Discussed in this paper is the effect of floating buffering on the execution time of a program.

An analytic model of floating buffering is developed and discussed. It is shown that with use of the model it is possible to compute the run time of a program as a function of the number of floating buffers it uses.

A model of floating buffering

by L. J. Woodrum

In computing systems, it is a common occurrence for the execution of a program to result in the concurrent processing of several related input files. A set of input files is considered related when at least one unprocessed record from each file must be present in memory before the program can proceed. In this paper, we discuss the effect of floating buffering on the total time required for execution of the program. An analytic model of floating buffering is developed, and conclusions about the number of floating buffers needed are presented. It is shown that an analytic model can provide valuable insight into a process, even though simplifying assumptions must be made in the model.

We are concerned here with the reading of q input files only. The model that is developed is appropriate for a program that causes the reading of q files from a single disk with one access arm or from a set of tapes on a single channel or any other situation where only one read at a time is permitted. It is assumed that the q files consist of blocks of records, and the number of records per block is the same for all q files, but the number of records in each of the files can vary. We assume that if output is produced, it does not interfere with the reading of the input files.

Using the model, we discover from computations that when one or two floating buffers are used, the program run time decreases sharply, and, as the number of buffers increases, the time asymptoti-

cally approaches the minimum possible. The effect of the number of floating buffers on the run time is principally a function of (1) the ratio of processing time per buffer to the time to read a buffer and (2) the coefficients of variation of the processing and reading times ($C(x) = \sigma(x) \div E(x)$). Other things being equal, the run time is slowest in approaching the minimum as the number of floating buffers is increased when the average central processing unit (CPU) processing time for a block is equal to the average read time for a block.

The following section begins by contrasting floating buffering with double buffering, two methods of buffering that are commonly used for input. On this basis, the analytic model is built and discussed.

Note that all subscripting and indexing in this paper is zero-origin, i.e., if A is a vector, A[0] is its first element.

Two buffering methods

double buffering

A common buffering method for input is called *double buffering*. In this method, a contiguous amount of main memory space capable of storing two blocks is allocated for each of the q input files. Processing of the records in a block is done one by one in the buffer area, and when all records in an input buffer have been processed, a read of the corresponding input file is initiated into the just-emptied buffer. While this read is in progress, processing can continue on the other buffer for that input file. The space required for double buffering is $2qbl$, where b is the number of records per block and l is the length of a record.

floating buffering initialization

Another method called *floating buffering* differs from double buffering in that a contiguous amount of space in main memory capable of storing a single block of records is allocated for each of the q input files, and f extra input buffers are used as well. There are $q + f$ input buffers altogether, and each buffer is exactly big enough to hold one block of records. The space required for floating buffering is $bl(q + f)$, where b is the number of records in a block and l is the length of a record. Initially, q buffers are filled by reading the first block from each of the q input files. Then, the program determines which of the q input buffers will empty first (if possible). A read is initiated into one of the f extra buffers from the file corresponding to the buffer that will empty first, while processing continues on the q buffers initially filled. If the q input files are being processed based on a collating sequence of keys present in the records, as is the case in merging while sorting, then the file to be read next can be determined by examining the last (highest) key present in memory for each of the q files, and finding the smallest of these. The file with the smallest last key is the file to be read next.

It is assumed that whenever a read is done, it is from the file that has been determined in this way. This assumption is an essential part of the model.

When the read of the first of the f buffers has been completed, the program again determines (if possible) which input file to read by an examination of the contents of q of the $q + 1$ nonempty buffers. Reading may continue in this fashion until there are $q + f$ nonempty buffers in main memory.

When a buffer has been emptied by processing all the records it contains, the empty buffer is exchanged for a full one, which has been filled in advance by reading using one of the f extra buffers. The empty buffer now becomes one of the f extra buffers, and the full buffer becomes one of the q buffers being processed. This situation is represented pictorially in Figure 1.

Following are definitions of terms found in Figure 1 and in later discussions:

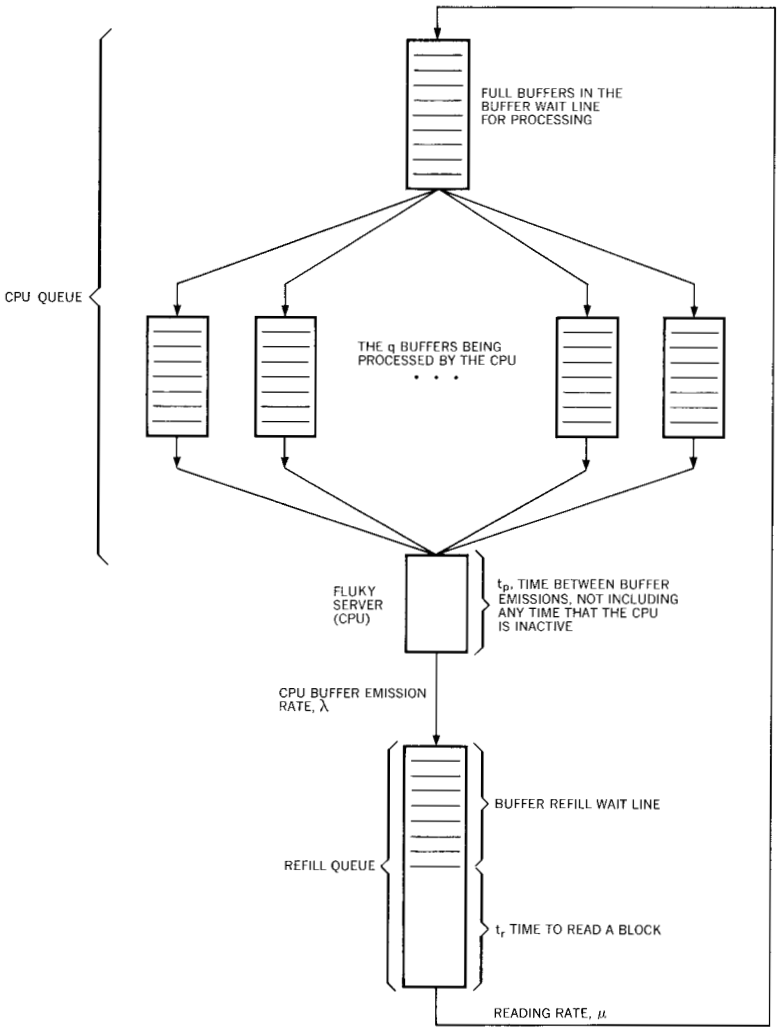
- λ — the average arrival rate (in buffers per second) of buffers from the CPU queue to the refill queue, given that the CPU is processing. When the CPU is processing, it emits (empties) buffers at the rate of λ buffers per second.
- μ — the average arrival rate (in buffers per second) of buffers from the refill queue into the CPU queue, given that there are buffers in the refill queue.
- t_r — the time for a read to take place. The random variable t_r is measured from the instant the seek is initiated to read into the buffer to the instant the read has been completed.
- n_c — the number of buffers currently in the CPU queue.
- n_r — the number of buffers currently in the refill queue, including the one currently being read into.
- $q + f$ — the total number of buffers in the system. Note the identity $n_c + n_r = q + f$.
- t_p — the time spent in the processing state between buffer emissions.

The CPU is always in one of two states here—it is either waiting for a read to be completed or not. If it is not waiting for a read to be completed, then it is actively processing records; hence, this state is called the processing state. If it is waiting for a read to be completed, it is not actively processing records; this state is called the nonprocessing state. For brevity we say that the CPU is processing when it is in the processing state, and say it is not processing when it is in the nonprocessing state. The term "CPU time" as used in this paper is the time spent in processing.

The expected processing time for the CPU to emit a buffer is

$$E(t_p) = \frac{1}{\lambda}$$

Figure 1 A model of floating buffering



model of floating buffering

In Figure 1, the processing of and reading into a buffer is represented by the CPU queue and the refill queue. The CPU queue consists of q nonempty buffers being concurrently processed (emptied) by the CPU and a wait line of full buffers waiting to be processed, whereas the refill queue consists of empty buffers being filled by reading. Initially, q buffers are filled by reading and join the CPU queue. The input files are related, and the CPU cannot continue processing until there are at least q nonempty buffers. The other f buffers join the refill queue.

The CPU is a "fluky" server in its processing of the q buffers. It determines, in a way analogous to flipping a q-sided die, from which buffer to select the next record to process. When a buffer is emptied

by this procedure (all records in it are processed), then the buffer is emitted and joins the refill queue.

If the CPU empties a buffer before another is ready to take its place (none are waiting in the buffer wait line for processing), processing stops until the completion of the needed read. Then the new buffer joins the CPU queue, the empty one joins the refill queue, and processing continues. At no time is the number of nonempty buffers (the number in the CPU queue) allowed to be less than $q - 1$. When the number of buffers in the CPU queue, n_c, is greater than or equal to q, the CPU continues processing, and when $n_c = q - 1$, processing stops while reading is completed.

Because of the way in which the file to be read is chosen in the refill queue, whenever $n_c = q - 1$, the file currently being read is the one from which records are needed to continue processing, and processing can resume as soon as the current read is completed. This is easy to see, because if no record is present in memory from a file, then the last record processed for that file was less than or equal to the last record present in memory for any of the other $q - 1$ files. But if a read is in progress, and no record is in memory from this file, then the read must be from this file because that is exactly the way to determine which file to read.

Note that this model differs from the classic problem of servicing machines in two basic respects:[1]

- In the machine repair model, the arrival rate of broken machines to the repair queue is dependent on how many machines are up and running, whereas in this model the arrival rate of empty buffers (corresponding to the broken machines) to the refill queue (corresponding to the repair queue) is always either λ or zero.
- In the machine repair model, the number of machines not broken can be 0, 1, 2, $\cdots$, m, where m is the number of machines. In this model, the number of nonempty buffers must always be at least $q - 1$ and must be at least q for the arrival rate of buffers to the refill queue to be nonzero.

Mathematical model

With the system of the two queues in Figure 1 in mind, we describe the way in which the total run time of the program can be determined, after first describing a way of looking at the problem in which the run time *cannot* be calculated.

pitfalls

We might think that if the average time for a buffer to go through both queues (the cycle time of the system) is known, then the total run time can be determined by multiplying the total number of

reads (buffer fillings) by the cycle time. This is not the right answer, which is obvious from the trivial example of exactly two files and three buffers. Suppose that the read time, t_r, in the read queue always takes a constant one second, and the time in the CPU queue also takes a constant one second, then the cycle time for the system is two seconds. Because the read time is always overlapped with the CPU time in the CPU queue, buffers can be considered to actually go through the system at the rate of one buffer per second.

Markov chain

We can determine the total run time by examining certain probabilities associated with a Markov chain and a semi-Markov process. The reader already familiar with these may skip immediately to the section on states in the semi-Markov process. Briefly, a Markov chain is a collection of s states and the probabilities associated with directed lines connecting the states.[2,3] A Markov chain is easiest to visualize as a directed graph, where each point is a state, and the lines are directed from state to state. Imagine a particle wandering around in this network. It can go from any specified point to another point along a directed line. If a point has more than one line emanating from it, then each of the lines has associated with it the probability that the particle will choose that line when departing from the point. The sum of the probabilities on the lines emanating from a given point must be one, since the particle always picks one of the lines when departing from the point.

A step in a Markov chain is the transition of the particle from one state to another. The process is said to be in a state i from the instant the particle enters state (point) i until it enters another state j. State i may be the same as j if there is a directed line from i to i. If we assume that each step takes some period of time, then we may talk about the time it takes to go from state to state, as well as the number of steps it takes. It is customary to label the s states $0, 1, 2, \cdots, s-1$, as all subscripting is done in zero-origin. We can represent a Markov chain by a matrix of transition probabilities, P, where $P[i; j]$ is the probability that when the system is in state i it will go to state j next. From the P matrix it is possible to calculate the various statistical moments of the number of steps required to go from any state i to any state j. In a Markov chain, it is usually assumed that it takes a constant unit of time to go from one state to another.

Of special interest is the probability that during a step in the process the system will be in any state i after a long time has passed (the process has reached a steady state, where the probability of being in any state during a step is independent of the state in which the process was initially). If π is a vector such that π_i is the steady-state probability of being in state i at an arbitrary step in the process, then the steady-state probability of being in state i in any step is the same as the steady-state probability that it will be in state i on the next step. This leads to the matric equation $\pi P = \pi$, which the π vector must satisfy.

The vector π may be computed using the following four-line derivation, where I is the identity matrix, U is the matrix of all ones, $\underline{I} = U - I$, and $\mathbf{u}$ is the row vector of all ones.

$$\pi P = \pi \tag{1}$$

$$\pi U = \mathbf{u}, \qquad \text{because } \sum_i \pi_i = 1. \tag{2}$$

By adding Equations 1 and 2 together, we get

$$\pi P + \pi U = \pi + \mathbf{u} \tag{3}$$

and, factoring, we get

$$\pi(P + U - I) = \mathbf{u}, \qquad \text{or} \qquad \pi = \mathbf{u}(P + \underline{I})^{-1} \tag{4}$$

Thus by adding up the columns of the inverse, we get π. Most of the other calculations are also just as easily obatined.

semi-Markov processes

A semi-Markov process (hereafter abbreviated as SMP) is essentially the same as a Markov chain, except that the time for the particle to traverse a line is a random variable, and each of the lines can have its own *arbitrary* statistical distribution for the time it takes for the particle to cross a line. In an SMP, we are more interested in the statistical moments of the *time* required to go from a state i to a state j than the number of *steps*. Therefore, it is customary to define a matrix of the first moments of the time to traverse each line (i, j), called $T1$, so that $T1[i; j]$ is the first moment of the time to traverse the line (i, j). Similarly, matrices $T2$, $T3$, etc., may be used to record second, third, etc., moments. These matrices are called the matrices of the statistical moments of *conditional* holding times. Analogous to the π vector is a vector $\underline{\mathbf{P}}$, where $\underline{\mathbf{P}}_i$ is the steady-state probability of being in state i at an arbitrary instant in time after all initial effects have worn off (i.e., independent of the state it started in).

For a full discussion of semi-Markov processes, see Barlow and Proscham[3] and also Appendix A, where methods of computing the various items in an SMP are discussed.

The semi-Markov process

state space

For the model of floating buffers, we define a collection of states labeled 0, 1, 2, $\cdots$, $s - 1$, so that the process is in state 0 when no reading is taking place and is in state i, providing $i \neq 0$, when a read is in progress that was started when exactly i of the buffers were in the refill queue. When, for example, the process is in state 1, it means that a read is in progress that started when there was exactly one buffer in the refill queue. If no other buffers enter the refill queue during this read, the process goes to state 0, meaning there are no buffers in the refill queue. If a read starts with one buffer in the refill queue, then the process remains in state 1 until the read

is finished, no matter how many buffers join the refill queue while the read is in progress. If a read starts with two buffers in the refill queue and four buffers join the refill queue while the process is in state 2, the process remains in state 2 until the read is completed, next enters state 5, and another read is started. State 5 is then held until the new read is completed.

A transition from state 0 to a nonzero state is allowed as soon as the CPU has a buffer empty and ready to be refilled. This means that state 0 can only change to state 1. For analysis involving disks or rotary devices this is a simplified treatment of the transition from state 0, but for analysis of reading tapes, or devices where no rotational delay is present, this is a reasonable way to define the transition from state 0. (See Appendix B, item 2, for further discussion of this point.)

number of states needed

The states in the process correspond to the number of buffers in the refill queue when a read is started. Then if there are f floating buffers, there cannot be more than f buffers in the refill queue when a read begins. Since we allowed state 0 to only go to state 1, when a transition to state f occurs, it must always be just after a read is completed, and whenever a read is completed, the CPU is able to start processing immediately as mentioned earlier. Since the CPU can always start processing when state f is entered, there must be exactly q buffers in the CPU queue. But if there are a total of $q + f$ buffers, and q of these are in the CPU queue, there can only be f of them in the refill queue when the next read is started. Hence, we may take the s states to be $0, 1, 2, \cdots, f$, where $s = f + 1$. Observe that q is not present in the model, as the effect of floating buffers does not depend on q when q is sufficiently large. Accordingly, we have dequeued our model.

probability transition matrix P

Let P be an s by s probability transition matrix, where an entry $P[i; j]$ in row i and column j is the probability that when the process is in state i it will next be in state j. Since there are i buffers in the refill queue when a read starts, and one buffer leaves the refill queue when the read is finished, the number of buffers in the refill queue at the completion of the read is $(i - 1)$ plus the number that join the refill queue during the read. If a transition to state j is made, then $1 + j - i$ buffers have joined the refill queue during the read. Therefore, the probability that when the system is in state i it goes to j next is also the probability that exactly $1 + j - i$ buffers join the refill queue during a read. This is only true if $i \neq 0$ because from state 0 it always goes to state 1.

The entries in the probability transition matrix P depend upon the statistical distributions of the CPU processing time for a buffer and the time to read a buffer of records. We will here assume that the inter-emission processing time, t_n, for buffers to be emptied and join the refill queue has a negative exponential distribution,

with mean $1 \div \lambda$. For the arguments justifying this assumption, see Appendix B. This assumption depends on the fact that the distribution of the superposition of a number of renewal processes can be estimated by the negative exponential distribution. This estimate is easily made precise; see Reference 4, and also see Reference 3, pages 18–22, for the exponential as the failure law of complex equipment.

refill time distribution

We take the read time, t_r, to be a discrete random variable, given by the two vectors $\mathbf{R}$ and $\mathbf{H}$, where the probability that $t_r = \mathbf{R}_k$ is $\mathbf{H}_k$. $\mathbf{R}$ is the vector of possible values of t_r, and $\mathbf{H}$ is the corresponding vector of the probabilities that t_r will assume the values in $\mathbf{R}$. Since the completion of a read can only occur at the end of a revolution when reading full-track records from a disk, the seek plus read time *must* be a multiple of one rotation time. Since a seek and read is started at the end of another seek and read if there is another read request pending, it makes sense to restrict the values of t_r to multiples of a full revolution. Note that the division of a refill operation into seek, wait for one-half a revolution (average), and read will not yield correct results when reading full-track records. For example, if seek time were always distributed uniformly in the range of 25 to 35 milliseconds, three full revolutions would be needed for every full-track record, whereas using seek plus wait plus read would yield $25 + 30 + 12.5 = 67.5$ milliseconds instead of 75.

When records are being read that are not full-track records, this must be modified to always be a multiple of the rotational time to pass over one record. The exact values for $\mathbf{H}$ and $\mathbf{R}$ are not, generally, easy to obtain and must depend on the particular application.

With these assumptions, the entries in the probability transition matrix P for the model are given by

$$P[i; j] = 0 \quad \text{if} \quad i = 0, \quad j \neq 1 \tag{5a}$$

$$P[i; j] = 1 \quad \text{if} \quad i = 0, \quad j = 1 \tag{5b}$$

$$P[i; j] = 0 \quad \text{if} \quad i > j + 1 \tag{5c}$$

$$P[i; j] = \sum_k \mathbf{H}_k e^{-\lambda \mathbf{R}_k} \frac{(\lambda \mathbf{R}_k)^{1+j-i}}{!(1 + j - i)}, \quad \text{if} \quad i \neq 0, \quad i \leq j + 1 \leq f \tag{5d}$$

$$P[i; j] = 1 - \sum_{k=0}^{f-1} P[i; k], \quad \text{if} \quad i \neq 0, \quad j = f \tag{5e}$$

Equations 5a through 5c are justified in the previous discussion; Equation 5d is an application of the fact that the Poisson distribution is the distribution function of the number of arrivals in a given period of time when the interarrival time has the negative exponential distribution; and Equation 5e expresses the fact that the last entry in each row of P is set to one minus the sum of the other entries.

Equation 5e is justified because the emission of buffers continues as a Poisson process until emission stops and resumes as a Poisson process exactly as if no interruption had occured. Besides, probabilities must sum to one by convention.

matrices of conditional holding time

Next, we define a matrix $T1$, which is an s by s matrix containing the first moments of conditional holding time for each of the states. $T1[i;j]$ is the first moment of the time spent in state i, given that a transition to state j will be made next. In general, it is not necessary for any of the $T1$ entries to be the same in an SMP; they can be completely different for each (i, j) pair of states. $T1$ is called the matrix of first moments of *conditional* holding times because the holding time can depend on the next state. For this model, the $T1$ elements are given by

$$T1[i; j] = 0 \quad \text{if} \quad P[i; j] = 0 \tag{6a}$$

$$T1[i; j] = \frac{1}{\lambda} \quad \text{if} \quad i = 0, \quad j = 1 \tag{6b}$$

$$T1[i; j] = \sum_k \mathbf{H}_k \mathbf{R}_k \quad \text{elsewhere} \tag{6c}$$

Similarly, we define matrices $T2$ and $T3$ to be matrices of the second and third moments of conditional holding time. $T2$ and $T3$ are given by

$$T2[i; j] = 0 \quad \text{if} \quad P[i; j] = 0 \tag{7a}$$

$$T2[i; j] = \frac{2}{\lambda^2} \quad \text{if} \quad i = 0, \quad j = 1 \tag{7b}$$

$$T2[i; j] = \sum_k \mathbf{H}_k \mathbf{R}_k^2 \quad \text{elsewhere} \tag{7c}$$

$$T3[i; j] = 0 \quad \text{if} \quad P[i; j] = 0 \tag{8a}$$

$$T3[i; j] = \frac{6}{\lambda^3} \quad \text{if} \quad i = 0, \quad j = 1 \tag{8b}$$

$$T3[i; j] = \sum_k \mathbf{H}_k \mathbf{R}_k^3 \quad \text{elsewhere} \tag{8c}$$

run-time computation

We are now in a position to compute the execution time for a program using floating buffering. The total execution time for the program is equal to the time spent reading all the blocks of input data, plus the time spent with the CPU processing and no reading taking place. State 0 in the SMP is the only state where reading is not taking place, and $\underline{P}_0$ is the fraction of the total execution time that is spent in state 0. Then $(1 - \underline{P}_0)$ is the fraction of total execution time spent reading, so that if N blocks are read we have

$$\textit{Runtime}\ (1 - \underline{P}_0) = N(E(t_r)) \tag{9}$$

Clearly,

$$\textit{Runtime} = \frac{N\underline{E}(t_r)}{1 - \underline{P}_0} \tag{10}$$

With the model now complete, we may compute run times, using the APL program EVL shown in Figure 2. (EVL uses the supporting function SMP, described in Appendix A.) EVL produces a matrix result Z that has four rows and nine columns. Figure 3 contains sample executions of EVL. Row 0 of Z is an index of the number of floating buffers used, e.g., $Z[0; 2]$ is a 2, meaning the other numbers in column 2 are the results of using two floating buffers.

Row 1 of Z contains the run times, in seconds, when the indicated number of floating buffers is used. The last entry in row 1, $Z[1; 8]$, is the minimum achievable run time (which occurs only if processing and reading are overlapped to the maximum extent possible).

Row 2 of Z is obtained from row 1 of Z by dividing row 1 by $Z[1; 8]$. This expresses the run time for each choice of floating buffers in terms of the minimum achievable run time. Thus, if $Z[2; 3] = 1.076$, using three floating buffers results in a run time that is 1.076 times as long as the minimum possible time. Since $Z[2; 8]$ would always be 1, $Z[2; 8]$ instead contains the expected value of buffer refill time, $E(t_r)$.

Row 3 of Z contains the fraction of recoverable time remaining when the indicated number of buffers is used. By recoverable time we mean the difference between the actual run time and the minimum achievable run time. When zero floating buffers are used, then the run time is the sum of the processing and refilling times. The minimum achievable time is the maximum of the processing and refilling times. The difference between run time using zero floating buffers and the minimum achievable is the maximum recoverable run time. By dividing the recoverable run time for i floating buffers by the maximum recoverable run time, we get the entry in column i, row 3 of Z, $Z[3;i]$. For example, if $Z[3;3] = 0.091$, then 91.9 percent of the recoverable time is recovered by using only three floating buffers, and 9.1 percent remains. Since $Z[3;8]$ would always be zero, $Z[3;8]$ instead contains the coefficient of variation of buffer refill time, t_r. The coefficient of variation of a random variable t, $C(t)$, is defined by $C(t) = \sigma(t) \div E(t)$.

To show how the coefficient of variation of refill time affects the output of the EVL program, several executions are shown in Figure 3, where $E(t_r)$, average refill time, is the same for all, and only $C(t_r)$ is different. Note how run time increases as the coefficient of variation increases.

The inputs to EVL are the vectors **R** and **H**, which give the refill distribution, and a three-element vector, **NBX**, containing the number of reads to be done during the run, the number of records per block, and the CPU processing time per record, respectively. In the examples in Figure 3, there are 1000 blocks, 50 records per block, and the processing time per record is 0.001667 second. This

Figure 2 APL program EVL, which computes run times using SMP

```
     ∇ Z←RH EVL NBX;H;N;B;X;A;I;R;K;L;PI;C;J;ET1;E1T;
       ET2;E2T;ET3;E3T;SS;SIG;SKW;T1;T2;T3;S;P;Q;R
[1]    R←(ρH←(0.5×ρRH)↓RH)↑RH
[2]    L←1↓C←(B←NBX[1])×X←NBX[2+0×N←NBX[0]]
[3]    Z← 2 9 ρ0×S←2
[4]    P←(SS←S,S)↑0⌈(-J)∘.+1+J←⍳S
[5]    P←(J∘.≤1+J)×((H××-L×R)+.×(L×R)∘.*P)÷!P
[6]    P[0;]←1=J
[7]    P[;S-1]←1-+/ 0 ¯1 ↓P
[8]    T1←SS↑ ¯1 ¯2 ↑C
[9]    T1[K←1+J;]←A←H+.×R
[10]   T2←SS↑ ¯1 ¯2 ↑2×C×C
[11]   T2[K;]←H+.×R×R
[12]   T3←SS↑ ¯1 ¯2 ↑6×C*3
[13]   T3[K;]←H+.×R*3
[14]   PRT SMP 0
[15]   Z[;S-1]←(S-1),(N×PI+.×C,(S-1)ρA)÷1-PI[0]
[16]   → 17 4[8≥S←S+1]
[17]   Z[;0]←0,N×C+A
[18]   Z[;8]←0,N×C⌈A
[19]   Z← 1 1 0 0 ⍉Z
[20]   Z[2;]←Z[1;]÷Z[1;8]
[21]   Z[3;]←(Z[1;]-Z[1;8])÷-/Z[1; 0 8]
[22]   Z[3;8]←(((H+.×R*2)-A*2)*0.5)÷Z[2;8]←A
     ∇
```

Figure 3 Examples of executions of EVL

```
     PRT←0
     ⍝ EXAMPLE 0, COEFFICIENT OF VARIATION IS ZERO.
     3 EDT .07 1 EVL 1000 50 .001667

   .000    1.000    2.000    3.000    4.000    5.000    6.000    7.000     .000
153.350  105.989   94.380   89.706   87.330   85.960   85.110   84.557   83.350
  1.840    1.272    1.132    1.076    1.048    1.031    1.021    1.014     .070
  1.000     .323     .158     .091     .057     .037     .025     .017     .000

     ⍝ EXAMPLE 1, C=0.214
     3 EDT .025 .075 .1 .9 EVL 1000 50 .001667

   .000    1.000    2.000    3.000    4.000    5.000    6.000    7.000     .000
153.350  106.679   94.892   90.075   87.606   86.170   85.272   84.684   83.350
  1.840    1.280    1.138    1.081    1.051    1.034    1.023    1.016     .070
  1.000     .333     .165     .096     .061     .040     .027     .019     .214

     ⍝ EXAMPLE 2, C=0.385
     3 EDT .025 .05 .075 .1 .1 .4 .1 .4 EVL 1000 50 .001667

   .000    1.000    2.000    3.000    4.000    5.000    6.000    7.000     .000
153.350  107.908   95.930   90.872   88.220   86.650   85.650   84.983   83.350
  1.840    1.295    1.151    1.090    1.058    1.040    1.028    1.020     .070
  1.000     .351     .180     .107     .070     .047     .033     .023     .385

     ⍝ EXAMPLE 3, C=0.525
     3 EDT .025 .1 .4 .6 EVL 1000 50 .001667

   .000    1.000    2.000    3.000    4.000    5.000    6.000    7.000     .000
153.350  109.767   97.329   91.889   88.988   87.247   86.129   85.360   83.350
  1.840    1.317    1.168    1.102    1.068    1.047    1.033    1.024     .070
  1.000     .377     .200     .122     .081     .056     .040     .029     .525

     ⍝ EXAMPLE 4, C=0.711
     3 EDT .025 .125 .55 .45 EVL 1000 50 .001667

   .000    1.000    2.000    3.000    4.000    5.000    6.000    7.000     .000
153.350  112.335   99.570   93.635   90.358   88.343   87.012   86.089   83.350
  1.840    1.348    1.195    1.123    1.084    1.060    1.044    1.033     .070
  1.000     .414     .232     .147     .100     .071     .052     .039     .711

     ⍝ EXAMPLE 5, C=1.286, AN UNREALISTIC SITUATION FOR A DISK.
     3 EDT .025 .250 .8 .2 EVL 1000 50 .001667

   .000    1.000    2.000    3.000    4.000    5.000    6.000    7.000     .000
153.350  120.231  108.206  101.200   96.690   93.651   91.512   89.942   83.350
  1.840    1.442    1.298    1.214    1.160    1.124    1.098    1.079     .070
  1.000     .527     .355     .255     .191     .147     .117     .094    1.286
```

results in a CPU processing rate of $\lambda = 50 \times 0.001667 = 0.08335$ buffer per second. In Example 2 of Figure 3, $\mathbf{R} = (0.025, 0.05, 0.075, 0.1)$ and $\mathbf{H} = (0.1, 0.4, 0.1, 0.4)$. The vectors $\mathbf{R}$ and $\mathbf{H}$ are catenated to form one vector, which is the left operand of EVL.

By examining the results of many executions of EVL, we found that, in general, the run time depends primarily upon the ratio of CPU speed to input/output speed, the number of floating buffers, and the coefficient of variation of the refill time t_r. However, these three parameters do not always result in the same run time. We have refrained from including a table of run times based on these parameters because it might mislead one into believing that run time depends solely on these parameters. It must be emphasized that there does not appear to be a simple way to calculate run times without using matrix inversion to solve for the vector π in the Markov chain. For a small number of floating buffers, it is possible to obtain formulas for the run time by symbolically inverting the $P + \underline{I}$ matrix. This is a mere exercise in algebra and is left to the reader.

use of semi-Markov process

More information about the behavior of a program than just run times can be obtained by examining results of other computations made using the SMP. In the following discussion, we will use the notation described in Appendix A.

Since the program always starts in state f, it is of interest to determine how long the program would have to run before reaching any selected state of interest. If we found, for example, that the expected time for the program to run before reaching state 0 from the initial state f is greater than the run time of the program itself, we would rightly suspect that using f buffers would be unnecessary. This time, $E({}_1t_{f0})$, is calculated in the APL function SMP described in Appendix A.

Related to this is the probability of the program being in a state at any instant in time. This information is given by the vector $\underline{\mathbf{P}}$, and if we find probabilities which are very small, we can conclude that more buffers are used than necessary.

As an aside, note that if a program were written using a fixed number of buffers, and the probability of the refill queue being empty (or full) were so small that it would not be likely to appear in the testing of the program, then special care must be taken to ensure proper functioning for such a rare event. This is an interesting use of modeling in general—to determine when conditions are so rare as to merit special care in debugging a program.

Since the probability π_0 is crucial in computing average run times, it is of interest to know how long it takes from the time the system enters state 0 to the time the system subsequently returns to it,

and also to know the standard deviation of the return time. If the standard deviation of the first passage time from state 0 to state 0 is large relative to its mean ($C({}_1t_{00})$ is large), then we can expect the run time of the program to have a larger standard deviation as well. This would mean that the computed average run time might not correspond very often with actual run times. This standard deviation, $\sigma({}_1t_{00})$, is given by the SMP function in the vector **SIG**. See Appendix A for further discussion of these computations.

assumptions

No model is complete without a list of the assumptions and a discussion of their importance, so such a list is given in Appendix B. Of particular interest are the assumptions that the steady-state probabilities do in fact describe the behavior of the program, and that the arrival pattern of buffers to the refill queue is Poisson. When a program begins execution, it usually does so with all the buffers full from the first read of each of the q inputs. It will be awhile before any of them empties, and then they will all tend to empty together. This situation will recur until enough records have been processed for the reads to become more independent of the initial conditions. The wearing off of these initial conditions is not the same as the wearing off of the initial conditions described previously when discussing the steady-state probability vector π.

In the previous case, we talk about the steady-state independence of the state the process starts in, which is the state f, with the refill queue full. Now we are concerned with the use of the negative exponential distribution to describe the inter-emission time of buffers by the CPU. The negative exponential distribution does not describe very well the behavior of the program when it is first being executed but more and more closely describes its behavior as time goes on and more records are processed.

This is analogous to (but not *exactly* the same as) shuffling q decks of cards together in a q-way shuffle, where only the last card of each deck is marked, and then examining the resulting single deck for the marked cards. They will tend to all fall together at the end. Under repeated q-way shuffles, we would expect the distance between marked cards to be better approximated by the negative exponential distribution. Obviously, the negative exponential distribution can never describe this exactly, since intercard distances are discrete random variables, whereas the negative exponential distribution applies to continuous random variables.

As a consequence of this tendency for buffers to initially empty together, run times would be greater than the computations indicated. The effect of this clustering persists for some significant portion of the run if there are not very many blocks in each of the input files. How long this persists has not been investigated extensively, but we do know that it causes run time to increase. Here is a case where the physical situation does not correspond to

mathematical reality. What must clearly be done is to change the physical situation in such a way that it does correspond to mathematical reality.

As in the case of a deck of marked cards, if we would like to have a shuffled deck, then it is best to begin with a shuffled deck. This may be done by writing a random number of records in the *first* block only of each input file when the input files are created. For further discussion of this point, see Appendix B, item 5.

Summary comment

An analytic model of floating buffering has been obtained with which program run times can be computed. The run time per block was found to depend principally on the number of floating buffers, the ratio of CPU processing time per block to the input/output refill time per block, and the coefficient of variation of the input/output refill time.

The various results of computation using the SMP model provide a method of estimating the magnitude of the variability of run time, and even call attention in such a program to possible errors that might go undetected due to their infrequent occurrence. When this model was compared to an actual run of a program, the run time for the program was 42.1 seconds, and the computed run time was 39.1 seconds.[5] Probably, the computed run time was less than the actual run time because the run was not long enough for the negative exponential distribution to be a good approximation to the time between the emptying of buffers. The run times obtained using the model are reasonable estimates of the run times of actual programs. Other things being equal, as the number of floating buffers used is increased, the run time is slowest in approaching the minimum when the average CPU processing time for a block is equal to the average read time for a block.

The computations involved in the analysis of an SMP model are as easy to obtain as the vector π, provided one has the necessary familiarity with vectors and matrices. In the investigations of this model, the use of APL\360 for computations was especially useful because the language is well-suited for the vector and matrix operations needed. It seems likely that the use of stochastic models for analyzing computing situations will become a more widely used technique.

ACKNOWLEDGMENT

The author is grateful to Dr. J. E. Flanagan, without whom this paper would not have been written. The motivation to present an SMP model of floating buffering was inspired by Dr. Flanagan.

Appendix A: APL functions for analyzing semi-Markov processes

This appendix describes a set of three APL functions for obtaining information about a semi-Markov process.[6,7] The primary function, called SMP, is used when its inputs have been precalculated and are already defined when its execution starts. When optional input data is omitted, the corresponding global variable must be set to zero before execution of the APL function SMP. In what follows, the random variable t_{ij} is the time it takes for the process to go directly from a state i to a state j. The random variable ${}_1t_{ij}$ is the time it takes for the process to go from a state i to a state j for the first time by any path. The time ${}_1t_{ij}$ is measured from the instant state i is entered until the instant state j is subsequently entered for the first time. The SMP function computes the following information provided the appropriate input data is supplied:

PI An S-element vector giving the steady-state probability of being in each of the states during any step of the process after the effects of the initial conditions have worn off, where S is the number of states in the system. **PI**[i] is the steady-state probability of being in state i at any instant in time in the associated Markov chain after the effects of the initial conditions have worn off. **PI**[i] = π_i.

ET1 An S-element vector giving the first moments of the unconditional holding time for each of the states. **ET1**[i] = $\mathbf{E}(t_i)$, where the random variable t_i is the unconditional holding time in state i regardless of the state it goes to next. $\mathbf{E}(t_i) = \sum_k P_{ik}\mathbf{E}(t_{ik})$, where t_{ik} is the random variable representing the time to go *directly* from i to k along a directed line.

P An S-element vector giving the steady-state probability of being in each of the states at any instant in time in the SMP after the effects of the initial conditions have worn off. $\mathbf{P}_i = [\pi_i E(t_i)] \div [\sum_k \pi_k E(t_k)]$.

E1T An S-element vector giving the first moments of the time required to go from each of the states to a selected state, called j, for the first time. **E1T**[i] is the first moment of the time to go from any state i to j for the first time. **E1T**[i] = $E({}_1t_{ij})$, where ${}_1t_{ij}$ is the random variable representing the time to go from state i to state j for the first time by *any* path. The time ${}_1t_{ij}$ is measured from the instant the system enters state i to the instant the system subsequently enters state j for the first time. Thus **E1T** contains the first moments of the first passage times from all states i to the selected state j.

ET2 An S-element vector giving the second moments of unconditional holding time. **ET2**[i] is the second moment of the unconditional holding time for any state i.

E2T An S-element vector giving second moments of first passage time. **E2T**[i] is the second moment of the time to go from any state i to the selected state j for the first time.

Table 1 Summary of notation

$S \equiv S$, the number of states	$\mathbf{PI} \equiv \pi$	$\mathbf{SIG} \equiv \{\sigma({}_1t_{ij})\}$
$J \equiv j$, the selected state of interest	$\mathbf{ET1} \equiv \{E(t_i)\}$	$\mathbf{SKW} \equiv \left\{\frac{E[{}_1t_{ij} - E({}_1t_{ij})]^3}{\sigma^3({}_1t_{ij})}\right\}$
$P \equiv \{P_{ik}\}$	$\mathbf{ET2} \equiv \{E(t_i^2)\}$	$Q \equiv P$, except column j is all zeros
$\underline{P} \equiv \{P_i\}$	$\mathbf{ET3} \equiv \{E(t_i^3)\}$	$\underline{Q} \equiv \underline{IP} \equiv (I - Q)^{-1}$
$T1 \equiv \{E(t_{ik})\}$	$\mathbf{E1T} \equiv \{E({}_1t_{ij})\}$	
$T2 \equiv \{E(t_{ik}^2)\}$	$\mathbf{E2T} \equiv \{E({}_1t_{ij}^2)\}$	
$T3 \equiv \{E(t_{ik}^3)\}$	$\mathbf{E3T} \equiv \{E({}_1t_{ij}^3)\}$	

In every case, the braces, { }, are used to indicate a vector or matrix of the elements obtained by letting i and k range over the values from 0 to $S - 1$. Also, j is a fixed scalar.

SIG An S-element vector giving the standard deviation of the first passage times. **SIG**[i] is the standard deviation of the time to go from any state i to the selected state j for the first time.

ET3 An S-element vector giving the third moments of unconditional holding time for each of the states.

E3T An S-element vector giving the third moments of first passage time from each of the states to the selected state j.

SKW An S-element vector giving the skew of the first passage times from each of the states to the selected state j. The skew given is the normalized third central moment, i.e., the third moment with respect to the mean, divided by the cube of the standard deviation.

The following input data must be supplied to use the program:

J The state of special interest (see the previous descriptions of output).

P The S by S probability transition matrix. $P[i; k]$ is the probability that if the system is in state i, it will be in state k next.

$T1$ The S by S matrix of first moments of conditional holding time. $T1[i; k]$ is the first moment of the time required to go to state k from state i directly, given that the system is in state i and will be in state k next.

The following input data is optional and must be set to zero if not used:

$T2$ The S by S matrix of second moments of conditional holding time.

$T3$ The third moments of conditional holding time.

If $T2$ is not supplied, the program will not produce the outputs **ET2**, **E2T**, **ET3**, **E3T**, **SIG**, and **SKW**. If $T3$ is not supplied, the program will not produce the outputs **ET3**, **E3T**, and **SKW**. Table 1 is a summary of the notation.

using the functions

The mnemonics for the three functions are SMP for semi-Markov process, INV for matrix inversion, and EDT for editing or displaying data after it has been entered. The syntax of SMP is "PRT SMP J," where PRT is a zero if no typewriter display of the results of computations is desired, and PRT is a one if typed output is desired. J is the selected state of interest, as described earlier. In every case, SMP will leave its computed results as variables global to it with the names described above.

general notes

In the SMP function, there are three diagnostic messages. The first, on line 4 of SMP, "ROW SUMS OF P ARE BAD.", appears when the rows of P don't sum to ones. However, the function continues anyway since the difference may be caused by slight imprecisions in the calculation of the entries in P. Accompanying this message is a display of the actual row sums that are not ones, with the rows indicated in the display. Two other diagnostic messages may occur, on lines 8 and 12 of the SMP function, when the second or third moments are in error. If either of these messages occurs, it will exit by means of a right arrow (→) by itself because when this happens there is an error in the computation of the entries in one or more of the conditional holding time matrices.

If any problem should arise while executing these functions, such as running out of workspace, so that execution is suspended, an exit can be made completely from all function execution by entering a right arrow.

A variable can be displayed when no execution is taking place by typing its name or using the EDT function. The syntax for EDT is "f EDT M", where f is the number of fractional digits to be displayed and M is the name of the matrix or vector to be displayed. To display scalars, always do so by entering the name of the variable; do not use the EDT function. When displaying variables by name, or when data is typed from SMP, the number of digits displayed can be restricted by the system command ")DIGITS d". If this is entered before the display, only d significant digits will be displayed.

moments of first passage time

We now describe the way the statistical moments of first passage times, $E({}_1t_{ij}^n)$ are computed. The random variable ${}_1t_{ij}$ is the sum of a number of random variables t_{kh}, the times to go from states k to other states h, where k and h are states passed through on the way from i to j. With probability P_{ij}, the random variable ${}_1t_{ij}$ is equal to t_{ij}, since if the system goes directly from state i to state j, ${}_1t_{ij} = t_{ij}$. With probability P_{ik}, $k \neq j$, the system goes to some state k directly, and then from k to j for the first time. Thus with probability P_{ik}, ${}_1t_{ij} = t_{ik} + {}_1t_{kj}$. This means that

$$E({}_1t_{ij}) = P_{ij}E(t_{ij}) + \sum_{k \neq j} P_{ik}E(t_{ik} + {}_1t_{kj}) \text{ and also}$$

$$E({}_1t_{ij}^n) = P_{ij}E(t_{ij}^n) + \sum_{k \neq j} P_{ik}E(t_{ik} + {}_1t_{kj})^n$$

If $(t_{ik} + {}_1t_{kj})^n$ is expanded using the binomial theorem, then

$$(t_{ik} + {}_1t_{kj})^n = \sum_{x=0}^{n} \mathsf{C}_x^n t_{ik}^{n-x} {}_1t_{kj}^x$$

where C_x^n is the number of combinations of n things taken x at a time. We can then see that

$$E({}_1t_{ij}^n) = P_{ij}E(t_{ij}^n) + \sum_{k \neq j} P_{ik}E\left(\sum_{x=0}^{n} \mathsf{C}_x^n t_{ik}^{n-x} {}_1t_{kj}^x\right)$$

Since the expected value of the sum of a number of independent random variables is equal to the sum of their expected values, and all the random variables are independent of one another in the above sum that has x as the index of summation, we can write the expected value operation inside the summation sign. Then we get

$$E({}_1t_{ij}^n) = P_{ij}E(t_{ij}^n) + \sum_{k \neq j} P_{ik} \sum_{x=0}^{n} E(\mathsf{C}_x^n t_{ik}^{n-x} {}_1t_{kj}^x)$$

Now when $x = 0$, the summation term is $E(\mathsf{C}_0^n t_{ik}^n {}_1t_{kj}^0)$, which is $E(t_{ik}^n)$. Taking this term out of the summation, we let x go from 1 to n instead of from 0 to n, and get

$$E({}_1t_{ij}^n) = P_{ij}E(t_{ij}^n) + \left[\sum_{k \neq j} P_{ik}E(t_{ik}^n)\right] + \sum_{k \neq j} P_{ik} \sum_{x=1}^{n} E(\mathsf{C}_x^n t_{ik}^{n-x} {}_1t_{kj}^x)$$

Observe that the first summation is over $k \neq j$, and the term inside the summation, $P_{ik}E(t_{ik}^n)$, is the same as the first term in the above expression, $P_{ij}E(t_{ij}^n)$, except that k is present as a subscript instead of j. Since the first summation is for all $k \neq j$, we can put $P_{ij}E(t_{ij}^n)$ inside the range of the first summation and let k range over all values from 0 to $s - 1$, including j. Then we find that

$$E({}_1t_{ij}^n) = \left[\sum_{k=0}^{s-1} P_{ik}E(t_{ik}^n)\right] + \sum_{k \neq j} P_{ik} \sum_{x=1}^{n} E(\mathsf{C}_x^n t_{ik}^{n-x} {}_1t_{kj}^x)$$

The part of the expression in brackets, $\sum_k P_{ik}E(t_{ik}^n)$, is the expected value of the nth moment of the unconditional holding time in state i, because that is the definition of $E(t_i^n)$. The equation is then

$$E({}_1t_{ij}^n) = E(t_i^n) + \sum_{k \neq j} P_{ik} \sum_{x=1}^{n} E(\mathsf{C}_x^n t_{ik}^{n-x} {}_1t_{kj}^x)$$

When $x = n$, the term inside the range of the second summation sign is the last term in the binomial expansion of $E(t_{ik} + {}_1t_{kj})^n$, which is $E({}_1t_{kj}^n)$. Taking this last term out of the second summation, and letting x go from 1 to $n - 1$ instead of from 1 to n, we see that the equation becomes

$$E({}_1t_{ij}^n) = E(t_i^n) + \left[\sum_{k \neq j} P_{ik} \sum_{x=1}^{n-1} E(\mathsf{C}_x^n t_{ik}^{n-x} {}_1t_{kj}^x)\right] + \sum_{k \neq j} P_{ik}E({}_1t_{kj}^n)$$

Now if the matrix P is replaced in the expression by a matrix Q, where Q is formed from P by setting column j to all zeros, we can let k range over all values from 0 to $s - 1$, including j, since Q_{ij} is always zero. Doing this, we see the expression is

$$E({}_1t_{ij}^n) = E(t_i^n) + \left[\sum_k Q_{ik} \sum_{x=1}^{n-1} E(\mathsf{C}_x^n t_{ik}^{n-x} \, {}_1t_{kj}^x) \right] + \sum_k Q_{ik} E({}_1t_{kj}^n)$$

Suppose that j is held at a fixed value and we imagine that we have written the above equation s times, once for each possible choice of i. Then we get a system of s equations. These s equations can be represented by a single equation involving matrices and vectors. The term on the left side of the equation, $E({}_1t_{ij}^n)$, gives a column vector of s elements, one for each i from 0 to $s - 1$. Let the symbols $\{E({}_1t_{ij})\}$ stand for this column vector. Similarly the term $E(t_i^n)$ yields a column vector, represented by the symbols $\{E(t_i^n)\}$.

Looking at the last term, $\sum_k Q_{ik} E({}_1t_{kj}^n)$, we see that the summation over k is equivalent to taking the vector inner product of row i of Q with the vector $\{E({}_1t_{kj}^n)\}$, formed by letting k assume all values from 0 to $s - 1$. We can then write this term as $Q \overset{+}{\times} \{E({}_1t_{kj}^n)\}$, where $\overset{+}{\times}$ represents the usual matrix or vector product. This last term is the column vector that results from taking the inner product of each row of q with the vector $\{E({}_1t_{kj}^n)\}$.

In the term

$$\sum_k Q_{ik} \sum_{x=1}^{n-1} E(\mathsf{C}_x^n t_{ik}^{n-x} \, {}_1t_{kj}^x)$$

interchanging the order of summation gives the same answer, so that

$$\sum_k Q_{ik} \sum_{x=1}^{n-1} E(\mathsf{C}_x^n t_{ik}^{n-x} \, {}_1t_{kj}^x) = \sum_{x=1}^{n-1} \sum_k Q_{ik} E(\mathsf{C}_x^n t_{ik}^{n-x} \, {}_1t_{kj}^x)$$

The expected value of the product of independent random variables is equal to the product of their expected values, thus

$$\sum_{x=1}^{n-1} \sum_k Q_{ik} E(\mathsf{C}_x^n t_{ik}^{n-x} \, {}_1t_{kj}^x) = \sum_{x=1}^{n-1} \sum_k Q_{ik} E(\mathsf{C}_x^n t_{ik}^{n-x}) E({}_1t_{kj}^x)$$

Let Z be the above expression. Taking the C_x^n outside the summation, we get

$$Z = \sum_{x=1}^{n-1} \mathsf{C}_x^n \sum_k Q_{ik} E(t_{ik}^{n-x}) E({}_1t_{kj}^x)$$

Now let the symbols $\{E(t_{ik}^{n-x})\}$ stand for the whole matrix of $(n - x)$th moments of holding time, and, as before, $\{E({}_1t_{kj}^x)\}$ represents the vector of xth moments of first passage times from each state k to the fixed state j. Let $\underline{Z}$ be the column vector that results from the s equations formed by letting i take on all values from 0 to $s - 1$. Then

$$\underline{Z} = \sum_{x=1}^{n-1} [\mathsf{C}_x^n [Q \times \{E(t_{ik}^{n-x})\}] \overset{+}{\times} \{E({}_1t_{kj}^x)\}]$$

where the symbol $\times$ after the Q means the multiplication of corresponding elements of Q and the matrix $\{E(t_{ik}^{n-x})\}$, and, as before,

$\overset{+}{\times}$ represents the usual inner product of a matrix and a vector. The summation involving k has been replaced by the matrix product operation, and the remaining single summation represents the sum of corresponding elements of $n - 1$ column vectors.

Collecting all these observations, we can write

$$\{E({}_1t_{ij}^n)\} = \{E(t_i^n)\} + \left[\sum_{x=1}^{n-1} \mathsf{C}_x^n[Q \times \{E(t_{ik}^{n-x})\}] \overset{+}{\times} \{E({}_1t_{kj}^x)\}\right] + Q \overset{+}{\times} \{E({}_1t_{kj}^n)\}$$

Looking at the last vector in the equation, $\{E({}_1t_{kj}^n)\}$ observe that it is the same as the vector on the left side of the equal sign, which is the vector of nth moments of first passage time from each of the states to the selected state j. Subtracting $Q \overset{+}{\times} \{E({}_1t_{kj}^n)\}$ from both sides, we get

$$\{E({}_1t_{ij}^n)\} - Q \overset{+}{\times} \{E({}_1t_{ij}^n)\} = \{E(t_i^n)\} + \sum_{x=1}^{n-1} \left[\mathsf{C}_x^n[Q \times \{E(t_{ik}^{n-x})\}] \overset{+}{\times} \{E({}_1t_{kj}^x)\}\right]$$

where $\{E({}_1t_{kj}^n)\}$ has been replaced by $\{E({}_1t_{ij}^n)\}$.

Since x goes from 1 to $n - 1$, all the statistical moments of the form $E({}_1t_{kj}^x)$ are the xth moments of first passage times from state k to state j, but x is less than or equal to $n - 1$, so that the nth moments of first passage time are now all on the left side of the equation, and only smaller moments are on the right side.

The left side of the equation is a vector minus a matrix times the same vector, which can be factored to be

$$\{E({}_1t_{ij}^n)\} - Q \overset{+}{\times} \{E({}_1t_{ij}^n)\} = (I - Q) \overset{+}{\times} \{E({}_1t_{ij}^n)\}$$

where I is the identity matrix.

Then, multiplying both sides of the previous equation by the matrix inverse of $(I - Q)$, which is $(I - Q)^{-1}$, obviously

$$\{E({}_1t_{ij}^n)\} = (I - Q)^{-1} \overset{+}{\times} \left(\{E(t_i^n)\} + \sum_{x=1}^{n-1} \left[\mathsf{C}_x^n[Q \times \{E(t_{ik}^{n-x})\}] \overset{+}{\times} \{E({}_1t_{kj}^x)\}\right]\right)$$

Now we have the nth moments of first passage time expressed as a function of the nth moments and lower moments of holding times and of moments of first passage times lower than n.

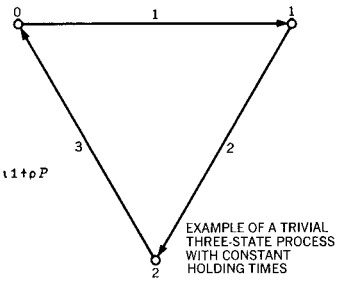
If the statistical distributions of the holding times are known, any statistical moments of first passage times can be computed by a recursive procedure using the above results.

Figure 4 An example of the computation of moments of first passage time

```
    ∇ Z←SET JN;I;Q;Q
[1]   Q←INV(I∘.=I)-Q←P×(JN[0]⌈0×I)∘.≠I←⍳1↑⍴P
[2]   Z←NTH JN[1]
    ∇

    ∇ Z←NTH N;X
[1]   → 2 4[N≤⍴⍴Z←I×0⌊X←1]
[2]   Z←Z+(X!N)×(Q×T N-X)+.×NTH X
[3]   → 4 2[N>X←X+1]
[4]   Z←Q+.×Z++/P×T N
    ∇

    ∇ Z←T N
[1]   Z←T1*N
    ∇
      SET 0 0
3  2  1
      SET 0 1
6  5  3
      SET 0 2
36  25  9
      SET 0 3
216  125  27
      SET 0 4
1296  625  81
      SET 0 5
7776  3125  243
      SET 0 6
46656  15625  729
      SET 1 0
1  3  2
      SET 1 1
1  6  4
      SET 1 2
1  36  16
```



EXAMPLE OF A TRIVIAL THREE-STATE PROCESS WITH CONSTANT HOLDING TIMES

	0	1	2
0	0	1	0
1	0	0	1
2	1	0	0

PROBABILITY TRANSITION MATRIX P

	0	1	2
0	0	1	0
1	0	0	1
2	0	0	0

Q(FOR j = 0)

	0	1	2
0	1	1	1
1	0	1	1
2	0	0	1

$\underline{Q} = (I - Q)^{-1}$

	0	1	2
0	0	1	0
1	0	0	2
2	3	0	0

T1

	0	1	2
0	0	1	0
1	0	0	4
2	9	0	0

T2

	0	1	2
0	0	1	0
1	0	0	2^N
2	3^N	0	0

TN

MATRICES FOR MOMENTS OF HOLDING TIME

Let $\underline{Q}$ be the inverse of $(I - Q)$, and suppose that an APL function called T exists in a workspace that computes the matrix of nth moments of conditional holding time. The syntax of T is given by its header line, $\nabla Z \leftarrow T\ N$, so that T has only one input parameter, N, and it returns a value, Z, which is the matrix of nth moments of conditional holding time.

Then the recursive computation of any statistical moments of first passage time is trivial, and the moments are given by the APL function NTH, displayed in Figure 4. The function SET, also displayed in Figure 4, computes the Q and the $\underline{Q}$ for use in NTH. Sample executions are shown using a simple process whose statistical moments of first passage time are obvious so that the results can be checked by inspection.

Although the function SMP only computes the first three moments of first passage time, any moments may be obtained in this fashion.

The standard deviation of first passage time is

$$\sigma({}_1t_{ij}) = [E({}_1t_{ij}^2) - [E({}_1t_{ij})]^2]^{1/2}$$

Figure 5 Computations in the APL function SMP

```
     ∇ PPT SMP J;I;I;IP;T2;T3;S
[1]    → 3 2[PRT⌊S←(⍴P)[0]]
[2]    □←(1 1 ⍴' ';'S IS ';S;', J IS ';J)
[3]    → 4 6[0=⍴I←(1≠+/P)/I←⍳S]
[4]    □←'ROW SUMS OF P ARE BAD.'
[5]    □←(2,⍴I)⍴I,+/P[I;]
[6]    →14[⍳T2←~1∈T2≠0
[7]    → 10 8[0∈I←T2≥T1×T1]
[8]    □←('THESE 2ND MOMENTS ARE < 1ST MOMENTS SQUARED:';'×□'[I])
[9]    →
[10]   →14[⍳T3←~1∈T3≠0
[11]   → 14 12[0∈I←T3≥T2*1.5]
[12]   □←('THESE 3RD MOMENTS ARE < 2ND MOMENTS TO THE 1.5:';'×□'[I])
[13]   →
[14]   PI←+⌿INV P+I←I∘.≠I←⍳S
[15]   I←I÷+/I←PI×ET1←+/P×T1
[16]   E1T←(IP←INV 1-I+P×(S⍴J)∘.≠I)+.×ET1
[17]   → 22 18[PRT]
[18]   □←('PI IS  ';PI)
[19]   □←('I IS   ';I)
[20]   □←('ET1 IS ';ET1)
[21]   □←('E1T IS ';E1T)
[22]   →⍳T2
[23]   ET2←+/P×T2
[24]   E2T←IP+.×ET2+2×(P[;I]×T1[;I])+.×E1T[I←(I≠J)/I]
[25]   SIG←(I←E2T-E1T×E1T)*0.5
[26]   → 28 27[PRT]
[27]   □←('SIG IS ';SIG)
[28]   →⍳T3
[29]   ET3←+/P×T3
[30]   E3T←IP+.×ET3+3×((P[;I]×T2[;I])+.×E1T[I])+(P[;I]×T1[;I])+.×E2T[I]
[31]   SKW←(E3T+(2×E1T*3)-3×E1T×E2T)÷(I=0)+I*1.5
[32]   →⍳~PRT
[33]   □←('SKW IS ';SKW)
     ∇

     ∇ Z←INV C;H;I;J;S;P;Q;R
[1]    H←P←R←C←⍳1↑⍴Z←C
[2]    S←(1-Z[;J]←I←H)×Q←Z[;J←C[(⌈Z[I←P[Q⍳S];C])⍳S←⌈/Q←⌈/⌈Z[R;C]]]
[3]    Z←Z-S∘.×Z[I;]←Z[I;]÷Q[I]
[4]    C←(C≠P[I]←J)/C
[5]    → 6 2[0≠⍴R←(I≠R)/R]
[6]    Z←Z[⍋P;P]
     ∇

     ∇ Z←F EDT X;W;WF;S;P;Q;R
[1]    WF←(3+F+⌈10⌈.⍟WF),F,3+F+⌈10⍟WF←WF+1E¯9>WF←⌈/⌈0⌋X
[2]    S←' ¯ 0123456789','.'[WF⌈1]=0]
[3]    Z←Z-Q←¯10=(Z←(F≠P←ΦP)\¯10+Z←10⌊⌊(⌊0.5+X×10*F←WF[1])∘.÷10*Φ⍳W←1)⌈.⌊
       ¯11×P∘.>P←⍳W←WF[0]
[4]    Z←Z-((Q←X∘.<W⍴0)⌈.×P+1)∘.=P
[5]    Z←((¯1↓⍴X),W×¯1↑⍴X)⍴S[Z+12]
[6]    →⍳2=⍴WF
[7]    Z←(,(2+WF)∘.>P)/Z
     ∇
```

The skew of first passage time is given by

$$\text{skew } ({}_1t_{ij}) = E\left[\frac{[{}_1t_{ij} - E({}_1t_{ij})]^3}{\sigma^3({}_1t_{ij})}\right]$$

$$\text{skew } ({}_1t_{ij}) = E\left[\frac{{}_1t_{ij}^3 - 3\,{}_1t_{ij}^2 E({}_1t_{ij}) + 3\,{}_1t_{ij} E^2({}_1t_{ij}) - E^3({}_1t_{ij})}{\sigma^3({}_1t_{ij})}\right]$$

$$\text{skew } ({}_1t_{ij}) = \frac{E({}_1t_{ij}^3) - 3E({}_1t_{ij}^2)E({}_1t_{ij}) + 3E^3({}_1t_{ij}) - E^3({}_1t_{ij})}{\sigma^3({}_1t_{ij})}$$

$$\text{skew } ({}_1t_{ij}) = \frac{E({}_1t_{ij}^3) - 3E({}_1t_{ij}^2)E({}_1t_{ij}) + 2E^3({}_1t_{ij})}{\sigma^3({}_1t_{ij})}$$

description of the SMP function

With the above derivations of the information about first passage times in mind, we describe the computations in the APL function SMP, shown in Figure 5, along with the functions INV and EDT:

- Lines 1 and 2 test PRT to determine whether a typewriter display is desired. If PRT = 1, then the values for S, the number of states, and J, the selected state of interest, are displayed.
- Lines 3 through 5 check the row sums of P to see if they are all equal to one. If any are not, they are displayed along with the diagnostic message "ROW SUMS OF P ARE BAD." Nevertheless, execution will continue since the fact that the row sums are not all equal to one might be due to imprecisions in the computation of P.
- Line 6 checks the second moment matrix, $T2$, to see if the second moments are supplied. If all elements of $T2$ are zeros, then all calculations involving second and third moments are bypassed in the rest of the function. If the second moments are supplied, then they are checked for validity on lines 7 to 9. If they are found to be in error, then SMP exits completely from function execution by means of the $\rightarrow$ operation.
- Lines 11 to 13 perform a similar check for the third moments, if supplied.
- Line 14 computes the vector π by using the matrix inversion function INV, also displayed in Figure 4.
- Line 15 computes the vector $\underline{P}$.
- Line 16 computes the vector **E1T**, the first moments of first passage times from each of the states to the selected state J.
- Lines 17 to 21 display the results of the computations thus far if PRT = 1.
- Line 22 exits from SMP if no matrix of second moments was supplied.
- Line 23 computes the second moments of unconditional holding times, and line 24 computes the second moments of first passage times from each of the states to state J. These are then used on line 25 to calculate the standard deviations of first passage times.
- Lines 26 and 27 display the standard deviations of first passage times if PRT is a one.
- Line 28 exits from the function if no matrix of third moments was supplied, i.e., $T3$ has only zeros in it.
- Line 29 obtains the unconditional third moments of holding time, **ET3**.
- Lines 30 and 31 compute the third moments of first passage times, **E3T**, and the skew of first passage times, **SKW**.
- Lines 32 and 33 display **SKW** if PRT is a one.

A sample execution of EVL, which uses SMP, is shown in Figure 6, with PRT = 1, so that the intermediate displays appear from SMP. Observe that when $S = 6$, five floating buffers are used, and the expected value of the time to go from state 5 to state 0 is 1.352 seconds with a standard deviation of 0.8951 and a skew of 1.895. The skew tells us that this passage time is not normally distributed, and, since the run time for this case is 86.410 seconds, the effects of starting the program in state 5, with all five buffers in the refill queue, do not last very long compared to the total run time.

Figure 6 Sample of additional information obtainable from SMP function

```
      PRT←1
      3 EDT .025 .05 .075 .1 .1 .2 .5 .2 EVL 1000 50 .001667

 S IS 2, J IS 0
PI IS  0.3091  0.6909
P IS   0.3476  0.6524
ET1 IS 0.08335  0.07
E1T IS 0.2398  0.1564
SIG IS 0.1467  0.1208
SKW IS 1.49  2.014

 S IS 3, J IS 0
PI IS  0.2336  0.5222  0.2442
P IS   0.2663  0.4999  0.2338
ET1 IS 0.08335  0.07  0.07
E1T IS 0.313  0.2296  0.3861
SIG IS 0.2609  0.2472  0.2751
SKW IS 2.159  2.461  1.955

 S IS 4, J IS 0
PI IS  0.1972  0.4407  0.2061  0.156
P IS   0.2263  0.4248  0.1986  0.1503
ET1 IS 0.08335  0.07  0.07  0.07
E1T IS 0.3683  0.285  0.5146  0.671
SIG IS 0.3827  0.3736  0.4479  0.4639
SKW IS 2.735  2.92  2.107  1.932

 S IS 5, J IS 0
PI IS  0.1769  0.3954  0.1849  0.1399  0.1029
P IS   0.2038  0.3825  0.1789  0.1353  0.09959
ET1 IS 0.08335  0.07  0.07  0.07  0.07
E1T IS 0.4091  0.3257  0.6107  0.8403  0.9967
SIG IS 0.4958  0.4887  0.6151  0.6629  0.6739
SKW IS 3.23  3.362  2.34  1.997  1.913

 S IS 6, J IS 0
PI IS  0.1645  0.3676  0.1719  0.1301  0.09573  0.0701
P IS   0.1899  0.3565  0.1667  0.1261  0.09282  0.06797
ET1 IS 0.08335  0.07  0.07  0.07  0.07  0.07
E1T IS 0.4389  0.3555  0.6813  0.9662  1.196  1.352
SIG IS 0.5951  0.5892  0.7655  0.8518  0.8869  0.8951
SKW IS 3.677  3.782  2.599  2.133  1.943  1.895

 S IS 7, J IS 0
PI IS  0.1565  0.3497  0.1635  0.1238  0.09106  0.06668  0.04881
P IS   0.1809  0.3396  0.1588  0.1202  0.08842  0.06475  0.0474
ET1 IS 0.08335  0.07  0.07  0.07  0.07  0.07  0.07  0.07
E1T IS 0.4607  0.3774  0.7329  1.059  1.344  1.573  1.73
SIG IS 0.6798  0.6747  0.8957  1.02  1.087  1.114  1.121
SKW IS 4.087  4.177  2.861  2.305  2.027  1.906  1.876

 S IS 8, J IS 0
PI IS  0.1511  0.3376  0.1579  0.1195  0.08792  0.06438  0.04713  0.0345
P IS   0.1748  0.3282  0.1535  0.1161  0.08545  0.06258  0.04581  0.03353
ET1 IS 0.08335  0.07  0.07  0.07  0.07  0.07  0.07  0.07  0.07
E1T IS 0.4767  0.3934  0.7708  1.126  1.452  1.737  1.967  2.123
SIG IS 0.7508  0.7461  1.006  1.166  1.264  1.318  1.341  1.347
SKW IS 4.468  4.549  3.116  2.49  2.148  1.961  1.877  1.856

     .000    1.000    2.000    3.000    4.000    5.000    6.000    7.000     .000
  153.350  107.294   95.410   90.474   87.913   86.410   85.460   84.832   83.350
    1.840    1.287    1.145    1.085    1.055    1.037    1.025    1.018     .070
    1.000     .342     .172     .102     .065     .044     .030     .021     .311
```

Note that $\sigma({}_1t_{00})$ is 0.5951, compared with $E({}_1t_{00}) = 0.4389$, which indicates that the run time for the program will not vary wildly, at least. Unfortunately, it does not seem to be easy to calculate the standard deviation of run time directly from this information.

Following these displays, the results of EVL have been edited using the EDT function and displayed.

Appendix B: Assumptions for the model

1. The writing of the output of the program does not in any way affect the reading and processing of the input. When the output is on a separate channel, the time for the writing can be ignored if and only if it can never cause either processing or reading to stop. This will be the case when the time to write is always less than or equal to the processing time for a block of records, and will also be the case if the time for a read is never smaller than the time for a write.

   If the output is written sequentially on another channel and arm, then this will usually be the case, except for one extra rotation when a cylinder is crossed. This is a negligible portion of the total run time.

2. The holding time $E(t_{01})$ is $1 \div \lambda$. Since the program, when reading full-track records from a disk, usually cannot read in less than two full revolutions of the disk, it might be argued that the holding time in all cases should be a multiple of a revolution time. If this were done for state 0, and if the seek can be initiated in the middle of a revolution, it becomes more difficult to determine the effect of this on the time to read the next record. For tape inputs, this problem does not exist.

   On running EVL repeatedly in two ways—the way it is, and with $E(t_{01})$ rounded to the nearest multiple of 0.025—there was little effect on the results. When this change is made, the entries in the first row of the probability transition matrix P must also be changed.

   This holding time, t_{01}, is also dependent on Assumption 3.

3. The arrival pattern of buffers to the refill queue is Poisson. This is in part based on experience, and also on results in References 3 and 4. This point is addressed in the text in the paragraph labeled assumptions.

4. All reading is done from a single channel, so that reads cannot be overlapped with one another.

5. The initial effects of priming all the buffers have worn off, i.e., that the initial tendency for them to all empty simultaneously has disappeared. This assumption is partly related to the assumption in item 3.

   The consequences are that the run times are somewhat lower than they would be if the initial effects were considered. This can be compensated for by making the program correspond to mathematical reality, i.e., by getting it into steady state immediately.

This can be done by writing out a variable length block as the first block of each sequence. For example, if there were 50 records in a block and ten input files, writing five records in the first block of the first input, ten records in the first block of the second input, etc., the effect achieved is similar to the steady-state effect. We emphasize the point that the steady state here discussed is not the same as the steady state of the semi-Markov process.

6. Whenever a buffer is placed in the refill queue, it is in fact the next one needed, i.e., the prediction of which file to read next is based on examining the last record in each block.

7. The random variable t_r is a discrete random variable given by the vectors **R** and **H**. This important assumption is discussed in the text in the paragraph labeled refill time distribution.

CITED REFERENCES AND FOOTNOTE

1. L. Takacs, "A process of servicing machines," *Introduction to the Theory of Queues*, Oxford University Press, New York, New York, Chapter 5 (1962).
2. W. Chang, "Single-server queuing processes in computing systems," *IBM Systems Journal* 9, No. 1, 36–71 (1970).
3. R. E. Barlow and F. Proscham, *Mathematical Theory of Reliability*, John Wiley & Sons, Inc., New York, New York, 121–151 (1965).
4. W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. 2, John Wiley & Sons, Inc., New York, New York, 355–356 (1966).
5. The actual program run time was furnished by R. F. Zorn.
6. The APL functions are based on the notes from a course taught by Dr. J. E. Flanagan at the IBM Systems Research Institute, New York, in 1968.
7. A. D. Falkoff and K. E. Iverson, *APL\360 User's Manual*, International Business Machines Corporation, Thomas J. Watson Research Center, Yorktown Heights, New York (August 1968).