S. S. Lavenberg
G. S. Shedler

Stochastic Modeling of Processor Scheduling with Application to Data Base Management Systems

Abstract: This paper is concerned with the stochastic modeling of processor scheduling and of queuing due to contention for resources in data base management systems. The processing services rendered in searching the data base and retrieving and processing information are modeled explicitly, as is the algorithm used to schedule these services on the processor. The scheduling of the processor is based on a total priority ordering of a set of queues for processing service. A queuing model incorporating the processor scheduling algorithm for IMS (Information Management System) is formulated in order to illustrate the modeling ideas. The model is analyzed under rather general distributional assumptions, based on the observation that certain stochastic processes in the model are cumulative processes defined over the same embedded semi-Markov process. The model is not used in a performance study of IMS, nor is it proposed that the model developed here is one upon which a performance study of IMS should be undertaken. The model should be viewed as illustrative of stochastic models which can be constructed to incorporate algorithms for processor scheduling.

Introduction

The algorithms used to schedule a computer system's resources can affect system performance. This paper is concerned with the formulation of stochastic models of the contention for resources in computer systems, which incorporate explicit representations of algorithms for processor scheduling. The modeling is illustrated in the context of data base management systems. A data base management system is a processing program that operates under the operating system of a computer and facilitates the accessing of large quantities of data shared in common by several applications. When a data base management system runs on a computer there can be contention by the data base management program and the application programs for the processing and input-output resources of the system. The processor provides several types of service in searching the data base and retrieving and processing information, and the scheduling of these services on the processor can affect system performance significantly.

The several types of service rendered by the processor are modeled explicitly. The scheduling of these services on the processor is represented as follows. There can be more than one queue for each type of processor service and the queue entered by a particular customer requiring processor service depends on the type of service required and on the service just completed for that customer. The scheduling of the processor is based on a total static priority ordering of the queues for processor service. This formulation allows rather complex processor scheduling algorithms to be represented. To illustrate these modeling ideas, a model is formulated which represents the processor scheduling algorithm for a particular data base management system, IMS (Information Management System) [1].

Most of the previous literature analyzing priority scheduling in queuing models has been concerned with single resource models, i.e., the single server queue (see Kleinrock [2] for an account of such models). Here priority scheduling is incorporated in a multiple resource model; i.e., both processor and input-output resources are represented. An explicit representation of different types of processor service and processor scheduling in a multiple resource queuing model can be found in Lewis and Shedler [3]. Lavenberg [4] formulated a class of multiple resource models incorporating priority scheduling based on a total ordering of queues. However, unlike the model developed here, there can be only one queue for each type of service. An alternative representation of multiple types of service is possible via the notion of customer classes as used by Baskett, et al. [5]. (The models considered in [5] do not incorporate priority scheduling, however.)

The model developed here is analyzed based on the existence of an embedded finite state semi-Markov process, using techniques similar to those in [3]. In order that an embedded semi-Markov process exist, it is assumed that certain service times are mutually independent and exponentially distributed random variables. However, all services that cannot be performed simultaneously (e.g., all processor services) and that are not

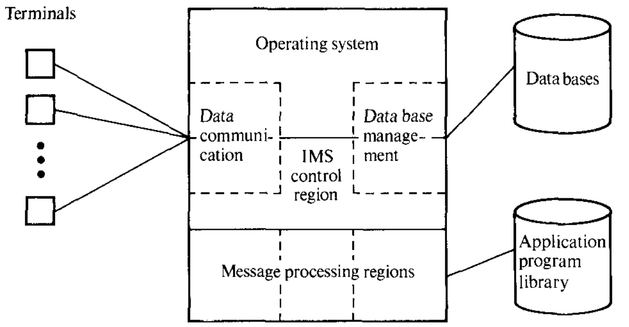

Figure 1 Configuration of a computer system running IMS.

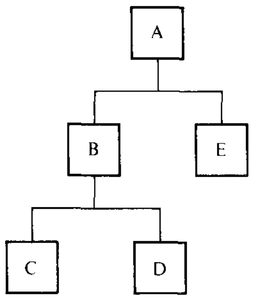

Figure 2 Hierarchical tree structure.

interruptable are assumed to be mutually independent random variables having arbitrary distributions, where the distribution depends on the service type.

Although the particular model described here has been formulated in the context of a computer running IMS, this paper does *not* contain a performance study of IMS. The model should be considered illustrative of stochastic models that can be constructed to incorporate algorithms for processor scheduling in data base management systems. Although the model is in general consistent with the processor scheduling algorithm of IMS, it is not proposed as the model upon which a performance study of IMS should be undertaken. In particular, there is no claim that all system features relevant to such a performance study have been incorporated in the model. Moreover, there has been no attempt here to validate the particular probabilistic assumptions that have been made. To do so would involve the collection of data from representative running systems and the statistical analysis of those data (cf. Gaver, Lavenberg and Price [6]). Validation of the total model could be based on the comparison of one or more stochastic processes observed in the running system with corresponding processes predicted by the model. Techniques for the statistical analysis of such stochastic processes that may be applicable in a total model validation can be found in Lewis and Shedler [7].

The next section of the paper is a description of IMS that is intended to provide a background for the remainder of the paper. In the third section, contention for the resources of a computer system running IMS and processor scheduling in IMS are discussed. A queuing model of IMS incorporating processor scheduling is formulated in the fourth section. The last section contains an analysis of the queuing model.

Description of IMS

IMS operates under the operating system of a computer system and extends the data communication and data base management capabilities of the operating system. Only the data base management facility of IMS called Data Language/I (DL/I) will be considered in this paper (IMS/360, Version 2, Release 2.3).

In IMS, users can access the data bases from remote terminals by entering messages called *transactions*. A particular transaction uniquely identifies an *application program*, which processes the message and accesses the data bases. The execution of an application program gives rise to a sequence of *data base calls* to the IMS program. (Application programs also issue data communication calls to IMS, but these are not considered in this paper.)

A conceptual diagram of a computer system running IMS is given in Fig. 1. As indicated, the operating system resides in memory. The IMS program occupies a portion of memory called the *IMS control region*. Application programs reside in secondary storage in an application program library. For execution, an application program must be loaded into one of several regions in memory, called *IMS message processing regions*. The data bases reside in secondary storage and data are transferred into memory for processing in response to transactions.

Application programs written to use IMS deal with logical data structures and are independent of the physical data structure, "physical" referring to the manner in which the data are stored in secondary storage. In IMS, the logical data structure is a hierarchical tree structure of one or more *segment types*, a segment type being a data element composed of related data fields. Figure 2 depicts a hierarchical tree structure consisting of five segment types. Segment A at the top of the tree is called the *root segment*. The segments below A in the tree and directly connected to A, i.e., segments B and E, are called the *children* of A, and A is called the *parent* of B and E. Similarly, C and D are the children of B, and B is the parent of C and D. All segments below A and connected to A by a path in the tree, i.e., segments B, C, D, and E, are said to be *dependent* on A.

In general, many *instances* of each segment type exist in the tree. Lower case letters with multiple subscripts can be used to designate segment instances and to make segment dependencies explicit. Thus, for the hierarchical tree structure in Fig. 2,

a_i = *i*th instance of root segment A,

b_{ij} (resp. e_{ij}) = *j*th instance of segment type B(resp. E) whose parent is a_i, and

c_{ijk}(resp. d_{ijk}) = *k*th instance of segment type C(resp. D) whose parent is b_{ij}.

A tree consisting of a single instance of a root segment and all its dependent segment instances is called a *logical data base record.* The tree shown in Fig. 3, which depicts all segment instances dependent on a_i, is a logical data base record. The set of all logical data base records having the same root segment type comprises a *logical data base.* In a logical data base a particular ordering of segment instances is assumed, the *top-down left-to-right* (TDLR) *ordering.* To illustrate TDLR ordering, suppose $j = 1$ in Fig. 3; i.e., the logical data base record shown in Fig. 3 is the first record in a logical data base. The segment instances that comprise the logical data base record are shown numbered in TDLR order in Fig. 4. If there were a second record in the logical data base, its root segment instance would be numbered 13, and the numbering of its remaining segment instances would proceed in a manner similar to that shown in Fig. 4.

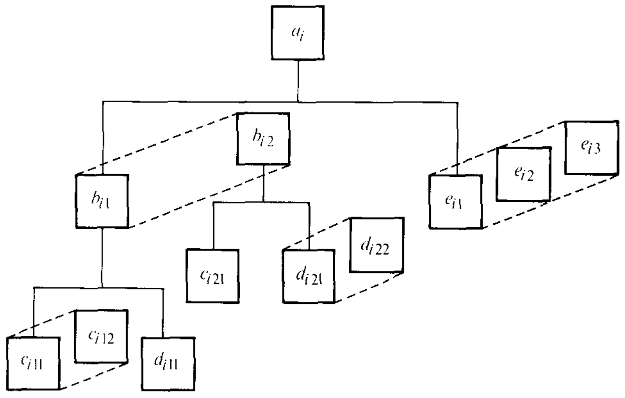

Figure 3 Logical data base record.

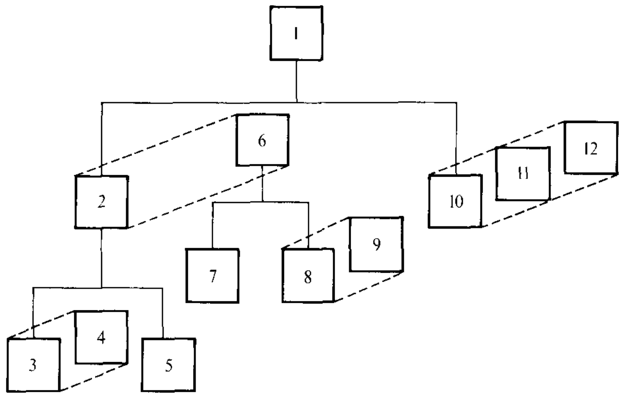

Figure 4 TDLR ordering.

Each data base call issued by an application program constitutes a request to IMS to perform an operation of specified type on a segment instance in a logical data base. Examples of operations specified by data base calls are: *get* a *unique* segment instance, *get* the *next* sequential (in the TDLR ordering) segment instance, *replace* the data in an existing segment instance, *delete* an existing segment instance, and *insert* a new segment instance. The segment instance to be operated on is called the *target segment* and is explicitly specified by the call and, in the case of *get next* calls, by the current position of a cursor for the data base. In general, it is necessary to search the data base, i.e., to access a number of distinct segment instances, in order to find the target segment; the segment instances that are accessed are called *path segments.* (The final segment instance in the access path is the target segment.) At the successful completion of a data base call, the cursor is positioned at the target segment.

IMS maintains a *buffer pool* of *physical blocks* in memory; each block contains one or more segment instances. A physical block is the unit of transfer of data between memory and the data bases residing in secondary storage. When a particular segment is to be accessed, IMS first determines if the block containing the segment is in the buffer pool. If it is, the segment can be accessed without performing a physical I/O operation (e.g., a disk access and data transfer); if it is not, an I/O operation must be performed to transfer the block containing the segment from the data bases to the buffer pool.

IMS supports two physical storage organizations, *hierarchical sequential* and *hierarchical direct.* The order in which segment instances are accessed differs for these two organizations. Under the hierarchical sequential storage organization, segment instances in a logical data base are accessed consecutively in TDLR order from some position in the data base. (The position is the beginning of the data base for *get unique* calls, and the position is the current cursor position for *get next* calls.) Under the hierarchical direct storage organization, certain segment instances can be related by pointers, and segment instances related by pointers can be accessed successively (intervening segment instances are not accessed). Segment instances of the same particular type that are children of the same parent segment instance are called *twins.* (All root segment instances in a logical data base are twins.) Under the direct organization, adjacent twins can be related by forward, or forward and backward, pointers; e.g., if $\{b_{ij}\}$ are twins which are children of a_i, then b_{ij} and b_{ij+1} can be related by pointers. A parent segment instance can be related by forward pointers to the first, or first and last, instances of each of its children; e.g., if $\{b_{ij}:j = 1, \cdots, n\}$ and $\{e_{ij}:j = 1, \cdots, m\}$ are all the instances of children of segment instance a_i, then a_i and b_{i1} can be related by a pointer as can a_i and b_{in}, a_i and e_{i1}, a_i and e_{im}. Each instance of a child can be related by a backward pointer to its parent segment instance. Segment instances that are consecutive in TDLR order can also be related by pointers. In addition, indices can be maintained under either storage organization that allow root segment instances to be accessed without accessing any other segment instances. Clearly, the physical storage organization affects the number of path segments that must be accessed in order to find the target segment. The number of path segments accessed is called the *access path length.*

Resource contention and processor scheduling in IMS

In the next section a queuing model of processor scheduling in IMS is presented. The interactions between two abstract servers, a *processor* and an *I/O unit* are considered in the model. The model contains an explicit representation of the several processor activities involved in servicing data base calls (e.g., determining the next path segment to be accessed, searching the buffer pool, processing the target segment), along with the algorithm used in scheduling these services on the processor. The transfer of physical blocks from the data bases to the buffer pool is also represented in the model. The collection of devices (e.g., disks and channels) that perform the data transfers comprises the I/O unit. Processor activities related to data communication and the activity of the terminals are not represented in the model, nor is the loading of application programs from the application program library into the message processing regions. The model addresses the congestion problems arising from the contention for processor and I/O unit resources when several message processing regions are active simultaneously.

When a data base call is issued from a particular IMS message processing region, a sequence of processor and I/O unit services is performed in response to that call. Six types of services can be distinguished. Services of the processor are required to:

1. determine the first path segment to be accessed,
2. search the buffer pool for the path block, i.e., the block containing the path segment,
3. find the path segment within the path block and examine the path segment in order to determine whether or not the path segment is the target segment (if not, another path segment must be retrieved),
4. perform overhead activities associated with a block transfer,
5. process the target segment.

Service of the I/O unit is required to:

6. transfer a block from the data base to the buffer pool.

The order in which these processor and I/O unit services are rendered by IMS to an *individual* data base call can be described as follows:

Service 1) is performed first and then services 2) and 3) are performed alternately until either a) a buffer pool exception occurs, i.e., the block containing the path segment is not in the buffer pool, or b) the target segment is found, i.e., the path segment is the target segment. In the former case, services 4) and 6) are performed between services 2) and 3). In the latter case, service 5) is performed after service 3) and is the last service performed in response to the data base call.

Several data base calls can compete concurrently for the processing and I/O resources of the system. The allocation of the processing resources to the competing data base calls, i.e., the scheduling of the processor, is summarized next. In IMS, a data base call receiving processor services 1), 2), 3) or 4) maintains sole access to the processor until either processor service 5) or I/O unit service 6) is to be performed. At that time, the processor may be allocated to a competing data base call as determined by a priority rule. The priority rule implemented in IMS is essentially the following:

Highest priority is given to resuming processor service [i.e., providing service 3)] for a data base call whose block transfer [service 6)] has been completed. Next highest priority is given to initiating processor service [i.e., providing service 1)] for previously unserviced data base call. Lowest priority is given to processing the target segment [i.e., providing service 5)].

Processor service 5), once begun, is interruptable at the completion of I/O unit service 6). This interruption is of the preemptive-resume type and the next processor service to be performed is determined by the above priority rule. No other services are interruptable.

Description of the queuing model

The servicing of competing data base calls can be represented mathematically as a network of interconnected queues that will now be specified. It is assumed in this model that at any instant of time there is a fixed number of data base calls issued, one from each message processing region; i.e., the number of active message processing regions is constant. No distinction is made among different types of data base calls. When service is completed in response to a particular data base call, the message processing region that issued the call is assumed to issue another call immediately. The model is depicted in Fig. 5.

The interpretation of the diagram for this model differs from that of a conventional queuing model in that *services* are distinguished from the *servers* which perform them. Thus, circles in the figure represent services rather than servers. Five types of service, denoted by α_0, α_1, α_2, α_3 and β are represented in this model. The α services (α_0, α_1, α_2 and α_3) are performed by a single server, representing the processor, and the β service is performed by a single server, representing the I/O unit. It is assumed in the model that no two α services can be performed concurrently, but that any α service can be performed concurrently with a β service. Each of the α_0, α_1, α_2 and β services is assumed to be non-interruptable. The α_3 service is, however, interruptable at the completion of a β service, this interruption being of the preemptive-resume type. The interpretation of the α and β services in terms of the services described in the previous section that are performed when servicing a data base call is shown in Table 1.

A fixed number of customers, each customer representing a data base call, circulates in the network. At time zero all customers are assumed to be in the queue denoted by q_0 and an α_0 service is about to begin. At any subsequent instant of time a customer is either receiving service or waiting for service in one of the queues. Each of the queues is assumed to be a first-come first-served queue. Note that there can be more than one queue for a particular type of service, e.g., there are two queues, denoted by $q_{1,1}$ and $q_{1,2}$, for the α_1 service. The flow of customers is indicated by the arrows in Fig. 5. Note that there are two branches leaving the α_1 service and two branches leaving the α_2 service. These branches are labeled by binary-valued variables ψ_1 and ψ_2 defined as follows:

$$\psi_1 = \begin{cases} 1, \text{ if buffer pool exception occurs} \\ 0, \text{ else;} \end{cases}$$

$$\psi_2 = \begin{cases} 1, \text{ if path segment is target segment} \\ 0, \text{ else.} \end{cases}$$

Upon completion of an α_1 or α_2 service, the customer just served follows the branch whose label has value 1.

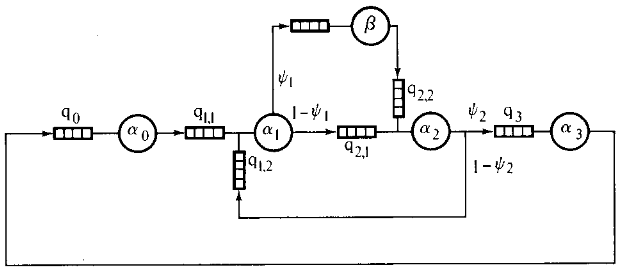

Figure 5 Queuing model incorporating processor scheduling.
1. Processor renders α_0, α_1, α_2 and α_3 services.
2. I/O unit renders β service.
3. α_0, α_1, α_2 and β services are not interruptable.
4. α_3 service has pre-emptive resume type interruption at completion of β service.
5. Processor scheduled according to priority ordering of queues q_0, $q_{1,1}$, $q_{1,2}$, $q_{2,1}$, $q_{2,2}$ and q_3.
6. Routing determined by binary valued random variables ψ_1 and ψ_2.

Table 1 Interpretation of services in model.

Service in model	*Interpretation of service*
α_0	1)
α_1	2) if no buffer pool exception occurs; 2) followed by 4) if a buffer pool exception occurs
α_2	3)
α_3	5)
β	6)

Thus, for example, if upon completion of an α_1 service the current value of ψ_1 is zero, the customer just served enters queue $q_{2,1}$.

The epoch of completion of any α service or the epoch of completion of a β service at which either no α service is in progress or an α_3 service is in progress is called a *scheduling decision epoch* and is an epoch at which the next processor service to be performed, if any, is determined by a *processor scheduling algorithm*. It is assumed that the customer whose service has just been completed immediately enters the next queue on its route at such an epoch. The next processor service is the service having highest priority, where priority is determined by a total ordering of queues q_0, $q_{1,1}$, $q_{1,2}$, $q_{2,1}$, $q_{2,2}$ and q_3. The processor scheduling algorithm $\mathscr{A}$ employs the total ordering $q_{1,2}$, $q_{2,1}$, $q_{1,1}$, $q_{2,2}$, q_0, q_3 from highest to lowest priority. A flowchart of this processor scheduling algorithm is given in Fig. 6. Theorem 1, below, establishes some consequences of applying algorithm $\mathscr{A}$ at each scheduling decision epoch. The following lemma will be used in the proof of this theorem.

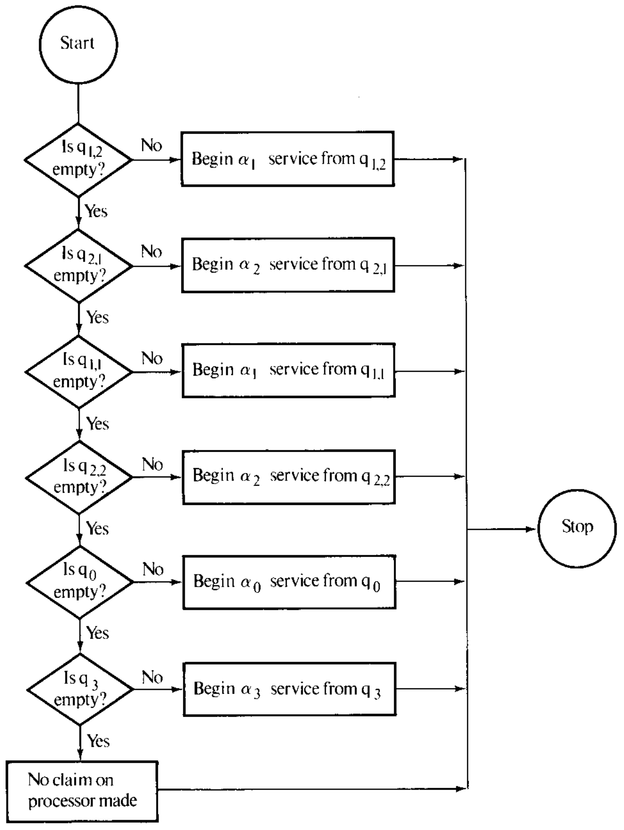
Start
Is $q_{1,2}$ empty?
No
Begin α_1 service from $q_{1,2}$
Yes
Is $q_{2,1}$ empty?
No
Begin α_2 service from $q_{2,1}$
Yes
Is $q_{1,1}$ empty?
No
Begin α_1 service from $q_{1,1}$
Yes
Is $q_{2,2}$ empty?
No
Begin α_2 service from $q_{2,2}$
Yes
Is q_0 empty?
No
Begin α_0 service from q_0
Yes
Is q_3 empty?
No
Begin α_3 service from q_3
Yes
No claim on processor made
Stop

Figure 6 Processor scheduling algorithm.

Lemma 1 If at time zero all customers are in queue q_0 with α_0 service about to begin, then under processor scheduling algorithm $\mathscr{A}$ queues $q_{1,1}$, $q_{1,2}$, and $q_{2,1}$ will always be empty, except instantaneously at epochs of completion of α_0, α_2 (if $\psi_2 = 0$) and α_1 (if $\psi_1 = 0$) services, respectively.

Proof A customer enters $q_{1,2}$ only upon completion of an α_2 service. Consider the first time a customer enters $q_{1,2}$. Since $q_{1,2}$ has highest priority (see Fig. 6) it empties immediately and an α_1 service starts. By induction on k it can be shown that for any k, the kth time a customer enters $q_{1,2}$ this queue empties immediately and an α_1 service starts. A customer enters $q_{2,1}$ only upon completion of an α_1 service. At any such time $q_{1,2}$ is empty as just shown. Consider the first time a customer enters $q_{2,1}$. Because $q_{1,2}$ is empty and $q_{2,1}$ has second highest priority, $q_{2,1}$ empties immediately and an α_2 service starts. Again by induction it can be shown that whenever a customer enters $q_{2,1}$, this queue empties immediately and an α_2 service starts. By a similar argument it can be shown that whenever a customer enters $q_{1,1}$, this queue empties immediately and an α_1 service starts.

Theorem 1 If processor scheduling algorithm $\mathscr{A}$ is applied at each scheduling decision epoch, then processor services are scheduled so that a customer receiving processor services α_0, α_1 or α_2 maintains sole access to the processor until either an α_2 service is completed and $\psi_2 = 1$ or an α_1 service is completed and $\psi_1 = 1$. At such a point in time or at the completion of a β service while an α_3 service is in progress, the next processor service to be performed is chosen according to a priority rule as follows:

Highest priority is given to a customer in queue $q_{2,2}$ (awaiting α_2 service). Next highest priority is given to a customer in queue q_0 (awaiting α_0 service). Lowest priority is given to a customer in queue q_3 (awaiting α_3 service).

Proof It follows from Lemma 1 that a customer completing an α_0 service immediately begins an α_1 service, a customer completing an α_1 service with $\psi_1 = 0$ immediately begins an α_2 service and a customer completing an α_2 service with $\psi_2 = 0$ immediately begins an α_1 service. Furthermore, at an α_2 service completion with $\psi_2 = 1$ or at an α_1 service completion with $\psi_1 = 1$, $q_{1,1}$, $q_{1,2}$ and $q_{2,1}$ are empty. At such scheduling decision epochs algorithm $\mathscr{A}$ gives highest priority to beginning an α_2 service from $q_{2,2}$, next highest priority to beginning at α_0 service from q_0 and lowest priority to beginning an α_3 service from q_3. Finally, since an α_3 service has lowest priority under algorithm $\mathscr{A}$, an α_3 service can only be in progress if q_0, $q_{1,1}$, $q_{1,2}$, $q_{2,1}$ and $q_{2,2}$ are empty. Thus, if a β service completes while an α_3 service is in progress, an α_2 service is begun from $q_{2,2}$.

This theorem implies that the scheduling of processor services in the model is identical to the scheduling of processor services in IMS as described in the previous section.

The queuing model just described is analyzed in the next section under the following eight *probabilistic assumptions*:

1. For $i = 0, 1, 2, 3$ successive α_i service times form a sequence $\{a_{ik}: k = 1, 2, \cdots\}$ of independent and identically distributed (i.i.d.) random variables, each distributed as a random variable a_i.
2. Successive β service times form a sequence $\{b_k: k = 1, 2, \cdots\}$ of i.i.d. random variables, each distributed as a random variable b.
3. For $i = 1, 2$, ψ_i is a random variable and values of ψ_i at successive α_i service completions form a sequence $\{\psi_{ik}: k = 1, 2, \cdots\}$ of i.i.d. random variables.
4. The sequence of pairs of random variables $\{(a_{1k}, \psi_{1k}): k = 1, 2, \cdots\}$ is a sequence of i.i.d. random vectors, each distributed as the random vector (a_1, ψ_1). The random variables a_{1k} and ψ_{1k} may be dependent

random variables since a_{1k} is interpreted as the duration of processor service 2) if $\psi_{1k} = 0$ and as the sum of the durations of processor services 2) and 4) if $\psi_{1k} = 1$ (see Table 1). If a_{1k} and ψ_{1k} are assumed to be dependent, then a joint distribution of the random variables a_1 and ψ_1 must be specified.

5. All other random variables are mutually independent.
6. The random variables a_3 and b are exponentially distributed.
7. Each of the random variables a_0, a_1 and a_2 is non-negative with non-zero finite mean but otherwise arbitrary distribution function.
8. $\Pr\{\psi_2 = 1\} > 0$, so that customers eventually receive α_3 service.

Some comments about these probabilistic assumptions are appropriate. In analyzing this model, epochs of time at which no α service is in progress play a special role. If b is exponentially distributed, the elapsed β service time at these epochs need not be part of the definition of the state of the model. Similarly, if a_3 is exponentially distributed, the elapsed α_3 service time at an interruption caused by a β service completion need not be part of the state definition. Since α_0, α_1 and α_2 services are not interruptable, a_0, a_1 and a_2 may have distributions of arbitrary form.

Note that the assumptions that the random variables ψ_{2k} are i.i.d. and independent of all other random variables implies that the number of segments accessed in response to a data base call, i.e., the access path length, has a geometric plus one distribution.

Analysis of the queuing model

In this section an analysis is given of the queuing model of processor scheduling in IMS as defined in the previous section. The analysis yields expressions for the long-run fractions of time during which the processor performs α_0, α_1, α_2 and α_3 services, the long-run fraction of time during which the I/O unit is busy, the long-run number of data base calls serviced per unit time and the average response time for a data base call. Techniques of analysis similar to those in Lewis and Shedler [3] will be used. The analysis given is based on the observation that the processes of accumulated service times for each type of service in the model are cumulative processes defined over the same embedded semi-Markov process.

• *State definition*

For $t \geq 0$, denote by $n_0(t), n_{1,1}(t), \cdots, n_3(t)$ the number of customers in queue $q_0, q_{1,1}, \cdots, q_3$, respectively, at time $t+$ and by $n_4(t)$ the number of customers in the queue for the β service at time $t+$. Let

$$i_k(t) = \begin{cases} 1, \text{ if an } \alpha_k \text{ service is in progress at time } t+ \\ 0, \text{ else}, \qquad k = 0, 1, 2, 3, \end{cases}$$

$$i_4(t) = \begin{cases} 1, \text{ if a } \beta \text{ service is in progress at time } t+ \\ 0, \text{ else}. \end{cases}$$

It follows from Lemma 1 of the previous section that $n_{1,1}(t) = n_{1,2}(t) = n_{2,1}(t) = 0$ for all $t \geq 0$.

The *state* of the queuing network at any time $t \geq 0$ is defined to be the 5-tuple $\mathbf{v}(t) = (v_0(t), v_1(t), v_2(t), v_3(t), v_4(t))$ where $v_0(t) = n_0(t) + i_0(t)$, $v_1(t) = i_1(t)$, $v_2(t) = n_{2,2}(t) + i_2(t)$, $v_3(t) = n_3(t) + i_3(t)$ and $v_4(t) = n_4(t) + i_4(t)$. The components of $\mathbf{v}(t)$ sum to N, where N is the number of customers in the queuing network. Since a_0, a_1 and a_2, respectively, the α_0, α_1 and α_2 service times, have arbitrary (i.e., in general, non-exponential) distributions the stochastic process $\mathbf{v}(t)$ is not a Markov process, but is Markovian only at certain epochs of time.

• *Embedded Markov chain and embedded semi-Markov process*

Let $\{t_k : k = 1, 2, \cdots\}$ be the increasing sequence of all epochs of completion of α_0, α_1 and α_2 services. Since neither an α_0, α_1 nor α_2 service can be in progress at t_k, under the probabilistic assumptions that have been made (e.g., a_3 and b, the α_3 and β service times, are exponentially distributed) it can be verified that the embedded discrete time stochastic process $\{\mathbf{v}(t_k) : k = 1, 2, \cdots\}$ is a Markov chain. In addition, it can be verified that the time between epochs t_{k-1} and t_k is, given the state at these epochs, independent of the time between previous epochs and the state at previous epochs. Thus, the embedded continuous time stochastic process $\mathbf{v}'(t)$ defined by $\mathbf{v}'(t) = \mathbf{v}(t_{k-1})$, $t_{k-1} \leq t < t_k$, $k = 1, 2, \cdots$, where $t_0 = 0$, is a finite state semi-Markov process. Note that both the *embedded Markov chain* and the *embedded semi-Markov process* have finite state space given by $\mathscr{S}_N = \{\mathbf{v} = (v_0, v_1, v_2, v_3, v_4) : v_0, \cdots, v_4$ are non-negative integers, $v_0 + \cdots + v_4 = N$, $v_1 = 0$ or $1\}$ where the dependence on N, the number of customers in the network, has been denoted explicitly.

A number of states in the embedded Markov chain and hence in the embedded semi-Markov process are transient. The following lemma is used to identify the transient states.

Lemma 2 If at time zero all customers are in queue q_0 with an α_0 service about to begin, then after a finite time queue q_0 will be empty and will always be empty thereafter (except instantaneously at epochs of completion of an α_3 service).

Proof A customer enters queue q_0 only upon completion of an α_3 service. Since an α_3 service is interruptable only at completions of β service, i.e., when a customer enters queue $q_{2,2}$, and α_3 service has lowest priority under algorithm $\mathscr{A}$, it follows that queues $q_{1,2}$, $q_{2,1}$, $q_{1,1}$, $q_{2,2}$ and q_0 are empty immediately prior to an α_3 service completion. Thus, under algorithm $\mathscr{A}$, an α_0 service starts immediately for the customer whose α_3 service was just completed. Since all service times are finite and $\Pr\{\psi_{2k} = 1\} > 0$, an α_3 service completion will occur in finite time. The proof is complete.

It follows from Lemma 2 that $\lim_{k\to\infty} \Pr\{n_0(t_k) = 0\} = 1$. Also, note that $i_0(t_k) = 0$ for all k. Thus, the set of states $\mathscr{T}_N = \{\mathbf{v} \in \mathscr{S}_N : v_0 > 0\}$ is transient. For any state $\mathbf{v} \in \mathscr{S}_N - \mathscr{T}_N$, $v_0 = 0$. It is shown in Appendix A that all states in $\mathscr{S}_N - \mathscr{T}_N$ communicate. Thus, $\mathscr{S}_N - \mathscr{T}_N$ is the single irreducible class of the embedded Markov chain and hence of the embedded semi-Markov process.

Since all service times are assumed to have finite mean, it is straightforward to show that the mean holding times are finite for all states of the embedded semi-Markov process. Thus, all states in the irreducible class $\mathscr{S}_N - \mathscr{T}_N$ are positive recurrent in the embedded semi-Markov process (cf. Ross [8]). This result guarantees the existence of limits for the processes of accumulated service times.

• *Cumulative processes*
Let $\mathbf{v}_0$ be any state in $\mathscr{S}_N - \mathscr{T}_N$. Let $\{\tau_k : k = 1, 2, \cdots\}$ be the subsequence of $\{t_k : k = 1, 2, \cdots\}$ having the property that for each k, $\mathbf{v}(\tau_k) = \mathbf{v}_0$. Then $\{\tau_k - \tau_{k-1} : k = 1, 2, \cdots\}$, where $\tau_0 = 0$, is a delayed renewal process, i.e., $\tau_k - \tau_{k-1}$ are i.i.d. for $k = 2, 3, \cdots$ and independent of $\tau_1 - \tau_0$. Since $\mathbf{v}_0$ is positive recurrent, $\mathrm{E}[\tau_2 - \tau_1] < \infty$. Denote by $W_i(t)$ the total amount of time that α_i service is performed in $[0, t)$, $i = 0, 1, 2, 3$, and by $W_4(t)$ the total amount of time that β service is performed in $[0, t)$. Then $W_i(t)$ is a *cumulative process* (cf. Smith [9]) defined with respect to the delayed renewal process $\{\tau_k - \tau_{k-1}\}$, and since $W_i(t) \le t$, $\mathrm{E}[W_i(\tau_2) - W_i(\tau_1)] < \infty$. Thus, $U_i = \lim_{t\to\infty} W_i(t)/t$ exists with probability one and

$$U_i = \frac{\mathrm{E}[W_i(\tau_2) - W_i(\tau_1)]}{\mathrm{E}[\tau_2 - \tau_1]}. \tag{1}$$

The quantities U_i, $i = 0, \cdots, 4$, are by definition, respectively, the utilization of the processor due to α_0, α_1, α_2 and α_3 services and the I/O unit utilization. These utilizations need not be determined independently, however. It is shown in Lavenberg and Shedler [10] that

$$\frac{U_i}{U_1} = \begin{cases} p_2\mathrm{E}[a_0]/\mathrm{E}[a_1], & i = 0 \\ \mathrm{E}[a_2]/\mathrm{E}[a_1], & i = 2 \\ p_2\mathrm{E}[a_3]/\mathrm{E}[a_1], & i = 3 \\ p_1\mathrm{E}[b]/\mathrm{E}[a_1], & i = 4, \end{cases} \tag{2}$$

where $p_i = Pr\{\psi_i = 1\}$, $i = 1, 2$, and E[ ] denotes expectation. Thus, if any one of the utilizations, say U_1, is known, the other four utilizations can be easily determined by using (2). The total processor utilization is given by $U_\alpha = U_0 + U_1 + U_2 + U_3$.

Note that the ratio of processor to I/O unit utilization is given by

$$\frac{U_\alpha}{U_4} = \frac{\mathrm{E}[a_0] + (\mathrm{E}[a_1] + \mathrm{E}[a_2])/p_2 + \mathrm{E}[a_3]}{\mathrm{E}[b]p_1/p_2} \tag{3}$$

and is independent of N, the number of customers in the network.

Let $\{R_k : k = 1, 2, \cdots\}$ denote the sequence of successive response times in the model, i.e., for the kth customer to enter queue q_0 (the customer may spend zero time in queue q_0), R_k is the time from when this customer enters queue q_0 until this customer next completes an α_3 service. (The first customer to enter queue q_0 is the customer in the last position in queue q_0 at time zero.)

It is shown in [10] that $R = \lim_{n\to\infty} \sum_{k=1}^{n} R_k/n$ exists with probability one and that

$$R = \frac{N\mathrm{E}[a_0]}{U_0}. \tag{4}$$

The quantity R is interpreted as the average response time for a data base call. It is also shown in [10] that the long-run number of completions of α_3 service per unit time, here denoted by Λ, is related to R by the formula

$$\Lambda = N/R. \tag{5}$$

The quantity Λ is interpreted as the throughput, i.e., the long-run number of data base calls serviced per unit time.

The expectations $\mathrm{E}[\tau_2 - \tau_1]$ and $\mathrm{E}[W_1(\tau_2) - W_1(\tau_1)]$ are computed in Appendix B. All utilizations, the average response time and the throughput can then be determined easily by using (1), (2), (4) and (5). Numerical studies for the queuing model, obtained via the analysis of this section and Appendix B, can be found in [10].

Acknowledgment
The authors are indebted to J. Rodriguez-Rosell and W. G. Tuel for sharing their insights into the operation of IMS. The present work benefited from conversations with them about resource contention and processor scheduling in IMS, although this model does not apply directly to IMS.

Appendix A
Here it is shown that all states in $\mathscr{S}_N - \mathscr{T}_N$ communicate. It is convenient to represent any state $\mathbf{v} = (0, v_1, v_2, v_3, v_4) \in \mathscr{S}_N - \mathscr{T}_N$ by a triple $\mathbf{x} = (x_1, x_2, x_3)$ where $x_1 = v_2$, $x_2 = v_3$ and $x_3 = v_4$ so that $x_1 + x_2 + x_3 = N - 1$ or N. Let Ω_N be the set of all such triples, i.e.,

$$\Omega_N = \{\mathbf{x} = (x_1, x_2, x_3): x_1, x_2, x_3 \text{ are non-negative integers, } x_1 + x_2 + x_3 = N - 1 \text{ or } N\}.$$

The cardinality of Ω_N can be shown to be equal to $(N + 1)^2$. In what follows the members of Ω_N will also be called states.

Let $\mathbf{x} \rightarrow \mathbf{y}$ denote that a transition in the embedded Markov chain from state $\mathbf{x} \in \Omega_N$ to state $\mathbf{y} \in \Omega_N$ occurs with positive probability. It can be shown (e.g., see (B1)-(B8) of Appendix B) that:

if $x_1 + x_2 + x_3 = N - 1$, then

$$\mathbf{x} \rightarrow (x_1 + i, x_2, x_3 + 1 - i), \quad i = 0, \cdots, x_3 + 1; \quad \text{(A1)}$$

if $x_1 + x_2 + x_3 = N$ and $x_1 > 0$, then

$$\mathbf{x} \rightarrow (x_1 - 1 + i, x_2, x_3 - i), \quad i = 0, \cdots, x_3 \quad \text{(A2)}$$

$$\mathbf{x} \rightarrow (x_1 - 1 + i, x_2, x_3 - i), \quad i = 0, \cdots, x_3; \quad \text{(A3)}$$

if $x_1 + x_2 + x_3 = N$, $x_1 = 0$, $x_2 > 0$ and $x_3 > 0$, then

$$\mathbf{x} \rightarrow (i, x_2, x_3 - 1 - i), \quad i = 0, \cdots, x_3 - 1 \quad \text{(A4)}$$

$$\mathbf{x} \rightarrow (i, x_2 + 1, x_3 - 1 - i), \quad i = 0, \cdots, x_3 - 1 \quad \text{(A5)}$$

$$\mathbf{x} \rightarrow (i, x_2 - 1, x_3 - i), \quad i = 0, \cdots, x_3 \quad \text{(A6)}$$

and

$$(0, 0, N) \rightarrow (i, 0, N - 1 - i), \quad i = 0, \cdots, N - 1 \quad \text{(A7)}$$

$$(0, 0, N) \rightarrow (i, 1, N - 1 - i), \quad i = 0, \cdots, N - 1 \quad \text{(A8)}$$

$$(0, N, 0) \rightarrow (0, N - 1, 0). \quad \text{(A9)}$$

Let $\mathbf{x} \xrightarrow{r} \mathbf{y}$ denote that state $\mathbf{y} \in \Omega_N$ is reachable from state $\mathbf{x} \in \Omega_N$ in a finite number of transitions.

Lemma 3 For any $\mathbf{y} \in \Omega_N$, $(0, 0, N) \xrightarrow{r} \mathbf{y}$.

Proof From (A7),

$$(0, 0, N) \xrightarrow{r} (i, 0, N - 1 - i), i = 0, \cdots, N - 1 \quad \text{(A10)}$$

From (A1), $(i, 0, N - 1 - i) \rightarrow (i, 0, N - i)$, $i = 0, \cdots, N - 1$, and $(N - 1, 0, 0) \rightarrow (N, 0, 0)$. Thus, using (A10),

$$(0, 0, N) \xrightarrow{r} (i, 0, N - i), i = 0, \cdots, N. \quad \text{(A11)}$$

From (A3), $(i + 1, k - 1, N - k - i) \rightarrow (i, k, N - k - i)$, $k = 2, \cdots, N$, $i = 0, \cdots, N - k$. Thus, starting with (A8) it follows that

$$(0, 0, N) \xrightarrow{r} (i, k, N - k - i), k = 1, \cdots, N, \; i = 0, \cdots, N - k. \quad \text{(A12)}$$

From (A2), $(i + 1, k, N - 1 - k - i) \rightarrow (i, k, N - 1 - k - i)$, $k = 1, \cdots, N - 1$, $i = 0, \cdots, N - 1 - k$. Thus,

$$(0, 0, N) \xrightarrow{r} (i, k, N - 1 - k - i), k = 1, \cdots, N - 1, \; i = 0, \cdots, N - 1 - k. \quad \text{(A13)}$$

The triples on the right side of (A10)-(A13) exhaust the members of Ω_N. Hence, the proof is complete.

Lemma 4 For any $\mathbf{y} \in \Omega_N$, $\mathbf{y} \xrightarrow{r} (0, 0, N)$.

Proof From (A1), (A2) and (A6) it follows that

$$(i, k - i - 1, N - k) \rightarrow (i, k - i - 1, N - k + 1), \; k = 1, \cdots, N, i = 0, \cdots, k - 1 \quad \text{(A14)}$$

$$(i, k - i, N - k) \rightarrow (i - 1, k - i, N - k), \; k = 1, \cdots, N, i = 1, \cdots, k \quad \text{(A15)}$$

$$(0, k, N - k) \rightarrow (0, k - 1, N - k), \; k = 1, \cdots, N - 1. \quad \text{(A16)}$$

First using (A14)-(A16) for $k = 1$, then using (A14)-(A16) for $k = 2, \cdots$, then using (A14)-(A16) for $k = N - 1$, then using (A14) and (A15) for $k = N$ and finally using (A9) yields

$$(i, k - i - 1, N - k) \xrightarrow{r} (0, 0, N), k = 1, \cdots, N, \; i = 0, \cdots, k - 1 \quad \text{(A17)}$$

$$(i, k - i, N - k) \xrightarrow{r} (0, 0, N), k = 1, \cdots, N, \; i = 0, \cdots, k. \quad \text{(A18)}$$

The triples on the left side of (A17) and (A18) exhaust all members of Ω_N except $(0, 0, N)$, which was shown to be reachable from itself in Lemma 3. Hence, the proof is complete.

It follows directly from Lemmas 3 and 4 that all states in $\mathscr{S}_N - \mathscr{T}_N$ communicate.

Appendix B

The expectations $E[\tau_2 - \tau_1]$ and $E[W_1(\tau_2) - W_1(\tau_1)]$ are computed here. The first step in the computations is to compute the stationary probability vector of the embedded Markov chain. Only states in the single irreducible class $\mathscr{S}_N - \mathscr{T}_N$ need be considered. As in Appendix A let Ω_N be the set of triples which represent the states in $\mathscr{S}_N - \mathscr{T}_N$, i.e.,

$$\Omega_N = \{\mathbf{x} = (x_1, x_2, x_3): x_1, x_2, x_3 \text{ are non-negative integers, } x_1 + x_2 + x_3 = N - 1 \text{ or } N\}.$$

Thus x_1 represents the number of customers in queue $q_{2,2}$ or receiving α_2 service, x_2 represents the number of customers in queue q_3 or receiving α_3 service, and x_3 represents the number of customers in queue for or receiving β service.

The transition probabilities for the states in Ω_N are straightforward to obtain. A few of these are derived below for illustrative purposes. Let $F_i(t) = \Pr\{a_i < t\}$, $i = 0, 2$ and let $F_{1j}(t) = \Pr\{a_1 < t | \psi_1 = j\}$, $j = 0, 1$. Recall that $p_j = \Pr\{\psi_j = 1\}$, $j = 1, 2$. Denote by $\lambda = 1/E[a_3]$ and

$\gamma = 1/\mathrm{E}[b]$ the rate parameters of the exponentially distributed random variables a_3 and b. Let $\sigma_n = \Sigma_{k=1}^{n} b_k$, $n = 1, 2, \cdots$, and $\sigma_0 = 0$. Let

$$p_i(n) = \int_0^\infty (1 - F_i(t)) \frac{\gamma^n t^{n-1} e^{-\gamma t}}{(n-1)!} dt,$$

$$n = 1, 2, \cdots, p_i(0) = 1, i = 0, 2,$$

$$p_{1j}(n) = \int_0^\infty (1 - F_{1j}(t)) \frac{\gamma^n t^{n-1} e^{-\gamma t}}{(n-1)!} dt,$$

$$n = 1, 2, \cdots, p_{1j}(0) = 1, j = 0, 1.$$

Denote by $p(\mathbf{x}, \mathbf{y})$ the probability of transition from state $\mathbf{x}$ to state $\mathbf{y}$.

Consider now an $\mathbf{x}$ such that $x_1 + x_2 + x_3 = N - 1$ and $x_3 > 0$, i.e., a state for which an α_1 service is about to begin and there is at least one β service to be performed. A state transition occurs at the completion of the α_1 service and the possible states at the completion of the α_1 service are $(x_1 + i, x_2, x_3 + 1 - i)$, $i = 0, 1, \cdots, x_3$ if $\psi_1 = 1$ and $(x_1 + 1 + i, x_2, x_3 - i)$, $i = 0, 1, \cdots, x_3$ if $\psi_1 = 0$, where i is the number of β services which complete during the α_1 service. Thus, it can be shown that

$$p(\mathbf{x},(x_1 + i, x_2, x_3 + 1 - i)) = \begin{cases} \Pr\{a_1 < \sigma_1 | \psi_1 = 1\} p_1, & i = 0 \\ \Pr\{\sigma_i < a_1 < \sigma_{i+1} | \psi_1 = 1\} p_i + \Pr\{\sigma_{i-1} < a_1 < \sigma_i | \psi_1 = 0\}(1 - p_1), & i = 1, \cdots, x_3 - 1 \\ \Pr\{\sigma_i < a_1 | \psi_1 = 1\} p_1 + \Pr\{\sigma_{i-1} < a_1 < \sigma_i | \psi_1 = 0\}(1 - p_1), & i = x_3 \\ \Pr\{\sigma_i < a_1 | \psi_1 = 0\}(1 - p_1), & i = x_3 + 1. \end{cases}$$

Hence, if $x_1 + x_2 + x_3 = N - 1$ and $x_3 > 0$, then

$$p(\mathbf{x}, (x_1 + i, x_2, x_3 + 1 - i)) = \begin{cases} p_1(1 - p_{11}(1)), & i = 0 \\ p_1(p_{11}(i) - p_{11}(i+1)) + (1 - p_1)(p_{10}(i-1) - p_{10}(i)), & i = 1, \cdots, x_3 - 1 \\ p_1 p_{11}(x_3) + (1 - p_1)(p_{10}(x_3 - 1) - p_{10}(x_3)), & i = x_3 \\ (1 - p_1) p_{10}(x_3), & i = x_3 + 1. \end{cases} \qquad \text{(B1)}$$

As a second example consider an $\mathbf{x}$ such that $x_1 + x_2 + x_3 = N$, $x_1 = 0$, $x_2 > 0$, $x_3 > 1$, i.e., a state for which an α_3 service is about to begin and there are at least two β services to be performed. A state transition occurs at the next α_2 service completion if a β service completes before the α_3 service completes [see Figure 7(a)] and the possible states at the α_2 service completion are $(i, x_2, x_3 - 1 - i)$ if $\psi_2 = 0$ and $(i, x_2 + 1, x_3 - 1 - i)$ if $\psi_2 = 1$, $i = 0, \cdots, x_3 - 1$, where i is the number of β services which complete during the α_2 service. A state transition occurs at the next α_0 completion if the α_3 service completes before a β service completes [see Figure 7(b)] and the possible states at the α_0 service completion are $(i, x_2 - 1, x_3 - i)$, $i = 0, \cdots, x_3$, where i is the number of β services which complete during the α_3 and α_0 services. Using Fig. 7(a) it can be shown that

$$p(\mathbf{x}, (i, x_2, x_3 - 1 - i)) = \Pr\{a_3 > \sigma_1\}(1 - p_2) \begin{cases} \Pr\{a_2 < \sigma_2 - \sigma_1\}, & i = 0 \\ \Pr\{\sigma_{i+1} - \sigma_1 < a_2 < \sigma_{i+2} - \sigma_1\}, & i = 1, \cdots, x_3 - 2 \\ \Pr\{\sigma_{i+1} - \sigma_1 < a_2\}, & i = x_3 - 1; \end{cases}$$

$$p(\mathbf{x}, (i, x_2 + 1, x_3 - 1 - i)) = p(\mathbf{x}, (i, x_2, x_3 - 1 - i)) p_2/(1 - p_2), \quad i = 0, \cdots, x_3 - 1.$$

Using Fig. 7(b) it can be shown that

$$p(\mathbf{x}, (i, x_2 - 1, x_3 - i)) = \begin{cases} \Pr\{a_3 + a_0 < \sigma_1\}, & i = 0 \\ \Pr\{a_3 < \sigma_1; \sigma_i < a_3 + a_0 < \sigma_{i+1}\}, & i = 1, \cdots, x_3 - 1 \\ \Pr\{a_3 < \sigma_1; \sigma_i < a_3 + a_0\}, & i = x_3. \end{cases}$$

It follows from the above after algebraic manipulation that if $x_1 + x_2 + x_3 = N$, $x_1 = 0$, $x_2 > 0$ and $x_3 > 1$, then

$$p(\mathbf{x}, (i, x_2, x_3 - 1 - i))/(1 - p_2) = p(\mathbf{x}, (i, x_2 + 1, x_3 - 1 - i))/p_2 = \gamma/(\gamma + \lambda) \begin{cases} p_2(i) - p_2(i+1), & i = 0, \cdots, x_3 - 2 \\ p_2(x_3 - 1), & i = x_3 - 1; \end{cases}$$

$$p(\mathbf{x}, (i, x_2 - 1, x_3 - i)) = \lambda/(\gamma + \lambda) \begin{cases} p_0(i) - p_0(i+1), & i = 0, \cdots, x_3 - 1 \\ p_0(x_3), & i = x_3. \end{cases} \qquad \text{(B2)}$$

The remaining non-zero transition probabilities are given as follows:

if $x_1 + x_2 + x_3 = N - 1$ and $x_3 = 0$, then

$$\left.\begin{aligned} p(\mathbf{x}, (x_1, x_2, 1)) &= p_1 \\ p(\mathbf{x}, (x_1 + 1, x_2, 0)) &= 1 - p_1 \end{aligned}\right\}; \tag{B3}$$

if $x_1 + x_2 + x_3 = N$, $x_1 > 0$ and $x_3 = 0$, then

$$\left.\begin{aligned} p(\mathbf{x}, (x_1 - 1, x_2, 0)) &= 1 - p_2 \\ p(\mathbf{x}, (x_1 - 1, x_2 + 1, 0)) &- p_2 \end{aligned}\right\}; \tag{B4}$$

if $x_1 + x_2 + x_3 = N$, $x_1 > 0$ and $x_3 > 0$, then

$$\begin{aligned} &p(\mathbf{x}, (x_1 - 1 + i, x_2, x_3 - i))/(1 - p_2) \\ &= p(\mathbf{x}, (x_2 - 1 + i, x_2 + 1, x_3 - i))/p_2 \\ &= \begin{cases} p_2(i) - p_2(i+1), & i = 0, \cdots, x_3 - 1 \\ p_2(x_3), & i = x_3; \end{cases} \end{aligned} \tag{B5}$$

if $\mathbf{x} = (0, N - 1, 1)$, then

$$\left.\begin{aligned} p(\mathbf{x}, (0, N-1, 0)) &= (1 - p_2)\gamma/(\gamma + \lambda) \\ p(\mathbf{x}, (0, N, 0)) &= p_2\lambda/(\gamma + \lambda) \\ p(\mathbf{x}, (0, N-2, 1)) &= (1 - p_0(1))\lambda/(\gamma + \lambda) \\ p(\mathbf{x}, (1, N-2, 0)) &= p_0(1)\lambda/(\gamma + \lambda) \end{aligned}\right\} \tag{B6}$$

if $\mathbf{x} = (0, 0, N)$, then

$$\begin{aligned} &p(\mathbf{x}, (i, 0, N - 1 - i))/(1 - p_2) \\ &= p(\mathbf{x}, (i, 1, N - 1 - i))/p_2 \\ &= \begin{cases} p_2(i) - p_2(i+1), & i = 0, \cdots, N - 2 \\ p_2(N - 1), & i = N - 1; \end{cases} \end{aligned} \tag{B7}$$

if $\mathbf{x} = (0, N, 0)$, then

$$p(\mathbf{x}, (0, N - 1, 0) = 1. \tag{B8}$$

The equations $\pi\ T = \pi$, $\Sigma_{\mathbf{x} \in \Omega_n} \pi_{\mathbf{x}} = 1$, where T is the $(N + 1)^2$ by $(N + 1)^2$ stochastic matrix whose non-zero entries are given by (B1)-(B8), can be solved numerically to obtain the unique stationary probability vector $\vec{\pi}$ of the embedded Markov chain.

The next step in the computations is to compute μ, the mean holding time in state $\mathbf{x} \in \Omega_N$ for the embedded semi-Markov process and $\mu'_{\mathbf{x}}$, the mean amount of α_1 service time during the holding time in state $\mathbf{x}$. It is simple to show that

$$\mu_{\mathbf{x}} = \begin{cases} \mathrm{E}[a_1], & \text{if } x_1 + x_2 + x_3 = N - 1 \\ \mathrm{E}[a_2], & \text{if } x_1 + x_2 + x_3 = N \text{ and } x_1 > 0 \end{cases}$$

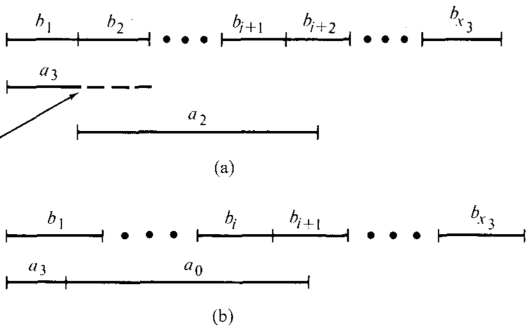

Figure 7 Transitions from state $\mathbf{x} = (0, x_2, x_3)$; $x_2 + x_3 = N$, $x_2 > 0$, $x_3 > 1$.

$$\mu_{\mathbf{x}} = \begin{cases} (1 + \lambda \mathrm{E}[a_0] + \gamma \mathrm{E}[a_2])/(\gamma + \lambda), & \text{if } x_1 + x_2 + x_3 = N,\ x_1 = 0,\ x_2 > 0 \text{ and } x_3 > 0 \\ \mathrm{E}[a_2] + 1/\gamma, & \text{if } \mathbf{x} = (0, 0, N) \\ \mathrm{E}[a_0] + 1/\lambda, & \text{if } \mathbf{x} = (0, N, 0); \end{cases} \tag{B9}$$

$$\mu'_{\mathbf{x}} = \begin{cases} \mathrm{E}[a_1], & \text{if } x_1 + x_2 + x_3 = N - 1 \\ 0. & \text{if } x_1 + x_2 + x_3 = N. \end{cases} \tag{B10}$$

Finally,

$$\mathrm{E}[\tau_2 - \tau_1] = \sum_{\mathbf{x} \in \Omega_N} \pi_{\mathbf{x}} \mu_{\mathbf{x}} / \pi_{\mathbf{x}_0}, \tag{B11}$$

$$\mathrm{E}[W_1(\tau_2) - W_1(\tau_1)] = \sum_{\mathbf{x} \in \Omega_N} \pi_{\mathbf{x}} u'_{\mathbf{x}} / \pi_{\mathbf{x}_0}, \tag{B12}$$

where $\mathbf{x}_0 \in \Omega_N$ represents $\mathbf{v}_0 \in \mathscr{S}_N - \mathscr{T}_N$. Combining (1) and (B9)-(B12), it follows that

$$\begin{aligned} U_1 = \mathrm{E}[a_1]Z_1/(&\mathrm{E}[a_1]Z_1 + \mathrm{E}[a_2]Z_2 \\ &+ (1 + \lambda\mathrm{E}[a_0] + \gamma\mathrm{E}[a_2])Z_3/(\gamma + \lambda) \\ &+ (\mathrm{E}[a_2] + 1/\gamma)\pi_{(0,0,N)} + (\mathrm{E}[a_0] + 1/\lambda)\pi_{(0,N,0)}), \end{aligned}$$

where

$$Z_1 = \sum_{\mathbf{x}: x_1+x_2+x_3=N-1} \pi_{\mathbf{x}};$$

$$Z_2 = \sum_{\mathbf{x}: x_1+x_2+x_3=N,\, x_1>0} \pi_{\mathbf{x}};$$

$$Z_3 = \sum_{\mathbf{x}: x_1+x_2+x_3=N,\, x_1=0,\, x_2>0,\, x_3>0} \pi_{\mathbf{x}}.$$

Acknowledgment

The authors are indebted to J. Rodriguez-Rosell and W. G. Tuel for sharing their insights into the operation of IMS. The present work benefited from conversations with them about resource contention and processor scheduling in IMS, although this model does not apply directly to IMS.

References

1. *Information Management System/360, Version 2, General Information Manual*, Report GH20-0765, IBM Corporation, White Plains, NY, 1973.
2. L. Kleinrock, *Queuing Systems Volume II: Computer Applications*. John Wiley and Sons, Inc., New York 1976.
3. P. A. W. Lewis and G. S. Shedler, "A cyclic-queue model of system overhead in multiprogrammed computer systems," *J. ACM* **18**, 199 (1971).
4. S. S. Lavenberg, "Queueing Analysis of a Multiprogrammed Computer System Having a Multilevel Storage Hierarchy," *SIAM J. Comput.* **2**, 232 (1973).
5. F. Baskett, K. M. Chandy, R. R. Muntz and F. G. Palacios, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *J. ACM* **22**, 248 (1975).
6. D. P. Gaver, S. S. Lavenberg and T. G. Price, Jr., "Exploratory Analysis of Access Path Length Data for a Data Base Management System," *IBM J. Res. Develop.* 20, 449 (1976; this issue).
7. P. A. W. Lewis and G. S. Shedler, "Statistical Analysis of Non-stationary Series of Events in a Data Base System," IBM J. Res. Develop. 20, 465 (1976, this issue).
8. S. M. Ross, *Applied Probability Models with Optimization Applications*. Holden-Day, Inc., San Francisco 1970.
9. W. L. Smith, "Renewal theory and its ramifications," *J. Roy. Statist. Soc. Ser. B* **20**, 243 (1958).
10. S. S. Lavenberg and G. S. Shedler, "A Queueing Model of the DL/I Component of IMS," *Research Report RJ 1561*, IBM J. Res. Develop. 20, 465 (1976, this issue).

Received January 9, 1976

Dr. Lavenberg is located at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 and Dr. Shedler is located at the IBM Research Laboratory, 5600 Cottle Road, San Jose, CA 95193.