THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

COMPUTING SYSTEM SIMULATION USING MARKOV PROCESSES

Mark C. Maletz

CRL-TR-12-83

APRIL 1983

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

[1] Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

ABSTRACT

Models of computing systems involving fixed system resources and probabilistically specified user state transition behavior must account for interactions between the system users and the system states that result from resource limitations. This is normally accomplished by the introduction of queues for the system states. Because of the probabilistic definition of user behavior, queue transition probabilities must usually be determined by a simulation of the computing system. This paper develops a new technique for simulating computing systems where the hardware configuration is specified by a processor/state service matrix and is fixed throughout the simulation, and user state transition behavior is specified using a basic state transition probabilities matrix satisfying the requirements for a Markov process.

INTRODUCTION

Computing system models employing a Markov methodology partition the computing system resources into a set of states which may be occupied by user tasks. No user task may occupy more than one state at any given time, and every user task must occupy some state at all times (i.e., the states form a true partition). User task state transition behavior is specified using a Markov basic state transition probabilities matrix (STPM). This matrix specifies the probabilities that a user task will move from its current state to every other state in the system. Implicit in the use of Markov processes is the assumption that a state will be available when a user task attempts to move to that state. This assumption is not generally satisfied by state sets that correspond to processing activity undertaken by a user task. The problem is that user tasks are not always involved in processing, but rather, must occasionally await processing on queues for system states. These queues may be added to the state set and be used to produce an extended STPM which includes the newly created queue states.

Construction of an extended STPM requires the derivation of queue entry transition probabilities (the probabilities of moving from the original states to the queue states) and the queue maintenance transition probabilities (the probabilities of remaining in and of leaving a queue at the end of the current time step). These queue-related transition probabilities may be calculated from performance statistics for existing systems, but such statistics are not always available, and can often be costly to collect. A less costly and more practical method for deriving these probabilities is through the use of simulation techniques. This paper will introduce a simulation technique for deriving the information needed to calculate the extended STPM for a computing system where user state transition behavior is specified using Markov processes. The states of the basic STPM will be referred to simply as states, and the new states of the extended STPM that function as queues for the basic STPM states will be referred to as queue states. Therefore, there will be a queue state associated with every state of the basic STPM.

A simulation of such computing systems probabilistically moves users among the states of the system and maintains a queue for each state. A state's queue becomes active once all of that state's processors are in use servicing other users. During the simulation, queue length and user queue wait time statistics are collected. A queue transition probabilities matrix which contains the probabilities of moving from every state to the queue for every other state is produced at the end of the simulation. This matrix can be combined with the basic STPM and user queue wait time statistics to form an extended STPM which includes the original states of the basic STPM and a queue state associated with each of the original states. The extended STPM can then be used to specify user behavior in analytical models of the given computing system. In fact, a Markov-based model for studying computing system performance has been developed by the author that utilizes an extended STPM to specify user state transition behavior [7].

COMPUTING SYSTEM SPECIFICATION

The computing systems that we will consider can be completely specified by providing a basic STPM (**P**), a processor/state service matrix (**S**), and the number of user tasks (N). This specification is given as a triple {**P**,**S**,N}. The basic STPM specifies the state transition probabilities for user tasks. A Markov process is a discrete time event model. This means that all user task state transitions occur at the end of discrete time intervals called time steps. The ijth entry of the basic STPM indicates the probability that a user task that is currently in state i will move to state j at the end of the current time step (or remain in state i if i=j). A user task can only enter a state if one of the processors associated with that state is available. Otherwise, the simulation must place the user task on a queue for that state. We will assume that none of the states of the basic STPM are queue states.

The processor/state service matrix is a boolean matrix whose rows correspond to system processors, and whose columns correspond to system states. The entries are calculated as follows:

Let **S** be the processor/state service matrix. If there are M system processors and N system states then **S** will be a MxN boolean matrix.

S_{ij} = TRUE if processor i services state j

= FALSE otherwise

Consider a simple computing system in which the user tasks cycle through three states: Input, Compute, and Output. If we denote the Input state by I, the Compute state by C, and the Output state by O, then we can construct a basic STPM **P**:

$$\mathbf{P} = \begin{array}{c|ccc} & I & C & O \\ \hline I & .3 & .7 & 0 \\ C & .3 & .2 & .5 \\ O & .6 & .1 & .3 \end{array}$$

This matrix models a computing system in which the dominant user state cycle is Input-Compute-Output, but where it is also possible to move to the Input state from the Compute state or to move to the Compute state from the Output state.

For an example computing system, we will assume that there are 13 system processors; 3 dedicated to the Input state, 5 dedicated to the CPU state, 2 dedicated to the Output state, and 3 which may be used by either the Input state or the Output state. Using these assumptions, we can construct a processor/ state service matrix S for the computing system:

	Input	CPU	Output
Proc 1:	TRUE	FALSE	FALSE
Proc 2:	TRUE	FALSE	FALSE
Proc 3:	TRUE	FALSE	FALSE
Proc 4:	FALSE	TRUE	FALSE
Proc 5:	FALSE	TRUE	FALSE
Proc 6:	FALSE	TRUE	FALSE
Proc 7:	FALSE	TRUE	FALSE
Proc 8:	FALSE	TRUE	FALSE
Proc 9:	FALSE	FALSE	TRUE
Proc 10:	FALSE	FALSE	TRUE
Proc 11:	TRUE	FALSE	TRUE
Proc 12:	TRUE	FALSE	TRUE
Proc 13:	TRUE	FALSE	TRUE

We will also assume that there are 24 user tasks in the system. The example computing system is completely specified by the triple {**P**,S,24}. Figure 1 contains a system diagram for the example computing system which includes a graphic representation of user task state transition behavior.

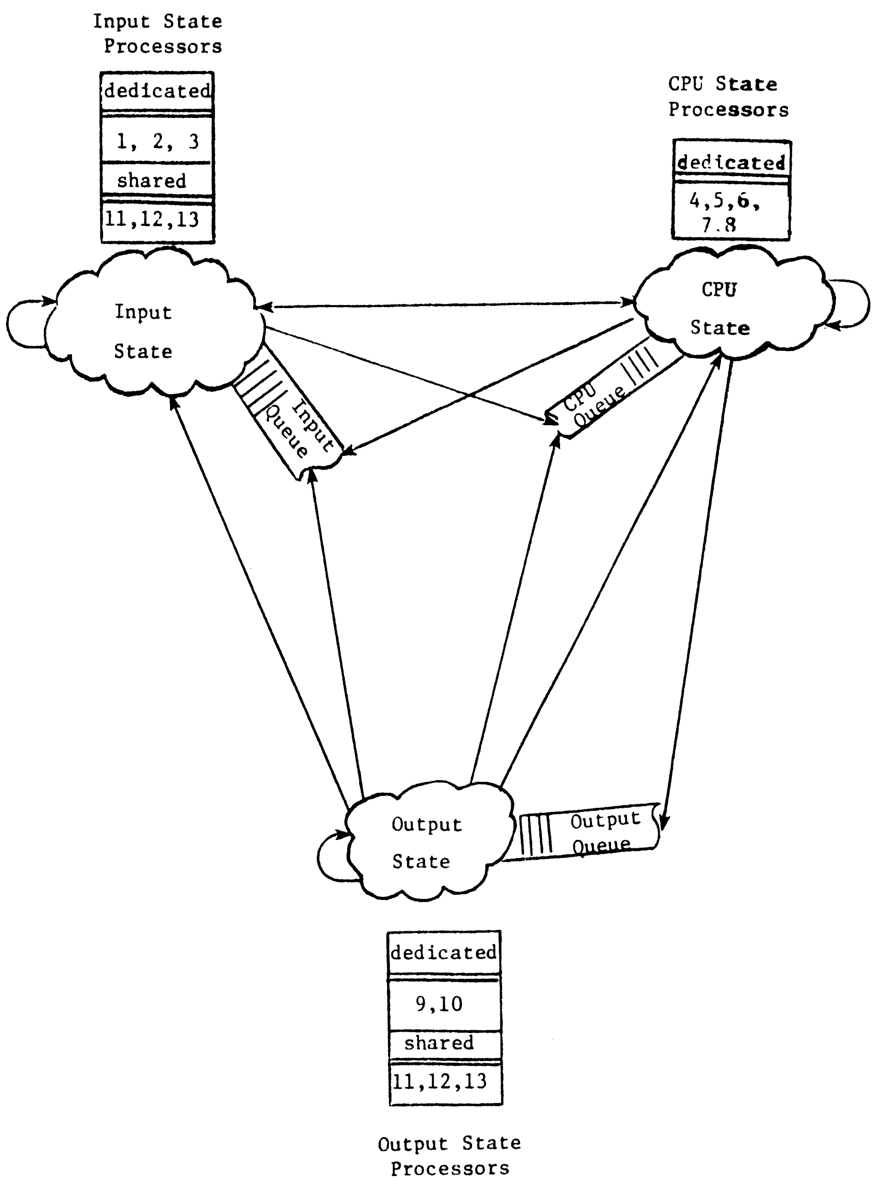

Figure 1. System Diagram for Example Computing System Including User Task State Transitions

Let us consider finite first-order ergodic Markov processes. This imposes several constraints on the user task state transition behavior that we can model:

1. User task state transition behavior can depend only on the current state.

2. There must be a non-zero probability that a user task leaves a state at the end of each time step (no absorbing states are permitted).

3. Once a user task enters a state, it must remain for an entire time step due to the discrete time nature of the Markov process.

4. A fixed time step must be used, and the process must reach equilibrium in a finite number of time steps.

5. The basic STPM state set must be fixed throughout the process.

6. The state transition probabilities must be fixed throughout the process.

GENERAL SIMULATION REQUIREMENTS

The Markov process can be viewed as consisting of two domains: the state/time domain and the transition domain. The state/time domain contains information about the states of the model and the amount of time spent in these states. The transition domain contains information about the way in which user tasks move among the states. Markov processes are highly flexible in the transition domain, but are fairly inflexible with respect to the state/time domain. It is because queues are part of the state/time domain that they are particularly difficult to incorporate into the Markov model unless they are modelled as states.

A queue is used when a user task attempts to move into a state that is not available because all of the processors associated with the state are being used by other user tasks. The order in which user tasks are removed from the queue is determined by the queue maintenance policy. The processor that is assigned to a user task once it enters a new state is determined by the processor selection policy.

The complexity of the queue maintenance algorithm depends on the complexity of the processor/state service matrix (S). The simplest possible situation arises when each row of S has exactly one TRUE entry (corresponding to a unique state assignment for each processor). A more complex and more realistic processor assignment strategy permits multiple state assignments for each processor. In either case, a user task must be placed on a queue when all of the associated state's processors are in use. We will use the more general processor/state mapping strategy.

When a user task enters a new state, either from another state or from the state's queue, the processor selection algorithm assigns that user task one of the state's available processors. If more than one processor is available, then the processor selection algorithm must consider the other states that each available processor can service, since selecting a processor will make it unavailable to these other states.

Therefore, to implement the queue maintenance and processor selection algorithms, a simulation must maintain records of available processors for each state.

To include queues as states of the Markov model, the transition probabilities for entering a queue when attempting to move from one state to another (queue entry transition probabilities) must be known. To simplify this problem somewhat, we will assume that once a user task enters a state, it can retain control of the processor that it is assigned until it leaves the state. This eliminates the possibility of a user task leaving a state, entering a queue for that same state, and then returning to the state. This assumption seems reasonable in general, since most computing systems do behave in this manner. There are, of course, exceptions such as the time slice interrupt which forces a user task to relinquish the CPU after some prescribed time limit. Such exceptions can be handled by using the flexibility that is available directly through the Markov model (e.g., a ready state can be defined, so that a user task can go from use of a processing state to the ready state and then back to the same processing state).

Given a computing system specification {P,S,N}, we simulate user task state transitions through the system. First, a set of starting state assignments must be constructed. This set can either be supplied to the simulation or can be randomly constructed. The simulation uses the basic STPM to move the user tasks among the various states of the model. While many different simulations are possible, based on the queue management and processor selection policies, their goals are the same:

1. To collect statistics on average queue lengths for each queue.

2. To calculate the queue-related transition probabilities for each queue: queue entry and queue maintenance transition probabilities.

3. To collect statistics on average time spent on each queue in the system.

4. To collect statistics on the percentage of time spent in every state of the model.

Using the queue entry transition probabilities and the average time spent on each queue, we can add queue states to the basic STPM to form the extended STPM.

SIMULATION STATISTICS AND OUTPUT

Let us now move from the general specification of Markov model simulations involving queues to a simulation that has been developed as an example. This simulation uses a First-In

First-Out (FIFO) queue maintenance policy. The processor selection policy is also a simple one: from the set of processors that can service the required state, the processor that can service the fewest other states is selected. This policy attempts to select those processors that are least likely to be needed by other states.

The simulation was written in ALGOL-W on the Michigan Terminal System (MTS). It begins by obtaining the computing system specification {P,S,N} and the starting assignment of user tasks among system states. It also requests the number of time steps that are to be simulated. For each time step, the simulation moves each user task from the state that it occupies at the beginning of the time step to the state that it will occupy at the beginning of the next time step (it may be the same state). This is done by generating a pseudo-random number, and using this number and the basic STPM to determine the state transition. During each time step, the simulation program checks for state availability, manages the system resource queues, and updates the following statistics:

1. Cumulative queue length for each state's queue during the simulation.

2. Average queue length for each queue. This is cumulative queue length divided by the number of time steps.

3. Cumulative task queue wait time. This is the cumulative time spent by user tasks on each queue during the simulation.

4. Average task queue wait time. This is the cumulative task queue wait time divided by the total number of user tasks that were on the queue during the simulation.

5. Average distribution of user tasks among system states and queues. This approximates the equilibrium user task distribution. The average distribution of user tasks may be used to identify potential system bottlenecks, where additional processors might be needed.

6. Queue entry transition probabilities matrix. This matrix specifies the probabilities of moving from every state to the queue for every other state. This matrix will always have zero entries along the diagonal since a user task need not leave a state until ready.

A possible problem with the simulation concerns the order in which user tasks are processed. The first tasks to be processed during each time step have a greater implicit priority, since they move among states before other user tasks. This is a function of the fixed time step, and the fact that every user task must move at the end of a time step, even if the move is simply a return to the same state. This problem is alleviated somewhat by the assumption that a user task can remain in a state as long as necessary once it arrives. This prevents user tasks from being bumped from a state to a queue for that state.

In the simulation, user tasks are indexed 1 to n. Task 1 has the highest implicit priority, since it is always moved first. One way to eliminate this problem would be to randomly select the order in which user tasks are moved among the states of the model during each time step. This was not done in the simulation because we are only interested in average quantities, and not in the behavior of individual user tasks. This amounts to a statement of the importance of equilibrium conditions, which are used by the Markov model as opposed to more user task specific information.

A related problem is the implicit priority of states of the model. Depending on the order in which the queues are purged at the end of a time step, certain states could have a higher implicit priority than others. This results from the fact that a processor can potentially service more than one state. If state i and state j each have non-empty queues, and there is one processor available that can service both state i and state j, then depending on the order in which the queues for state i and state j are purged, one of these states will be able to service a new user task, while the other will not be able to remove any user tasks from its queue.

The problem of user task priorities was resolved by considering the fact that the statistics about individual user tasks are averaged into aggregate statistics. This same reasoning does not apply to the states of the system. It is important to ensure that certain states do not receive implicit priority based solely on the order in which the state queues are processed. For this reason, the simulation processes the queues in a random order at the end of each time step.

If a particular computing system required a prioritization of states of the model, a revised simulation could be produced to incorporate such priorities. One possibility would be to impose a fixed order on queue processing, although other mechanisms could also be used. In the current simulation, an implicit priority scheme can be imposed simply by restricting the state assignments of some of the processors. As an example, consider a case in which the input state is to have a greater priority than the output state. If the computing system has six I/O processors, each capable of servicing either the input or the output state, then five of these processors could be assigned to both the input and output states, while the sixth processor could be assigned only to the input state, thereby giving the input state a greater implicit priority.

Every time a user task is moved from a state to another state, to a queue, or is assigned to remain in its current state, a matrix of user task transitions is updated to reflect the move. This simulation transition matrix is only updated when a user task leaves a system state. Therefore, when a user task is removed from a system queue and enters the associated state, the simulation transition matrix is not updated. If a user task enters and leaves a queue during the same time step, then the simulation does not record the transition to the queue or the time spent on the queue. This queue statistic recording policy is used because the simulation processing is serial, while the user task state movements can be thought of as being somewhat parallel. If no processor is available when a user task is moved to a new state, and then one becomes available during the current time step, the wait on the queue is not recorded.

One consequence of the simulation transition matrix updating policy concerns starting conditions. User tasks that begin the simulation in system states cause the simulation transition matrix to be updated during their first time step move, while those starting on queues do not cause an update until they arrive in the associated state and then leave that state. Any problems that this policy may cause are resolved by the fact that the simulation program processes the user tasks through enough time steps so that the starting conditions have little impact on the final results.

Another situation that arises when considering starting conditions is the fact that a user task that begins the simulation on a queue and is then moved into the associated state during the first time step does not get charged for time spent on the queue. This is the result of the queue statistic updating policy.

Zero and 1 valued entries in the queue entry transition probabilities matrix require special consideration. A 0 value indicates that a queue was never needed in moving between the two states involved. This can occur if there is a 0 probability of transition between the two states, or if the state to which the user task moves has enough processors to handle the user tasks that move into the state. A 1 value indicates that every transition from the row state to the column state required that the user task involved be placed on a queue. This does not necessarily mean that the transition between the states requires a queue with probability 1, but it does indicate a very high likelihood and for the purposes of the Markov model, we will use probability 1. As the number of time steps in the simulation is increased, the likelihood of encountering a probability 1 queue transition decreases, unless the associated state has been assigned too few processors.

Table 1 displays the results of a run of the simulation program using the sample computing system of the previous section. The simulation was run for 2500 time steps.

Table 1. Simulation program summary statistics with 13 processors.

The queue length and wait time statistics (in time step units) are:

	Cumul. Queue Length	Average Queue Length	Cumul. Queue Wait Time	Average Queue Wait Time
Input:	7493	2.498	7487	1.321
CPU:	1280	0.427	1278	1.018
Output:	27117	9.039	27111	3.989

The average distribution of user tasks among system states and queues is:

	Input	CPU	Output
In State:	.200	.172	.129
On Queue:	.104	.018	.377

The queue entry transition probabilities matrix is:

	Input Queue	CPU Queue	Output Queue
Input:	0.000	0.128	0.000
Output:	0.561	0.000	1.000
CPU:	0.609	0.074	0.000

In table 1, there is a difference between the cumulative queue length and cumulative task wait time for each of the three states. This difference is based on the time at which these quantities are updated. Cumulative task queue wait time is only updated when a user is removed from the queue. User tasks on queues at the end of the simulation are not reflected in the cumulative task queue wait time, but they are included in the cumulative queue length.

The determination of the number of time steps to be simulated is somewhat arbitrary as long as the simulation is sufficiently long so that the average distribution of user tasks among system states and queues represents an equilibrium distribution. In the simulation whose results are displayed in table 1, the average distribution for 2500 time steps was checked against a simulation run involving 5000 time steps. The average distributions from these two runs were very close, suggesting that an equilibrium had been reached after 2500 time steps.

Table 2. Extended State Transition Probabilities Matrix with Queue States.

	Input State	Input Queue	CPU State	CPU Queue	Output State	Output Queue
Input State	.30	0	.61	.09	0	0
Input Queue	.76	.24	0	0	0	0
CPU State	.13	.17	.20	0	0	.50
CPU Queue	0	0	.98	.02	0	0
Output State	.23	.37	.09	.01	.30	0
Output Queue	0	0	0	0	.25	.75

The extended STPM is constructed using the average task queue wait times and the queue entry transition probabilities matrix. The average task queue wait time is used to calculate the probability that a user task will remain on a queue during the next time step or leave the queue and enter the associated state. The queue entry transition probabilities matrix is used to calculate the probability that a user task will move from every state to the queue for every other state (recall that we assume that a user task never moves from a state to the queue for that state). A more detailed discussion of the derivation of the extended STPM can be found in Maletz [7]. The extended STPM derived for the example computing system using the statistics found in table 1 appears in table 2.

CONCLUSIONS

A simulation technique was developed that can be used to model computing systems in which the hardware configuration is specified by a processor/state service matrix and user task state transition behavior is determined probabilistically by using a Markov basic STPM. The simulation calculates queue-related statistics that can be used to derive an extended STPM which includes queue states. It also produces an average distribution of user tasks among the system states and queues which can be used to verify that the system is in equilibrium and to identify potential system bottlenecks.

The simulation was initially developed to provide the queue-related statistics needed to derive an extended STPM for a computing system specified by the {P,S,N} triple. This extended STPM was used to specify a computing system involving processing states and queue states. The information provided by the simulation is also of interest for other applications where queue-related statistics and equilibrium distributions are required.

Some of the constraints imposed by selecting finite first-order ergodic Markov processes could be eliminated to provide a less restrictive definition of user task state transition behavior. This might provide some interesting results. The first-order condition requires that the current time step state transition depend only on the current state. By using higher order Markov processes, user state transition behavior could depend on the recent state history of a user task. One consequence of this would be a direct ability to model the time slice interrupt that was referred to earlier. The ergodic condition could also be removed to admit the possibility of absorbing states if a particular application had need of such states.

The class of simulations developed for this paper are relatively easy to program and provide a wealth of information. By adjusting the queue maintenance and processor selection policies, a wide variety of computing systems, specified by the {P,S,N} triple can be modelled. This makes computing system simulation using Markov processes an attractive alternative to costly performance monitors for existing systems, and provides a means of modelling new systems.

Acknowledgments. The author is grateful to Toby Teorey for his comments and advice during the development of this paper and his thorough review of a draft of this paper.

REFERENCES

[1] Feeley, J.M. A Computer Performance Monitor and Markov Analysis for Multiprocessor System Evaluation, *Statistical Computer Performance Evaluation*, New York, 1972, pp. 165-225.

[2] Feller, W. *An Introduction to Probability Theory and its Applications, Volume 1*, John Wiley, New York, 1967.

[3] Foley, J.D. A Markovian Model of the University of Michigan Executive System, *CACM* 10,9 (September 1967).

[4] Kemeny, J.G., and Snell, J.L. *Finite Markov Chains*, Van Nostrand, New Jersey, 1960.

[5] Kemeny, J.G., and Snell, J.L. *Mathematical Models in the Social Sciences*, MIT Press, Massachusetts, 1978.

[6] Lewis, P.A.W., and Yue, P.C. Statistical Analysis of Series of Events in Computer Systems, *Statistical Computer Performance Evaluation*, New York, 1972, pp. 265-272.

[7] Maletz, M.C. On Enhancing the Idealized CPU-I/O and I/O-I/O Overlap Models Through the Use of Markov Processes, Computing Research Laboratory Technical Report CRL-TR-7-83, University of Michigan, Ann Arbor, Michigan, February 1983.

[8] Peterson, J., and Bulgren, W. Studies in Markov Models of Computer Systems, Department of Computer Science, University of Kansas, Lawrence, Kansas.

[9] Smith, J.L. An Analysis of Time-Sharing Computer Systems Using Markov Models, Systems Engineering Laboratory, University of Michigan, Ann Arbor, Michigan.

[10] Snell, J.L. *Introduction to Probability Theory with Computing*, Prentice-Hall, New Jersey.