Optimal Dynamic Control of Resources in a Distributed System

KANG G. SHIN, SENIOR MEMBER, IEEE, C. M. KRISHNA, MEMBER, IEEE, AND YANN-HANG LEE, MEMBER, IEEE

Abstract—The various advantages of distributed systems can be realized only when their resources are "optimally" (in some sense) controlled and utilized. For example, distributed systems must be reconfigured dynamically to cope with component failures and workload changes. Owing to the inherent difficulty in formulating and solving resource control problems, the resource control strategies currently proposed/used for distributed systems are largely ad hoc.

It is our purpose in this paper to 1) quantitatively formulate the problem of controlling resources in a distributed system so as to optimize a reward function, and 2) derive optimal control strategies using Markov decision theory. The control variables treated here are quite general: for example, they could be control decisions related to system configuration, repair, diagnostics, files, or data. Two algorithms for resource control in distributed systems are derived for time-invariant and periodic environments, respectively. A detailed example to demonstrate the power and usefulness of our approach is provided.

Index Terms—Markov decision process, optimization, performability, reconfiguration, repair, resource control.

I. Introduction

BECAUSE distributed computer systems are usually composed of a large number of resources to achieve higl performance and reliability and can function (albeit sometimes in a degraded condition) in a very wide variety of configurations and environments, managing the resources in such systems is much more difficult than managing a conventional uniprocessor machine. Such difficulty has led to an undesirable trend: the resource control strategies currently proposed/used are largely qualitative and ad hoc. We counter this trend with the development of a rigorous, systematic method for "optimally" (to be defined below) controlling the resources that make up distributed systems. That is, we quantitatively formulate a resource control problem, for which solution algorithms are derived using Markov decision theory.

A resource control decision is needed whenever there is a significant change in either the operating environment or in the system, e.g., component failures and workload changes.

Manuscript received November 9, 1987; revised April 3, 1989. Recommended by B. W. Wah. This work was supported in part by NASA under Grant NAG-1-296, by the National Science Foundation under Grant NSFDMC-8504971, and by the Florida High Technology and Industry Council under Grant 9E071.

K. G. Shin is with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109.

C. M. Krishna is with Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003.

Y.-H. Lee is with the Department of Computer and Information Sciences, University of Florida, Gainesville, FL 32611.

IEEE Log Number 8930136.

Two parameters characterize the operating environment: the *reward structure*, and the *imposed load*. The former needs some elaboration. The operating environment imposes a *value* on each of the many services it receives from the computer. Put more formally, there is a *reward* (which could be negative) which accrues from each job execution, and this reward is a function of the needs of the application. When the operating environment changes, such a change can be quantified by the change in the reward structure or failure rates that this causes. For example, the reward accruing from a transaction-handling machine in a bank is different at peak banking hours than it is at, say, midnight. Naturally, it would be useful to be able to optimally configure or service the system as a function of the prevailing application needs.

Resource control decisions also have to be made when the computer changes due to component failures. When, for instance, is it appropriate to summon a repairman? Which (degraded) configuration should the system switch to, prior to repair? Or, consider the problem of allocating channel bandwidth optimally to members of a set of token-ring networks. Each network has a set of users each of which pays a certain amount of money for a given quality of service (e.g., waiting time). Additionally, the system response time is a function of the load offered to the system. How does one allocate bandwidth amongst the various networks so as to maximize reward (i.e., customer payment)? When systems are simple, such decisions can be made on the basis of intuition alone. When they get complex, unsupported intuition is insufficient, and must be supplemented by rigorous methods enabling a more precise control. This is especially true when the performance of the system is an intricate function of parameters which may act at cross purposes to one another.

As we shall show in Section II, resource control decisions can be viewed as semi-Markovian decision processes. The measure of performance used here is based on Meyer's *performability* [1]. It is one of the most powerful application-sensitive metrics available today, and incorporates both the traditional measures of performance (e.g., throughput) and of reliability. Performability formally accounts for the requirements of the application by defining *accomplishment levels*. The vector of probabilities of meeting the accomplishment levels is performability. Suppose we identify a reward with each accomplishment level. Then, the expected reward rate can be obtained from the performability of the system. This reward rate is also defined as *reward structure* by Furchtgott [2], and as *reward function* by Donatiello and Iyer [3], [4]. It is this reward rate that we use to characterize the performance of the system. The average reward received over infinite time horizon is used as an optimization criterion for resource control in a distributed computer system.

The expected total reward accumulated during a mission lifetime, using reward rates as an optimization criterion, has been studied in [5] for configuring degradable and nonrepairable systems. The results suggest that the system should perform not only *passive reconfiguration* to respond the occurrence of a failure, but also *active reconfiguration* during the course of operation to respond to changes in the reward or loading structure with a change in configuration. The problem is formulated as a dynamic programming problem with a finite horizon. It can be simplified by identifying the relationships between configurations and switch times (the time instants that the system performs active reconfiguration).

Algorithms from Markov decision theory are applied to solve the above resource control problem. Note that, despite the importance of the resource control problem and a voluminous applied statistics literature on decision processes, there is very little in the computing literature on using the results of decision theory to control computing resources optimally.

This paper is organized as follows. In Section II, the optimal resource control problem is stated formally. In Section III, we show how to use the *strategy improvement procedure* from decision theory in taking optimal resource control decisions for the case in which the reward rate is constant. In Section IV, we turn to the case in which the reward rate is periodic: Sections III and IV between them encompass the majority of fault-tolerant systems. In Section V we provide a numerical example. We conclude with Section VI.

II. Problem Formulation

Consider a system which may exist in one of several states. A *state* is a compact description of everything about the system that it is relevant to know. At predefined instants of time, an action or input is applied to it. The system response to that action a is characterized by the matrix of transition probabilities $p_{ij}(a)$, i.e., the probability of a direct transition from state i to state j under action a. Assume that there is a reward that the system generates for its owners per unit time; a reward which is clearly a function of the system state. Assume also that taking an action costs something (zero is a permissible cost).

Maximizing the net reward per unit time by suitably choosing the actions a is one of the most important problems of decision theory. We refer the reader to [6], [7], and [8] for an excellent introduction to the subject. In the remainder of this section, we formalize and elaborate on what we have said above.

Suppose our computer system has n units of some resource. A *unit* of resource is defined to be the smallest part of the whole system which may fail to operate, and which can be repaired, reloaded, or replaced. Examples are processors, memory modules, I/O channels, shared tables, file units, and data sets. A unit can provide useful services when it is fault-free and may become unavailable or invalid in the event of failure or loss of control.

The system state is the aggregation of all states, denoted by Φ, which is a finite set. The system state at time t can be defined as a stochastic process $S(t)$.

For a state $i \in \Phi$, the system may be in one of various operational modes or may take certain actions. For instance, the system might choose to reconfigure itself. Let A_i be the set of all available actions or operational modes when $S(t) = i$, and let the system choose action $a(i, t)$ from A_i. It is easy to see that transitions between system states depend upon the current state and the current action or operational mode. When the availability/functionality of units are uncorrelated and the failure process in each unit are Markovian, state transitions will be independent of the past states and actions. Thus, the system's behavior at time t can be fully specified by the pair $(S(t), a(S(t), t))$, where $S(t) \in \Phi$, and $a(S(t), t) \in A_{S(t)}$.

Let $\rho(i, a(i))$ be a reward rate associated with the system state i and the action $a(i) \in A_i$. This reward rate represents what the system can achieve, or may lose, per unit time, with the pair $(i, a(i))$. In addition, we assume that there exists a *cost*, $c(i, a(i))$, with the action $a(i)$ taken when the system enters state i. The cost $c(i, a(i))$ represents the instantaneous cost (if any) of taking the action $a(i)$.

$A = \bigcup_{i \in \Phi} A_i$ is defined to consist of all choices of action or operational mode that the system will take during its lifetime. When the choice of action at time t depends only on the system's state at time t, the strategy is called *stationary*. In other words, if we take an action or enter a particular operational mode when the system enters a new state, that action or operational mode will be used until the next state transition takes place. Thus the same action or operational mode will be used continuously between two successive state transitions. A *stationary strategy* can be specified by a set of actions, $\{a(i) \mid a(i) \in A_i, i = 1, 2, \cdots, n\}$. With a given stationary strategy π, the action taken at time t is denoted by $a_\pi(S(t))$. Also, we assume that, when the system has no units available, the only action that can be chosen is to repair all or part of the system. Thus, under a stationary strategy π, the reward accumulated during $[0, t)$ can be expressed as

$$W_{\pi,i}(t) = \int_0^t \rho\big(S(\tau), a_\pi(S(\tau))\big)\, d\tau - \sum_{j \in \Phi} c\big(j, a_\pi(j)\big)\, k_j(t) \qquad (2.1)$$

where $S(0) = i$ is the initial state of the system, and $k_j(t)$ the number of visits to state j during $[0, t)$.

If, instead of being stationary, there is a periodicity to the reward, failure, or cost structures, then the system is *periodic*. If, as sometimes happens, cost and failure rates have different periods, the least common multiple of these determines the system period. If the period is T, then the choice of action at time t of a periodic system will depend only on the system state at time t and on $t - [t/T]T$, where $[x]$ is the maximum integer not exceeding x. Clearly, any periodicity in the job arrival process can be expressed through the reward function.

The period of reward, cost, etc., will depend on the application in question. For example, the arrival process might be diurnal, as in most computing centers.

The resource control problem arising from this model is to determine a strategy π^* such that the average expected reward $\lim_{t\to\infty} W_{\pi^*,i}(t)/t$ is maximized with respect to all feasible strategies. This is a common problem: for instance, if a two-dyad system were to suffer a processor failure, does it reconfigure into a one-triad system, or into a one-dyad, one-simplex system? Another example is choosing the recovery action to be taken when a file becomes inaccessible.

III. Optimal Strategy for Homogeneous Systems

A system is ***homogeneous*** if transition characteristics between states, the cost of actions, and the reward rate do not change in time. The distribution of the time that the system spends in state i before leaving it is independent of the time at which the system entered state i. Thus, given an arbitrary stationary strategy, the system can be modeled by a semi-Markov process. Moreover, when Φ is finite, the optimal strategy of a homogeneous system must be stationary as shown in [10].

While few—if any—distributed systems exhibit exact homogeneity, there are many which are approximately homogeneous. Aging of components and a change in the reward structure are the most frequent causes of a departure from homogeneity. However, in many instances, components age, and the reward structure changes so slowly that the system can, for all practical purposes, be regarded as homogeneous.

Because we are interested in optimizing the asymptotic reward rate, the transient states have no effect on our objective and are therefore not considered in this model. Also, we limit ourselves to the case where there is only one recurrent class in this semi-Markov process. (Recurrence follows from the fact that repair is allowed.) This is not a limiting factor: even if there are multiple classes, each of them can be considered separately.

Since repair is allowed and the total number of states is finite, the expected period between two consecutive visits to state i, denoted by T_i, is finite. Under a stationary strategy, $S(t)$ is a regenerative process. Thus, $W_{\pi,i}(t)$ can be regarded as a renewal reward process with regenerative period T_i. It follows that, under the strategy π, the average expected reward per unit time can be given as

$$V_\pi(i) = \lim_{t\to\infty} \frac{E[W_{\pi,i}(t)]}{t} = \frac{E[W_{\pi,i}(T_i)]}{E[T_i]}. \quad (3.1)$$

This equality can be proved easily from the facts that $E[T_i] \le \infty$ and that the strategy is stationary. The optimal resource control problem in which $V_\pi(i)$ is maximized then becomes a semi-Markov decision problem. Let the optimal strategy be π^* such that $V_{\pi^*}(i) = \max_\pi V_\pi(i)$.

We denote the transition probability from state i to state j under action $a(i)$ by $p_{ij}(a(i))$, and the mean holding time at state i by $\alpha(i, a(i))$. Then, the mean reward accrued from the moment the system enters state i until the moment the next transition occurs can be represented by $\gamma(i, a(i)) \equiv \rho(i, a(i))\,\alpha(i, a(i)) - c(i, a(i))$. According to Theorem 7.6 of [8], the optimal stationary strategy can be obtained by the following theorem.

Theorem 1: If there exists a bounded function $h(i)$, $i = 1, 2, \cdots, n$, and a constant g such that

$$h(i) = \max_{a\in A_i} \left\{ \gamma(i, a) + \sum_{j=1}^{n} p_{ij}(a)\, h(j) - g\alpha(i, a) \right\}, \quad (3.2)$$

then there exists a stationary strategy π^* such that $g = V_\pi^*(i) = \max_\pi V_\pi(i)$.

Notice that the condition in the above theorem will be met automatically when $\gamma(i, a(i))$ is bounded for all i and action $a(i)$, and all stationary strategies give rise to a finite and irreducible state space, despite the fact that we do not know g in advance. The optimal strategy can be determined by Howard's strategy improvement procedure [6], [7]. In the discussion that follows, we shall present an algorithm on the basis of the strategy improvement procedure that is embedded in the solution of Howard's equations:

$$g\alpha(i, a(i)) + h(i) = \gamma(i, a(i)) + \sum_{j=1}^{n} p_{ij}(a(i))\, h(j) \quad (3.3)$$

where $a(i) \in A_i$. Equation (4) stems from Theorem 1.

It is easy to see from (4) that the functions $h(\bullet)$ cannot be uniquely determined, since $[p_{ij}(a(i))]$ is a transition matrix: the set of equations is dependent. From Theorem 1, the absolute value of the $h(\bullet)$'s does not matter in our search for the optimal actions: only the relative value does. So, we can set one of the $h(\bullet)$'s to some value, and solve for the rest. In the algorithm below, we set $h(n) = 0$ in step 3 as the most convenient value.

Algorithm 1:

1) Select an arbitrary strategy $\pi = \{a(i) \mid a(i) \in A_i, i = 1, \cdots, n\}$.

2) Solve (4) to obtain $h(i)$, $i = 1, \cdots, n-1$, and g, under the strategy π. To do this, set $h(n) = 0$.

3) Generate the strategy π' as follows. For each i, determine an action $a'(i)$ for which $h(i)$ is maximized. That is, find $a'(i)$ to maximize:

$$\gamma(i, a(i)) + \sum_{j=1}^{n} p_{ij}(a'(i))\, h(j) - g\alpha(i, a'(i)).$$

If more than one action maximizes the function, choose the one with the smaller label. π' consists of these actions: $\{a'(1), \cdots, a'(n)\}$. If $\pi = \pi'$, then $\pi = \pi^*$ and we can stop, having found an optimal strategy. Otherwise, set $\pi = \pi'$ and go to step 2.

A proof that policy improvement procedures of this kind converge is presented, for example, in [9]. The worst-case complexity of this algorithm is simply the total number of possible strategies: this is because the policy improvement procedure is essentially a contraction mapping, and so the same nonoptimal strategy is never encountered twice.

IV. Systems with Periodic Rewards

In this section, we consider the case where the reward rate is a periodic function. Suppose that, at time t, the reward rate associated with state i and the action $a(i)$ is $\rho(\tau, i, a(i))$, where $\tau = t - [t/T]T$. The period T is obtained from practical considerations: it may be a day, a week, or any other period natural to the application. Our objective is to find a strategy

$$\pi = \{a(i, \tau) | i = 1, 2, \cdots, n, \tau \in [0, T)\} \quad (4.1)$$

such that the average expected reward is maximized.

For tractability, we assume that the holding time at state i under action $a(i)$ is exponentially distributed with mean $\alpha(i, a(i))$. Then, the system can be modeled by an embedded Markov chain in which the system's state is examined every period T. Let the system state at time mT be s_m. With a strategy π, the system will transfer into state s_{m+1} at time $(m + 1)T$. The sequence $\{s_m\}$ is a Markov process, and the state transition probability is defined as $q_{ij}(\pi) = \text{Prob}\{s_{m+1} = j | s_m = i\}$. Also, denote the reward accumulated between mT and $(m + 1)T$ by $w(i, \pi)$ when $s_m = i$. Thus, as in Theorem 1, an optimal strategy exists if there exists a bounded function $h(i)$ for $i = 1, 2, \cdots, n$, and a constant g such that

$$h(i) = \max_{\pi} \left\{ w(i, \pi) + \sum_{j=1}^{n} q_{ij}(\pi) h(j) - g \right\} \quad (4.2)$$

where g is the maximal reward.

It is easy to see that these conditions are satisfied in our case. Although we do not know g *a priori*, it is clear from the problem definition that it is finite. So is w.

The policy improvement procedure can also be applied to find the optimal strategy, π^* in this case. However, for each possible π, we need to determine both $q_{ij}(\pi)$ and $w(i, \pi)$. A modified algorithm is presented below, in which dynamic programming is used to find $q_{ij}(\pi)$, $w(i, \pi)$, and the improved strategy, at each iteration.

The first step is to discretize the period to make the algorithm suitable for digital implementation. Let K be a large natural number, $\delta = T/K$, and let $a(i, k)$ be the action applied at the system state i when $t \in [mT + k\delta, mT + (k + 1)\delta)$. We are concerned with the optimal strategy

$$\pi^*(K) = \{a(i, k) | i = 1, 2, \cdots, n, \\ k = 0, 1, \cdots, K - 1\}.$$

Also, it is convenient here to regard $p_{ij}(a(i, k))$ as the probability that the state transition $i \to j$ occurs in one time period of duration δ.

Algorithm 2:

1) Select an arbitrary strategy

$$\pi(K) = \{a(i, k) | i = 1, 2, \cdots, n, \\ k = 0, 1, \cdots, K - 1, \quad a(i, k) \in A_i\}.$$

2) For each $i = 1, 2, \cdots, n$, calculate $q_{ij}(\pi)$ and $w(i, \pi)$ by the following equations:

$$q_{ij}(\pi) = \tilde{q}_{ij}(\pi, K), \quad (4.3)$$

where

$$\tilde{q}_{ij}(\pi, k + 1) = \sum_{l=0}^{n} p_{lj}(a(l, k))\, \tilde{q}_{il}(\pi, k) \quad (4.4)$$

$$w(i, \pi) = \sum_{k=0}^{K-1} \sum_{j=1}^{n} \tilde{q}_{ij}(\pi, k)\, \rho(k\delta, j, a(j, k))\delta \\ - C(a(j, k)) \quad (4.5)$$

with $\tilde{q}_{ii}(\pi, 0) = 1$ and $\tilde{q}_{ij}(\pi, 0) = 0$ $(j \neq 0)$. $C(a(j, k))$ is the cost of taking action $a(j, k)$.

3) Calculate $h(i)$ as follows:

$$h(i) = w(i, \pi) + \sum_{j=1}^{n} q_{ij}(\pi)\, h(j). \quad (4.6)$$

4) For each $i = 1, 2, \cdots, n$, define $\hat{h}(i, K) = h(i)$, $\hat{w}(i, K) = 0$, and $\hat{q}_{ij}(K) = 0$ if $j \neq i$, $\hat{q}_{ij}(K) = 1$ otherwise. For $k = K - 1$ to 0, find actions $\hat{a}(i, k)$ for $i = 1, 2, \cdots, n$, which maximize

$$\hat{h}(i, k) = \rho(k\delta, i, \hat{a}(i, k))\delta \\ + \sum_{j=1}^{n} p_{ij}(\hat{a}(i, k))\, \hat{h}(j, k + 1) \quad (4.7)$$

where $a(i, k) \in A_i$.

5) Let $\pi'(K) = \{a(i, k) | i = 1, \cdots, n, k = 0, 1, \cdots, K - 1\}$. If $\pi'(K) = \pi(K)$, then stop the algorithm with $\pi = \pi'(K)$. Otherwise, set $\pi(K) = \pi'(K)$, $w(i, \pi) = \hat{w}(i, 0)$, and $q_{ij}(\pi) = \hat{q}_{ij}(i, 0)$, and go to step 2.

In (10), one can set $h(n)$ equal to some positive constant and solve for the rest of the $h(i)$'s.

For finite K, this algorithm produces nearly-optimal actions and not necessarily optimal ones, since the actions take place at specific epochs in the operating interval (namely at multiples of δ). It is easy to show, from the fundamentals of dynamic programming, that the algorithm tends to optimal as $K \to \infty$ [10]. It is trivial to show (by contradiction) that if there are two numbers k_1 and k_2, with $k_2 = mk_1$ for some natural number $m > 1$, then the algorithm with $K = k_2$ produces a policy which is at least as good as that with $K = k_1$. This nearly optimal issue is less troublesome from a practical standpoint than it first appears, since, for example, computers are not expected to reconfigure themselves more than a certain number of times an hour because of the overhead required for each reconfiguration. δ can then be chosen appropriately. For instance, in the numerical example that follows, we use $\delta = 3$ minutes. This means that reconfiguration can take place up to 20 times an hour: something that should be perfectly adequate for the example in question.

Theorem 2: Algorithm 2 converges.

Proof: We exploit the well-known fact that the strategy-improvement procedure converges for homogeneous systems. For every periodic system S_p with *finite* action set A_p, transition functions $q_{ij}^{(p)}(\bullet)$, and a reward structure, we can construct a discrete-time (with time-period T) homogeneous system S_h with finite action set A_h, transition functions $q_{ij}^{(h)}(\bullet)$, and a reward structure, such that

- for every policy π_α in the periodic system, there is a corresponding action $a_\alpha \in A_h$ such that $q_{ij}^{(p)}(\pi_\alpha) = q_{ij}^{(h)}(a_\alpha)$,
- the expected reward per time T of taking action a_α with S_h in state h is equal to that per period (of length T) due to policy π_α with S_p in the corresponding state p at the beginning of the period.

Clearly, running the strategy-improvement procedure for S_h is equivalent to running it for S_p. Convergence is thus established. Q.E.D.

V. Example

Let us consider a fault-tolerant multiprocessor system which uses n-modular redundancy, where $n = 2$ or 3, i.e., it can operate either in dyads or in triads. When it operates in dyads, it can detect one failure in any dyad, but not mask it. As a result, the affected computation must be rolled back or restarted. The cost of this is expressed through a penalty function, described in greater detail below. When the system operates in triads, however, it can mask up to one failure per triad by voting. Clearly, rollback or restart is not needed in this case.[1] There is a tradeoff here: if the system is configured in triads, the system can sustain failures without having to roll back or restart, while if the system is configured in dyads, there will tend to be more of them, and as a result, the throughput (when there is no failure) can be expected to be greater.

There is a repairman on call. There is a cost associated with summoning the repairman, and a nonzero time taken by him to get there. There is also a cost per unit time of keeping the repairman at the site. Handling the repairman resource also consists of balancing tradeoffs: sending him away too early might mean he will have to be called back—thus incurring a cost—while keeping him too long will also result in the incurral of some cost.

The state of the system expresses two things: whether the repairman is present, absent, or summoned, and how many processors are functional. In response to the states

[1]We assume that near-simultaneous failure of more than one processor in the same triad is vanishingly unlikely.

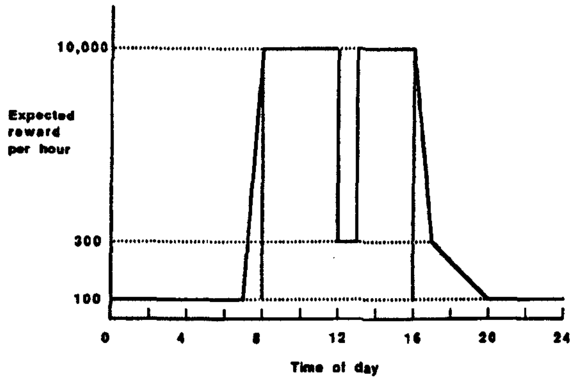
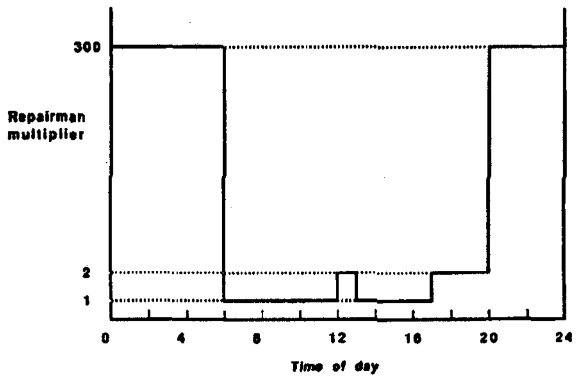


Fig. 1. Reward function.



Cost rate for keeping repairman = r_k • (Repairman multiplier)

Fig. 2. Cost of keeping the repairman.

are a set of actions, which are decisions to do with the repairman (keep him, send him away, or summon him) and decisions to do with the configuration (configure in triads or in dyads). Naturally, the actions that are available depend on the current system state. To take a trivial example, the action of summoning the repairman is meaningless when the repairman is already at hand.

The action depends on the system state and the reward structure. The reward structure in this example is particularly simple. The system receives a reward per unit time equal to the number of clusters (dyads or triads) functioning, minus any costs incurred due to the action at that state. The penalty incurred when a processor in a dyad fails is the product of a *penalty multiplier* and the reward rate. Because a triad has the failure-masking capability, there is no corresponding penalty for a processor failure when processors are configured in triads.

Processors fail, and are repaired (if the repairman is present), according to an exponential distribution with mean μ_f^{-1} and μ_r^{-1}, respectively. More specifically, the following symbols will be used for this example.

μ_r	repair rate.
μ_f	failure rate.
r_k	cost per hour of keeping the repairman.
r_{ss}	cost per hour of summoning the repairman.

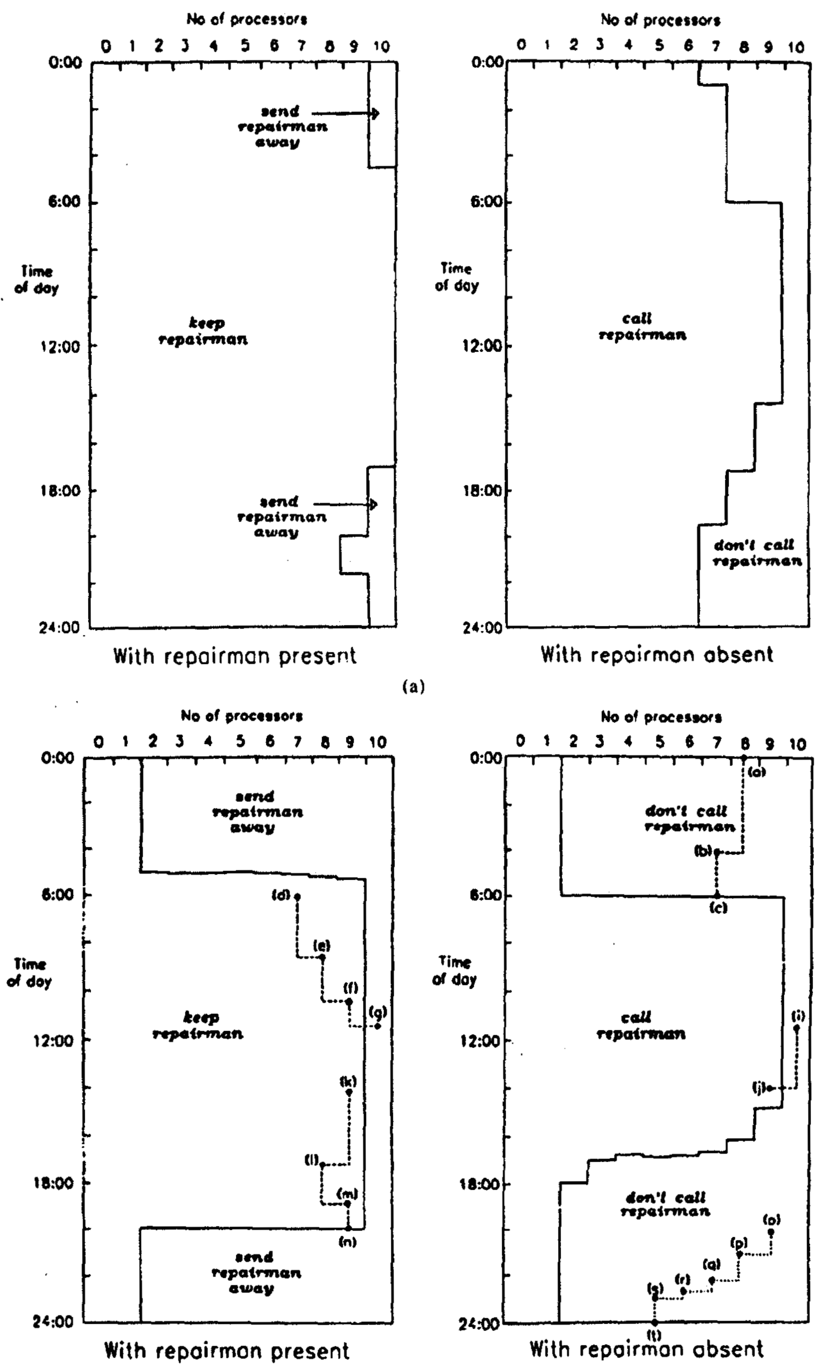


Fig. 3. Effect of changing r_k. r_s = 30 000, μ_f = 0.01, μ_r = 1.0, penalty multiplier = 300. (a) r_k = 1000. (b) r_k = 5000.

If the repairman is called at time $n\delta$, he arrives at time $(n + 1)\delta$.

The reward rates and the repairman costs are periodic with a period of 24 hours. They are shown in Figs. 1 and 2, respectively. The reward rates per processor group (dyad or triad) are low until about 7:00 in the morning, and then rise to their peak value by 8:00. This value is maintained, with a break of an hour at noon, until 5:00 in the evening, after which it declines. This reward rate is directly proportional to the job arrival rate, and is the means by which changes in the arrival rate are accounted for. Repairman costs are greatly magnified when he is called in after normal business hours. We have set K = 480, i.e., the day is divided down into 480 3-minute segments.

Numerical results are contained in Figs. 3–7. Fig. 3 deals with the effect of r_k, i.e., the cost of having the repairman on-site on the optimal action. As expected, the

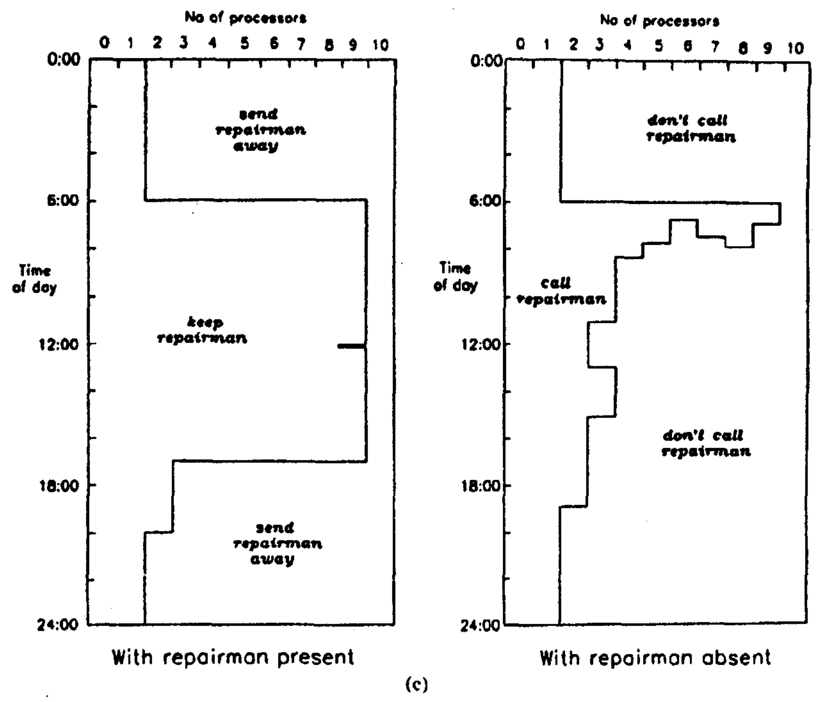


Fig. 3. (*Continued.*) (c) r_k = 25 000.

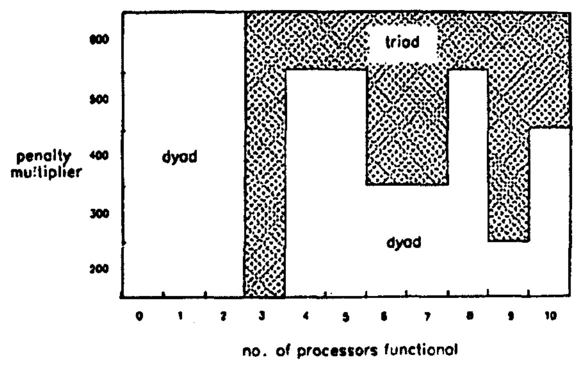


Fig. 4. Effect of failure penalty on configuration r_k = 1000, r_s = 30 000, μ_v = 0.01, μ_r = 1.0.

system tends to use him less when he is more expensive. As r_k increases, it becomes better to send the repairman away during particularly expensive periods (e.g., during the lunch hour or in the evening), and to accept the additional cost r_{ss} of summoning him later. A sample trajectory is plotted in Fig. 3(b). The system starts the day (0:00 hour) with eight processors functional, and the repairman away. At 4:00 AM, a processor fails (point *b*). Still, the repairman is not called until 6:00 AM, when the cost of keeping him is sufficiently low. With the repairman present, the system is brought up to eight functional processors by about 9:00 AM, (point *e*), to nine processors (point *f*) by about 10:00 AM, and fully functional (point *g*) around 11:00 AM. At this point, the repairman is sent away. The sample path for the second half of the day can be interpreted in the same way: at night, for example, with the reward rate down and the repairman expensive, even a bad succession of failures does not prompt the summoning of the repairman: not unless there are fewer than two processors functional is the repairman summoned.

Fig. 4 considers the effect of the penalty of dyad/triad failure: as the penalty increases, it shows that the system configures itself more and more into triads. When the penalty is 200, the only triad formed is when there are only three processors functional: this is obvious since with just that many processors available, there will be as many triads as there can be dyads. As the penalty for failure increases, triads are preferred more and more, despite the reduction in throughput that results. When the penalty is 300, the system now configures into triads for states 10 and 20 (nine functional processors) as well. As the penalty rises to 400, this is the case for 6 and 7 functional processors in addition to those mentioned above. Finally, when the penalty is 600, the system is always configured into triads except when there are not sufficient processors to make up even one triad.

The change in penalty of failure also affects how the repairman is handled. In Fig. 5, we plot the repairman curves for two penalty multipliers: 200 and 500. As one might expect, when the failure penalty is large, the repairman is called sooner and retained longer.

In Fig. 6, we consider the case when the penalty is constant and not a function of time, and show how the changing reward rate affects the optimum configuration. Below three functional processors, there is no decision to be taken: the system has to work in a dyad. When, for instance, there are nine processors functioning, the system

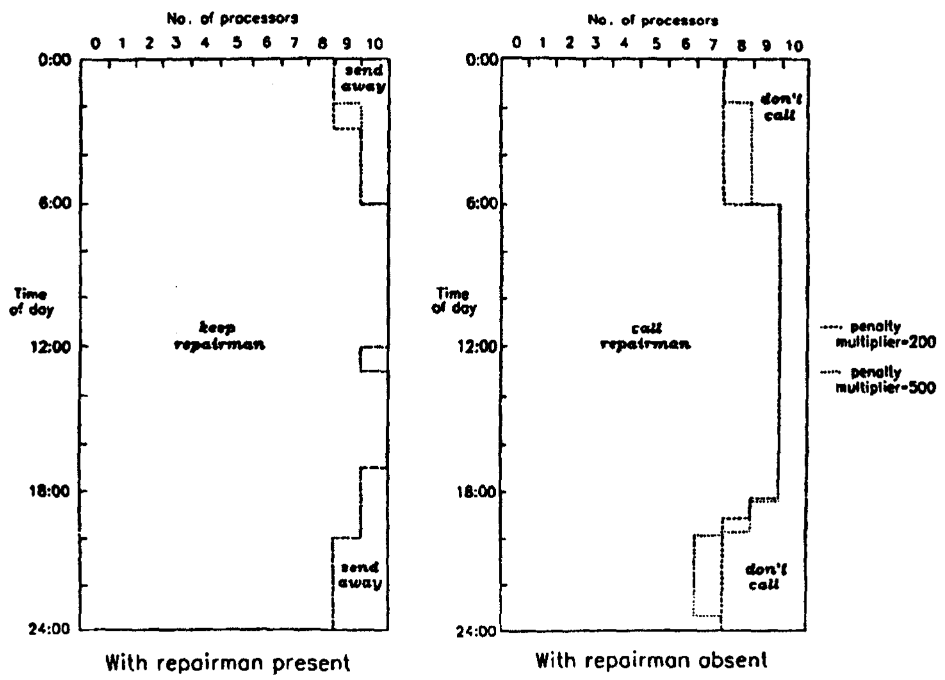


Fig. 5. Effect of failure penalty on repairman. $r_k = 1000$, $r_s = 30\ 000$, $\mu_f = 0.01$, $\mu_r = 1.0$.

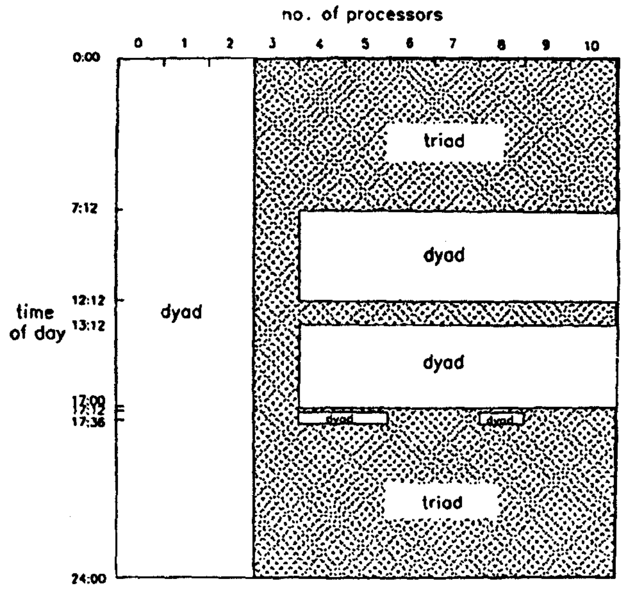
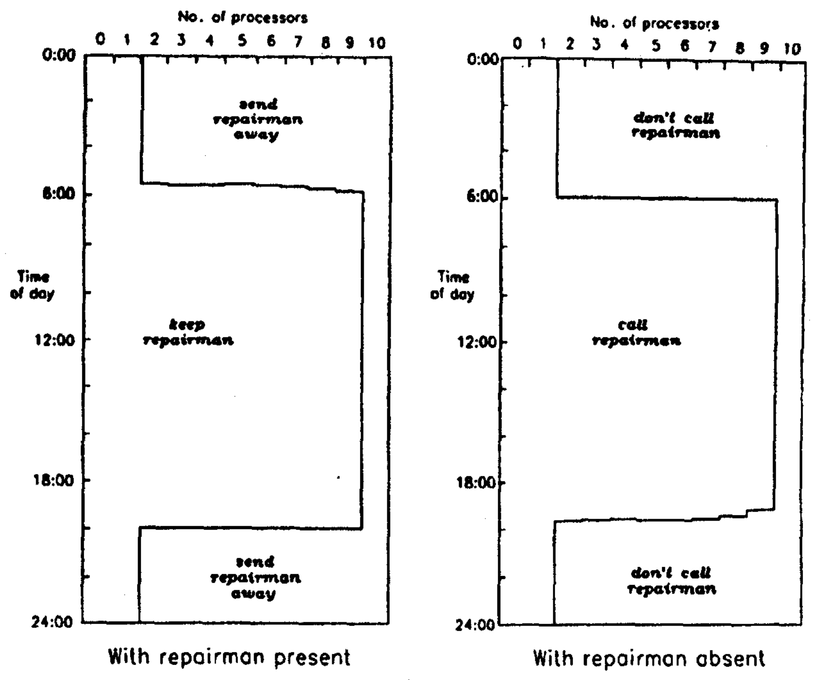


Fig. 6. Effect of time of day on configuration with a constant penalty of 10 000.

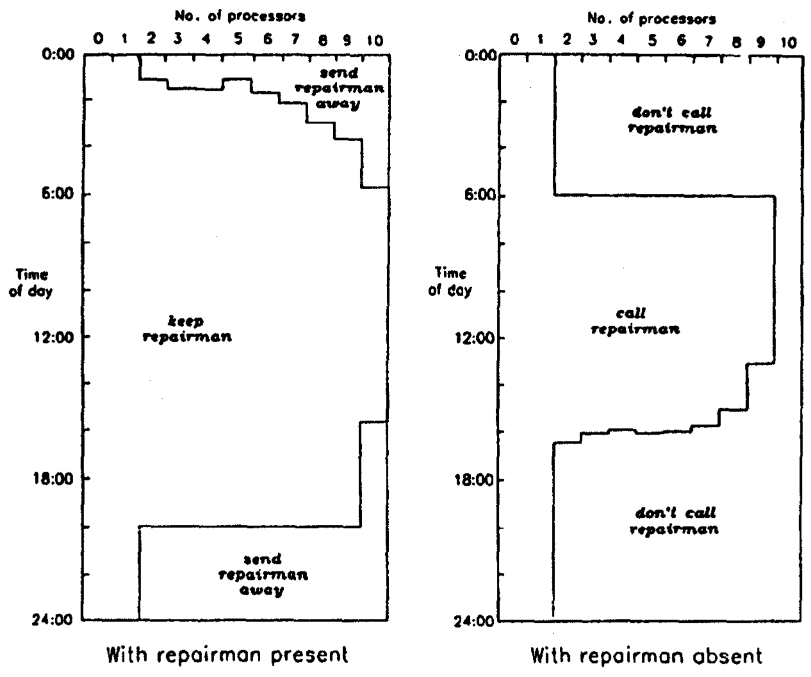
is configured into triads from midnight to 7:12 AM, when it switches to dyads. It switches back to triads at 12:12 PM, and back again to dyads at 1:12 PM. Finally, at 5:00 PM, when the reward rate begins to drop, the system goes back to operating in triads.

Fig. 7 considers the effect of increasing the cost of summoning the repairman. The repairman is now kept for a greater number of states when the cost of summoning him becomes very great: it is better to pay the cost of keeping the repairman under such circumstances. When $r_{ss} = 10^6$, the repairman is sent away only when all processors are functioning, and it is eight o'clock in the evening (the time when the reward rate is very small and the cost of keeping the repairman is especially large).

While all these trends are intuitively clear and do not need a sophisticated algorithm to determine, the exact epochs at which the repairman should be called or sent away, and the system configured into triads or dyads, cannot be



(a)



(b)

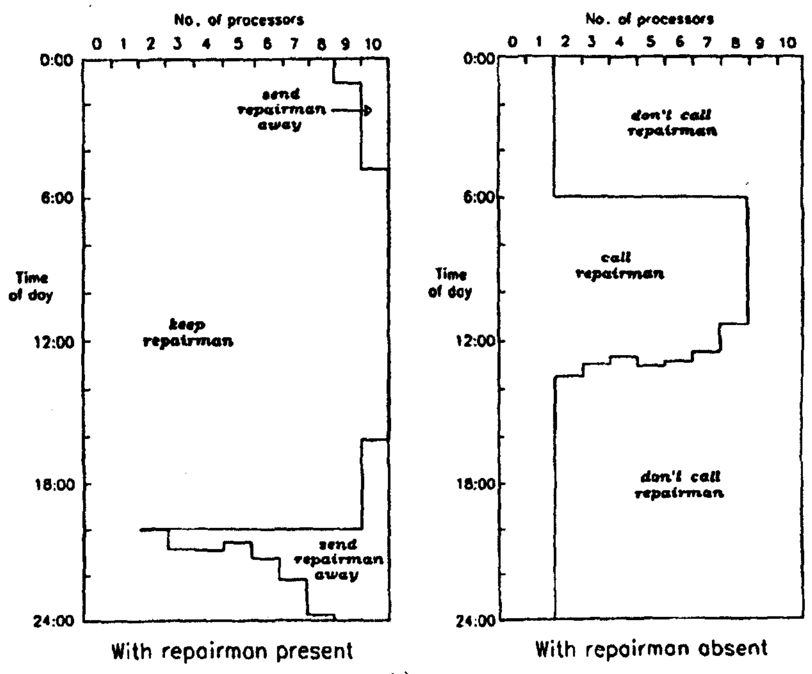
Fig. 7. Effect of summoning-cost on repairman. $r_k = 3000$, $\mu_r = 0.01$, $\mu_s = 1.0$, penalty multiplier = 300. (a) $r_s = 5000$. (b) $r_s = 50\,000$.

obtained through intuition alone, and do require an algorithm such as this one.

VI. Discussion

Because of its inherent difficulty, the problem of controlling resources in distributed systems has usually been treated in an ad hoc manner. Such a trend could remove the various advantages of distributed systems over uniprocessor systems. In this paper, we have countered this trend with the development of a rigorous, quantitative method for optimally controlling resources in a distributed system.

The resource control problem is of great practical significance because gracefully degrading systems are being used increasingly in such commercial fields as banking and travel. In such systems it is possible to estimate run-



(c)

Fig. 7. (*Continued.*) (c) r_s = 100 000.

ning costs, the benefits from having a certain throughput at various times of the day, repair costs, etc., much more accurately than for the computers used in, say, a university computing center. Another application is in real-time embedded systems, such as those which control airliners or spacecraft, where the reward rates for various jobs may vary with the phase of the mission. A third application is, as was mentioned in the Introduction, in local-area networks such as token rings, where one has the problem of allocating bandwidth to each of the rings, subject to a constraint on the total bandwidth allowed.

References

[1] J. F. Meyer, "Closed-form solutions of performability," *IEEE Trans. Comput.*, vol. C-31, no. 7, pp. 648-657, July 1982.
[2] D. G. Furchtgott and J. F. Meyer, "A performability solution method for degradable nonrepairable systems," *IEEE Trans. Comput.*, vol. C-33, no. 6, pp. 550-554, June 1984.
[3] L. Donatiello and B. R. Iyer, "Analysis of a composite performance reliability measure for fault tolerant systems," IBM Thomas J. Watson Research Center, Yorktown Heights, NY, Res. Rep. RC-10325, Jan. 1984; also in *J. ACM*, vol. 34, no. 1, pp. 179-199, Jan. 1987.
[4] B. R. Iyer, L. Donatiello, and P. Heidelberger, "Analysis of performability for stochastic models of fault-tolerant systems," IBM Thomas J. Watson Research Center, Yorktown Heights, NY, Res. Rep. RC-10719, Sept. 1984; also in *IEEE Trans. Comput.*, vol. C-35, no. 10, pp. 902-907, Oct. 1986.
[5] Y. H. Lee and K. G. Shin, "Optimal reconfiguration strategy for a degradable multi-module computing system," Comput. Res. Lab., Univ. Michigan, Ann Arbor, MI, Tech. Rep. CRL-TR-41-84, Sept. 1984; also in *J. ACM*, vol. 34, no. 2, pp. 326-348, Apr. 1987.
[6] R. A. Howard, *Dynamic Probabilistic Systems, vol. II, Semi-Markov and Decision Processes*. New York: Wiley, 1971.
[7] C. Derman, *Finite State Markovian Decision Processes*. New York: Academic, 1970.
[8] S. M. Ross, *Applied Probability Models with Optimization Applications*. San Francisco, CA: Holden-Day, 1970.
[9] A. Federgruen and H. C. Tijms, "The optimality equation in average cost denumerable state semi-Markov decision problems, recurrency conditions and algorithms," *Appl. Prob.*, vol. 15, pp. 356-373, 1978.
[10] R. Bellman, "Functional equations in the theory of dynamic programming-iv: A direct convergence proof," *Ann. Math.*, vol. 65, pp. 215-223, Mar. 1957.

Kang G. Shin (S'75-M'78-SM'83) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and the M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, NY, in 1976 and 1978, respectively.

He is a Professor in the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, which he joined in 1982. He has been very active and authored/coauthored over 150 technical papers in the areas of fault-tolerant computing, distributed real-time computing, computer architecture, and robotics and automation. In 1987, he received the Outstanding Paper Award from the IEEE TRANSACTIONS ON AUTOMATIC CONTROL for a paper on robot trajectory planning. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are currently building a 19-node hexagonal mesh multicomputer, called **HARTS**, to validate various architectures and analytic results in the area of distributed real-time computing. From 1970 to 1972 he served in the Korean Army as an ROTC officer and from 1972 to 1974 he was on the research staff of the Korea Institute of Science and Technology, Seoul, Korea, working on the design of VHF/UHF communication systems. From 1978 to 1982 he was an Assistant Professor at Rensselaer Polytechnic Institute, Troy, NY. He was also a visiting scientist at the U.S. Airforce Flight Dynamics Laboratory in Summer 1979 and at Bell Laboratories, Holmdel, NJ, in Summer 1980. During the 1988-1989 academic year, he was a Visiting Professor in the CS Division, Electrical Engineering and Computer Science, UC Berkeley.

Dr. Shin was the Program Chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the General Chairman of the 1987 RTSS, and the Guest Editor of the 1987 August special issue of IEEE TRANSACTIONS ON COMPUTERS on Real-Time Systems. He is a Distinguished Visitor of the IEEE Computer Society. He is a member of ACM, Sigma Xi, and Phi Kappa Phi.

C. M. Krishna (S'78–M'79) received the B.Tech. degree from the Indian Institute of Technology, Delhi, the M.S. degree from Rensselaer Polytechnic Institute, Troy, NY, and the Ph.D. degree from the University of Michigan, Ann Arbor, all in electrical engineering.

Since September 1984, he has been on the faculty of the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst. He was a Visiting Scientist at the IBM Thomas J. Watson Research Center during the Summer of 1986. His research interests include distributed systems architectures and operating systems, real-time systems, reliability modeling, and queueing and scheduling theory.

Yann-Hang Lee (S'81–M'84) received the B.S. degree in engineering science and the M.S. degree in electrical engineering from National Cheng Kung University in 1973 and 1978, respectively, and the Ph.D. degree in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1984.

From 1984 to 1988, he was a Research Staff Member in the Architecture Design and Analysis Group at IBM Thomas J. Watson Research Center, Yorktown Heights, NY. Since August 1988, he has been an Associate Professor in the Department of Computer and Information Sciences, University of Florida, Gainesville. His research interests include distributed computing, parallel processing, performance modeling, database management systems, fault-tolerant computing, and VLSI testing.

Dr. Lee is a member of the IEEE Computer Society and the Association for Computing Machinery.