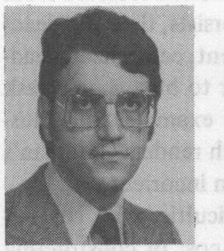
[7] H. C. Sun, "Deadlock detection in distributed systems–A preliminary evaluation," Masters report, Dep. Comput. Sci., Univ. Maryland, 1978.



Virgil D. Gligor received the B.Sc., M.Sc., and Ph.D. degrees in electrical engineering and computer science in 1972, 1973, and 1976, respectively, all from the University of California, Berkeley.

In 1976, he joined the Department of Computer Science, University of Maryland, College Park, as an Assistant Professor. Between 1977 and 1980 he was also associated with the Computer Systems Group of Burroughs Corporation. He has lectured in computer science at the University of California at Santa Barbara (1974–1976) and at the Institutul Politechnic Bucuresti in Romania (1972). His research interests include reliability and integrity of operating systems, computer architecture, and distributed database systems.

Susan H. Shattuck received the B.Sc. degree from Duke University, Durham, NC, in 1967 and the M.Sc. degree from the University of Maryland, College Park, in 1977.

Since 1974 she has worked on various research projects in the Department of Computer Science, University of Maryland. In 1967–1969 she was a Systems Programmer for the Naval Command Systems Support Activity, Washington, DC. Her research interests include distributed operating systems and reliability and integrity of operating systems.

Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets

C. V. RAMAMOORTHY, FELLOW, IEEE, AND GARY S. HO, MEMBER, IEEE

Abstract–Some analysis techniques for real-time asynchronous concurrent systems are presented. In order to model clearly the synchronization involved in these systems, an extended timed Petri net model is used. The system to be studied is first modeled by a Petri net. Based on the Petri net model, a system is classified into either: 1) a consistent system; or 2) an inconsistent system. Most real-world systems fall into the first class which is further subclassified into i) decision-free systems; ii) safe persistent systems; and iii) general systems. Procedures for predicting and verifying the system performance of all three types are presented. It is found that the computational complexity involved increases in the same order as they are listed above.

Index Terms–Asynchronous, concurrent, performance, Petri net, real time.

I. Introduction

THE RECENT advances in solid-state technology have provided computer designers with powerful functional capabilities at low cost. This enables computer designers to isolate the functions of a computing element and add intelligence to functional units that can optimize their circuits locally. Such an approach eliminates the cost involved in time and bandwidth to submit these local events for a central judgment in real time. This trend has led to the emergence of distributed systems. Furthermore, the recent growth in microprocessor architectures, their capabilities, and their low cost, have motivated system designers to design computer systems as distributed networks of microprocessors. By using multiple processing elements, system throughput can be improved and processing requirements and capabilities unobtainable by uniprocessors can be satisfied. However, the success of multiple processor systems greatly depends on the effectiveness of the synchronization among the processing elements.

In this paper, we concentrate our discussion on techniques for the prediction and verification of performance of distributed systems. We consider a distributed system as a set of loosely or tightly coupled processing elements working cooperatively and concurrently on a set of related tasks. In general, there are two approaches for performance evaluation [6], [2]: 1) deterministic models and 2) probabilistic models. In deterministic models, it is usually assumed that the task arrival times, the task execution times, and the synchronization involved are known in advance to the analysis. With this information, a very precise prediction of the system performance can be obtained. This approach is very useful for performance evaluation of real-time control systems with hard deadline requirements.

In probabilistic models, the task arrival rates and the task service times are usually specified by probabilistic distribution functions. The synchronization among tasks is usually not

Manuscript received February 00, 1979; revised December 00, 1979. This work was supported by the Ballistic Missile Defense Advanced Technology Center under Contract DASG60-77-C-0138, the U.S. Air Force under Grant F49620-79-C-0173, and the National Science Foundation under Grant MCS77-27293. The work of G. S. Ho was done while he was at the University of California, Berkeley, CA.

C. V. Ramamoorthy is with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.
G. S. Ho is with Bell Laboratories, Naperville, IL 60540.

modeled, because otherwise the number of system states becomes so large that it would be impossible to perform any analyses. Probabilistic models usually give a gross prediction on the performance of a system and are good for early stages of system design when the system characteristics are not well understood. In this paper, we focus on performance analysis of real-time systems and therefore we have chosen the deterministic approach. In particular, in order to model clearly the synchronization involved in concurrent systems, the Petri net model [16] is chosen.

In this approach, the system to be studied is first modeled by a Petri net. Based on the Petri net model, a given system is classified as either (Fig. 1): 1) a consistent system; or 2) an inconsistent system (the definitions are given in later sections of the paper). Most real-world systems fall into the first class and so we focus our discussion on consistent systems. Due to the difference in complexity involved in the performance analyses of different types of consistent systems, they are further subclassified into: i) decision-free systems; ii) safe persistent systems; and iii) general systems. Procedures for predicting and verifying the system performance of all three types are presented. It is found that the computational complexity involved increases in the same order as they are listed above.

The paper is divided into four sections. In Section II, a brief introduction to Petri nets is given. In Section III, consistent systems, decision-free systems, safe persistent systems, and general systems are defined. Analysis techniques for each type of systems are discussed. Examples are used extensively to illustrate the realism and the applicability of the approaches. Finally, in Section IV, the results are summarized and some areas of future research are discussed.

II. An Introduction to Petri Nets

Petri nets [16], [1] are a formal graph model for modeling the flow of information and control in systems, especially those which exhibit asynchronous and concurrent properties. A Petri net contains two types of nodes (Fig. 2): the circles (called *places*) represent conditions and the bars (called *transitions*) represent events. A black dot (called a *token*) at a place indicates the holding of the condition of the place. A pattern of tokens in a Petri net (called a *marking*) represents the state of the system. For example, Fig. 2 models a communication protocol between two processes. The process on the left is ready to send and the process on the right is ready to receive.

To model the dynamic behavior of a system, the execution of a process is represented by the firing of the corresponding transition. The changes in system state are represented by the movements of tokens in the net. The firing rules of Petri nets are as follows.

1) A transition is enabled if and only if each of its input places has at least one token.
2) A transition can fire only if it is enabled.
3) When a transition fires:
   i) a token is removed from each of its input places; and
   ii) a token is deposited into each of its output places.

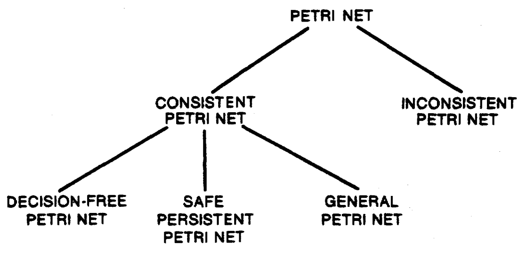


Fig. 1. System classification.

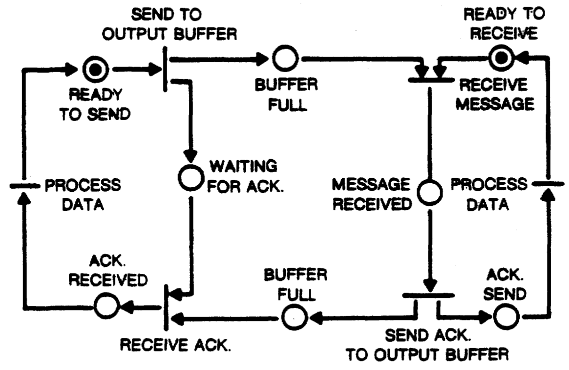


Fig. 2. Petri net model of a communication protocol between two processes.

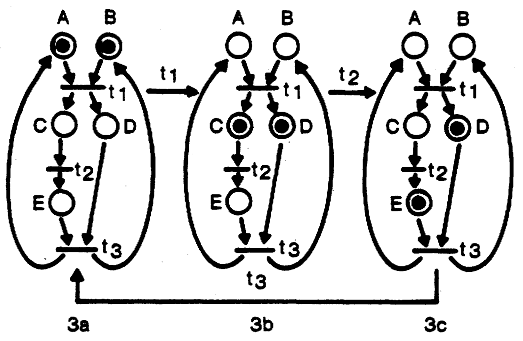


Fig. 3. Execution of Petri net.

Fig. 3 shows the execution of a Petri net. At the beginning [Fig. 3(a)], transition t_1 is enabled because both of its input places, A and B, have a token in them. Firing transition t_1 removes one token from places A and B, and deposits a token into each of its output places, namely, C and D [Fig. 3(b)]. At this point, transition t_3 remains disabled because one of its input place, E, still has no token in it. However, transition t_2 is enabled. Firing t_2 removes a token from place C and deposits a token into place E [Fig. 3(c)]. Now transition t_3 is enabled. Firing transition t_3 removes a token from places D and E and deposits a token into places A and B returning the system to its initial configuration.

A. Control Flow Analyses

Petri nets have been used extensively to study the control flow of computer systems. By analyzing the liveness, boundedness, and proper termination properties of the Petri net model of a computer system, many desirable properties of the system can be unveiled.

A Petri net is *live* [9], [11] if there always exists a firing sequence to fire each transition in the net. By proving that the Petri net is live, the system is guaranteed to be deadlock free.

A Petri net is *bounded* [12], [14] if for each place in the next, there exists an upper bound to the number of tokens that can be there simultaneously. If tokens are used to represent intermediate results generated in a system, by proving that the Petri net model of the system is bounded, the amount of buffer space required between asynchronous processes can be determined and therefore information loss due to buffer overflow can be avoided. If the upper bound on the number of tokens at each place is one, then the Petri net is *safe*. Programming constructs like critical regions [3] and monitors [4], [10] can be modeled by safe Petri nets.

A Petri net is *properly terminating* [8], [17] if the Petri net always terminates in a well-defined manner such that no tokens are left in the net. By verifying that the Petri net is properly terminated, the system is guaranteed to function in a well behaved manner without any side effects on the next initiation.

B. Extended Timed Petri Nets

In order to study the performance of a system, the Petri net model is extended to include the notion of time [18]. In such extended nets, an execution time r is associated with each transition. When a transition initiates its execution, it takes r units of time to complete its execution. With the extended Petri net model, the performance of a computer system can be studied.

For example, Fig. 4 shows a simple computer system with a processor and two tape units. The input queue contains a set of similar tasks to be processed by the system. Each task first requires some computation by the processor together with a tape unit for r_1 units of time. Then the system does some input and output on the tape unit for r_2 time units while the processor continues its computation for r_3 time units. After that, the processor works on the task with the tape unit for r_4 time units. The processor is then ready to start the next task with the other tape unit while the current tape unit is being rewound. Strictly speaking, Fig. 4 is not a Petri net as queues are not defined in the Petri net model. However, when the system performance is evaluated, only the control portion of a system (i.e., the portion inside the square in Fig. 4) is used and therefore is a Petri net. For the rest of this paper, queues will be used to represent the system structure but the readers must keep in mind that only the control portion of a system is used in the evaluation of the system performance.

The problem studied in this paper is to find the maximum performance of a system, i.e., to find the minimum cycle time (for processing a task) of a system. As pointed out in the introduction, different computational complexities are involved in the analyses of systems of different types. The approaches for analyzing each type of system are studied separately in detail in the following section. Before we come to the analyses, some definitions are in order.

Definition: In a Petri net, a sequence of places and transitions, $P_1 t_1 P_2 t_2 \cdots P_n$, is a *directed path* from P_1 to P_n if transition t_i is both an output transition of place P_i and an input transition of place $P_{(i+1)}$ for $1 \leqslant i \leqslant n-1$.

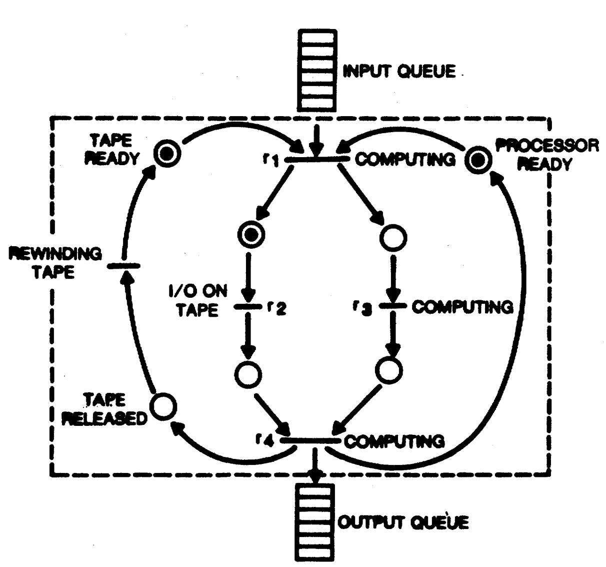


Fig. 4. Petri net model of a computer configuration.

Definition: In a Petri net, a sequence of places and transitions, $P_1 t_1 P_2 t_2 \cdots P_n$, is a *directed circuit* if $P_1 t_1 P_2 t_2 \cdots P_n$ is a directed path from P_1 to P_n and P_1 equals P_n.

Definition: A Petri net is *strongly connected* if every pair of places is contained in a directed circuit.

In the next section, we discuss performance analysis techniques for strongly connected nonterminating [14] Petri nets. Extensions to analyze weakly connected Petri nets are quite straightforward and are left to the readers.

III. Performance Evaluation

A. Consistent and Inconsistent Systems

The first step involved in our approach to analyze the performance of a system is to model it by a Petri net. A system is a consistent (inconsistent) system if its Petri net model is consistent (inconsistent). A Petri net is *consistent* (condition A) if and only if there exists a nonzero integer assignment to its transitions such that at every place, the sum of integers assigned to its input transitions equals the sum of integers assigned to its output transitions; otherwise, the system is inconsistent. If a transition has n input arcs to a place, it is counted as n input transitions to that place.

Fig. 5(a) is an inconsistent system and Fig. 5(b) is a consistent system. In Fig. 5(a), there does not exist an integer assignment to its transitions to satisfy condition A. This can be verified by assigning an integer variable to each transition and getting a contradiction in trying to solve the simultaneous equations provided by condition A:

$$\begin{array}{lll} \text{for place } A: & x + y = z & (1) \\ \text{for place } B: & x = z & (2) \\ \text{for place } C: & y = z & (3) \\ (2) + (3) & x + y = 2z & (4) \end{array}$$

and therefore (4) contradicts (1).

Fig. 5(b) is a consistent Petri net. If each transition is assigned an integer of value 1, condition A is satisfied.

The practical implication behind this system classification is that the integer assigned to a transition is the relative num-

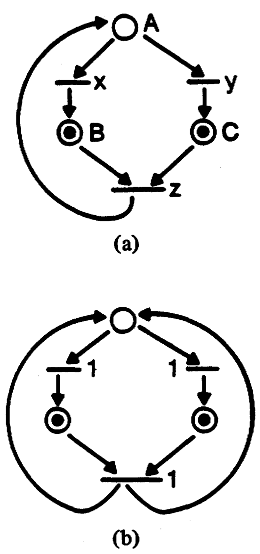


Fig. 5. (a) An inconsistent system. (b) A consistent system.

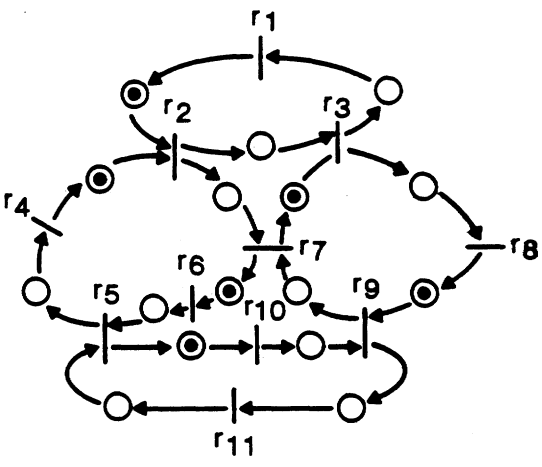


Fig. 6. Petri net model of a train configuration.

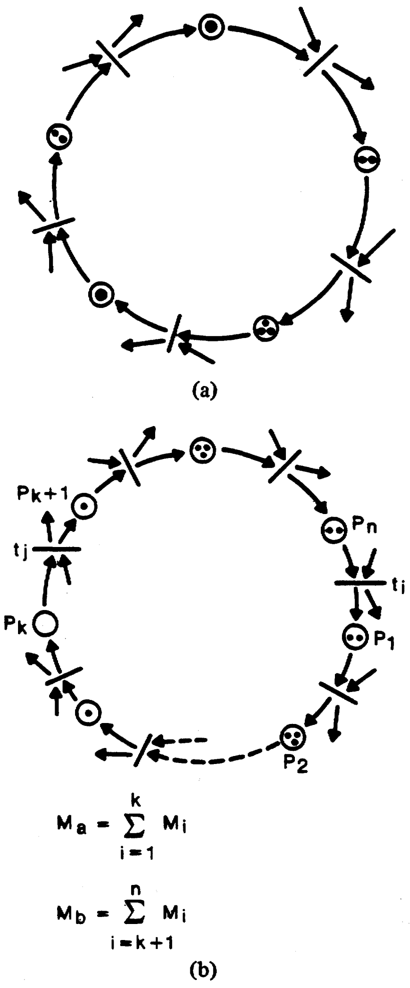


Fig. 7. (a) A cycle in a decision-free Petri net.

ber of executions of that transition in a cycle. If a system is live and consistent, the system goes back to its initial configuration (state) after each cycle and then repeats itself. If a system is inconsistent, either it produces an infinite number of tokens (i.e., it needs infinite resources) or consumes tokens and eventually comes to a stop. Most real-world systems which function continuously with finite amount of resources fall into the class of consistent systems. For the rest of this paper, we focus our discussion on consistent systems and further subclassify them into decision-free systems, persistent systems, and general systems. Performance analysis techniques for each subclass are discussed in the following subsections.

B. Decision-Free Systems

A system is a *decision-free* system if its Petri net model is a decision-free Petri net (also known as marked graph [5], [15].) A Petri net is decision-free if and only if for each place in the net, there is one input arc and one output arc. This means that tokens at a given place are generated by a predefined transition (its only input transition) and consumed by a predefined transition (its only output transition).

The computer configuration shown in Fig. 4 is a decision-free system. The train system shown in Fig. 6 is another decision-free system. The tokens in the net are used to represent trains. For the convenience of the passengers, trains wait at stations for the next train to arrive so as to allow passengers to transfer between trains before leaving stations. Similarly, the chaining operations in the CRAY-1 computer [20] can be modeled by a decision-free Petri net as shown in Fig. 6. The results issued from one functional unit are immediately fed into another functional unit and so on. For a decision-free system, the maximum performance can be computed quite easily. However, before we come to that result, we need the following two theorems.

Theorem 1: For a decision-free Petri net, the number of tokens in a circuit remains the same after any firing sequence.

Proof: Without loss in generality, a circuit containing five places and five transitions in a decision-free Petri net is shown in Fig. 7(a). Tokens in the circuit can only be produced or consumed by transitions in the circuit. When a transition consumes a token, it produces one back into the circuit; therefore, the number of tokens in a circuit remains the same after any firing sequence. Q.E.D.

This result has been proven by many researchers [5], [15]. The proof is included here just for the completeness of this paper.

Definition: Let $S_{i(n_i)}$ be the time at which transition t_i

initiates its n_ith execution. The *cycle time* C_i of transition t_i is defined as

$$\lim_{n_i \to \infty} \frac{S_i(n_i)}{n_i}.$$

Theorem 2: All transitions in a decision-free Petri net have the same cycle time.

Proof: For any two transitions t_i and t_j in a decision-free Petri net, choose a circuit that contains both transitions. Such a circuit must exist because the net is strongly connected. Without loss in generality, assume that there are M_i tokens in place i in the initial marking, Fig. 7(b). Let

$$M_a = \sum_{i=1}^{k} M_i \quad \text{and} \quad M_b = \sum_{i=k+1}^{n} M_i.$$

At time $S_{i(n_i)}$,

$$n_i - M_b \leqslant \text{number of initiations of transition } t_j \leqslant n_i - M_a$$

$$\lim_{n_i \to \infty} \frac{S_i(n_i)}{n_i - M_b} \geqslant C_j \geqslant \lim_{n_i \to \infty} \frac{S_i(n_i)}{n_i - M_a}.$$

Since M_a and M_b are finite, as $n_i \to \infty$, the left- and right-hand side expressions approach C_i, i.e.,

$$C_i \geqslant C_j \geqslant C_i$$

$$C_i = C_j.$$

Therefore, all transitions in a decision-free Petri net have the same cycle time C. Q.E.D.

Theorem 3: For a decision-free Petri net, the minmum cycle time (maximum performance) C is given by

$$C = \max\left\{\frac{T_k}{N_k}: k = 1, 2, \cdots, q\right\}$$

such that $S_{i(n_i)} = a_i + Cn_i$ where

$T_k = \sum_{t_i \in L_k} r_i$ = sum of the execution times of the transition in circuit k

$N_k = \sum_{P_i \in L_k} M_i$ = total number of tokens in the places in circuit k

q = number of circuits in the net

a_i = constant associated with transition t_i

L_k = loop (circuit) k

M_i = number of tokens in place P_i.

Similar results have been obtained in [19], [15]. The proof given here uses a graph theoretical approach and is different from the previous approaches. Based on this approach, we develop a very fast procedure to verify the performance of a system.

Before we prove Theorem 3, we would like to give an example of the usage of Theorem 3 to clarify our notations. The computer configuration with the execution time of each transition is shown in Fig. 8. According to Theorem 3, for

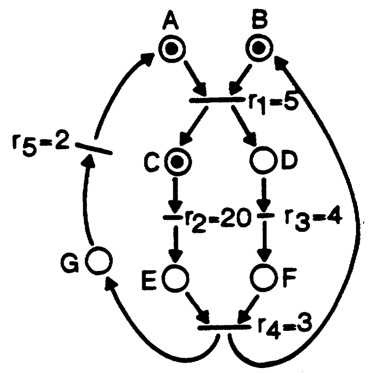


Fig. 8. Computer configuration with execution times.

circuit $At_1Ct_2Et_4Gt_5$: $\dfrac{T_k}{N_k} = \dfrac{5+20+3+2}{2} = 15$

circuit $At_1Dt_3Ft_4$: $\dfrac{T_k}{N_k} = \dfrac{5+4+3+2}{1} = 14$

circuit $Bt_1Ct_2Et_4$: $\dfrac{T_k}{N_k} = \dfrac{5+20+3}{2} = 14$

circuit $Bt_1Dt_3Ft_4$: $\dfrac{T_k}{N_k} = \dfrac{5+4+3}{1} = 12.$

By enumerating all circuits in the net, the minimum cycle time is 15.

Proof of Theorem 3: The proof is in two parts.

a) Minimum cycle time, $C \geqslant \max\{T_k/N_k, k = 1, 2, \cdots, q\}$.

b) For $C = \max\{T_k/N_k, k = 1, 2, \cdots, q\}$ there exists a_i, such that $S_i(n_i) = a_i + Cn_i$ and the firing rules are not violated.

Proof of a):

number of transitions that are enabled simultaneously $\leqslant$ number of tokens in circuit $= N_k$ (Theorem 1)

processing power required by circuit per cycle $= T_k = \sum_{t_i \in L_k} r_i$ $\leqslant$ maximum processing power of the circuit per cycle time $= CN_k$.

Therefore,

$$T_k \leqslant CN_k \quad \text{for every circuit}$$

and

$$C \geqslant \max\left\{\frac{T_k}{N_k}, k = 1, 2, \cdots, q\right\}.$$

Lemma: For $C = \max\{T_k/N_k, k = 1, 2, \cdots, q\}$

$$0 \geqslant T_k - CN_k \quad \text{for all circuit } k.$$

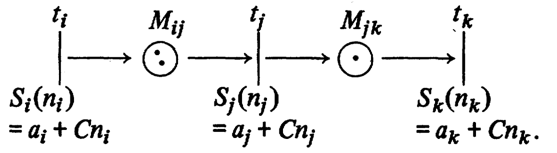
Proof:

$$C \geqslant \frac{T_k}{N_k} \quad k = 1, 2, \cdots, q$$

$$0 \geqslant T_k - CN_k.$$

Proof of b): Let

$$C = \max\left\{\frac{T_k}{N_k}, k = 1, 2, \cdots, q\right\}$$



In order not to violate the firing rules:

finish time of the n_ith execution of transition $t_i \leqslant$ initiation time of the $n_i + M_{ij}$ execution of transition t_j

$$S_i(n_i) + r_i \leqslant S_j(n_i + M_{ij})$$

$$a_i + Cn_i + r_i \leqslant a_j + C(n_i + M_{ij})$$

$$a_i - CM_{ij} + r_i \leqslant a_j. \tag{1}$$

Similarly,

$$a_j - CM_{jk} + r_j \leqslant a_k \tag{2}$$

$$(1) + (2) \quad a_i - C(M_{ij} + M_{jk}) + r_i + r_j \leqslant a_k.$$

In general,

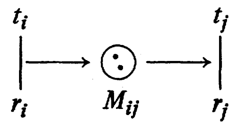
$$a_i - C \sum_{(u,v)\in R} M_{uv} + \sum_{w\in R} r_w \leqslant a_s \tag{3}$$

where R is a path from transition i to transition s. In order not to violate the firing rules, we have to find a_i's such that (3) is satisfied.

Procedure for assigning a_i's such that

$$a_i - C \sum_{(u,v)\in R} M_{uv} + \sum_{w\in R} r_w \leqslant a_s. \tag{3}$$

1) Define the distance from transition t_i to transition t_j (t_i adjacent to t_j) to be $r_i - CM_{ij}$:



2) Find a transition t_s, which is enabled initially and assign 0 to a_s.

3) Assign a_u to each transition t_u such that a_u is the greatest distance from t_s to t_u,

i.e., $a_u = \max\left\{\sum_{w\in R} r_w - C \sum_{(u,v)\in R} M_{uv}\right\}$

where R is a path from t_s to t_u.

Such an assignment of a_i's exists because by the lemma, $T_k - CN_k \leqslant 0$, the greatest distance between any two nodes is finite and the corresponding path would never contain a loop. Q.E.D.

A drawback of the above approach is that all circuits in the net must be enumerated; this can be very tedious. In the design of computer systems, the required performance is usually given. With this information, the performance of a system can be verified very efficiently. By the lemma, the performance requirement (expressed in cycle time C) can be satisfied if and only if $CN_k - T_k \geqslant 0$ for all circuits. This can be verified by the following procedure.

A Procedure for Verifying System Performance:

1) Express the token loading in an $n \times n$ matrix P, where n is the number of places in the Petri net model of the system. Entry (A, B) in the matrix equals x if there are x tokens in place A, and place A is connected directly to place B by a transition; otherwise (A, B) equals 0. Matrix P of the example system in Fig. 8 is shown below:

	A	B	C	D	E	F	G
A	0	0	1	1	0	0	0
B	0	0	1	1	0	0	0
C	0	0	0	0	1	0	0
D	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0

Matrix P

2) Express transition time in an $n \times n$ matrix Q. Entry (A,B) in the matrix equals to r_i (execution time of transition i) if A is an input place of transition i and B is one of its output places. Entry (A,B) contains the symbol "w" if A and B are not connected directly as described above. Matrix Q for the example system is

	A	B	C	D	E	F	G
A	w	w	5	5	w	w	w
B	w	w	5	5	w	w	w
C	w	w	w	w	20	w	w
D	w	w	w	w	w	4	w
E	w	3	w	w	w	w	3
F	w	3	w	w	w	w	3
G	2	w	w	w	w	w	w

Matrix Q

3) Compute matrix $CP - Q$ (with $n - w = \infty$ for $n \in N$), then use Floyd's algorithm [7] to compute the shortest distance between every pair of nodes using matrix $CP - Q$ as the distance matrix. The result is stored in matrix S. There are three cases.

a) All diagonal entries of matrix S are positive (i.e.,

$CN_k - T_k > 0$ for all circuits)—the system performance is higher than the given requirement.

b) Some diagonal entries of matrix S are zero's and the rest are positive (i.e., $CN_k - T_k = 0$ for some circuits and $CN_k - T_k > 0$ for the other circuits)—the system performance just meets the given requirement.

c) Some diagonal entries of matrix S are negative (i.e., $CN_k - T_k < 0$ for some circuits)—the system performance is lower than the given requirement.

In the example, for $C = 15$, $CP - Q$ is

	A	B	C	D	E	F	G
A	∞	∞	10	10	∞	∞	∞
B	∞	∞	10	10	∞	∞	∞
C	∞	∞	∞	∞	-5	∞	∞
D	∞	∞	∞	∞	∞	-4	∞
E	∞	-3	∞	∞	∞	∞	-3
F	∞	-3	∞	∞	∞	∞	-3
G	-2	∞	∞	∞	∞	∞	∞

After applying Floyd's algorithm to find the shortest distance between every pair of places, we have

	A	B	C	D	E	F	G
A	0	2	10	10	5	6	2
B	0	2	10	10	5	6	2
C	-10	-8	0	0	-5	-4	-8
D	-9	-7	1	1	-4	-4	-7
E	-5	-3	5	5	0	1	-3
F	-5	-3	5	5	0	1	-3
G	-2	0	8	8	3	4	0

Matrix S

Since the diagonal entries are nonnegative, the performance requirement of $C = 15$ is satisfied. Moreover, since entries (A, A), (C, C), (E, E), and (G, G) are zero's, $C = 15$ is optimal (i.e., it is the minimum cycle time). In addition, when a decision-free system runs at its highest speed, CN_k equals to T_k for the bottleneck circuit. This implies that the places that are in the bottleneck circuit will have zero diagonal entries in matrix S. In the example, the bottleneck circuit is $At_1Ct_2Et_4Gt_5$. With this information, the system performance can be improved by either reducing the execution times of some transitions in the circuit (by using faster facilities) or by introducing more concurrency in the circuit (by introducing more tokens in the circuit). Which approach should be taken is application dependent and beyond the scope of this paper.

The above procedure can be executed quite fast. The formulation of matrices P and Q takes $0(n^2)$ steps. The Floyd algorithm takes $0(n^3)$ steps. As a whole, the procedure can be executed in $0(n^3)$ steps. Therefore, the performance requirement of a decision-free system can be verified quite efficiently.

C. Safe Persistent Systems

A system is a safe persistent system if its Petri net model is a safe persistent Petri net. A Petri net is a safe persistent Petri net if and only if it is a safe Petri net and for all reachable markings, a transition is disabled only by firing the transition. It differs from a decision-free Petri net in that it may have more than one input (output) arcs to (from) a place. However, like a decision-free Petri net, it models a deterministic system. In a persistent Petri net, if a token enables a transition, it will be consumed by that transition only, i.e., a token will never enable two or more transitions simultaneously. As a result, a safe persistent Petri net can always be transformed into a decision-free Petri net.

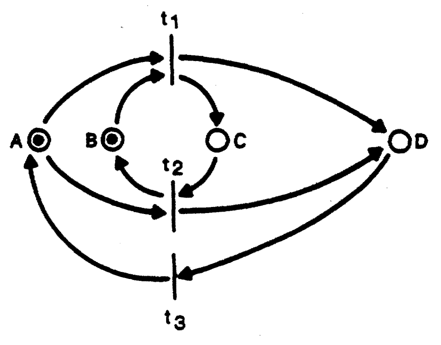
Fig. 9(a) shows a persistent Petri net. It models the operations of a double buffer input port. Transitions t_1 and t_2 represent fetching the contents of buffer 1 and buffer 2, respectively. Transition t_3 represents storing the input into the memory. To compute the performance of the system, we first transform it into a decision-free system and then use the algorithm discussed in the previous subsections to compute the system performance.

A persistent Petri net can be transformed into a decision-free Petri net by tracing the execution of the system for one cycle. For example, Fig. 9(b) is the decision-free system corresponding to the persistent Petri net shown in Fig. 9(a). Places A_1 and A_2 in Fig. 9(b) represent two different occurrences of place A in Fig. 9(a) in a cycle. Condition A_1 holds when transition t_1 is enabled and condition A_2 holds when transition t_2 is enabled. Similarly, place D is duplicated into D_1 and D_2. Condition D_1 holds when transition t_3 is enabled and transition t_2 will be enabled after firing t_3. Condition D_2 holds when transition t_3 is enabled and transition t_1 will be enabled after firing t_3.

Initially, there is a token in places A and B and transition t_1 is enabled in Fig. 9(a), and therefore there is a token in places A_1 and B in Fig. 9(b). After firing t_1, a token is deposited into places C and D in Fig. 9(a). This is represented by depositing a token in places D_1 and C in Fig. 9(c). By following the execution of the system for a cycle, the corresponding decision-free system can be generated. The system performance can then be computed by the procedure discussed in Section III-B.

D. General Systems

A system is a general system if its Petri net model is a general Petri net. A Petri net is a *general Petri net* if it is a consistent Petri net and there exists a reachable marking such that the firing of a transition disables some other transitions. Figs. 10 and 11 show two general Petri nets. Fig. 10 models the communication protocol from process P_1 to processes P_2 and P_3, such that the difference in the number of messages sent from P_1 to P_2 and P_3 is always less than three. It is not a decision-free Petri net because place A has more than one



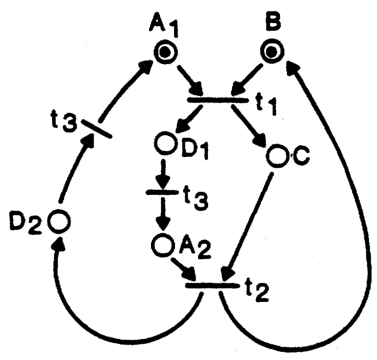


Fig. 9. (a) A persistent system. (b) The decision-free system corresponding to (a).

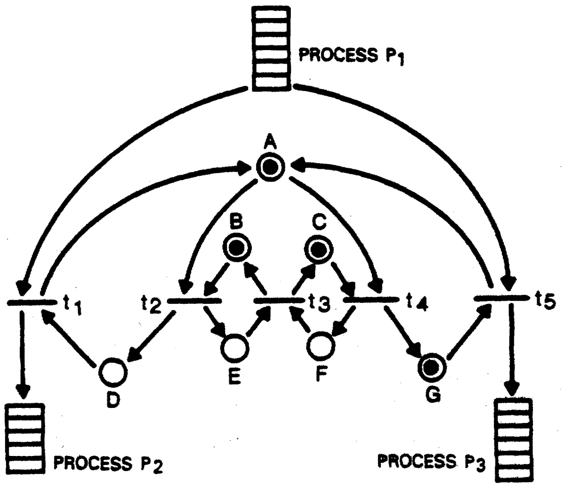


Fig. 10. A general system (a communication protocol).

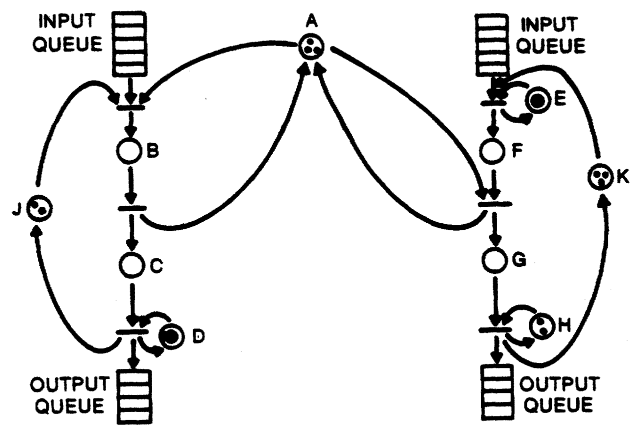


Fig. 11. A general system (shared resource pipelines).

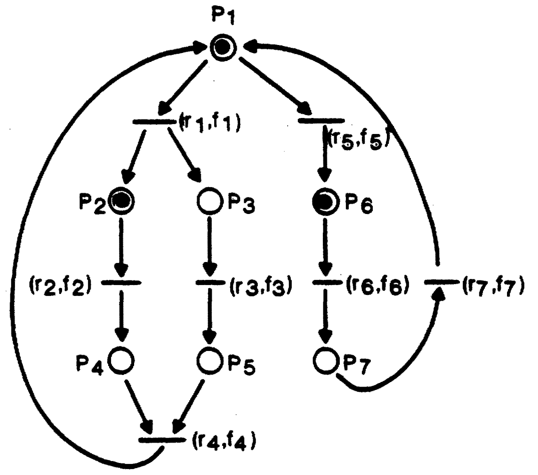


Fig. 12. A general system with specified execution times and relative firing frequencies.

input and output arcs. It is not a persistent Petri net because, in the configuration shown, the execution of either transition t_2 or transition t_4 disables the other transition. This introduces the nondeterministic characteristic of a system.

Fig. 11 models two shared resource pipelines. Place A represents the condition that resource A is free. The system has three units of resource A and they are used by the first and second stages of the pipeline on the left, and by the second stage of the pipeline on the right. Places J and K represent the conditions that buffers for the left and right hand side pipelines are free, respectively. By the same reasons given for Fig. 10, it is a general system.

Fig. 12 shows another general Petri net. Tokens at place P_1 can be consumed by either transition t_1 or t_5. In addition to this, the execution frequencies of the two transitions are independent of each other. This introduces another degree of freedom into the system. In order to fully specify the dynamic characteristics of the system, the number of executions of each transition per cycle has to be defined. In the example, it is specified by the ordered pair, (r_i, f_i), where r_i is the execution time and f_i is the execution frequency per cycle of transition t_i. General systems are very difficult to analyze. In the next theorem, we show that it is unlikely that a fast algorithm exists to verify the performance of a general system. A method of computing the upper and lower bounds of the performance of a conservative general system [14] is proposed. For a nonconservative general system, no good heuristics are known to the authors and further research is needed.

Theorem 4: Verifying the performance of a conservative general Petri net is an *NP*-complete problem [12].

Proof:

i) It is in *NP* because we can guess the optimal schedule (execution sequence). The nondeterminisms of the general Petri net are resolved and therefore the general Petri net can be transformed into a decision-free Petri net according to the guessed schedule. The performance can then be verified by the procedure discussed in Section III-B.

ii) It is *NP*-complete because the set partitioning problem (an *NP*-complete problem) can be reduced to the above problem.

The Set Partition Problem [13]: Given a set of integers, $S = \{x_1, x_2, \cdots, x_n\}$, partition it into two subsets, S_1 and S_2 such that (condition B)

$$S = S_1 \cup S_2 \quad \text{and} \quad S_1 \cap S_2 = \phi$$

$$\text{and} \sum_{i \in S_1} x_i = \sum_{i \in S_2} x_i = \frac{\sum_{i=1}^{n} x_i}{2}.$$

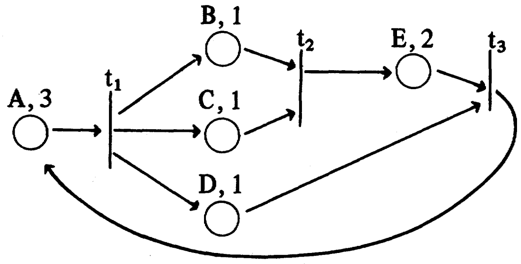
Reduction: Given $S = \{x_1, x_2, \cdots, x_n\}$, generate the general Petri net shown in Fig. 13. It is easy to see that the system has minimum cycle time, $C = (\Sigma_{i=1}^{n} x_i)/2$, if and only if the set S satisfies condition B.

A Method to Compute Upper and Lower Bounds of the Performance of a Conservative General System:

1) Upper Bound: We choose a schedule which satisfies the execution frequency requirement and then use the algorithm discussed in Section III-B to find the cycle time of the system.

2) Lower Bound (Fig. 12):

a) Find a nonzero integer assignment to the places such that the sum of integers assigned to the input places of a transition equals the sum of integers assigned to the output places. Such an integer assignment must exist because the system is conservative [14]. Intuitively, the integer assigned to a place represents the relative processing capability of a token at that place. The weighted execution time of a transition is equal to the product of the transition execution time and the sum of the integers assigned to its input places. For example,



A token at place E has twice the processing capability of a token at either place B, C, or D and transition t_2 has weighted execution time $r_2(1 + 1)$.

b) Assume that all tokens in the net are busy all the time. Then

$$C \times \text{weighted processing capability} \geq \text{sum of weighted execution time}$$

$$C \times \Sigma s_i M_i \geq \Sigma s_i f_i r_i$$

$$C \geq \frac{\Sigma s_i f_i r_i}{\Sigma s_i M_i}$$

where s_i = sum of integers assigned to the input places of transition i.

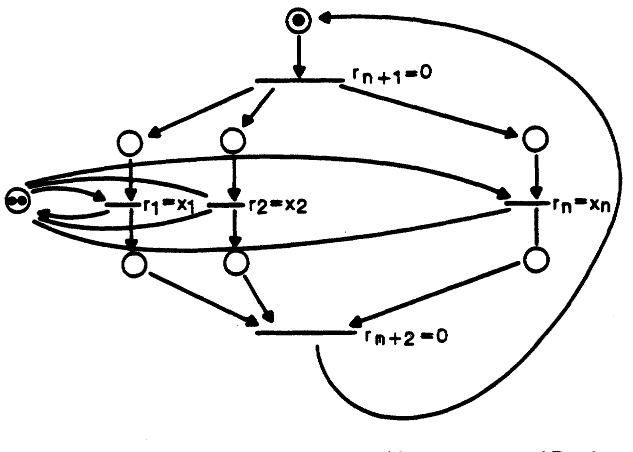


Fig. 13. Reduction of the set partition problem to a general Petri net.

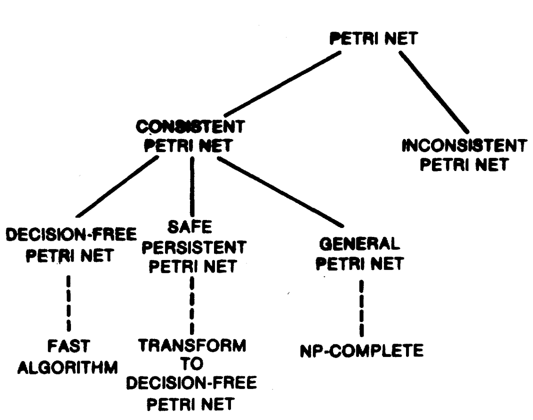


Fig. 14. Summary of results.

IV. Conclusions

In this paper, we have discussed a systematic method to evaluate and verify the performance of concurrent systems. The system to be studied is first modeled by a Petri net. Based on the Petri net model, the system is classified into either 1) a consistent system or 2) an inconsistent system. A consistent system is further subclassified into: i) a decision-free system; ii) a safe persistent system; and iii) a general system. The system classification and the results are summarized in Fig. 14. The performance of decision-free systems and safe persistent systems can be computed quite efficiently. In the case of general systems, we have proven that the verification of system performance is *NP*-complete. An approach for computing the upper and lower bounds of the performance of a conservative general system is proposed. However, the bounds produced may be loose. For a nonconservative general system, no good heuristics are known. Further research is needed.

Another difficulty that may arise in the proposed approach is the inaccuracy in estimating the execution times of the processes in a system. However, in real-time systems such as the air traffic control systems, chemical plant control systems, nuclear power plant control systems, etc., the execution times of the processes may be predicted quite accurately. In the case that the execution times cannot be estimated accurately, the worst case execution times of the processes can be used. The performance prediction obtained will be the worst case system performance.

Acknowledgment

We would also like to thank Dr. C. R. Vick and J. E. Scalf for many helpful discussions related to this work.

References

[1] T. Agerwala and M. J. Flynn, "On the completeness of representation schemes for concurrent systems," presented at the Conf. Petri Nets and Related Methods, M.I.T., Cambridge, MA, July 1975.

[2] F. Baskett, K. M. Chandy, R. R. Muntz, and F. Palacios-Gomez, "Open, closed and mixed networks of queues with different classes of customers," *J. Ass. Comput. Mach.*, vol. 22, Apr. 1975.

[3] P. Brinch Hansen, "Structured multiprogramming," *Commun. Ass. Comput. Mach.*, vol. 15, July 1972.

[4] P. Brinch Hansen, "Concurrent programming concepts," *Computing Surveys*, vol. 5, Dec. 1973.

[5] F. Commoner *et al.*, "Marked directed graphs," *J. Comput. Syst. Sci.*, vol. 5, 1971.

[6] D. Ferrari, *Computer Systems Performance Evaluation.* Englewood Cliffs, NJ: Prentice-Hall, 1978.

[7] R. W. Floyd, "Algorithm 97, shortest path," *Commun. Ass. Comput. Mach.*, vol. 5, 1962.

[8] K. P. Gostelow, "Flow of control, resource allocation, and the proper termination of programs," Ph.D. dissertation, School Eng. Appl. Sci., Univ. California, Los Angeles, Dec. 1971.

[9] M. Hack, "Decidability questions for Petri nets," Ph.D. dissertation, Dep. Elec. Eng., M.I.T., Cambridge, MA, Dec. 1975.

[10] C. A. R. Hoare, "Monitors: An operating system structuring concept," *Commun. Ass. Comput. Mach.*, vol. 17, Oct. 1974.

[11] R. C. Holt, "On deadlock in computer systems," Ph.D. dissertation, Dep. Comput. Sci., Cornell Univ., Ithaca, NY, Jan. 1971.

[12] R. M. Karp and R. E. Miller, "Properties of a model for parallel computation: Determinacy, termination, queuing," *SIAM J. Appl. Math.*, vol. 14, Nov. 1966.

[13] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations.* New York: Plenum, 1972.

[14] Y. E. Lien, "Termination properties of generalized Petri nets," *SIAM J. Comput.*, vol. 5, no. 2, June 1976.

[15] T. Murata, "Petri nets, marked graphs, and circuit-system theory," *Circuits Syst.*, vol. 11, June 1977.

[16] J. L. Peterson, "Petri nets," *Comput. Surveys*, vol. 9, Sept. 1977.

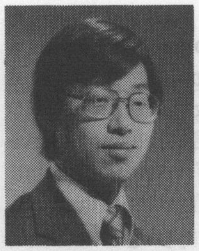
[17] J. B. Postel, "A graph model analysis of computer communication protocols," Ph.D. dissertation, Dep. Comput. Sci., Univ. California, Los Angeles, Jan. 1974.

[18] C. Ramchandani, "Analysis of asynchronous concurrent systems by Petri nets," Project MAC, TR-120, M.I.T., Cambridge, MA, 1974.

[19] R. Reiter, "Scheduling parallel computations," *J. Ass. Comput. Mach.*, vol. 15, Oct. 1968.

[20] R. M. Russell, "The CRAY-1 computer system," *Commun. Ass. Comput. Mach.*, vol. 21, Jan. 1978.

C. V. Ramamoorthy (M'57–SM'76–F'78), for a photograph and biography, see page 117 of the March 1980 issue of this TRANSACTIONS.



Gary S. Ho (S'77–M'79) was born in Hong Kong on March 2, 1953. He received the B.S., M.S., and Ph.D. degrees in computer science from the University of California, Berkeley, in 1975, 1977, and 1979 respectively.

Currently he is working for Bell Laboratories, Indian Hill, IL. His research interests include distributed computer system, design methodology, performance evaluation, and computational complexity.

Dr. Ho is a member of the Association for Computing Machinery.