

Should I Add a Processor?

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

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

October 11, 2001

Abstract

It is generally assumed that the availability of a multiprocessor system increases with the number of processors in the system. However, when reconfiguration delays are allowed, this assumption is no longer valid. In this paper we develop a model that enables us to show that if availability is the only measure of system effectiveness of interest, then even a small reconfiguration delay leads to a violation of the monotonic increase in availability with the number of processors. We propose a new measure of system effectiveness, namely loss probability of a task, to equitably capture aspects of performance and availability.

1 Introduction

In a multiprocessor system it is well known that if at least one processor is needed for the system to be up (parallel redundancy), then system reliability, availability and the mean time to failure are all monotonically increasing with the number of processors. However, this statement is only conditionally true. It only holds if a failed processor can be mapped out of the system and if a repaired processor can be readmitted into the system without causing any reconfiguration delay. In practice, however, some reconfiguration delay is encountered when a failed processor is being mapped out of the system and when a repaired processor is being readmitted into the system.

In this paper we develop a Markov model of the availability of a multiprocessor system that enables us to show that even a small reconfiguration delay is enough to ensure that increasing the number of processors will not always increase the system availability. This points out the deficiency of such measures as availability and the need to consider combined measures of performance and availability [?, ?, ?, ?]. The throughput-oriented availability [?, ?] increases monotonically with the number of processors for realistic parameter values; however, this measure is nearly unaffected by failure/repair behavior.

We propose several measures of system effectiveness that combine the effects of failure/repair with system performance. These include the probability of rejection of an incoming task due to the fact that the system is either *down overloaded*, the probability of interruption of an accepted task, and the probability of the late completion of an accepted task.

The paper is organized as follows. In Section 2 we develop the basic model. In Section 3 we provide numerical results for various measures and illustrate possible performance/availability tradeoffs. Finally, concluding remarks are given in Section 5.

2 Basic Models

2.1 The Availability Model

Consider a multiprocessor system with n processors. Assume that the failure rate of each processor is γ . A processor fault is *covered* [?] with probability c and is not covered with probability $1 - c$. Subsequent to a covered fault, the system comes up in a degraded mode after a brief reconfiguration delay while after an uncovered fault, a longer, reboot action is required. The reconfiguration times are assumed to

Figure 1: Markov Chain for the Basic Model

be exponentially distributed with mean $1/\delta$, the reboot times are exponentially distributed with mean $1/\beta$, and the repair times are exponentially distributed with mean $1/\tau$. Assume all failure events are mutually independent, and that a single repair facility is shared by all the processors. The Markov model for this system is shown in Figure 1. Solving for the steady-state probabilities, we get

$$P_{n-i} = \frac{n!}{(n-i)!} (\gamma/\tau)^i P_n \quad i = 0, 1, \dots, n \quad (1)$$

$$P_{x,n-i} = \frac{n!}{(n-i)!} \frac{\gamma(n-i)c}{\delta} (\gamma/\tau)^i P_n \quad i = 0, 1, \dots, n-2 \quad (2)$$

$$P_{y,n-i} = \frac{n!}{(n-i)!} \frac{\gamma(n-i)(1-c)}{\beta} (\gamma/\tau)^i P_n \quad i = 0, 1, \dots, n-2 \quad (3)$$

where

$$P_n = \frac{1}{\sum_{i=0}^n (\gamma/\tau)^i \frac{n!}{(n-i)!} + \sum_{i=0}^{n-2} (\gamma/\tau)^i \frac{\gamma(n-i)cn!}{\delta(n-i)!} + \sum_{i=0}^{n-2} (\gamma/\tau)^i \frac{\gamma(n-i)(1-c)n!}{\beta(n-i)!}}$$

where P_i is the steady-state probability that the Markov chain is in state i . Let the availability, $A(n)$, be defined as a function of n . Then

$$\begin{aligned} A(n) &= \sum_{i=0}^{n-1} P_{n-i} \\ &= \frac{\sum_{i=0}^{n-1} \rho_a^i / (n-i)!}{\sum_{i=0}^n \rho_a^i / (n-i)! + \sum_{i=0}^{n-2} \gamma(n-i) \rho_a^i \left\{ \frac{c}{\delta} + \frac{(1-c)}{\beta} \right\} / (n-i)!} \end{aligned} \quad (4)$$

where $\rho_a = \gamma/\tau$. System unavailability, $U(n)$, is given by $U(n) = 1 - A(n)$; and downtime during an observation interval of duration T is given by $D(n) = U(n) \times T$.

The availability measure $A(n)$ ignores different levels of performance of various system states. We can take different levels of performance into account by attaching a reward rate to each state of the

Markov chain of Figure 1. The resulting Markov reward model can then be analyzed for various combined measures of performance and availability. Let reward rate r_i be assigned to a state with i processors functioning properly. Let the reward assigned to state 0 be r_0 . Similarly, the reward rates of reconfiguration states are $r_{x,i}$ and those for the reboot states are $r_{y,i}$. Then the expected reward rate in the steady-state, $EXRSS(n)$, is given by

$$EXRSS(n) = \sum_{i=0}^{n-1} r_{n-i} P_{n-i} + \sum_{i=0}^{n-2} r_{x,n-i} P_{x,n-i} + \sum_{i=0}^{n-2} r_{y,n-i} P_{y,n-i} + r_0 P_0. \quad (5)$$

In this reward-based approach, the availability $A(n)$ is simply $EXRSS(n)$ when the reward rate 1 is assigned to all system up states and a reward rate zero is assigned to all down states. By reversing the reward assignment, we obtain system unavailability.

It may be observed that users may be tolerant to short downtimes but will consider the system to have failed in case the downtime exceeds a threshold. Assume that h_R time units is a threshold on reconfiguration times. Now since no other event is allowed to take place while the reconfiguration is going on, the steady-state results derived above hold even under an arbitrary distribution (with a finite mean) of reconfiguration times. Let the distribution function of reconfiguration time be denoted by $F_R(t)$. Then for earlier equations to hold, we let $1/\delta = \int_{t=0}^{\infty} t dF_R$. The probability that an individual reconfiguration interval is short is given by $F_R(h_R)$. If we assign the reward rate

$$r_{x,n-i} = T[1 - F_R(h_R)] / \int_{t=0}^{\infty} t dF_R = \delta T[1 - F_R(h_R)]$$

and assign a zero reward rate to all other states, $EXRSS(n)$ will yield the average number of disruptive reconfigurations,

$$NR(n, h_R) = \delta T[1 - F_R(h_R)] \sum_{i=0}^{n-2} P_{x,n-i}, \quad (6)$$

in the interval of duration T . Note that the number of ‘long’ reconfigurations, plotted as a function of the threshold, h_R , is essentially the complementary distribution of reconfiguration times with a multiplying factor.

If we imposed a tolerance threshold h_B on reboot times and the distribution function of reboot times is $F_B(t)$, then it follows that the average number of disruptive reboots,

$$NB(n, h_B) = \beta T[1 - F_B(h_B)] \sum_{i=0}^{n-2} P_{y,n-i}, \quad (7)$$

in the interval of duration T . Note that the number of ‘long’ reboots, plotted as a function of the threshold, h_B , is essentially the complementary distribution of reboot times with a multiplying factor.

We can also compute the overall number of disruptive down epochs in the interval of duration T as:

$$ND(n, h_R, h_B) = T\{\tau P_0 + \sum_{i=0}^{n-2} [\delta(1 - F_R(h_R))P_{x,n-i} + \beta(1 - F_B(h_B))P_{y,n-i}]\} \quad (8)$$

Assigning performance-based reward rates to various states of the Markov model of Figure 1 is the subject of the next subsection.

2.2 Performance Models

The reward rate in a state with i processors functioning properly should correspond to some measure of performance in that configuration. An upper bound on system performance is obtained by equating performance with system capacity. Thus, the simplest reward assignment is to let the reward rate $r_i = i$ and let the reward rate of any down state be 0. Then the expected reward rate in the steady-state is specialized to the capacity-oriented availability and is given by

$$\begin{aligned} COA(n) &= \sum_{i=0}^{n-1} (n-i)P_{n-i} \\ &= \frac{\sum_{i=0}^{n-1} \rho_a^i / (n-i-1)!}{\sum_{i=0}^n \rho_a^i / (n-i)! + \sum_{i=0}^{n-2} \gamma(n-i)\rho_a^i \left\{ \frac{c}{\delta} + \frac{(1-c)}{\beta} \right\} / \{(n-i)!\}} \end{aligned} \quad (9)$$

We can also define the capacity-oriented unavailability, $COU(n) = n - COA(n)$, and capacity-oriented downtime, $COD(n) = T * COU(n)$.

Besides the obvious drawback that capacity is merely an upper bound on performance, the capacity-oriented availability seems to obliterate the effect of failure/repairs. We consider several different performance models that will tradeoff performance and availability.

First consider an arriving task. If the system is in one of the up states, then the task will be accepted; otherwise it is rejected. The steady-state probability for the task to be accepted is equal to the system availability $A(n)$. The probability that an incoming task is rejected due to the system being down is given by $P_{RSD}(n) = U(n)$.

Once the task is accepted, the probability of its completion will be close to 1, unless it is a long task. The probability of interruption of a long task that is accepted while the system is up with i processors can be derived with several different sets of assumptions. First assume that a fault on a processor that is scheduled to process the task cannot be tolerated and that a covered fault on any of the remaining $(i-1)$ processors can be tolerated. Further assume that there is a ‘software’ failure probability q_f for

the task. Then the interruption probability for a task that needs x units of execution time is given by

$$P_I1_i(x) = 1 - (1 - q_f)e^{-[\gamma+(i-1)\gamma(1-c)]x}. \quad (10)$$

Now if the task needs a random amount of execution time with the distribution function $F_X(x)$ and Laplace-Stieltjes transform (LST) $\tilde{F}_X(s) = \int_{x=0}^{\infty} e^{-sx}dF_X$, then the probability of interruption of an accepted task is derived by unconditioning the above expression with respect to x :

$$\begin{aligned} P_I1_i &= \int_{x=0}^{\infty} P_I1_i(x)dF_X = 1 - (1 - q_f) \int_{x=0}^{\infty} e^{-[\gamma+(i-1)\gamma(1-c)]x}dF_X \\ &= 1 - (1 - q_f)\tilde{F}_X(\gamma + (i - 1)\gamma(1 - c)). \end{aligned} \quad (11)$$

If we assign the reward rate $r_i = P_I1_i(x)/A(n)$ then $EXRSS(n)$ will specialize to $P_I1(n, x)$, the interruption probability of a task that needs x amount of computing time, given that it is accepted. Likewise, by setting $r_i = P_I1_i/A(n)$, we specialize $EXRSS(n)$ to $P_I1(n)$, the interruption probability of a task, given that it is accepted. We can also compute the probability of task being rejected or interrupted as:

$$\begin{aligned} P_{RI1}(n, x) &= 1 - \sum_{i=1}^{n-1} [(1 - q_f)e^{-[\gamma+(i-1)\gamma(1-c)]x}]P_i \\ &= U(n) + A(n)P_I1(n, x). \end{aligned} \quad (12)$$

Next we assume that a covered fault on any processor is not disruptive although it introduces a processing delay. Under this assumption, the probability of interruption of a task arriving when the system is up with i processors is:

$$P_I2_i = \begin{cases} 1 - (1 - q_f)e^{-[i\gamma(1-c)]x} & i > 1 \\ 1 - (1 - q_f)e^{-\gamma x} & i = 1 \end{cases} \quad (13)$$

If we assign the reward rate $r_i = P_I2_i(x)/A(n)$ then $EXRSS(n)$ will specialize to $P_I2(n, x)$, the interruption probability of a task that needs x amount of computing time, given that it is accepted. Likewise, if the task requirement distribution is given, then

$$P_I2_i = \begin{cases} 1 - (1 - q_f) \int_{x=0}^{\infty} e^{-[i\gamma(1-c)]x}dF_X = 1 - (1 - q_f)\tilde{F}_X(i\gamma(1 - c)) & i > 1 \\ 1 - (1 - q_f) \int_{x=0}^{\infty} e^{-\gamma x}dF_X = 1 - (1 - q_f)\tilde{F}_X(\gamma) & i = 1 \end{cases} \quad (14)$$

Setting $r_i = P_I2_i/A(n)$, we specialize $EXRSS(n)$ to $P_I2(n)$, the interruption probability of a task, given that it is accepted.

In the third scenario, we assume that a short reconfiguration delay is acceptable but a long one is not. Under this assumption, the probability of interruption of an accepted task that arrives while the system is up with i processors is:

$$P_{I3_i}(x) = \begin{cases} 1 - (1 - q_f)e^{-[i\gamma(1 - cF_R(h_R))]x} & i > 1 \\ 1 - (1 - q_f)e^{-\gamma x} & i = 1 \end{cases} \quad (15)$$

If we assign the reward rate $r_i = P_{I3_i}(x)/A(n)$ then $EXRSS(n)$ will specialize to $P_{I3}(n, x)$, the interruption probability of a task that needs x amount of computing time, given that it is accepted. In case of a distribution of task requirement, we have,

$$P_{I3_i} = \begin{cases} 1 - (1 - q_f)\tilde{F}_X(i\gamma(1 - cF_R(h_R))) & i > 1 \\ 1 - (1 - q_f)\tilde{F}_X(\gamma) & i = 1 \end{cases} \quad (16)$$

Setting $r_i = P_{I3_i}/A(n)$, we specialize $EXRSS(n)$ to $P_{I3}(n)$, the interruption probability of a task, given that it is accepted.

So far, we have not included the effects of contention in computing the performance in various configurations. In order to introduce queueing delays, we need to make several simplifying assumptions. We assume that arriving tasks form a Poisson process of rate λ and that the service requirements of tasks are independent, identically distributed with the exponential distribution of mean $1/\mu$. We will also assume that no faults occur during the execution of a task and that the task software failure probability $q_f = 0$.

When i processors are functioning properly, we use an $M/M/i$ queueing model to describe the performance of the multiprocessor system. The index of performance that we use is the relative throughput of tasks. We let $\rho_p = \lambda/\mu$. In this model, tasks are rejected with probability $(1 - i\mu/\lambda)$ whenever the system is unstable, that is, when $\lambda \geq i\mu$. Let K be the smallest number of processors for which the system is stable. Then we assign $r_i = \lambda/\mu$ for $i \geq K$, $r_i = i$ for $1 \leq i < K$ and a zero reward rate to all the remaining states. With this assignment of reward rates, we rename the expected reward rate in the steady-state as the performance-oriented availability, $POA(n)$.

The probability of rejection of a task when the system is down is accounted for by $P_{RSD}(n) = U(n)$. The probability of a task being rejected due to instability (while the system is up) is accounted for by

$$P_{RUS}(n) = \sum_{i=n-K+1}^{n-1} \left[1 - \frac{i\mu}{\lambda}\right] P_{n-i}. \quad (17)$$

There are at least two ways in which we can make the performance model more interesting and useful. First, we impose the constraint that a maximum number (b) of tasks are allowed in the system at one time. We then model the system with i processors functioning as an $M/M/i/b$ queue. Unlike the infinite buffer $M/M/i$ model, the finite buffer queue is always stable. However, a task will be rejected whenever b tasks are already in the system. Then the probability of rejection of a task in a configuration with i processors functioning is given by [?],

$$q_b(i) = \begin{cases} \frac{\rho_p^b}{i^{b-i} i! [\sum_{j=0}^{i-1} \frac{\rho_p^j}{j!} + \sum_{j=i}^b \frac{\rho_p^j}{i^{j-i} i!}]} & b \geq i \\ \frac{\rho_p^b}{b! [\sum_{j=0}^b \frac{\rho_p^j}{j!}]} & b < i \end{cases} \quad (18)$$

We now set the reward rate $r_i = q_b(i)$ and we set the reward rate to 0 in all other states. With this reward assignment, the expected steady-state reward rate specializes to the probability of task rejection due to limited buffers,

$$P_R_L_B(n, b) = \sum_{i=0}^{n-1} q_b(n-i) P_{n-i}. \quad (19)$$

Next we return to the infinite buffer case but we now impose a deadline on the task response time. Thus if a task response time takes longer than d time units then we consider that task as lost. Since the formula for the response time distribution in an $M/M/i$ queue is known [?], we can assign the reward rate to a configuration with i functioning processors as (if $\lambda < i\mu$)

$$r_i = Pr(R_i \geq d) = \left[\frac{\mu(1 - W_i)}{\lambda - (i-1)\mu} e^{-(i\mu - \lambda)d} - \frac{i\mu - \lambda - \mu W_i}{\lambda - (i-1)\mu} e^{-\mu d} \right] \quad (20)$$

where R_i is the response time random variable with i functioning processors and where

$$W_i = 1 - \frac{i\rho_p^i}{i!(i - \rho_p)} \frac{1}{\sum_{j=0}^{i-1} \frac{1}{j!} \rho_p^j + \frac{1}{i!} \rho_p^i \frac{i\mu}{i\mu - \lambda}}. \quad (21)$$

We set $r_i = i\mu/\lambda$ whenever the system is unstable (since then the deadline is sure to be violated for accepted tasks) and $r_i = 0$ when the system is down. The steady-state expected reward rate is now renamed to be the probability of lost tasks due to deadline violations,

$$P_L_D(n, d) = \sum_{i=0}^{n-K} r_{n-i} P_{n-i} + \sum_{i=n-K+1}^{n-1} \frac{i\mu}{\lambda} P_{n-i}. \quad (22)$$

Next we return to the finite buffer case since a closed-form formula for the response time distribution can be derived based on the formula for waiting time given in [?]:

$$r_i = Pr(R_i(b) \geq d) \quad (23)$$

where $R_i(b)$ is the response time random variable with i functioning processors and is given by (if $i > 1$),

$$Pr(R_i(b) > d) = \left[\sum_{j=0}^{i-1} q'_j + \sum_{j=i}^{b-1} q'_j \left(\frac{i}{i-1} \right)^{j-i+1} \right] e^{-\mu d} - \sum_{j=i}^{b-1} q'_j \sum_{k=0}^{j-i} \frac{(\mu i d)^k}{k!} e^{-\mu i d} \left[\left(\frac{i}{i-1} \right)^{j-i-k+1} - 1 \right] \quad (24)$$

and by (if $i = 1$),

$$Pr(R_1(b) > d) = \sum_{j=0}^{b-1} q'_j \sum_{k=0}^j \frac{(\mu d)^k}{k!} e^{-\mu d}. \quad (25)$$

In the above two equations we have defined, $q'_j = q_j / (1 - q_b)$. Based on this reward assignment, the expected reward rate in the steady-state is now specialized to the probability of a lost task due to deadline violations with the limited buffer space, $P_{L_DLB}(n, b, d)$.

In the case of no limit on the buffer space, the overall probability of a task lost (due to system down or unstable or too slow) is given by,

$$\begin{aligned} P_L(n, d) &= P_{RSD}(n) + P_{RUS}(n) + P_{LD}(n, d) \\ &= \sum_{i=0}^{n-K} Pr(R_{n-i} > d) P_{n-i} + \sum_{i=n-K+1}^{n-1} P_{n-i} \\ &+ \sum_{i=0}^{n-2} [P_{x,n-i} + P_{y,n-i}] + P_0. \end{aligned} \quad (26)$$

Similarly, in the case of limited buffer space, the overall probability of a lost task (due to system down or system full or too slow) is given by,

$$\begin{aligned} P_{LLB}(n, b, d) &= \sum_{i=0}^{n-1} [q_b(n-i) + \{(1 - q_b(n-i)) Pr(R_{n-i}(b) > d)\}] P_{n-i} \\ &+ \sum_{i=0}^{n-2} [P_{x,n-i} + P_{y,n-i}] + P_0. \end{aligned} \quad (27)$$

3 Numerical Results

In Figure 2, we have plotted the downtime in minutes per year as function of the number of processors for different values of the coverage factor (c). Here the downtime $D(n) = [1 - A(n)] \times 8760 \times 60$ minutes per year. We use $\gamma = 1/6000$ per hour, $\beta = 12$ per hour and $\tau = 1$ per hour, throughout this paper. We use the mean reconfiguration delay, $1/\delta = 10$ seconds, unless otherwise specified. We see as expected the downtime is not monotonically decreasing in the number of processors. The reason for this behavior is that an uncovered fault brings the entire system down and induces a reboot of all the

processors. Therefore as the number of processors in the system increases the impact of an uncovered fault increases. The question of minimizing downtime is explored at length in [?].

It is often argued that downtime paints an overly pessimistic picture of reality. Users may well be tolerant to short downtimes. In Figure 3, we have plotted the number (per year) of disruptive reconfigurations, the number of disruptive reboots and the total number of disruptive interruptions including off-line repairs. A disruptive down epoch (e.g. reconfiguration) implies that the mean time to complete this epoch exceeds a given threshold value. We have assumed that the probability that a given reconfiguration interval is long is 0.4 and that the probability of a long reboot is 0.97. We see that both the number of disruptive reboots and the number of disruptive reconfigurations monotonically increase with the number of processors. On the other hand, the number of disruptive off-line repair decrease with the number of processors. The net effect is that there is an optimal number of processors (in this case, two).

In Figure 4, we have plotted the probability that an accepted task is interrupted due to a failure as function of the task execution requirement, x . We used a two processor system to illustrate these results. The software failure probability, q_f is assumed to be zero. We have shown all three cases $P_{I1}(2, x)$, $P_{I2}(2, x)$, and $P_{I3}(2, x)$. We have also shown the probability of rejection of an incoming task due to system down, $P_{RSD}(2)$. For very long tasks, the interruption probability dominates the rejection probability. For such long tasks, it is worthwhile to do checkpointing. Otherwise, the dominant cause of task failure is that the system is down.

In Figure 5, we have plotted the unconditional probability that an accepted task is interrupted due to a failure as function of the number of processors. We assumed that the task service requirement is exponentially distributed with a rate $\mu = 100$ per second. We have shown all three cases $P_{I1}(n)$, $P_{I2}(n)$, and $P_{I3}(n)$. We have also shown the probability of rejection of an incoming task due to system down, $P_{RSD}(n)$. Thus if we wish to minimize the probability of rejection or interruption of tasks, we find the that optimal number of processors is two.

In Figure 6, we have plotted $COA(n)$ and $POA(n)$ for different values of the arrival rate λ . We see that the capacity-oriented availability is nearly linear in n . On the other hand, performance-oriented availability reflects both the system capacity and offered load. Unlike all the previous cases, we see that in order to process a heavier workload, we may need more than two processors. The measures plotted in Figures 6, mainly show the effects of system capacity (n) and the offered load (λ).

Figure 2: Downtime Vs. the Number of Processors for Different Values of c .

Figure 3: Number of Disruptive Downtime Epochs Vs. the Number of Processors.

Figure 4: Probability of Task Interruption Vs. Execution Time Requirement, x .

Figure 5: Probability of Task Interruption Vs. the Number Processors, n .

Figure 6: Computation Availability and Performance Availability as a Function of the Number of Processors.

Figure 7: Total Loss Probability for different values of d Vs. the Number Processors, n .

In Figure 7, we have plotted the total loss probability of a task in the limited buffer case as a function of the number of processors for different values of the deadline. We have used $b = 10$, $\lambda = 80$ per second and $\mu = 100$ per second. We see that as the stringency of deadline increases, the optimal number of processors, n^* , increases as well. For instance, $n^* = 3$ for one second deadline on task response time, $n^* = 4$ for 0.1 second deadline on task response time, and $n^* > 8$ for 0.01 second deadline on task response time. Note that an added processor means an increase in the computing power and hence as the stringency of the deadline on task response time increases the lower the probability of losing a task due to deadline violation.

In Figure 7, we have plotted the total loss probability of a task in the limited buffer case as a function of the number of processors for different number of buffers. In this case we have used a response time deadline of 0.1 second. We see that as the number of processors increases, the loss probability decreases. The interesting phenomena of a smaller number of buffers having a smaller loss probability (at $n = 1$) is explained by the fact that a system with a larger number of buffers accepts more tasks and is then less likely to meet the deadline than a system with a smaller number of buffers.

Figure 8: Total Loss Probability for Different values of b Vs. the Number Processors, n .

4 Conclusion

In this paper, we have explored the effect of increasing the number of processors on the effectiveness of a multiprocessor system. We have used availability and several combined measures of performance and availability measures in our study. Measures such as downtime and the probability of a task interruption are minimized at a small number (two) of processors. Probability of task completion within a stringent deadline is minimized at a larger number of processors.

5 Acknowledgement

Discussions with David Heimann and his paper [?], and several documents by Ed Balkovich and John Kitchin provided inspiration for part of the work presented in this paper.

References

- [1] M. D. Beaudry, "Performance-related reliability measures for computing systems," *IEEE Transactions on Computers*, vol. C-27, pp. 540–547, June 1978.

- [2] D. Gross and C. M. Harris, *Fundamentals of Queueing Theory*, John Wiley and Sons, New York, 1985.
- [3] D. Heimann, "VAXcluster System Availability— Concepts and Metrics," Internal Report, DEC, January 1989.
- [4] M. C. Hsueh, R. K. Iyer, and K. S. Trivedi, "Performability modeling based on real data: a case study," *IEEE Transactions on Computers*, Apr. 1988.
- [5] J. F. Meyer, "On evaluating the performability of degradable computing systems," *IEEE Transactions on Computers*, vol. C-29, pp. 720–731, Aug. 1980.
- [6] R. M. Smith, K. S. Trivedi, and A. V. Ramesh, "Performability analysis: measures, an algorithm, and a case study," *IEEE Transactions on Computers*, Apr. 1988.
- [7] K. S. Trivedi, A. Sathaye, and O. Ibe, "Pitfalls of Redundancy," Internal Report, DEC, December 1988.
- [8] 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.