

Stochastic Petri Net Modeling of VAXcluster System Availability

Oliver C. Ibe Archana Sathaye Richard C. Howe
 Digital Equipment Corporation
 6 Tech Drive
 Andover, MA 01810

Kishor S. Trivedi
 Computer Science Dept.
 Duke University
 Durham, NC 27706

Abstract

A VAXcluster is a closely-coupled multicomputer system that consists of two or more VAX computers, one or more hierarchical storage controllers (HSCs), two or more discs and a star coupler. The Markov model of VAXcluster system availability suffers from state space explosion as the number of VAX computers in the cluster increases. This has necessitated the use of approximate Markov models. In this paper we develop exact availability models of VAXcluster systems by means of stochastic Petri nets.

1 Introduction

A VAXcluster is a closely-coupled multicomputer system with two or more VAX computers, one or more mass storage servers called Hierarchical Storage Controllers (HSCs), a set of disks and a star coupler [1]. From a reliability and availability point of view, a VAXcluster system (or cluster for short) is essentially a series-parallel system consisting of a parallel network of N VAX computers in series with a parallel network of HSCs and a parallel network of discs. The star coupler is a passive device and is generally assumed not to fail. A VAXcluster system with two VAX computers, two HSCs and two discs is shown in Figure 1, and its reliability block diagram is shown in Figure 2.

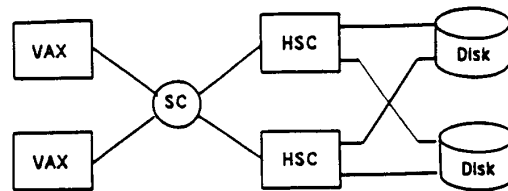


Figure 1: Physical Configuration of a Two-Computer VAXcluster

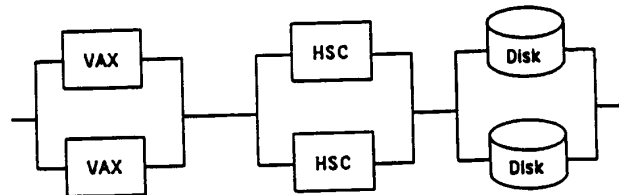


Figure 2: Reliability Block Diagram for the Two-Computer Cluster

One of the dependability measures of VAXcluster systems that has been greatly studied is the cluster *availability*, which is defined as the long-run probability that the cluster is operational (or up). For ease of analysis it is usually assumed that the failure times, reboot times, repair times and other times associated with events in the cluster are exponentially distributed. This leads to modeling the cluster availability by a continuous-time Markov chain. However, as the

number of nodes (i.e., VAX computers, HSCs and disks) increases, the number of states in the Markov chain increases quadratically, creating an extremely large number of states for large-scale clusters.

There are two principal approaches to deal with the largeness of the Markov chain state space. In the approach we call *largeness avoidance*, the generation and the solution of a large Markov model are avoided. This commonly involves the use of model hierarchies and often (but not always) implies an approximate rather than an exact solution to the original modeling problem [2]. An alternative approach is called *largeness tolerance*. Here we accept the fact that a large Markov model needs to be generated and solved. However, we automate the generation of the large Markov model. This can be done in several ways one of which is to use the stochastic Petri net (SPN) [3] to specify the problem. This usually gives a concise model of the system. Unfortunately, the resultant state space is still large. However, we call it a largeness tolerant scheme because once the model is specified in SPN we can use an SPN package to automatically generate and solve the Markov model. It is this second approach that we follow in this paper. In the balance of the paper we concentrate only on the processor subsystem of the VAXcluster. Since the three subsystems (i.e., processor, HSC and disk subsystems) are essentially connected serially, the availability of the VAXcluster is the product of the probabilities that the subsystems are up.

The paper is organized as follows. In Section 2 we consider the Markov model of a two-VAX computer cluster. In Section 3 we develop the stochastic Petri net model of cluster availability. In Section 4 we consider numerical examples, and in Section 5 we make some concluding remarks.

2 Markov Models of Cluster Availability

We consider a VAXcluster system with N VAX computers (hereafter referred to as an N -processor cluster), where $N \geq 2$. Each processor is either up or down. There are two classes of processor failures: *permanent* failure and *intermittent* failure. A permanent failure is any failure that requires the physical repair of the failed processor, and an intermittent failure is one that can be corrected by rebooting the affected processor. Each of these failure classes results in a cluster *fault*. We refer to a cluster fault that arises from a permanent failure as a *permanent fault* and to a cluster fault that arises from an intermittent failure as an *intermittent fault*. A cluster fault can be *covered* or *uncovered* [4]. A covered fault is one that can be recovered from by means of a *cluster reconfiguration* while an uncovered fault is one that can be recovered from by rebooting the cluster. An uncovered fault causes the cluster to go down until it is rebooted while a covered fault causes only a very brief (on the order of seconds) outage of the cluster.

We assume that the time until an up processor suffers a permanent failure is exponentially distributed with mean $1/\lambda_P$ hours, and the time until it suffers an intermittent failure is exponentially distributed with mean $1/\lambda_I$ hours. Let c and k denote the coverage factors for permanent faults and intermittent faults respectively. When a covered fault occurs, the up processors go through a cluster reconfiguration which results in suspension of all processing activities, canceling the cluster membership of the failed processor, taking a vote to see if the up processors still form a quorum¹, and recommencing processing if a quorum is formed. If a quorum is not formed, the cluster will *hang* and no process-

ing can be done. We assume that the duration of a cluster reconfiguration to map out a failed processor is exponentially distributed with mean $1/\mu_{TOUT}$ hours, the time to repair a processor that suffers a permanent failure is exponentially distributed with mean $1/\mu_P$ hours, the time to reboot a processor that suffers a covered intermittent failure is exponentially distributed with mean $1/\mu_{PB}$ hours, and the time to reboot the cluster when an uncovered intermittent (or permanent) fault occurs is exponentially distributed with mean $1/\mu_{CB}$ hours. After a processor that suffered a covered intermittent failure has been rebooted or a processor that suffered a permanent failure has been repaired, the cluster goes through another reconfiguration to readmit the repaired or rebooted processor. We assume that the duration of a cluster reconfiguration to readmit a repaired or rebooted processor is exponentially distributed with mean $1/\mu_{TIN}$ hours. Thus, we can model the cluster availability by a continuous-time Markov chain.

The Markov chain for the two-processor cluster of Figure 1 is shown in Figure 3 where we have assumed that $\mu_{TOUT} = \mu_{TIN} = \mu_T$. The states of the Markov chain are represented by (abc, d) , where

- a = number of processors that are down with permanent failure
- b = number of processors that are down with intermittent failure

$$c = \begin{cases} 0 & \text{if both processors are up} \\ p & \text{if a processor is being repaired} \\ b & \text{if a processor is being rebooted} \\ c & \text{if the cluster is being rebooted} \\ t & \text{if the cluster is undergoing} \\ & \text{reconfiguration} \\ r & \text{if both processors are rebooting} \\ s & \text{if one processor is rebooting and} \\ & \text{the other is being repaired} \end{cases}$$

$$d = \begin{cases} 0 & \text{if the system is up in this state} \\ 1 & \text{otherwise} \end{cases}$$

Let $P_{abc,d}$ denote the steady-state probability that the process is in state (abc, d) . Then the availability of the cluster is given by:

$$A_1 = P_{000,0} + P_{10p,0} + P_{01b,0} \quad (1)$$

One important comment on the diagram of Figure 3 is that when a processor that suffered an intermittent failure is being rebooted, it can suffer a permanent failure that then calls for a physical repair. This is represented by the transition from

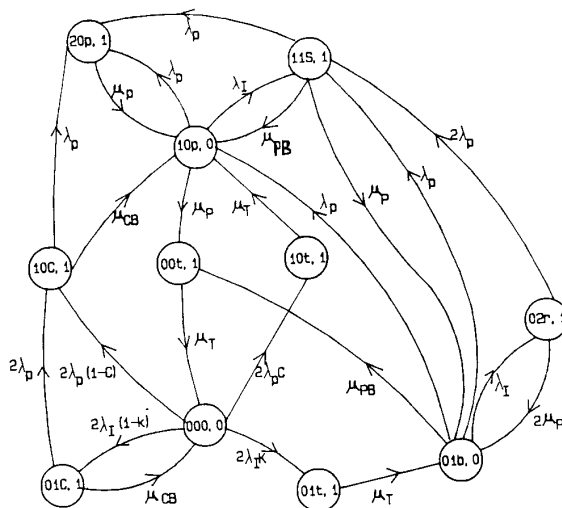


Figure 3: Markov Chain for the Two-Processor Cluster

¹Quorum is the minimum number of processors required for the VAXcluster to function.

state $(01b, 0)$ to state $(10p, 0)$. Such a failure does not disrupt the operation of the cluster since the failed processor has already been mapped out of the cluster.

The availability of the cluster can be obtained by solving the Markov chains directly by any of the standard techniques [5]. However, we solve it by means of SHARPE, a software package for dependability analysis [6].

3 Petri Net Models of Cluster Availability

One of the problems with the Markov model of cluster availability is the fact that as N , the number of processors, increases, the model suffers from state space explosion. Generally also, there is the inherent problem of generating the state space. And when the field service response time is required to be modeled explicitly, the state space becomes more difficult to generate. These facts have led to the development of an approximate model of the system [2]. In this section we consider an alternative way to model the cluster availability; that is, by means of stochastic Petri nets.

Petri nets are formal graph models that are well suited for representing the flow of information and control in systems that exhibit concurrency and synchronization characteristics [7,8,9]. A Petri net can be defined as a four-tuple:

$$PN = (P, T, A, M_0)$$

where

$$\begin{aligned} P &= \{p_1, p_2, \dots, p_n\} \text{ is the set of places} \\ T &= \{t_1, t_2, \dots, t_m\} \text{ is the set of transitions} \\ A &\subseteq \{P \times T\} \cup \{T \times P\} \text{ is the set of arcs} \\ M_0 &= \{m'_1, m'_2, \dots, m'_n\} \text{ is the initial marking} \\ &\quad m'_j = 0, 1, 2, \dots, \quad j \in [1, n] \end{aligned}$$

The places are represented by circles and the transitions are represented by bars. The arcs are directed arcs that connect the places to transitions and the transitions to places: A place may contain *tokens* (represented by dots). The number of tokens contained in each place in the net is usually specified by a tuple called the *marking*. The marking determines the state of the Petri net. A place is defined to be an *input place* of a transition if an arc exists from the place to the transition. Similarly, a place is defined to be an *output place* of a transition if an arc exists from the transition to the place. More than one arc may connect a place to a transition or a transition to a place. The number of arcs connecting a place to a transition is called the *multiplicity* of the input place. A transition is *enabled* if each of its input places contains as many tokens as its multiplicity. An enabled transition can *fire*. When a transition fires, it removes d_i tokens from each input place i of that transition, where d_i is the multiplicity of place i ; and it deposits in each of its output places as many tokens as there are output arcs from the transition to the place. Each firing generates a new marking of the net. The *reachability set* of a Petri net for a given initial marking is the set of all states (or markings) that can be generated from the initial state by a sequence of transition firings. And the *reachability graph* associated with a reachability set is the graph obtained by representing each state by a vertex and placing a directed edge from vertex s_1 to vertex s_2 if state s_2 can be obtained by the firing of some transition enabled in state s_1 .

Some extensions have been introduced to increase the modeling power of the standard Petri net described above. These include *inhibitor arcs* and *multiple arcs*. An inhibitor arc connects a place to a transition, and is represented by a line terminating in a circle rather than an arrow

head. It functions to prevent a transition from firing under certain markings. Thus a transition may fire iff each of its normal input places contains at least one token and none of its inhibitor input places contains any token. Multiple arcs permit the firing of a transition to deposit two or more tokens in an output place. Also, if a transition has an input place with k arcs, where $k > 1$, then the transition cannot fire when that input place contains less than k tokens. Often we need to flush a place of all the tokens it contains. For this purpose variable input/output arcs have been introduced [10]. We also allow for priorities and enabling/disabling functions to be specified on the transitions in the net [10].

Additional extensions of the Petri net have been proposed and this has led to the following types of Petri nets: the timed Petri net [11], the stochastic Petri net [3], the generalized stochastic Petri net (GSPN) [12], the extended stochastic Petri net (ESPN) [13], the stochastic activity networks (SAN) [14], and the generalized timed Petri net (GTPN) [15]. Because of our assumptions of exponentially distributed times for VAX-cluster activities, we use the GSPN model for our work.

The GSPN, which we use in the model described below, divides the transitions into two classes: *immediate* and *timed* transitions [12]. Immediate transitions fire in zero time once they are enabled. The time that elapses from the instant a timed transition is enabled until it fires is assumed to be exponentially distributed. In this paper we represent timed transitions by thick bars, and immediate transitions by small rectangular boxes. Firing rate of timed transitions and firing probabilities of immediate transitions are allowed to be marking dependent.

3.1 Model of a VAXcluster

We first consider the GSPN model of an N -processor VAXcluster that is subject to permanent failures only. Figure 4 shows the GSPN. The place P_{UP} with N tokens represents the initial condition in which the N processors are up. The place P_{PF} represents the condition that a permanent failure has occurred. The place P_{CPF} represents the condition that the failure is covered while the place P_{UPF} represents the condition that the failure is uncovered. The place P_{PRP} represents the condition that the cluster reboot or transition (as the case may be) is over and the failed processor is being repaired. We assume that there is only one repairman for the cluster. The place P_{TIN} represents the condition that the processor repair has been completed.

The firing rate of the timed transition tpf is marking dependent; this fact is denoted by the symbol $\#$ on the arc from P_{UP} to tpf . When a permanent failure occurs, it will be covered with probability c and with probability $1 - c$ it will be uncovered. A failure can be covered only if the number of up processors is at least l , the quorum. Thus, there is an input arc of multiplicity

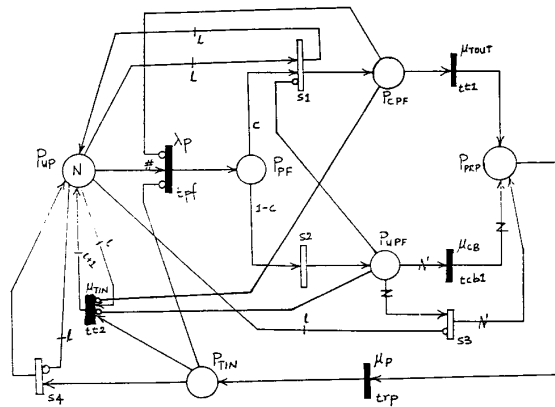


Figure 4: GSPN for a System with Only Permanent Failures

ity l from P_{UP} to the immediate transition $s1$. Failures that occur after the quorum is lost are essentially uncovered. Also, if a failure occurs after the quorum is lost (i.e., the number of tokens in $P_{UP} < l$), no cluster reboot takes place. This is modeled by the presence of an inhibitor arc of multiplicity l from P_{UP} to the immediate transition $s3$. The firing rate of the timed transition $tt1$ is μ_{TOUT} , the transition rate for the cluster to map out a failed processor. The firing rate of the timed transition $tt2$ is μ_{TIN} , the transition rate for the cluster to readmit a repaired (or rebooted) processor. The firing rate of trp is μ_P , the processor repair rate.

When a processor repair is completed (i.e., there is a token in P_{TIN}), one of two things will take place. If the number of up processors is at least l , or a quorum is formed, then the cluster will undergo a transition to readmit the repaired processor. This means that the timed transition $tt2$ is enabled, provided that the cluster is neither being rebooted nor undergoing a transition (i.e., there is no token in P_{CPF} and no token in P_{UPF}). This accounts for the inhibitor arcs from P_{CPF} and P_{UPF} to $tt2$. When it fires, $l+1$ tokens will be deposited in P_{UP} which includes the l tokens used to prime $tt2$ and the token from P_{TIN} . If the number of up processors is below quorum, then the immediate transition $s4$ is enabled. In this case the cluster does not undergo a transition since the up processors are effectively idle and the cluster is *hanging*. Note that when the cluster is rebooting, no covered failure can occur. This accounts for the inhibitor arc from P_{UPF} to $s1$. Also, the arc from P_{UPF} to $tcb1$ is a variable arc; this is denoted by the *zigzag* sign over the arc. This implies that when $tcb1$ fires it takes all the tokens in P_{UPF} and deposits them in P_{PRP} . This is due to the fact that if two or more processors suffer an uncovered failure, then after the

cluster reboot all the processors are mapped out and are ready for repair. By a similar argument the arcs from P_{UPF} to $s3$ and the arc from $s3$ to P_{PRP} are variable arcs.

We next consider an N -processor system in which only intermittent failures can occur. The GSPN for this system is shown in Figure 5. As

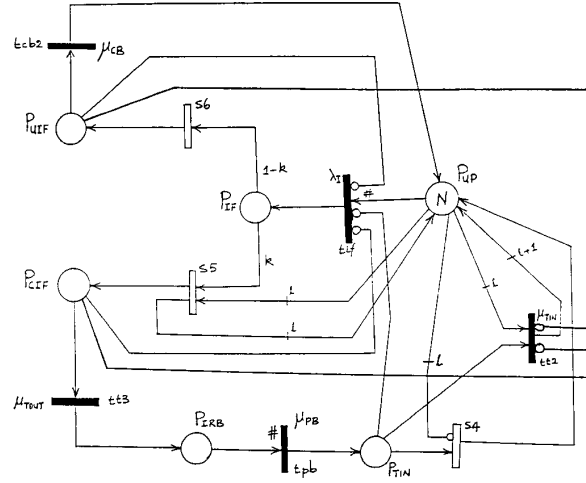


Figure 5: GSPN for a System with only Intermittent Failures

in Figure 4, the place P_{UP} initially contains N tokens and represents the condition that the system is up on all processors. The firing rate of the timed transition ttf is marking dependent. The factor λ_l is the intermittent failure rate of a processor. The place P_{IF} represents the condition that an intermittent failure has occurred. The place P_{CIF} represents the condition that an intermittent failure is covered while P_{UIF} represents the condition that it is an uncovered failure. The firing rate of the timed transition $tt3$ is μ_{TOUT} , the transition rate for the cluster to map out a processor. Similarly, the firing rate of the timed transition $tcb2$ is μ_{CB} , the cluster reboot rate. If the cluster is being rebooted, no other

intermittent failure can occur. This accounts for the inhibitor arc from P_{UIF} to tif .

The place P_{IRB} represents the condition that subsequent to a covered failure the cluster has completed a transition to map out the failed processor. The place P_{TIN} represents the condition that the failed processor has been rebooted. If there are at least l up processors, then the timed transition $tt2$ is enabled, otherwise the immediate transition $s4$ is enabled. Note that for $tt2$ to fire, the cluster must not be rebooting or undergoing a transition. That is, there is neither a token in P_{UIF} nor in P_{CIF} ; hence the inhibitor arcs to $tt2$ from these two places. When the system is either undergoing a cluster transition, no intermittent failure can occur. This accounts for the input inhibitor arcs to tif from both P_{CIF}

and P_{TIN} . Note that all processors that suffer an intermittent failure can be rebooted in parallel. Thus, the firing rate of tpb is marking dependent.

Figure 6 shows the complete model incorporating Figures 4 and 5 along with some crossover effects. These crossover effects are as follows:

1. While the cluster is being rebooted subsequent to an uncovered intermittent failure (i.e., there is a token in P_{UIF}), a covered permanent failure is impossible. This accounts for the inhibitor arc from P_{UIF} to the immediate transition $s1$.
2. While the cluster is undergoing a transition, no failure is assumed to occur. This accounts for the inhibitor arcs from P_{CPF} , P_{CIF} and P_{TIN} to tpf and tif .

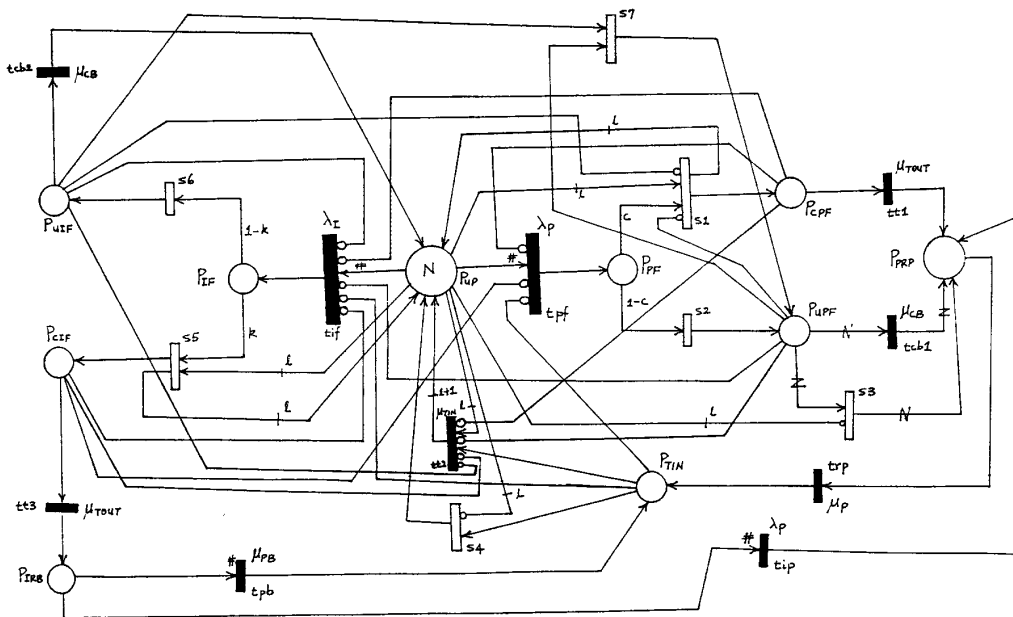


Figure 6: Complete Petri Net Model for the VAXcluster

3. After a failed processor has been mapped out of the cluster following a covered intermittent failure (i.e., there is a token in P_{IRB}), the processor can suffer a permanent failure while it is being rebooted. Thus, when there is a token in P_{IRB} , both tpb and tip are enabled. If the processor suffers a permanent failure before its reboot is completed, the timed transition tip fires and a token is deposited in P_{PRP} . Otherwise, the timed transition tpb fires and a token is deposited in P_{TIN} .
4. While the cluster is being rebooted following an uncovered intermittent failure (i.e., there is a token in P_{UIF}), a permanent failure of one of the processors can occur. Such a failure is essentially uncovered. Thus, if there is a token in P_{UIF} and tpf fires, the token deposited in P_{PF} when tpf fires will enable only $s2$ which fires immediately depositing a token in P_{UPF} . With a token in P_{UIF} and a token in P_{UPF} the immediate transition $s7$ is enabled. It fires and returns a token to P_{UPF} . The cluster reboot continues, and at the end of the reboot the repair of the failed processor commences.

The cluster is said to be available whenever

1. there are at least l tokens in place P_{UP} (that is, at least l up processors), and
2. there is no token in P_{UPF} or P_{UIF} (i.e., the cluster is not undergoing a reboot), and
3. there is no token in place P_{CDF} , place P_{CIF} or place P_{TIN} (i.e., the cluster is not undergoing a transition).

It is easy to show that the reachability graph of the GSPN in Figure 6 is isomorphic to the Markov chain of the two-processor cluster given in Figure 3.

4 Numerical Results

We consider a VAXcluster with the following parameter values:

$$\begin{aligned}
 1/\lambda_P &= 6000 \text{ hours} \\
 1/\lambda_I &= 1200 \text{ hours} \\
 1/\mu_P &= 2 \text{ hours} \\
 1/\mu_{CB} &= 15 \text{ minutes} \\
 1/\mu_{PB} &= 5 \text{ minutes} \\
 1/\mu_{TIN} &= 2 \text{ seconds} \\
 1/\mu_{TOUT} &= 5 \text{ seconds}
 \end{aligned}$$

We assume that $l = 1$; that is, at least one processor must be up for the cluster to be defined to be up. Since the state space of the reachability graph of the complete model is very large even for moderate values of N , we use the Stochastic Petri Net Package (SPNP) [10] to solve the model. We define the mean downtime in minutes per year as

$$D = (1 - A) \times 8760 \times 60$$

where A is the steady state cluster availability, as defined in Section 3.

Figure 7 shows how the mean downtime (in minutes per year) varies with N for different values of coverage factors c and k . Figure 8 shows how the mean downtime (in minutes per year) varies with N for different values of cluster re-configuration times $1/\mu_{TIN} = 1/\mu_{TOUT}$. As the two figures show, for the chosen set of parameter values the optimal value of N is 2. Since a cluster with a larger number of processors is likely to yield a higher performance than a smaller cluster, it follows that a composite measure of availability and performance is needed to assess the behavior of a cluster. Measures like availability and downtime provide only a partial picture of the cluster behavior [16].

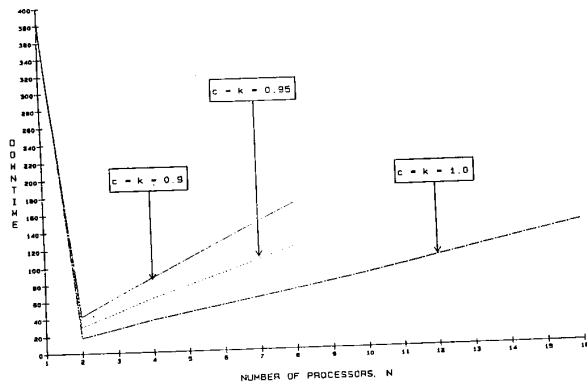


Figure 7: Variation of Mean Cluster Downtime (in Mins/Year) with N and Different Coverage Values

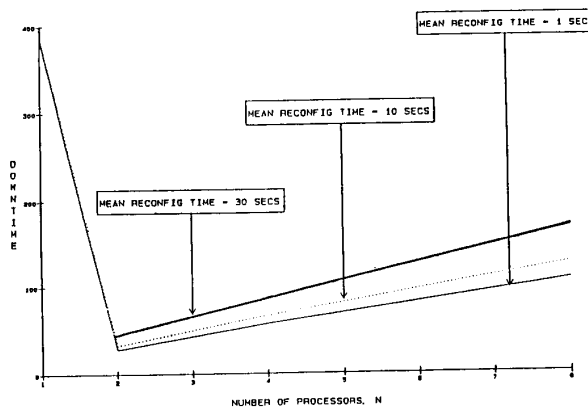


Figure 8: Variation of Mean Cluster Downtime (in Mins/Year) with N and Different Mean Reconfiguration Time Values

5 Conclusion

We have used the power of stochastic Petri nets to model VAXcluster system availability. Realistic features are modeled while keeping the size of the net independent of the number of processors in the cluster. End conditions such as the behavior of the processor fault breaking a quorum being different from a processor fault not

breaking a quorum are modeled by means of inhibitor arcs. Block departures (or token flushing) as implied by a cluster reboot operation are modeled by variable input/output arcs. Priority assignment on timed and immediate transitions are also used to advantage. This paper shows that stochastic Petri nets can be used for modeling the availability of realistic systems.

References

- [1] N.P. Kronenberg, H.M. Levy and W.D. Strecker, "VAXclusters: A Closely-coupled Distributed System," *ACM Trans. Computer Systems*, vol. 4, pp. 130 - 146, May 1986.
- [2] O.C. Ibe, R.C. Howe and K.S. Trivedi, "Approximate Availability Analysis of VAXcluster Systems," *IEEE Trans. Rel.*, vol. 38, pp. 146 - 152, 1989.
- [3] M.K. Molloy, "Performance Analysis Using Stochastic Petri Nets," *IEEE Trans. Comput.*, vol. C-31, pp. 913 - 917, 1982.
- [4] W.G. Bouricius, W.C. Carter and P.R. Schneider, "Reliability Modeling Techniques for Self-Repairing Computer Systems," *Proceedings of the 24th National Conference of the ACM*, pp. 295 - 309, 1969.
- [5] K.S. Trivedi, *Probability and Statistics with Reliability, Queueing and Computer Science Applications*, Prentice-Hall, 1982.
- [6] R. Sahner and K.S. Trivedi, "Reliability Modeling using SHARPE," *IEEE Trans. Rel.*, vol. R-36, pp. 186 - 193, 1987.
- [7] J.L. Peterson, "Petri Nets," *ACM Computing Surveys*, vol. 9, pp. 223 - 252, 1977.

- [8] T. Agerwala, "Putting Petri Nets to Work," *IEEE Computer*, vol. 12., No. 12, pp. 85 - 94, December 1979.
- [9] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, 1981.
- [10] G. Ciardo, J. Muppala and K.S. Trivedi, "SPNP: Stochastic Petri Net Package," *Proceedings of the Third International Workshop on Petri Nets and Performance Models*, Kyoto, Japan, December 1989.
- [11] W.M. Zuberek, "Timed Petri Nets and Preliminary Performance Evaluation," *Proc 7th Annual Symp. Computer Architecture*, pp. 89 - 96, 1980.
- [12] M.A. Marsan, G. Conte and G. Balbo, "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems," *ACM Trans. Computer Systems*, vol. 2, pp. 93 - 122, 1984.
- [13] J.B. Dugan, K.S. Trivedi, R.M. Geist and V.F. Nicola, "Extended Stochastic Petri Nets: Applications and Analysis," *Performance '84*, E. Gelenbe (ed.), pp. 507 - 519, North-Holland, 1984.
- [14] J.F. Meyer, A. Movaghar and W.H. Sanders, "Stochastic Activity Networks: Structure, Behavior, and Application," *Proc. Int. Workshop on Timed Petri Nets*, Torino, Italy, July 1985.
- [15] M.A. Holliday and M.K. Vernon, "A Generalized Timed Petri Net Model for Performance Analysis," *IEEE Trans. Software Eng.*, vol. SE-13, pp. 1297 -1310, December 1987.
- [16] K.S. Trivedi, A.S. Sathaye. O.C. Ibe and R.C. Howe, "Should I Add a Processor?," *Proceedings of the 23rd Hawaii International Conference on Systems Sciences*, January 1990.