THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY[1]

ON ENHANCING THE IDEALIZED CPU-I/O AND I/O-I/O OVERLAP MODELS THROUGH THE USE OF MARKOV PROCESSES

Mark C. Maletz

CRL-TR-7-83

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

A Modeling technique is developed for studying processor overlap in time-shared and multiprogrammed computing systems, based on the Markov process methodology. Measures of resource utilization, response time, and system throughput are derived from the statistics of the Markov process equilibrium. Techniques for identifying potential system bottlenecks and for determining summary statistics about user transition behavior are also discussed. The result is an enhancement of traditional overlap models with greater generality and improved system characterization capabilities.

1. Introduction

As the complexity of modern time-shared and multiprogrammed computing systems continues to increase, the need for cost-effective analytical models of the performance of these systems also increases. System complexity can be broadly classified as belonging to one of four areas:

1. Changes in the number and speed of Central Processing Units (CPU's).
2. Changes in the number and nature of system peripherals (e.g., I/O devices).
3. Changes in the complexity of user behavior.
4. Changes in the amount of main storage available.

A computing system model is developed that is highly flexible with respect to the first three of the above areas. This model provides information that is of value when evaluating the performance of existing computing systems, selecting new systems, and identifying system resource bottlenecks.

The idealized CPU-I/O Overlap model was first proposed by Hellerman and Smith to analyze the expected throughput of a computing system under certain "ideal" conditions.[3] This computing system consisted of one central processing unit (CPU), one or two I/O channels, and main storage that could be partitioned into one, two, or three equally sized partitions. The "ideal" conditions, aside from the particularly restrictive definition of the configuration of the computing system, were contained in a set of five assumptions:

1. Fixed record size.
2. For each input there must be an output.
3. Fixed number of records per block.
4. Fixed CPU compute time per record.
5. Fixed device access time per block.

The authors of this model acknowledged that these assumptions are not at all realistic; however, the assumptions did permit the development of the basic CPU-I/O overlap model, using analysis of

timing diagrams.

The basic CPU-I/O overlap model was generalized into a set of equations that were used to compute the maximum main storage utilization given the number of CPU's, the number of I/O channels, the number of main storage partitions, and the ratio of CPU time to I/O time to process a block of data.[1] With the development of these general equations, the use of timing diagrams and graphical analysis became obsolete. The generalized equations approach suffered, however, from an unrealistic definition of the system "user", restricted by a strictly deterministic precedence order of input, compute, output.

This paper develops a more general model of the user tasks involved in the CPU-I/O and I/O-I/O overlap models, based on Markov processes. Section Two develops the basic Markov model of user behavior. Sections Three and Four attempt to resolve the weaknesses of the basic Markov model by introducing queue states and limiting system resource availability. Finally, Section Five reviews the development of the Markov model, discusses the conclusions arrived at, and suggests areas for further research.

2. Development of the Basic Markov Model

We will consider finite first-order ergodic Markov chains, and begin by defining the states of the Markov process. A user task is said to be in a particular state when the computing system is involved in processing for that task using a processor associated with that particular state. This means that every state must have associated with it a unique class of processors. Five states are considered:

1. Input - user task initiated input of one record
2. Output - user task initiated output of one record
3. Compute - CPU computation performed on a user task
4. Page-in - system initiated paging disk activity
5. Page-out - system initiated paging disk activity

These states correspond to classes of processors which are functionally defined. The same physical processors may be assigned to different states at different times depending on the function that they are performing. As long as the states of the Markov model are unique with respect to occupancy by user tasks (state uniqueness criterion), the states may be defined in whatever way is most appropriate to the particular application.

A corollary to the state uniqueness criterion is the state exhaustion criterion. This requires that every state in which the user task might be found must be a state of the Markov model. This can be accomplished by a thorough enumeration of states.

After the states have been defined, we compute the state transition matrix. This is accomplished using the following

procedure:

1. Determine the "real time equivalent" of one step in the Markov process. This is the amount "real time" that corresponds to a single time step in the process.

2. Assign the non-returning transition probabilities (which will be modified later). A non-returning transition probability, $\mathbf{N}_{ij}$, is defined as the probability that a user task moves to state j given that it is currently in state i, where i and j must be different.

3. Calculate the expected number of time steps that a user task remains in a each state. Note that the "real time equivalent" of the Markov process time step must be sufficiently small so that a user task, on average, will remain in each state at least one time step.

4. Determine the probability of return for each state:

$$P(r)_i = (E(ts)_i - 1)/E(ts)_i \qquad [2.1]$$

where $E(ts)_i$ is the expected number of time steps that a user task will remain in state i, and $P(r)_i$ is the calculated probability of return.

The solution of this equation, for each state, yields that state's probability of return ($\mathbf{P}_{ii} = \mathbf{X}$).

5. Scale the non-returning transition probabilities using the probability of return for each state:

$$\mathbf{P}_{ij} = \mathbf{N}_{ij} \times (1 - \mathbf{P}_{ii})$$

for each state i, $\mathbf{P}_{ij}$ is the transition probability to state j (where $i \neq j$).

The Markov process transition probability matrix is given by **P**, at the conclusion of step five.

The problem with this development is that the Markov model based on the transition matrix **P** assumes that a user task will be able to enter any state that it is probabilistically directed to enter. Implicit in this is an assumption of unlimited system resources. Such an assumption is clearly unrealistic. Moreover, since an unlimited resource computing system degenerates into a series of single user systems, there is little benefit in considering such systems when constructing overlap models. To resolve this problem a queueing system is required.

3. Simulation of Queues in the Markov Model

There are four major limitations which result from the use of Markov processes:

1. The requirement that a fixed time step be used.
2. The fixed number of states requirement.
3. The requirement that transition probabilities be fixed throughout the life of the model.
4. The requirement that once a user task enters a state, it must remain for an entire time step. Related to this is the requirement that there must be a non-zero probability that the user task will leave the state at the end of the time step.

The problems caused by these limitations are greatly reduced when one considers the fact that this model was developed as an enhancement of the basic overlap models, where all processing times are required to be fixed and state transitions are strictly deterministic. Each of the four limitations discussed above is also present in the basic CPU-I/O and I/O-I/O overlap models.

One way to reduce the limitations of fixed states and transition probabilities would be to construct several Markov models for a given computing system, where each model presented information about the equilibrium conditions of the computing system using its respective parameters. This might, for example, be useful if the system was utilized differently during day and evening hours. The requirement that there be a non-zero probability of leaving a state at the end of the time step eliminates the possibility of absorbing states. This is not a serious limitation, however, since computing systems do not typically have absorbing states.

We are now ready to eliminate the unlimited system resources assumption by introducing queues into the Markov model. Ultimately, queues will be included as states of the Markov model. Before this can be accomplished, however, the transition probabilities for entering a queue when attempting to move from one state to another (queue 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 CPU to the ready state and then back to a CPU).

The determination of queue transition probabilities is dependent on the basic transition probabilities, the number of user tasks, and the number and types of system resources. While the queue transition probabilities may be obtained by empirical data collection (in the case of existing computing systems), it is easier to make use of their dependence on the basic transition probabilities and derive the queue transition probabilities using computing system simulation techniques. We will use a program that simulates a computing system in which user task transitions are governed by a basic transition probabilities matrix.' One of the summary statistics produced by this program is the queue transition probabilities matrix for the simulated computing system.

Table 1 displays the starting conditions for a sample run of the simulation program. This run involved eight processors and twelve user tasks, and was run for 40 time steps.

Table 1. Starting conditions for a sample run of the simulation program.

The processor/state service matrix is:

	Input	Output	CPU	Page_I	Page_O
proc 1:	Yes	Yes	No	No	No
proc 2:	Yes	Yes	No	No	No
proc 3:	Yes	Yes	No	No	No
proc 4:	No	No	Yes	No	No
proc 5:	No	No	Yes	No	No
proc 6:	No	No	Yes	No	No
proc 7:	No	No	No	Yes	Yes
proc 8:	No	No	No	Yes	Yes

The transition probabilities matrix is:

	Input	Output	CPU	Page_I	Page_O
Input:	.30	.00	.40	.20	.10
Output:	.10	.30	.30	.20	.10
CPU:	.25	.25	.30	.10	.10
Page_I:	.00	.00	.60	.30	.10
Page_O:	.00	.00	.50	.30	.20

The user task distribution matrix is:

	Input	Output	CPU	Page_I	Page_O
task 1:	In State				
task 2:			In State		

```
task  3:                                          In State
task  4:                          In State
task  5:                                                    In State
task  6:                          In State
task  7:                In State
task  8:                          On Queue
task  9:                                          On Queue
task 10:                In State
task 11:                          On Queue
task 12:    On Queue
```

The processor/state service matrix indicates which states are serviced by each processor in the simulation. The user task distribution matrix indicates the starting state of each user task and whether it is in the state or on the queue for the state. After the simulation program has completed processing, it produces a table of summary statistics, shown in Table 2.

Table 2. Simulation program summary statistics with 8 processors.

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

	Cumulative Queue Length	Average Queue Length	Cumulative Queue Wait time	Average Queue Wait time
Input:	26	0.650	26	1.368
Output:	1	0.025	1	1.000
CPU:	141	3.525	138	1.816
Page_I:	13	0.325	10	1.000
Page_O:	8	0.200	8	1.600

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

	Input	Output	CPU	Page_I	Page_O
In State:	.121	.092	.250	.087	.056
On Queue:	.054	.002	.294	.027	.017

The queue transition probabilities matrix is:

	Input Queue	Output Queue	CPU Queue	Page_I Queue	Page_O Queue
Input:	0.000	0.000	0.893	0.556	0.667
Output:	0.500	0.000	1.000	0.400	0.000
CPU:	0.457	0.034	0.000	0.417	0.375

Page_I:	0.000	0.000	0.926	0.000	0.000
Page_O:	0.000	0.000	0.917	0.200	0.000

In this table, there is a difference between the cumulative queue length and cumulative task wait time for the CPU and Page_I states. This difference is based on the time at which these quantities are updated and on the queue distributions at the end of the simulation. The average task queue wait time is of importance in computing the transition probability matrix. The queue transition probabilities matrix indicates the probabilities of moving from every state to the queue for every other state.

The first four statistics presented in Table 2 are useful in identifying system bottlenecks. In the example run, the CPU state has a cumulative task queue wait time of 138 which is greater than the other four states combined. This indicates that the CPU state represents a system bottleneck, and that it might be desirable to add a fourth CPU. A lesser bottleneck is the Input state which has a cumulative task queue wait time of 26. Because the Input and Output states share three processors (I/O processors), and because the Output state has a cumulative queue length of only one, it might be desirable to assign one of the three processors to the Input state alone, while the other two continue to service both the Input and Output states. Table 3 shows the results of adding a fourth CPU state processor, and assigning one of the three I/O processors uniquely to the input state. All other starting conditions are the same as the previous example.

Table 3. Simulation program summary statistics with 9 processors (4 CPU's).

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

	Cumulative Queue Length	Average Queue Length	Cumulative Queue Wait time	Average Queue Wait time
Input:	27	0.675	27	1.588
Output:	14	0.350	13	1.444
CPU	22	0.550	22	1.100
Page_I:	83	2.075	78	2.438
Page_O:	32	0.800	18	2.571

An analysis of this table shows that the addition of a CPU and the I/O processor assignment change has had the desired effect. Neither the CPU state nor the Input state appear to be bottlenecks. A new potential bottleneck, the Page_I state, has appeared. This could be resolved by adding another processor to the Page_I state.

4. The Generalized Markov Model with Queue States

This section presents an example of the generalized Markov model. With every state in the model of Section Two there is associated a queue state. To avoid confusion, we will refer to non-queue states simply as states and queue states as queues. The ten state transition probabilities matrix is derived from the basic transition probabilities matrix and the queue transition probabilities matrix provided by the simulation using the following procedure:

Let P denote the basic transition probabilities matrix.

Let Q denote the queue transition probabilities matrix output by the simulation.

Let R denote the new 10 state transition probabilities matrix. Row (or column) 2i-1 corresponds to state i, and row (or column) 2i corresponds to the queue for state i. Notice that states have odd numbers and queues have even numbers.

1. State to state: for each state i (i odd) and for each state j (j odd)

$$R_{ij} = P_{ij} \times (1-Q_{ij})$$

2. State to queue: for each state i (i odd) and for each queue j (j even)

$$R_{ij} = P_{ij} \times Q_{ij}$$

3. Queue to associated state: for each queue i (i even) and for the associated state j (j=i-1)

$$R_{ij} = 1 - P(r)_i$$

where $P(r)_i$ is the solution to equation 2.1 when $E(ts)_i$ is set to the average task queue wait time for queue i.

4. Queue to other state: for each queue i (i even) and for each state j ($j \neq i-1$)

$$R_{ij} = 0$$

5. Queue to same queue: for each queue i (i even)

$$R_{ii} = P(r)_i$$

where $P(r)_i$ is as computed in (3) above.

6. Queue to other queue: for each queue i (i even) and for each queue j (j even, j≠i)

$$R_{ij} = 0$$

Using the above procedure, we compute the ten state transition probabilities matrix for the the example computing system presented in Section Three. This matrix appears in Table 4.

Table 4. Ten state transition probabilities matrix.

	1	2	3	4	5	6	7	8	9	10
1	.300	.000	.000	.000	.043	.357	.089	.111	.033	.067
2	.731	.269	.000	.000	.000	.000	.000	.000	.000	.000
3	.050	.050	.300	.000	.000	.300	.120	.080	.100	.000
4	.000	.000	1.000	.000	.000	.000	.000	.000	.000	.000
5	.136	.114	.292	.008	.300	.000	.058	.042	.062	.038
6	.000	.000	.000	.000	.555	.445	.000	.000	.000	.000
7	.000	.000	.000	.000	.044	.556	.300	.000	.100	.000
8	.000	.000	.000	.000	.000	.000	1.000	.000	.000	.000
9	.000	.000	.000	.000	.042	.458	.240	.060	.200	.000
10	.000	.000	.000	.000	.000	.000	.000	.000	.625	.375

State 1: Input State
State 2: Input Queue
State 3: Output State
State 4: Output Queue
State 5: CPU State
State 6: CPU Queue
State 7: Page_In State
State 8: Page_In Queue
State 9: Page_Out State
State 10: Page_Out Queue

We can now calculate the limiting vector for the Markov process which gives the equilibrium distribution of user tasks among the ten states of the model, the mean first passage matrix, and the mean recurrence vector. The results of these calculations appear in Table 5.

Table 5. Markov model summary statistics.

The limiting vector is:

[.104 .046 .106 .002 .248 .291 .120 .034 .057 .006]

The mean recurrence vector is:

[9.65 21.8 9.42 505. 4.04 3.44 8.36 29.6 17.7 157.]

The mean first passage times matrix is:

	1	2	3	4	5	6	7	8	9	10
1	.000	28.4	13.1	503.	4.14	2.92	9.10	24.6	19.5	226.
2	1.37	.000	14.5	505.	5.51	4.92	10.5	25.9	20.9	228.
3	12.2	26.7	.000	504.	4.49	2.97	8.84	25.5	19.3	248.
4	13.2	27.7	1.00	.000	5.49	3.97	9.84	26.5	20.3	249.
5	9.64	24.3	8.97	499.	.000	4.40	9.24	25.5	19.7	247.
6	11.4	26.1	10.8	501.	1.80	.000	11.0	27.3	21.5	248.
7	13.0	27.7	12.3	503.	3.38	2.04	.000	28.6	19.7	250.
8	14.0	28.7	13.3	504.	4.38	3.04	1.00	.000	20.7	251.
9	13.3	27.9	12.6	503.	3.62	2.32	8.13	26.8	.000	250.
10	14.9	29.5	14.2	504.	5.22	3.92	9.73	28.4	1.60	.000

In comparing the limiting vector of Table 5 with the average distribution of user tasks among the states and queues of the model (found in Table 2), we find that the two vectors are very similar. This is expected, since the simulation program was designed to simulate a Markov process, and the same basic transition probabilities matrix was used for both the simulation and the Markov process calculations.

The other statistics reported by the simulation program are identical to results that can be derived from the Markov model (except for average queue length, which is not derivable from the Markov model directly). The average task queue wait time for each queue can be found in the mean first passage times matrix as follows:

> Let AW(i) be the average queue task wait time for queue i, where i must be even.
>
> Let j=i-1 be the number of the state associated with queue i.
>
> Let R be the ten state transition probabilities matrix.

$$AW(i) = R_{ij}$$

The queue transition probabilities matrix calculated by the

simulation program was used explicitly in the construction of the ten state transition probabilities matrix. It can therefore be recovered simply by reversing the procedure used to convert from the queue transition probabilities matrix to the ten state matrix.

4.1 System Bottlenecks and Resource Utilization

The limiting vector gives the average amount of time spent by each task in each state of the system. It can also be viewed as describing the equilibrium distribution of user tasks among the system states. In this way, it provides an implicit measure of state overlap in the computing system. The limiting vector has two primary interpretations. The first is in the identification of possible system bottlenecks. The queue states can be checked, and if any of them have particularly high values in the limiting vector, this is an indication that the associated state may be a system bottleneck. This can be resolved by adding more processors to the state (either new processors, or processors that were previously assigned to other states) or by restricting some of the processors that are assigned to the state and to other states so that they service only the bottleneck state (i.e., reducing the service scope of the processors that service the bottleneck state). One of the advantages of the Markov model over other computing system models, especially those that require a detailed analysis of the system every time a system specification parameter changes, is that it is easy to change system parameters and recompute the limiting vector to see if the changes have produced the desired effect.

A second interpretation of the limiting vector is that it provides a measure of resource utilization. This can be helpful in determining whether states are under-utilized or over-utilized relative to the number of processors that they have been assigned. If a state has been assigned a relatively large number of processors but has a relatively low-valued entry in the limiting vector, then it might be beneficial to either reduce the number of processors assigned to the state or assign some of them to additional states. Changes in processor assignment require that processors are flexible with respect to the states that they can service.

It is important to distinguish between processor utilization, which is considered in traditional overlap models, and state utilization. The limiting vector provides information about state utilization. This information can be interpreted along with the processor class information contained in the processor/state service matrix to produce one measure of processor utilization. It cannot be used to determine the percentage of time that a particular processor is in use or idle, but it can be useful in providing a general measure of processor utilization for all processors associated with each state.

4.2 System Response Time

The mean recurrence vector measures the amount of time that

elapses between the time that a user task enters a state, and the time that it returns to that same state. This vector can be used to measure system response time by assigning one of the states in the model to be a user response state in addition to its other assignment. The user response state is the state from which all user interfacing is done. In the example computing system, the CPU state could be designated as the response state. The response time would then be 3.4 time steps.

Another possibility for response state is to add a new state to the model which handles nothing but user interaction. This has the advantage that user tasks enter the state only for user interfacing, and therefore, the response time measure is more accurate.

4.3 System Throughput

The mean first passage times matrix can be used to construct a measure of system throughput. This is done by defining a sequence of states as representing a cycle through the system. The mean first passage times matrix can then be used to determine the average amount of time required to complete this cycle. If there are n user tasks, then this is the amount of time that is required to complete n cycles. As an example, if we define an Input state, CPU Queue state, CPU state, Output state cycle for all user tasks, then this cycle requires 25.9 time steps, and system throughput would be 12 cycles per 25.9 time steps. This construct allows for more than one cycle. In fact, each user task can perform its own cycle.

5. Conclusions

A general Markov model of computing systems was developed that enhances idealized CPU-I/O and I/O-I/O overlap models in a number of ways. First, it eliminates the need for a restrictive configuration definition, and because the Markov process is probabilistic, the user task state transitions no longer have to be deterministic, which makes for a far more realistic model.

The Markov model that was initially proposed assumed an unlimited number of resources. This assumption was removed when queues were added to the Markov model by adding a queue state for every non-queue state that was present in the unlimited-resource Markov model.

The Markov model can be used to analyze a computing system by deriving the three system parameters that are most often used by the idealized overlap models:

1. A utilization parameter
2. A response time parameter (in time step and real time equivalent formats)
3. A throughput parameter

A special case of the Markov model occurs when the basic transition probabilities are contained in a zero-one matrix. This models deterministic user behavior and gives the same results as the idealized overlap models (based on test cases that were analyzed). In fact, the Markov model provides more information than the idealized models. An important example of this concerns user task behavior. In the transition probabilities matrix, only the probabilities of moving from state i to state j (for every i,j combination) need to be specified. The model, however, provides us with a matrix containing the average number of time steps that are required by a user task to move from state i to state j.

One interesting possibility for enhancement of the Markov approach might be to consider Markov processes that are of order greater than one or that are cyclic processes. The existence of cycles might prove useful in further investigations of system throughput.

The simplicity of constructing the Markov models, along with the usefulness of the results derived from the model makes the Markov-based enhancement of the idealized CPU-I/O and I/O-I/O overlap models a useful tool for those interested in the analysis of time-shared and multiprogrammed computing 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] Hellerman, H., and Conroy, T.F. *Computer System Performance*, McGraw-Hill, New York, 1975, pp. 149-157.

[3] Hellerman, H., and Smith, H.J. Jr. Throughput Analysis of Some Idealized Input, Output and Compute Overlap Configurations, *Computing Surveys*, 2,2 (June 1970), pp. 111-118.

[4] Hynes, A.C. An Equation for Throughput Under Overlap Conditions, Data Translation Project Working Paper DE 7.3, University of Michigan, Ann Arbor, Michigan, April 1976.

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

[6] Maletz, M.C. Computing System Simulation Using Markov Processes, Computing Research Laboratory Technical Report CRL-TR-12-83, University of Michigan, Ann Arbor, Michigan, February 1983.

[7] Mills, P. The Overlapping of CPU and I/O Processing in Multiprogramming, Proceedings of the Seventh International Conference of the Computer Measurement Group, Inc., Atlanta, Georgia, November 16-19, 1976.

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

[9] Sekino, A. Throughput Analysis of Multiprogrammed Virtual Memory Computer Systems, Proceedings of the First Annual SIGME Symposium on Measurement and Evaluation, Palo Alto, California, 1973, pp. 47-53.

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

[11] Teorey, T.J. General Equations for Idealized CPU-I/O Overlap Configurations, *CACM* 21,6 (June 1978), pp. 500-507.

[12] Towsley, D., Chandy, K.M., and Browne, J.C. Models for Parallel Processing Within Programs: Application to CPU:I/O and I/O:I/O Overlap, *CACM* 21,10 (October 1978).