware is used to eliminate external storage fragmentation [3]. We require, however, that the entire storage requirement of a job must fit into main memory before the job can be loaded. Finally, we assume that jobs are loaded strictly in the order of their arrival, i. e. the loading policy is first come first loaded (FCFL).

The resource requirements of each job are represented by the pair of independent random variables (X_i, S_i). X_i denotes the storage requirement (in blocks) of the i^{th} job; the sequence $\{X_i\}$ is assumed to be i. i. d. (independent and identically distributed) with distribution $F(x)$ and density $f(x)$. The X_i's are required to be integer valued with $1 \leq X_i \leq m$ where $m \leq M$. X_i can be interpreted as either the program size of the the i^{th} job in a swapping system, or as the working set size of the i^{th} job in a paging system. S_i represents the service time requirement of the i^{th} job. The sequence $\{S_i\}$ is i. i. d. with an exponential distribution

of parameter μ.

Let L(t) denote the number of loaded jobs (the MPL) at time t and let P denote the number of processors on the system, $1 \leq P \leq \infty$. We assume that if $L(t) \leq P$ then each job is assigned its own processor and each job receives service at a unit rate. If $L(t) > P$, then processor sharing [13] takes place and each job receives service at the rate of $P/L(t)$. When a job's service requirement has been satisified, its memory space is freed and as many jobs as possible are loaded from the memory queue. This process is assumed to occur instantaneously. We let $t_1 = \emptyset$ and for $k > 1$ we let t_k denote the departure time of the $k-1^{st}$ job to leave the system; we note that jobs need not depart in the same order that they arrive. Let $L_k = L(t_k^+)$ be the MPL just after the $k-1^{st}$ job departs. We will call the sequence $\{L_k\}$ the MPL sequence.

Throughout this paper, we will use subscripts to indicate particular members of a sequence, e. g. X_k. Capital letters are (usually) used to indicate random variables, lower case letters indicate values for the associated random variables. Underbars are used to indicate vector quantities and superscripts are used to indicate elements of a vector. Thus z_k^j is the j^{th} element of $\underline{Z}_k$, the latter being a random vector. Square brackets are used to indicate events (such as [X=x]). The probability of this event is denoted by Pr{X=x}; the expected value of the random variable X is denoted by E{X} or $\bar{X}$. A box $\square$ is used

to indicate the end of a proof or the end of a theorem with no formal proof. A box is also used to set off extended examples from the text. The symbol .≡. is used to indicate "defined as" and the symbol ≡ is used to mean "identically equal to".

Summary of results. In Section 2 we explore the properties of the distribution of memory sizes of jobs in system at time t_k. We show that under the loading policy FCFL, this distribution is the same at time t_k as it is at time t_1. Then in Section 3 we use this result to determine properties of the MPL sequence $\{L_k\}$. In particular we determine the marginal distribution of $\{L_k\}$. Section 4 contains an extension of these results to the process L(t) and we consider some applications. Finally in Section 5 we generalize this framework to allow for multiple job classes.

2. THE MEMORY STATE VECTOR SEQUENCE $\{\underline{Z}_k\}$

Let the vector $\underline{Z}_k = (Z_k^1, Z_k^2, \ldots, Z_k^M)$ represent the memory sizes of the oldest M jobs in system at time t_k. Here Z_k^1 is the memory requirement of the oldest job, Z_k^2 is the memory requirement of the second oldest job, and so forth. (Note that Z_k^i is not the same as X_{i+k-1}, $k>1$, since jobs need not depart in the same order as they arrive.) Then we can calculate L_k as $h(\underline{Z}_k)$ where $h(\underline{z})$ is defined as:

$$h(\underline{z}) = \begin{cases} M \text{ if } \sum_{i=1}^{M} z^i = M \\ \\ n \text{ if } \sum_{i=1}^{n} z^i \leq M \text{ and } \sum_{i=1}^{n+1} z^i > M \end{cases}$$

where $\underline{z}=(z^1,z^2, \ldots ,z^M)$.

We will call $\underline{Z}_k$ the memory state vector at time t_k. By examining the sequence $\{\underline{Z}_k\}$ we will determine some of the properties of the MPL sequence $\{L_k\}$.

Now consider the set of loaded jobs at time t_k. Their memory requirements are given by the list $z_k^1, z_k^2, \ldots, z_k^n$ where $n=h(\underline{Z}_k)$. One of these jobs will be the next job to depart. $\underline{Z}_{k+1}$ can therefore be obtained by deleting one of the first $h(\underline{Z}_k)$ elements from the vector $\underline{Z}_k$, shifting the remaining values left, and then inserting a new memory size requirement z_{k+1}^M in the M^{th} position in the vector.

Let D_k be the position in the memory state vector of the job which departs at time t_k $(k>1)$. Let $a(i,j)=\Pr\{D_k=i \mid L_k=j\}$. Clearly, $a(i,j)=0$ if $i<1$ or $i>j$. To avoid trivial cases, let us assume that $a(1,j)>0$ for all j. Normally (due to our exponential service time assumptions), we will have $a(i,j)=1/j$, $1 \leq i \leq j$, and $a(i,j)=0$ otherwise. Then the sequence $\{\underline{Z}_k\}$ is a time-homogenous, vector-valued Markov chain. It follows that:

Lemma 2.1: $\{\underline{Z}_k\}$ is a positive recurrent Markov chain

with a unique stationary distribution.

Proof: $\underline{Z}_k$ can take on at most m^M values so that the state space is finite. Let $\underline{z}_1$ and $\underline{z}_2$ be any two possible states, that is any states which satisfy

$$\prod_{i=1}^{M} f(z_1^i) > \emptyset \text{ and } \prod_{i=1}^{M} f(z_2^i) > \emptyset.$$

With non-zero probability $X_{M+k} = z_2^k$ and $D_k = 1$ for $k = 1, \ldots, M$ (since $a(1,j) > \emptyset$ for each j). Therefore $\Pr\{\underline{Z}_{M+1}=\underline{z}_2 | \underline{Z}_1=\underline{z}_1\} > \emptyset$ and hence $\{\underline{Z}_k\}$ is irreducible. Since all states of an irreducible chain have the same period, we can show aperiodicity by finding a state with period one. To do so, choose any integer j with $f(j) > \emptyset$ and consider the state $\underline{z} = (j,j, \ldots ,j)$. We clearly have $\Pr\{\underline{Z}_2=\underline{z} | \underline{Z}_1=\underline{z}\}=f(j) > \emptyset$. Thus $\underline{z}$ is a state with period one. Since $\{\underline{Z}_k\}$ is irreducible, aperiodic, and has a finite state space, it follows from the elementary theory of Markov chains that $\{\underline{Z}_k\}$ is positive recurrent and has a unique stationary distribution (see, for example, [16]). □

We now present a Theorem which gives the form of the stationary distribution for $\{\underline{Z}_k\}$. In its proof, we use the following notation. Let $\underline{x} \oplus (i,d)$ denote the vector produced from $\underline{x}$ by inserting d in the i^{th} position of $\underline{x}$ and shifting the remaining elements to the right. Thus, if

$\underline{x}=(x^1, \ldots ,x^M)$ then

$\underline{x} \oplus (i,d))=(x^1, \ldots ,x^{i-1},d,x^i, \ldots ,x^{M-1})$.

Also, let $P(\underline{x},\underline{y})=\Pr\{\underline{Z}_{n+1}=\underline{y} \mid \underline{Z}_n=\underline{x}\}$.

Theorem 2.2: The stationary distribution π of the memory state vector Markov chain $\{\underline{Z}_k\}$ is given by

$$\pi(\underline{z}) = \prod_{i=1}^{M} f(z^i).$$

Proof: We show that $\pi(\underline{x}) = \sum_{\underline{y}} \pi(\underline{y}) P(\underline{y},\underline{x})$ when π has the given form. We have

$$\sum_{\underline{y}} \pi(\underline{y})\ P(\underline{y},\underline{x}) = \sum_{k=1}^{M} \sum_{\underline{y}\ni h(\underline{y})=k} \pi(\underline{y})\ P(\underline{y},\underline{x})$$

Let us consider those $\underline{y}$'s in the inner sum with $P(\underline{y},\underline{x}) > 0$. Any such $\underline{y}$ must be of the form $\underline{x} \oplus (j,d)$, $1 \le j \le k$, for some d. Using this representation for $\underline{y}$, we see that d must satisfy

$$M - \sum_{i=1}^{k} x^i + 1 \le d \le M - \sum_{i=1}^{k-1} x^i$$

since otherwise $h(\underline{x} \oplus (j,d)) \neq k$. Let $D_1(\underline{x},k)$ denote the lower bound in this inequality and $D_2(\underline{x},k)$ denote the upper bound. For any d in this range and any j, $1 \le j \le k$, we have

$$\Pr\{ \underline{Z}_{n+1} = \underline{x}, D_n = j \mid \underline{Z}_n = \underline{x} \oplus (j,d) \} = f(x^M)\ a(j,k)$$

because the events $[\underline{Z}_n=\underline{x}\oplus(j,d)]$ and $[L_n=k]$ are equivalent. We also note that

$$\pi(\underline{x}\oplus(d,j)) = f(d) \prod_{i=1}^{M-1} f(x^i).$$

Thus

$$\sum_{\underline{y}} \pi(\underline{y})\ P(\underline{y},\underline{x}) =$$

$$\sum_{k=1}^{M} \sum_{d=D_1(\underline{x},k)}^{d=D_2(\underline{x},k)} \sum_{j=1}^{k} \pi(\underline{x}\oplus(j,d))\Pr\{\underline{X}_{n+1}=\underline{x}, D_n=j \mid X_n=x\oplus(j,d)\}$$

$$= \sum_{k} \sum_{d} \sum_{j} f(d) \prod_{i=1}^{M-1} f(x^i) f(x^M)\ a(j,k)$$

$$= \pi(\underline{x}) \sum_{k} \sum_{d} f(d)$$

since $\sum_{j=1}^{k} a(j,k)=1$. But

$$\sum_{k=1}^{M} \sum_{d=D_1(\underline{x},k)}^{d=D_2(\underline{x},k)} f(d) =$$

$$\sum_{d=M-x^1+1}^{M} f(d) + \sum_{d=M-(x^1+x^2)+1}^{M-x^1} f(d) + \ . \ . \ . \ +$$

$$\sum_{d=M-(x^1+\ .\ .\ .\ +x^{M-1})+1}^{M-(x^1+\ .\ .\ .\ +x^M)} f(d) \quad = \quad \sum_{d=1}^{M} f(d) = 1$$

because each x^i is at least 1. □

The surprising aspect of Theorem 2.2 is that the stationary distribution π is the same as the one which would result if we had defined $h(\underline{Z}_k) \equiv 1$. Now $h(\underline{Z}_k)\equiv 1$ corresponds to a first-in first-out (FIFO) departure process, while the departure process we have been describing may be called a

random departure process because the next departing job is chosen at random from the set of loaded jobs. Thus Theorem 2.2 states that the distribution π is the same under the FIFO departure rule as it is under any departure rule which choses the next job to depart at random from the set of loaded jobs. (This observation is also made in [4] and [12].) Furthermore, the proof shows that statement is true for any departure rule in which the choice of the departing job is made independently of memory size, provided only that the first job has a non-zero probability of departing. Finally, we see that since the form of π does not depend in any way on $a(j,k)$, we may allow $\Pr\{D_n|L_n\}$ to be a function of n. The resulting Markov chain, while no longer time-homogenous, must still have the unique stationary distribution given in Theorem 2.2.

Another view of this result is that we could choose $\underline{Z}_{k+1}$ independently of $\underline{Z}_k$ according to the product density $\prod_i f(x^i)$ and still obtain the same marginal distribution for the MPL. This observation forms the basis for approximate, product form solutions of this dual resource queueing system under Poisson arrivals [8; 9; 11].

3. THE MPL SEQUENCE $\{L_k\}$ AS A FUNCTION OF $\{\underline{Z}_k\}$

Because $\{L_k\}$ can be represented as $\{h(\underline{Z}_k)\}$ we can immediately derive several basic properties of the MPL sequence.

Theorem 3.1: (i) If $\{X_i\}$ is an i. i. d. sequence, then the sequence $\{L_k\}$ is strongly stationary. In particular, $\{L_k\}$ is marginally identically distributed.

(ii) Suppose that (1) $X_1,\ldots,X_k$, $k \leq M$ have an arbitrary joint distribution; (2) $X_{k+1}, X_{k+2}, \ldots$ are i. i. d. with distribution F; and (3) $\{L_k\}$ is an MPL sequence resulting from this sequence of memory sizes $\{X_k\}$. Then as $k\to\infty$, L_k converges in distribution to L_∞ where $L_\infty = h(\underline{X})$ and $\underline{X}$ is an M-vector of i. i. d. random variables with distribution F.

(iii) In (i) or (ii) we have with probability one:

$$\lim_{k\to\infty} \frac{L_1+L_2+\ldots+L_k}{k} = E\{L_\infty\}.$$

Proof: (i) This is clear since $L_k = h(\underline{Z}_k)$ and $\{\underline{Z}_k\}$ is a stationary Markov Chain.

(ii) Even if $\underline{Z}_1 = (X_1, \ldots, X_M)$ is chosen arbitrarily the limiting distribution of $\underline{Z}_k$ as $k\to\infty$ is $\pi(\underline{z})$ provided that $X_{M+1}, X_{M+2}, \ldots$ are i. i. d. with distribution F and $\pi(\underline{z})$ has the form given in Theorem 2.2.

(iii) This follows from a theorem [29] which states that if $\{Y_n\}$ is a Markov chain with stationary distribution π and g is a measurable function on the state space of $\{Y_n\}$, then $\{g(Y_n)\}$ satisfies the strong law of large numbers and

$$\lim_{k\to\infty} \frac{g(Y_1)+\ldots+g(Y_k)}{k} = E_\pi\{g(Y_1)\}. \quad \square$$

Other authors have proven results similar to Theorem 3.1. Buzen and Rubin [12] evaluate the distribution

of residual (unused) core after all possible jobs have been loaded from a never-empty memory queue. They show that under the FCFL policy the distribution of the size of residual core is the same after the k^{th} loader activation as it was after the first one. In part (i) of the Theorem, we have shown that the distribution of the number of jobs in memory after the k^{th} loader activation is the same as the distribution of number of jobs in memory at the first loader activation. It appears that these two results are equivalent; however we will not pursue this matter any further. Betteridge [4] also proves a result similar to (i) in the case m=M. Parts (ii) and (iii) seem to be new results.

By our construction it is clear that $\{L_k\}$ does not form an independent sequence of random variables. As a matter of fact, $\{L_k\}$ need not be a Markov chain, as the following example indicates.

Example 3.2: Suppose m=2, M=3 and f(1)=f(2)=0.5. Consider the event $[L_{k-1}=2, L_k=3]$. The only way $L_k=3$ can occur is by $\underline{Z}_k=(1,1,1)$ while $L_{k-1}=2$ can occur as $\underline{Z}_{k-1}$ = (1,1,2),(1,2,1), or (1,2,2). But the event $[L_{k-1}=2, L_k=3]$ must occur as $[\underline{Z}_{k-1}=(1,2,1), \underline{Z}_k=(1,1,1)]$ since the other transitions are not allowed. Because the event $[L_{k-1}=2, L_k=3]$ can occur, $\Pr\{L_k=3 \mid L_{k-1}=2\} > 0$. Next consider the event $[L_{k-2}=3, L_{k-1}=2, L_k=3]$. The memory state vector sequence corresponding to this event must be

$$[\underline{Z}_{k-2}=(1,1,1), \underline{Z}_{k-1}=(1,1,2), \underline{Z}_k=(1,1,1)].$$

However this event has probability zero since the state (1,1,1) is unreachable in one transition from (1,1,2). The only possible destinations from (1,1,2) are (1,2,1) and (1,2,2). Thus $Pr\{L_k=3|L_{k-1}=2,L_{k-2}=3\}=\emptyset$ and therefore $\{L_k\}$ is not a (one-step) Markov chain. □

We now turn to the problem of evaluating the marginal distribution of L_k.

Theorem 3.3: The marginal distribution of L_k is given by

$$Pr\{L_k=n\}=F^{(n)}(M)-F^{(n+1)}(M)$$

where $F^{(n)}$ denotes the n-fold convolution of F with itself.

Proof: $Pr\{L_k=n\} = Pr\{L_1=n\}$ by Theorem 3.1. Thus

$$Pr\{L_k=n\} = Pr\{L_1\geq n\} - Pr\{L_1\geq n+1\}$$

$$=Pr\left\{\sum_{i=1}^{n} X_i\leq M\right\} -Pr\left\{\sum_{i=1}^{n+1} X_i\leq M\right\}$$

$$=F^{(n)}(M)-F^{(n+1)}(M). \square$$

Note: This result is a basic formula from renewal theory [30]. □

Bard [3] and Betteridge [4] both observed that

$$Pr\{L_k=n\} = \sum_{j=n}^{M} Pr\left\{\sum_{i=1}^{n} X_i=j,\ X_{n+1}>M-j\right\}.$$

However the fact that the right hand side was indeed a

difference between two convolutions was not observed until [1]. Because $Pr\{L_k=n\}$ can be expressed in this way, it can be evaluated exactly by directly performing the convolutions, or approximately by using a normal approximation for $F^{(n)}(M)$. Furthermore, Bard states this equation without proving $Pr\{L_k=n\}=Pr\{L_1=n\}$ so his result is not as strong as ours. Similarly, Brown et al [8] derive a formula for $Pr\{L_k=n\}$ under the loading policy FCFL (FCFL is called "first fit without skip" in [8]), while they do not consider the stationarity issues we have discussed. Indeed, the other loading policy discussed in [8] (the "first fit with skip" policy) yields an MPL sequence which is not strongly stationary and their formulas for that case are exact only for the distribution of L_1 [10].

4. TIME-AVERAGED BEHAVIOR OF THE MPL SEQUENCE

Since only loaded jobs receive service it is reasonable to assume that job interdeparture times depend only on the number of jobs loaded. So let $Y_k=t_{k+1}-t_k$ be the time between departures of the $k-1^{st}$ and k^{th} jobs to leave the system. For the moment, we drop our exponential service time assumption. Let $Pr\{Y_k\leq y|L_k=n\}=G_n(y)$ denote the distribution of Y_k given n jobs loaded. We assume for each n, $1\leq n\leq M$, that the following hold:

(i) $G_n(0)=0$

$$\text{(ii)}\quad \frac{1}{\mu_n} = \int_{\emptyset}^{\infty} y\, dG_n(y) < \infty$$

Furthermore, we require that Y_k depends on the other Y_i's only through the MPL sequence, that is

$$\Pr\{Y_i \leq y_i, i=1,\ldots,n \mid L_i=n_i, i=1,\ldots,n\} = \prod_{k=1}^{n} \Pr\{Y_k \leq y_k \mid L_i=n_i, i=1,\ldots,n\}.$$

Hence, given a particular MPL sequence, the associated sequence $\{Y_k\}$ is an independent sequence of random variables. Finally, let us redefine the memory state vector sequence $\{\underline{Z}_k\}$ as a function of t as follows:

(i) $\underline{Z}(\emptyset) = \underline{Z}_1$.

(ii) for $t>\emptyset$, $\underline{Z}(t)=\underline{Z}_k$ iff $t_k \leq t < t_{k+1}$.

As defined $\underline{Z}(t)$ is a vector valued semi-Markov process, and $L(t)=h(\underline{Z}(t))$.

Since $\{L_k\}$ obeys the strong law and since given $\{L_k\}$, the sequence $\{Y_k\}$ is independently distributed with finite mean, it follows that:

Lemma 4.1: With probability one

$$\lim_{k\to\infty} \frac{Y_1 + \ldots + Y_k}{k} = E\{Y_1\}. \quad \square$$

Let D(t) be the number of job departures which have occurred at time t. Because $\{Y_k\}$ obeys the strong law we can show, using standard arguments (see, for example, [30]), that:

Lemma 4.2: With probablity one,

$$\lim_{t\to\infty} \frac{D(t)}{t} = \frac{1}{E\{Y_1\}}. \quad \square$$

We now state the main result of this section.

Theorem 4.3: With probability one,

$$\bar{L} \equiv \lim_{t\to\infty} \frac{1}{t} \int_0^t L(t)\, dt = \frac{E\{L_1 Y_1\}}{E\{Y_1\}}.$$

Proof: We have

$$\frac{1}{t} \sum_{i=1}^{D(t)} L_i Y_i \le \frac{1}{t} \int_0^t L(t)\, dt \le \frac{1}{t} \sum_{i=1}^{D(t)+1} L_i Y_i$$

which implies that

$$\frac{D(t)}{t} \left(\frac{1}{D(t)} \sum_{i=1}^{D(t)} L_i Y_i \right) \le \frac{1}{t} \int_0^t L(t)\, dt \le \frac{D(t)+1}{t} \left(\frac{1}{D(t)+1} \sum_{i=1}^{D(t)+1} L_i Y_i \right).$$

Now as $t\to\infty$ so does $D(t)$ and therefore, with probability one, both the right and left hand sides converge to $E\{L_1 Y_1\}/E\{Y_1\}$. Therefore so must the middle term. $\square$

Corollary 4.4: Let $P\{L(t)=j\}$ be the time-averaged probability of finding j jobs loaded. Then

$$P\{L(t)=j\} = \frac{P\{L_1=j\}E\{Y_1 | L_1=j\}}{E\{Y_1\}}$$

Proof: We have

$$P\{L(t)=j\} = \lim_{t\to\infty} \frac{1}{t} \int_{\emptyset}^{t} \Phi_j(L(t))\, dt$$

where $\Phi_j(x) = 1$ if $x=j$ and zero otherwise. The argument of the Theorem applies to this integral as well so that the result follows. □

We now consider three applications of these results. For the purpose of these examples, we have chosen F(x) to be the discrete uniform distribution on the integers 1,. . .,100. While this is an admittedly artificial choice, it suffices to illustrate the ideas involved. The details of evaluating $F^{(n)}(M)$ for these examples are given in the Appendix.

Example 4.5: Calculating the time averaged mean number of jobs loaded in a saturated system. Let us consider a computer system with P processors and M blocks of main memory as described in Section 1. We assumed that job service times are exponentially distributed with mean $1/\mu$. It follows that $G_n(y) = 1-e^{(-\mu_n y)}$ where μ_n is defined by

$$\mu_n = \begin{cases} n\mu \text{ for } P \geq n \geq 1 \\ P\mu \text{ for } n > P. \end{cases}$$

Under these assumptions we can easily derive (using Corollary 4.4) a general formula for the mean number of jobs loaded given P processors and M blocks of memory, which we

shall denote by $\bar{L}(P,M)$:

$$\bar{L}(P,M) = \frac{\sum_{i=1}^{M} \frac{(F^{(i)}(M)-F^{(i+1)}(M))\,i}{\mu_i}}{\sum_{i=1}^{M} \frac{F^{(i)}(M)-F^{(i+1)}(M)}{\mu_i}}$$

In certain cases this equation takes a particularly simple form. For example, if P=1 then $\mu_i=\mu$ (pure processor sharing) and

$$\bar{L}(1,M) = \sum_{i=1}^{M} (F^{(i)}(M)-F^{(i+1)}(M))\,i$$

$$= \sum_{i=1}^{M} F^{(i)}(M) \ .$$

Similarly if P=∞ (sufficient servers), then $\mu_i=i\mu$ and

$$\bar{L}(\infty,M) = \frac{1}{1-\sum_{i=2}^{M} \frac{F^{(i)}(M)}{i(i-1)}} \ . \ \square$$

Example 4.6: Balancing the number of processors and the amount of memory attached to a computer system. We continue to consider the system of Example 4.5. If $P<\bar{L}$ then all processors are nearly always busy and the system is processor bound rather than memory bound. If $P\geq\bar{L}$ then the system becomes memory bound rather than processor bound. In such a case, one might wish to answer the question: "Given

that our system is memory bound, how much of the time will at least one processor be idle due to the fact that not enough jobs can be loaded to keep all processors busy?" Let us call this fraction of time $P_I(P,M)$. If P_I is large, then the system does not have enough main memory in relation to the number of processors it has and an increase in main memory size will cause an increase in service rate. Now $P_I(P,M)$ can easily be evaluated as

$$P_I(P,M) = \sum_{i=1}^{P-1} \Pr\{L(t)=i\} = 1 - \sum_{i=P}^{M} \Pr\{L(t)=i\}$$

$$= 1 - \frac{1}{E\{Y_1\}P\mu} \sum_{i=1}^{M} (F^{(i)}(M) - F^{(i+1)}(M))$$

$$= 1 - \frac{1}{E\{Y_1\}P\mu} F^{(P)}(M).$$

Examining the values of $P_I(P,M)$ for various memory sizes M would allow one to balance the amount of memory on the system to the number of processors P in an intelligent way.

Suppose for example that M=100, P=2 and F(x) is the discrete uniform distribution on the integers 1 to 100. Since the mean memory size of arriving jobs is 50.5, it might seem that both processors will be busy most of the time. Actually this is not quite so. Table 4.1 gives selected values of $P_I(2,M)$ for M between 100 and 200. From these values we see that the system as described is memory bound until M increases beyond 170. Thus it seems that the minimum amount of memory which should be attached to our

M	$P_I(2,M)$
100	0.6711
110	0.5811
120	0.4894
130	0.3981
140	0.3094
150	0.2262
160	0.1516
170	0.0889
180	0.0411
190	0.0109
200	0.0000

Table 4.1

Values of $P_I(P,M)$ when P=2 and F(x) is the discrete uniform distribution on 1,. . . ,100.

M	$R^*(20,M)$
100	1.3853
200	3.2554
300	5.2430
400	7.2355
500	9.2230
600	11.2076
700	13.1920
800	15.1834
900	17.2022
1000	19.2673

Table 4.2

Values of $R^*(P,M)$ when P=20 and F(x) is the discrete uniform distribution on 1,. . . ,100.

M	$R^*(20,M)$
400	7.2355
410	7.4344
420	7.6334
430	7.8323
440	8.0311
450	8.2300
460	8.4285
470	8.6271
480	8.8257
490	9.0243
500	9.2230

Table 4.3

This table is a finer resolution version of Table 4.2.

hypothetical system is about 170 blocks. □

Example 4.7: Maximum processing rate of a memory bound system. In this example we give a capacity bound similar to (but computable for much larger memory sizes than) that of [27]. We then use this capacity bound to estimate the amount of memory a hypothetical system requires in order to process its workload.

We begin by observing that the mean job inter-departure time in our model is $E\{Y_1\}$; therefore the job departure rate given that the memory queue is never empty is $1/E\{Y_1\}$. If the memory queue does become empty, then the departure rate can only decrease. Thus $1/E\{Y_1\}$ is the maximum service rate which the system can sustain. Let $R^*(P,M)$ denote this maximum service rate when there are P processors and M memory blocks on the system.

Now suppose that jobs arrive at our system according to a Poisson process of rate λ. Clearly, if $\lambda > R^*(P,M)$, then the system will be overloaded. Thus we may think of $R^*(P,M)$ as the maximum processing rate which the system can supply. We note that if $\lambda > P\mu$, then increasing the amount of memory will not change $R^*(P,M)$ since there are not enough processors to service the offered workload. So let us assume that $\lambda < P\mu$. In such a system, throughput increases with memory size until $\bar{L} \geq P$, and the system is memory rather than processor bound. For such a system, a reasonable question to ask is: "What is the smallest memory size M we

can use to serve the workload without overloading the system?" Clearly, this value of M is the smallest value which makes $R^*(P,M) > \lambda$.

In particular, let us assume that $\lambda=8$, $\mu=1$, $P=20$, $F(x)$ is the uniform discrete distribution on 1 to 100 blocks, and that the processor model of Example 4.5 applies. To get an approximate idea of the amount of memory required, we examine Table 4.2 (see page 21) and find that between 400 and 500 blocks of main memory are needed. Table 4.3 is a finer resolution version of Table 4.2 and examination of Table 4.3 indicates that approximately 440 blocks of memory are required in order to service the workload. Actually, one should probably attach more than 440 blocks of memory to the system, in order to avoid the congestion problems associated with heavily utilized systems. □

5. A MULTIPLE CLASS MODEL

In this section we generalize the framework used to solve the examples of Section 4.0 to allow for the case of multiple job classes. Distinct job classes may have distinct mean service times and processor scheduling models, so that a mixture of CPU bound and I/O bound job classes can be described. For notational simplicity, we restrict our attention to the case of two job classes; we will indicate how to extend the analysis to an arbitrary number of classes.

We give an approximate solution to this model and estimate its accuracy by comparison to a detailed simulation of the model. We will then describe an application of this solution to a problem similar to that of Example 4.7.

Let p_i be the probability that a job is a member of class-i, i=∅,1. The class of a job is assumed to be independent of the job's memory requirement and the class memberships of all other jobs. The service time requirement of a class-i job is exponentially distributed with parameter μ^i. We let $L_i(t)$ be the number of class-i jobs loaded at time t, and we define $L_{k,i}=L_i(t_k^+)$.

Let P_i be the number of class-i processors attached to the system, $1\leq P_i\leq\infty$. A class-i job may receive service only from a processor of class-i. The processor scheduling model assumes processor sharing within each class with a maximum of one processor assigned to each job. Thus if $P_i\geq L_i(t)$ then each class-i job is assigned its own processor and receives service at a unit rate. If $L_i(t)>P_i$ then the class-i processors are equally shared among the class-i jobs and each job receives service at the fractional rate $P_i/L_i(t)$.

Now let $Y_{k,i}$ be the time from t_k until the next job of class-i is ready to depart. If $L_{k,i}=\emptyset$ then we may put $Y_{k,i}=\infty$ with probability one. If $L_{k,i}>\emptyset$ then $Y_{k,i}$ has exponential distribution with parameter $\mu_{n,i}$ where $n=L_{k,i}$ and

$$\mu_{n,i} = \begin{cases} n\mu^i & P_i \geq n \\ P_i\mu^i & n > P_i. \end{cases}$$

The job departure model is based on the assumptions that the next job to depart will be a class-i job if $Y_{k,i} < Y_{k,1-i}$ and that all members of the chosen class are equally likely to depart. From these assumptions it follows that $Y_k \equiv t_{k+1} - t_k = \min(Y_{k,\emptyset}, Y_{k,1})$ from which one may calculate the distribution of Y_k in terms of $L_{k,i}$.

Now let $\underline{C}(t) = (c^1(t), c^2(t), \ldots, c^M(t))$, with $c^i(t) = \emptyset$ or 1, represent the class memberships of the jobs present in the memory state vector $\underline{Z}(t)$. Then it is clear that $\{\underline{Z}(t), \underline{C}(t)\}$ is a Markov process.

Before we state our solution, we need to define some new quantities. Let us suppose for the moment that $L(t) \equiv L$ for all t. Then whenever a job departs, precisely one job is loaded into core and $L_{k,\emptyset}$ (as well as $L_{k,1}$) can only increase or decrease by one. Therefore $L_{k,\emptyset}$ is a birth-death chain with transition probabilities given by

$$\Pr\{L_{k+1,\emptyset} = i+1 \mid L_{k,\emptyset} = i\} = \frac{p_\emptyset \mu_{L-i,1}}{\mu_{i,\emptyset} + \mu_{L-i,1}}$$

$$\Pr\{L_{k+1,\emptyset} = i \mid L_{k,\emptyset} = i\} = \frac{p_\emptyset \mu_{i,\emptyset} + p_1 \mu_{L-i,1}}{\mu_{i,\emptyset} + \mu_{L-i,1}}$$

$$\Pr\{L_{k+1,\emptyset} = i-1 \mid L_{k,\emptyset} = i\} = \frac{p_1 \mu_{i,\emptyset}}{\mu_{i,\emptyset} + \mu_{L-i,1}}$$

where we have adopted the convention that $\mu_{\emptyset,i}=\emptyset$. We can also evaluate the mean time between job departures given $L(t)\equiv L$ as

$$E\{Y_k \mid L(t)\equiv L,\ L_{k,\emptyset}=j\} = \frac{1}{\mu_{j,\emptyset}+\mu_{L-j,1}}.$$

Using these quantities we can evaluate the stationary distribution of the number of class-$\emptyset$ jobs loaded given $L(t)\equiv L$, which we shall denote by $\pi_L(x)$. From π_L we can calculate the limiting mean number of class-$\emptyset$ jobs loaded and the mean job interdeparture time given $L(t)\equiv L$ as

$$E\{L_\emptyset(t) \mid L(t)\equiv L\} = \sum_{i=\emptyset}^{L} i\pi_L(i)$$

$$E\{Y_k \mid L(t)\equiv L\} = \sum_{i=\emptyset}^{L} \frac{\pi_L(i)}{\mu_{i,\emptyset}+\mu_{L-i,1}}.$$

Now we are ready to state our solution. We conjecture that the time-averaged mean number of jobs loaded for this two class model can be approximated by

$$\bar{L} \equiv \lim_{t\to\infty} \frac{1}{t}\int_\emptyset^t L(t)dt \cong \sum_{L=1}^{M} \frac{L(F^{(L)}(M)-F^{(L+1)}(M))}{E\{Y_k \mid L(t)\equiv L\}} \quad (5.1)$$

and that the time-averaged number of class-$\emptyset$ jobs loaded can be approximated by

$$\bar{L}_{\emptyset} \equiv \lim_{t\to\infty} \frac{1}{t}\int_{\emptyset}^{t} L_{\emptyset}(t)\,dt \cong$$

$$\sum_{L=1}^{M} E\{L_{\emptyset}(t) \mid L(t)\equiv L\}\,\frac{F^{(L)}(M)-F^{(L+1)}(M)}{E\{Y_k \mid L(t)\equiv L\}}\,. \qquad (5.2)$$

We observe that the last term on the right in each equation is merely the probability of finding $L(t)=L$.

These last two equations are based on the assumption that each birth-death chain $\{L_{k,\emptyset}\}$ (obtained by conditioning on $L(t)\equiv L$) reaches the equilibrium distribution π_L. However this is not the way the system behaves since $L(t)$ is continually changing. On the other hand we observe that if $\{W_k\}$ is an identically distributed ergodic sequence of random variables and if for each j, $\{U_k(j)\}$ is a birth-death chain with stationary distribution π_j, then with probability one

$$\lim_{k\to\infty} \frac{U_1(W_1)+U_2(W_2)+\cdots+U_k(W_k)}{k} = \sum_j E_{\pi_j}\{U_k(j)\}\Pr\{W_k=j\}.$$

Since $\{L_k\}$ is known to be an identically distributed and ergodic sequence of random variables, the parallel between the above and equations (5.1) and (5.2) should be clear.

For the case of $i=\emptyset,1,\ldots,c$ job classes $(c>1)$, the birth-death chain $\{L_{k,\emptyset}\}$ given $L(t)\equiv L$ is replaced by the c dimensional random walk

$$(L_{k,\emptyset}, L_{k,1}, \ldots, L_{k,c-1}),$$

where $L_{k,i} \geq \emptyset$, $\sum_{i=\emptyset}^{c-1} L_{k,i} = L$.

Since the state space of this random walk is finite, its discrete-time stationary distribution can (theoretically) be determined. From the discrete-time stationary distribution and the quantities

$$E\{Y_{k,} \mid L(t)\equiv L,\ L_{k,i}=j_i,\ i=0,1,\ldots,c-1\}$$

we can calculate the time-averaged stationary distribution of job types loaded given $L(t)\equiv L$, which we shall call $\pi_L(\underline{x})$. Given π_L, $\bar{L}$ and $\bar{L}_i$ can be found by straightforward generalizations of equations (5.1) and (5.2).

In the following examples we compare the accuracy of the equations (5.1) and (5.2) to a simulation model and illustrate one application of these equations.

Example 5.1: Mean number of jobs loaded under an I/O and CPU bound mixed workload. We consider the case $P_0=1$ and $P_1=\infty$. If we assume that there is one CPU and a large number of I/O channels on the system, then class-0 can be thought of a class of completely CPU bound jobs and class-1 can be thought of as a class of completely I/O bound jobs. Thus p_0 represents the percentage of completely CPU bound jobs in the workload. We will use $m=100$ and $F(x)$ the discrete uniform distribution on the integers $1,\ldots,100$, as before. Only the ratio $\gamma=\mu^1/\mu^0$ is actually significant in the formulas; so we have stated our answers in terms of γ. In Tables 5.1, 5.2, and 5.3 we have given values of $\bar{L}$, $\bar{L}_0$ and $\bar{L}_1$ $(=\bar{L}-\bar{L}_0)$ calculated from equations (5.1) and (5.2) and values calculated from a simulation model based on 20,000 job departures per case. (The details of this

M	Analytic Method	Simulation Method[1]	Difference Between Means
100	$\bar{L}$ = 1.406	1.408±0.034[2]	0.002
	$\bar{L}_0$ = 0.316	0.350±0.037	0.014
	$\bar{L}_1$ = 1.090	1.058±0.036	-0.032
200	$\bar{L}$ = 3.339	3.390±0.097	0.051
	$\bar{L}_0$ = 1.012	1.089±0.093	0.077
	$\bar{L}_1$ = 2.327	2.300±0.082	-0.027
300	$\bar{L}$ = 5.416	5.406±0.203	-0.010
	$\bar{L}_0$ = 2.178	2.168±0.226	-0.010
	$\bar{L}_1$ = 3.238	3.238±0.206	-0.000+
400	$\bar{L}$ = 7.494	7.515±0.255	0.021
	$\bar{L}_0$ = 3.768	3.717±0.350	-0.051
	$\bar{L}_1$ = 3.726	3.798±0.209	-0.072
500	$\bar{L}$ = 9.535	9.477±0.235	-0.078
	$\bar{L}_0$ = 5.614	5.487±0.446	-0.127
	$\bar{L}_1$ = 3.921	3.990±0.322	0.069
600	$\bar{L}$ =11.543	11.652±0.386	0.109
	$\bar{L}_0$ = 7.561	7.599±0.494	0.038
	$\bar{L}_1$ = 3.982	4.053±0.292	0.071
700	$\bar{L}$ =13.532	13.605±0.422	0.073
	$\bar{L}_0$ = 9.535	9.615±0.535	0.080
	$\bar{L}_1$ = 3.997	3.990±0.220	-0.007
800	$\bar{L}$ =15.514	15.444±0.535	-0.070
	$\bar{L}_0$ =11.515	11.298±0.696	-0.217
	$\bar{L}_1$ = 3.999	4.146±0.380	0.147
900	$\bar{L}$ =17.495	17.384±0.525	-0.111
	$\bar{L}_0$ =13.495	13.227±0.636	-0.268
	$\bar{L}_1$ = 4.000	4.157±0.307	0.157
1000	$\bar{L}$ =19.475	19.493±0.523	0.018
	$\bar{L}_0$ =15.475	15.460±0.555	-0.015
	$\bar{L}_1$ = 4.000	4.033±0.375	0.033

Table 5.1

Analytic and simulation solutions to the two class model of Example 5.1. In this table γ=0.25 and p_0=0.50.

Notes: (1) For simulation details see Appendix.
(2) ± figures give standard deviations.

p_0	Analytic Method	Simulation Method[1]	Difference Between Means
0.0	$\bar{L} = 7.235$	7.336 ± 0.183[2]	0.101
	$\bar{L}_0 = 0.000$	0.000 ± 0.000	-----
	$\bar{L}_1 = 7.235$	7.336 ± 0.183	0.101
0.2	$\bar{L} = 7.254$	7.325 ± 0.180	0.071
	$\bar{L}_0 = 0.646$	0.686 ± 0.122	0.040
	$\bar{L}_0 = 6.608$	6.639 ± 0.212	0.031
0.4	$\bar{L} = 7.395$	7.460 ± 0.198	0.065
	$\bar{L}_0 = 2.476$	2.457 ± 0.295	-0.001
	$\bar{L}_1 = 4.919$	5.000 ± 0.249	0.081
0.6	$\bar{L} = 7.558$	7.641 ± 0.232	0.083
	$\bar{L}_0 = 4.944$	5.055 ± 0.302	0.111
	$\bar{L}_1 = 2.614$	2.586 ± 0.257	-0.028
0.8	$\bar{L} = 7.593$	7.542 ± 0.285	-0.051
	$\bar{L}_0 = 6.593$	6.525 ± 0.351	-0.068
	$\bar{L}_1 = 1.000$	1.018 ± 0.109	0.018
1.0	$\bar{L} = 7.594$	7.668 ± 0.266	0.074
	$\bar{L}_0 = 7.594$	7.668 ± 0.266	0.074
	$\bar{L}_1 = 0.000$	0.000 ± 0.000	-----

Table 5.2

Analytic and simulation solutions to the two class model of Example 5.1. In this table $\gamma = 0.25$ and $M = 400$.

Notes: (1) For simulation details see Appendix.
(2) $\pm$ figures give standard deviations.

γ	Analytic Method	Simulation Method[1]	Difference Between Means
0.005	$\bar{L}$ =7.246	7.332±0.260[2]	0.086
	$\bar{L}_0$=0.482	0.583±0.069	0.101
	$\bar{L}_1$=0.676	0.675±0.231	0.001
0.10	$\bar{L}$ =7.293	7.422±0.234	0.129
	$\bar{L}_0$=1.242	1.369±0.115	0.227
	$\bar{L}_1$=6.051	6.054±0.214	0.003
0.25	$\bar{L}$ =7.494	7.515±0.255	0.021
	$\bar{L}_0$=3.768	3.717±0.350	-0.057
	$\bar{L}_1$=3.726	3.798±0.209	0.072
0.5	$\bar{L}$ =7.579	7.517±0.183	-0.062
	$\bar{L}_0$=5.593	5.547±0.193	-0.046
	$\bar{L}_1$=1.986	1.969±0.123	-0.017
1.0	$\bar{L}$ =7.593	7.617±0.195	0.024
	$\bar{L}_0$=6.593	6.593±0.254	0.000+
	$\bar{L}_1$=1.000	1.024±0.103	0.024
2.0	$\bar{L}$ =7.594	7.658±0.251	0.064
	$\bar{L}_0$=7.094	7.152±0.259	0.058
	$\bar{L}_1$=0.500	0.506±0.038	0.006

Table 5.3

Analytic and simulation solutions to the two class model of Example 5.1. In this table M=400 and p_0=0.50.

Notes: (1) For simulation details see Appendix.
(2) ± figures give standard deviations.

simulation model are contained in the Appendix.) In each table we have held two of the three parameters γ,M and $p_\emptyset$ constant while allowing the other parameter to vary. In all cases the agreement between the simulation and analytic models is good. □

Example 5.2: Maximum processing rate of a memory bound system subjected to a mixed workload. Given the system described in the last example, let us suppose that jobs arrive at the system according to a Poisson process of rate λ and let us consider the question, "Does the system have sufficient capacity to handle the workload without being overloaded?" We can give an answer to this question by comparing the arrival rate of each class of job to the maximum service rate attainable by the system. As we pointed out in Example 4.7, this maximum service rate occurs when the system is subject to an saturation workload so that the analysis of this paper applies.

The arrival rate of CPU bound jobs is $p_\emptyset\lambda$, and their maximum service rate is $\mu^\emptyset$ jobs per second provided $\bar{L}_\emptyset \geq 1$. If not then the service rate drops to $\bar{L}_\emptyset\mu^\emptyset$. Therefore in order to process the CPU bound portion of the workload we must have $p_\emptyset\lambda \leq \mu^\emptyset$ if $\bar{L}_\emptyset \geq 1$; otherwise we must have $p_\emptyset\lambda \leq \bar{L}_\emptyset\mu^\emptyset$.

If this condition is satisfied, then we may determine if the system can process the I/O bound portion of the workload. The arrival rate of I/O bound jobs is $p_1\lambda$ and their maximum service rate is $\bar{L}_1\mu^1$. We must have $p_1\lambda \leq \bar{L}_1\mu^1$

in order to process this portion of the workload.

If the overall system is to be stable, then both of these conditions must be satisfied. We have therefore only argued that these are necessary conditions for system stability and not sufficient conditions.

If we consider the case λ=7.0, p_0=0.50, μ^0=4.0, and μ^1=1.0 (i. e. γ=0.25), then we see from Table 5.1 that $\bar{L}_0>1$ for $M\geq 200$ and hence the condition $p_0\lambda \leq \mu^0$ is satisfied for all memory sizes except M=100. Processing of I/O bound jobs is clearly the bottleneck. Examining the table we see that the condition $p_1\lambda \leq \bar{L}_1\mu^1$ is satisfied for memory sizes M=400 or larger. Therefore at least 400 blocks of memory need to be attached to this system in order to service the workload. Further refinements of this estimate could be made by calculating $\bar{L}_1$ for M values between 300 and 400.

We note that increasing memory size is not the only change which may have to be made in order to obtain desired system capacity. If in the current example the arrival rate were to increase to λ=10 then no amount of memory will allow the system to handle the workload. This is because the arrival rate of CPU bound jobs is 5 jobs per second while the maximum possible service rate is 4 jobs per second. The only choice given the system designer in this case is to increase the speed or number of processors. □

6. CONCLUDING REMARKS

We have constructed a model of storage allocation which includes the memory request size distribution as an explicit parameter. Using this model we were able to show that the MPL sequence under the loading policy was strongly stationary, and we were able to derive the marginal distribution of the MPL. From this distribution and the distribution of job interdeparture times given n jobs loaded ($G_n(y)$) we then determined the time averaged distribution of jobs loaded, and for the case of exponential service times, we derived the maximum processing rate of a system with P processors and M blocks of memory. These results were then generalized to handle the case of a mixed I/O and CPU bound workload. We also showed how to use these results to determine whether the main memory size of a computer system was sufficient to handle a particular workload. We believe these results to be applicable to real problems of computer system configuration design.

We have considered only the loading policy FCFL. To the best of the author's knowledge, this is the only nontrival loading policy for which a theorem like Theorem 2.2 (and hence Theorem 3.3) holds. Loading policies such as first fit with skip [8], random, and smallest (or largest) memory size first do not create strongly stationary MPL sequences [10]. Any formula which claims to give the distribution of the MPL under such a loading policy should

be considered an approximation, until a theorem analagous to Theorem 3.3 can be proven.

Although we have not considered the application of these results to an entire computer system model, it is clear how to proceed in this direction. If we represent the transitions of jobs in main memory by a closed network of queues, then the rate of transitions in this network is much higher than the rate at which the MPL changes. Such a model is therefore nearly completely decomposible [14], with the memory interface forming a natural boundary for the system aggregates. One can then approximate $G_n(y)$ as an exponential distribution with parameter μ_n where μ_n is calculated as the reciprocal of the mean job departure time from the closed network model when there are n jobs in the network. For an example of this approach, see [10].

7. ACKNOWLEDGEMENTS

Much of this paper is part of the author's Ph. D. thesis [10] which was written at the University of Maryland under the direction of Ashok Agrawala. The current proof of Theorem 2.2 as well as the current exposition were prepared while the author was with the Computer Sciences Department, University of Wisconsin-Madison. This preparation was supported in part by the Wisconsin Graduate School Research Committee.

APPENDIX

A.1 EVALUATION OF $F^{(n)}(x)$

To evaluate $F^{(n)}(x)$, $f^{(n)}(x)$ was evaluated by the recurrence relation

$$f^{(n)}(x)=\sum_{y=1}^{m} f^{(n-1)}(x-y)f(y) \qquad (A.1.1)$$

with $f^{(1)}(x)\equiv f(x)$. $F^{(n)}(x)$ was then evaluated from $f^{(n)}(x)$, and $F^{(n)}(M)$ was checked to see if $F^{(n)}(M)<10^{-6}$. If so then the calculations were stopped, otherwise the next convolution was evaluated. Eventually a value n_{max} was found so that $F^{(n)}(M)<10^{-6}$ for $n\geq n_{max}$. (For M=1000 and F(x) the uniform discrete distribution on 1,. . .,100, n_{max} is about 35.) All computations in Examples 5.1 and 5.2 are based on the assumption that $F^{(n)}(M)=0$ for $n>n_{max}$. This computation requires about $m^2n_{max}^2$ floating point operations and requires $m(n_{max}+2)$ words of storage as shown below. The convolution calculations used for the examples required less than two minutes on a Univac 1110 computer.

Each evaluation of equation (A.1.1) clearly requires m multiplications and additions per evaluation. Since $f^{(n)}(x)=0$ if $x<m$ or $x>nm$ it follows that equation (A.1.1) needs to be evaluated n(m-1) times in order to find $f^{(n)}(x)$

throughout its non-zero range. The total number of operations required is therefore

$$\sum_{n=2}^{n_{max}} 2m(m-1)n = m(m-1)n_{max}(n_{max}+1)-2m(m-1),$$

which is approximately $m^2n_{max}^2$.

To calculate $f^{(n+1)}(x)$ from $f^{(n)}(x)$ one clearly needs to save $f^{(n)}(x)$. This takes a maximum of mn_{max} words. m more words are required to store f(x). Finally, the same storage space can be used to store $f^{(n)}(x)$ and $f^{(n+1)}(x)$ provided the most recently calculated m values of $f^{(n+1)}(x)$ are saved temporarily somewhere else. This gives a total storage requirement of $m(n_{max}+2)$ words.

A.2 DETAILS OF THE SIMULATIONS FOR EXAMPLE 5.1

These simulations were written in SIMSCRIPT II.5 which is a product of C. A. C. I., Inc. A description of this language is available in the language textbook [17]. The standard pseudo-random number generator for the Univac 1100 series implementation is the Lehmer generator described in [25]. This is the same generator as used in the IBM 360 implementation. Since extensive statistical tests of this pseudo-random number generator were not available and because we wanted to take advantage of the larger word size (36 bits) on Univac 1100 series machines, we replaced the standard pseudo-random number generator with the following

one: $R_{n+1} \equiv 5^{15} R_n \pmod{2^{35}}$. This pseudo-random number generator is known to be good by the spectral test [19] which is considered to be one of the most powerful tests of a congruential pseudo-random number generator.

These simulations were straightforward implementations of the model as described in Section 5. Since the simulations of the example were so simple, and in order to obtain maximum efficiency, we decided not to use the SIMSCRIPT II.5 event set mechanism to schedule events in the simulations. Instead we used a simple event scan loop. The steps in this loop were:

(1) Determine how many jobs from the memory queue fit into memory. Count the number of jobs of each type present in memory.

(2) Assign a departure time to each loaded job, according to the job class service time distribution.

(3) Find the job with the smallest departure time and choose this job to be the departing job. Remove this job from the system. Advance the simulation clock to the time of the job's departure.

(4) Create a new job and place it at the end of the memory queue. Set the class of this job according to a simple Bernoulli trial.

(5) Update statistics recording the number of jobs of each type loaded during this interval.

(Note: In step (1) it is not necessary to know which jobs

were previously loaded since under the FCFL loading policy this information can be calculated from the memory sizes of the jobs in the memory queue.)

These steps were repeated 1,000 times per experiment. At the end of an experiment, the mean number of jobs loaded of each type was output and the simulation statistics and parameters were reset. Another run of 1,000 job departures was then performed. After 10 runs had been executed, the random number seeds were reset to their original values and the sequence of 10 runs was repeated with the antithetic sequence of pseudo-random numbers being used to drive the simulations. The simulation data reported in Example 5.1 were calculated on the resulting set of 20 simulation outputs. The mean values reported in Tables 5.1, 5.2, and 5.3 are thus averages of experimental means; the standard deviations in the Tables are the standard deviations among the experimental means.

The purpose of using the antithetic sequence of pseudo-random variables was two-fold. First, this is a standard variance reduction technique [18]. Second, if the first run tends to overestimate the true mean value, then an execution with the antithetic random variates tends to underestimate the mean (provided that the generated random variates are monotonic functions of the base random number stream). The combined estimate is therefore more accurate.

BIBLIOGRAPHY

[1] Agrawala, A. K. and Bryant, R. M. Models of memory scheduling. Proceedings of Fifth Symposium on Operating System Principles, Austin, Texas, 1975, 217-222.

[2] Badel, M., Gelenbe, E., Leroudier, J. and Potier, D. Adaptive optimization of a time-sharing system's performance. Proceedings of the IEEE 63, 5 (June 1975), 958-965.

[3] Bard, Y. An analytic model of CP-67 and VM/370. IBM Research Report G320-2101, Cambridge Scientific Center, Cambridge, Massachusetts, 1974.

[4] Betteridge, T. An analytic storage allocation model. Acta Informatica 3 (1974), 101-122.

[5] Brandwajn, A. A model of a time sharing virtual memory system solved using equivalence and decomposition methods. Acta Informatica 4 (1974), 11-47.

[6] Brandwajn, A. A model of a virtual memory system. Acta Informatica 6 (1976), 365-386.

[7] Brandwajn, A. A queueing model of a multiprogramming computer system under full load conditions. Journal of the ACM 24, 2 (April 1977), 222-240.

[8] Brown R. M., Browne, J. C. and Chandy, K. M. Memory management and response time. Communications of the ACM 20, 3 (March 1977), 153-165.

[9] Bryant, R. M. and Agrawala, A. K. An evaluation of M/M/R queues as finite memory size models of computer systems. Proceedings of the 1977 Sigmetrics/CMG VIII Conference on Computer Performance Modelling, Measurement, and Management, Washington, D. C., November 28-December 1, 1977.

[10] Bryant, R. M. The storage limited exponential queue and applications to finite memory size models of computer systems. Ph. D. Thesis, Applied Mathematics, University of Maryland, College Park, May 1978.

[11] Bryant, R. M. Approximate, product form solutions to the storage limited exponential queue. Computer Sciences Department Techical Report, University of Wisconsin-Madison, in preparation.

[12] Buzen, J. P. and Rubin, D. B. Effects of compaction on memory utilization in multiprogramming systems. Proceedings of the International IRIA Workshop on Modeling and Evaluation of Computer Architectures and Networks, Rocquencourt, France, 1974, 113-124.

[13] Coffman, E. G., Jr. and Denning, P. J. Operating Systems Theory. Prentice-Hall, Englewood Cliffs, 1973, p. 172.

[14] Courtois, P. J. Decomposability: Queueing and Computer System Applications. Academic Press, New York, 1977.

[15] Gelenbe, E. and Kurinckx, A. Random injection control of multiprogramming in virtual memory. IEEE Transactions on Software Engineering SE4, 1 (January 1978), 2-17.

[16] Hoel, P. G., Port, S. C. and Stone, C. J. Introduction to Stochastic Processes. Houghton Mifflin, Boston, 1972, p. 62.

[17] Kiviat, P. J., Villanueva, R. and Markowitz, H. M. Simscript II.5 Programming Language. C. A. C. I., Inc., Los Angeles, 1973.

[18] Kleijnen, J. P. C. Statistical Techniques in Simulation, Parts I and II. Marcel Dekker, New York, 1974, 1975.

[19] Knuth, D. E. The Art of Computer Programming, Volume 2: Seminumerical Algorithms. Addison-Wesley, Reading, 1969, p. 88.

[20] Konheim, A. G. and Reiser, M. A queueing model with finite waiting room and blocking. Journal of the ACM 23, 2 (April 1976), 328-341.

[21] Konheim, A. G. and Reiser, M. Finite capacity queueing systems. SIAM J. Computing 7, 2 (May 1978), p. 210-229.

[22] Little, J. D. C. A proof for the queueing formula L=λW. Operations Research 9, 3 (May 1961), 383-387.

[23] Marathe, V. P. Priority queueing systems with simultaneous server requirements. Ph. D. thesis, Operations Research, Cornell University, May, 1972.

[24] Muntz, R. R. Analytic modeling of interactive systems. Proceedings of the IEEE 63, 6 (June 1975), 946-953.

[25] Payne, W. H. and Rabung, J. R. Coding the Lehmer pseudo-random number generator. Communications of the ACM 12, 2 (February 1969), 85-86.

[26] Omahen, K. J. Analytic models of multiple resource systems. Ph. D. thesis, Committee on Information Sciences, University of Chicago, June, 1973.

[27] Omahen, K. J. Capacity bounds for multiresource queues. Journal of the ACM 24, 4 (October 1977), 646-663.

[28] Omahen, K. and Marathe, V. A queueing model for a multiprocessor system with partitioned memory. Computer Science Technical Report 132, Purdue University, January 1975.

[29] Revesz, P. The Laws of Large Numbers. Academic Press, New York, 1968, p. 130.

[30] Ross, S. M. Applied Probability Models with Optimization Applications. Holden-Day, San Francisco, 1970, pp. 31-60.