# Optimal Resource Control in Periodic Real-Time Environments

Kang G. Shin*, C. M. Krishna**, and Yann-Hang Lee***

*Real-Time Computing Laboratory
Dept. of EECS
The University of Michigan
Ann Arbor, MI 48109

**Dept. of ECE
University of Massachusetts
Amherst, MA 01003

***Computer Science Dept.
University of Florida
Gainesville, FL 32611

## Abstract

Three factors determine the optimum configuration of a multiprocessor at any epoch: the workload, the reward structure, and the state of the computer system. We present an algorithm for the optimal (more realistically, quasi-optimal) configuration of such systems used in real-time applications with periodic reward rates and workloads. This algorithm is based on Markov decision theory.

Reconfiguration in most real-time systems is currently limited to reacting to individual component failure. We suggest here that a change in the workload or the reward structure be as powerful a motivation for reconfiguration as component failure. Such changes occur naturally over the course of operation: we present as an example an on-line transaction processing system with a workload and reward structure that has a period of a day.

## 1.0 Introduction

Because multiprocessor systems can function (albeit sometimes in a degraded condition) in a very wide variety of configurations and environments, managing such systems is much more difficult than managing a conventional uniprocessor machine. In this paper, we deal with the problem of optimally controlling the resources that make up such systems.

In real-time applications, both component failure and workload variation can lead to significant changes in the ability of the computer to meet the demands of the operating environment. It is quite easy to identify several cases where job arrival and workload follow certain periodic laws. For instance, the loading of on-line transaction processing systems typically reaches its peak from 9 to 5 and diminishes at night. This kind of periodic load distribution occurs also in other real-time systems, for example those used in air traffic control, as also in telecommunication and computer networks. To achieve the greatest rewards from a given computer system, it is important to reconfigure it in such a way as to optimally match (a) the current state of the hardware (e.g., the number of processors which are up) and (b) the current demands of the operating environment.

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. Putting it 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 imposed load 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. Or, a penalty may be introduced if job processing delays become excessive under heavy loading. 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 for a given quality of service (e.g. waiting time). Additionally, the system response time is a function of the load imposed on 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

The work was performed when Yann-Hang Lee was with IBM T. J. Watson Research Center, Yorktown Heights, NY. This work was supported in part by NASA under grant NAG-1-296, and by the National Science Foundation under grant NSFDMC-8504971. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors, and do not necessarily express the view of the funding agencies.

insufficient, and must be supplemented by algorithms 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.

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 and Meyer [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 real-time computer system.

The expected total reward accumulated during a mission lifetime as an optimization criterion has been studied in [5] for configuring degradable systems. The results suggest that the system should perform not only *passive reconfiguration* to respond the occurrence of a failure, but also *active reconfiguration* to better adapt itself to the current needs of the operating environment. 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 that the system performs active reconfiguration.)

The solution provided in [5] is limited to applications with non-repairable system and with a finite mission lifetime. Its underlying system model is represented by an acyclic Markov process. To examine the optimal system operation in a periodic real-time environment, additional considerations must be kept in mind: (1) time dependence due to the periodic nature of the system, (2) cyclic Markov process for system states if the system is repairable, and (3) average reward criterion. A solution methodology is suggested in the paper which consists of two steps: in the first step, an embedded Markov chain and decision process are established to model the system between successive periods, and in the second step, recursive computations are conducted to search for an improved strategy and to calculate state transition probabilities of the embedded Markov chain. We shall show that the suggested approach converges to the optimal solution.

This paper is organized as follows. In Section 2, the optimal system operation problem in periodic real-time environment is stated formally. In Section 3, we construct a solution methodology based on the *strategy improvement procedure* from decision theory. In Section 4 we provide a numerical example, and the paper concludes with Section 5.

## 2.0 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 rates $\mu_{ij}(a)$, i.e., the rate 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 [6,7,8] 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 space is the aggregate of all states, denoted by $\Phi$. $\Phi$ 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 $A_{i,t}$ be the set of all available actions or operational modes at time $t$ when $S(t) = i$, and let the system choose an action $a$ from $A_{i,t}$. It is easy to see that transitions between system states depend upon the current state and the current action or operational mode. When the causes for changing the availability/functionality of units are uncorrelated and the occurrences of failure in each unit are Markovian, state transitions will be conditional independent of the past states and the past actions, given the present. Thus, the system's behavior at time $t$ can be fully specified by $S(t)$, $a(S(t),t)$ and the associated $\mu_{ij}(a(S(t),t))$ , where $S(t) \in \Phi$ and $a(S(t),t) \in A_{S(t),t}$ is the action chosen at time $t$ with system state $S(t)$.

Let $\rho(i,a,t)$ be a reward rate at time $t$ associated with the system state $i$ and the action $a \in A_{i,t}$. This reward rate represents what the system can achieve, or may lose, per unit time, with $(i,a,t)$. In addition, we assume that there exists a *cost*, $c(i,a,t)$ ,

with the action $a$ taken when the system enters state $i$ at time $t$. The cost $c(i,a,t)$ represents the instantaneous cost (if any) of taking the action $a$. Certainly, both the reward rate and the cost of the system should be defined based on the job arrivals or load condition. For instance, a reward rate in a telecommunication system could be the number of calls connected within a unit time. Also, the cost of carrying on subsystem maintenance during the heavy load period is much higher than that during the light load period.

If there is a periodicity in the reward, state transition, available actions, or cost structures, then the system is *periodic*. If, as sometimes happens, their periods are different, the least common multiple of these becomes the system period. Let $T$ be the system period. The period, $T$, is obtained from practical considerations: it may be a day, a week, or any other period natural to the application. We shall represent the functions $\rho(i,a,t)$, $c(i,a,t)$, $\mu_{i,j}(a,t)$, and $A_{i,t}$ in terms of the relative time within a period. For instance, at time $t$, the reward rate associated with state $i$ and the action $a$ is $\rho(i,a,\tau)$, where $\tau = t - \lfloor t/T \rfloor T$ and $\lfloor x \rfloor$ is the greatest integer less than or equal to $x$.

With the system described above, we need to choose an action at time $t$, given that the system is in state $S(t)$. The action chosen not only controls the system's behavior (or system state transitions), but also determines the reward or cost the system receives during its operation. A *strategy* $\pi$ is specified by a set of actions, $\{a(i,t) \mid a(i,t) \in A_{i,t},\ i \in \Phi\}$. Thus, given a strategy $\pi$, the reward accumulated during $[0,t)$ can be expressed as

$$W_{\pi,i}(t) = \int_0^t \rho(S(\nu), a(S(\nu), \nu), \nu)d\nu - \int_0^t c(S(\nu), a(S(\nu), \nu), \nu)I(a(S(\nu), \nu), a(S(\nu^-), \nu^-))d\nu \quad (1)$$

where $S(0) = i$, $\nu^- = \lim_{\epsilon \to 0}(\nu - \epsilon)$ and $I(i,j) = 1$ if $i \neq j$, or 0 otherwise. Note that, in the above equation, the cost of switching an action is represented by the second term.

The resource control problem arising from this model is to determine a strategy $\pi^*$ such that the average expected reward which is defined as

$$\lim_{t \to \infty} \{ \frac{W_{\pi^*,i}(t)}{t} \} \quad (2)$$

is maximized with respect to all acceptable 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.

## 3.0 Optimization Approach

In this section, we propose an optimization approach to solve the problem formulated in Section 2. The solution is divided into two steps: in the first step, we show that an embedded Markov chain can be used to model the system behavior between successive periods, and then in the second step, an algorithm based on Chapman-Kolmogorov equations is developed to search for the optimal strategy.

### 3.1 Embedded Markov Chain and Decision Process

When the reward structure is periodic, it is convenient to partition a strategy into separate sub-strategies. At time $t \in [mT,(m+1)T)$, an action $a(\tau, i)$ of the m-th sub-strategy $\pi_m$ is applied where $m = \lfloor t/T \rfloor$ and $\tau = t - mT$. In other words, each sub-strategy consists of actions for a period $T$ and is expressed as $\pi_m = \{a(\tau,i) \mid i \in \Phi, \tau \in [0,T)\}$. Similarly, the optimal strategy $\pi^*$ is given by a set of $\pi_m^*$ where $m = 0,1,2,....$

Given a set of sub-strategies, the system state transitions are determined by the current state and the current action which, in turn, specify the transition rate $\mu_{i,j}(a,t)$. Thus, the system can be viewed as a time-varying Markov process. Let us examine the system at times $mT$, $m = 0,1,2,...$, and let the system state at time $mT$ be $s_m$. When a sub-strategy $\pi_m$ is applied, the system will transfer into state $s_{m+1}$ at time $(m+1)T$ with probability $q_{ij}(\pi_m) = Prob\{s_{m+1} = j \mid s_m = i\}$. The system can then be modelled by an embedded Markov chain specified by the sequence $\{s_m\}$. Also, let $w(s_m, \pi_m)$ denote the performability accumulated between $mT$ and $(m+1)T$, which is determined by the state $s_m$ and the sub-strategy $\pi_m$. The original optimization problem defined in Section 2 is then, transformed into a discrete Markov decision process [6,7,8] where actions are equivalent to the sub-strategies in our periodic system and the objective function becomes $\lim_{M \to \infty} \frac{1}{M} \sum_{m=0}^{M} w(s_m, \pi_m)$.

Because we are interested in maximizing the asymptotic reward rate, the transient states have no effect on our objective and are therefore not considered. Also, we limit ourselves to the case where there is only one recurrent class in this 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.

There are several existing results of Markov decision process which we shall apply to solve our optimization problem. First, since the system behavior is periodic and the state space $\Phi$ is finite, the optimal sub-strategies must be stationary [10,11]. That is, $\pi_k^* = \pi_l^*$ for all $k,l$. This implies that an optimal sub-strategy be used periodically. From Theorem 7.6 of [8], an optimal strategy exists if there exists a bounded function $h(i)$ for $i \in \Phi$, and a constant $g$ such that

$$h(i) = \max_{\pi_m}\{w(i, \pi_m) + \sum_{j=1}^{n} q_{ij}(\pi_m)h(j) - g\} \quad (3)$$

where $g$ is the maximal performability.

Notice that the condition in this theorem will be met automatically when $w(i, \pi_m)$ is bounded for all $i$ and sub-strategy $\pi_m$, and all stationary strategies give rise to a finite and irreducible state space. The above equation can be solved either by linear programming or by using the Howard's strategy improvement procedure [6]. For each possible $\pi_m$, we need to determine both $q_{ij}(\pi_m)$ and $w(i, \pi_m)$, which, in turn, depend upon the sub-strategy $\pi_m$ applied.

### 3.2 Optimal Sub-strategy

To apply the policy improvement procedure in our decision process, we need to solve the following linear equation first:

$$\tilde{h}(i, \pi_m) = w(i, \pi_m) + \sum_{j=1}^{n} q_{ij}(\pi_m)\tilde{h}(j, \pi_m) - g \quad (4)$$

and then to search for an improved strategy.

In our algorithm (to be presented), the period is digitized to make it suitable for digital implementation. Let $K$ be a large natural number, $\delta = T/K$, and let $a(i, k)$ be the action taken when the system is in state $i$ and $\tau \in [k\delta,(k + 1)\delta)$ . We want to determine the optimal sub-strategy

$$\pi_m^*(K) = \{a^*(i,k) \mid i \in \Phi, \text{ and } k = 0, 1, ..., K - 1\}$$

Also, it is convenient here to regard $p_{ij}(a, k)$ as the probability that the state transition $i \to j$ occurs in the k-th time period of duration $\delta$. This probability can be easily calculated based on the transition rate $\mu_{ij}(a,t)$. With $p_{ij}(a, k)$, $\rho(i, a, k)$ and $c(i,a, k)$ as the state transition probability, reward rate, and cost of action $a$ at the k-th duration under an action $a$, the Chapman-Kolmogorov backward equations can be used to calculate the state transition probability and accumulated reward during each period. These are represented by the following equations:

$$q_{ij}(k) = \sum_{l=1}^{n} p_{il}(a(i, k),k)q_{lj}(k + 1) \quad (5)$$

$$\begin{aligned} w(i, k) = {} & \rho(i, a(i, k),k)\delta - \\ & - \sum_{l=1}^{n} p_{il}(a(i,k),k)c(l,a(l,k + 1),k + 1)I(a(i,k),a(l,k + 1)) \\ & + \sum_{l=1}^{n} p_{il}(a(i, k))w(l, k + 1) \end{aligned} \quad (6)$$

where $q_{ij}(k)$ is the conditional probability that the system is in state $j$ at time $K\delta$ given it is in state $i$ at time $k\delta$ and $w(i, k)$ is the reward accumulated during $[k\delta, K\delta)$. In the right hand side of Eq. (6), the first term indicates the reward received during the k-th duration. The second term is the possible cost due to a new action or transition from the k-th duration to the k+1-th. The final term gives the accumulated reward starting from the k+1-th duration.

Similarly, to find an improvement in $\pi_m$, we shall search for actions $\hat{a}(i,k)$ , $k = K - 1$ to 0 and $i \in \Phi$ , which maximize the following recursive equation:

$$\begin{aligned} \hat{h}(i,k) = {} & \rho(i, a(i,k),k)\delta \\ & - \sum_{l=1}^{n} p_{il}(a(i,k))c(l,a(i,k + 1),k + 1)I(a(i,k),a(l,k + 1)) \\ & + \sum_{l=1}^{n} p_{il}(a(i,k),k)\hat{h}(l, k + 1) \end{aligned} \quad (7)$$

The initial condition of the above equations should be set to $\hat{h}(i,K) = \tilde{h}(i, \pi_m)$ which is the solution of Eq. (4).

We summarize the computation suggested above in the following algorithm which determines the optimal sub-strategy $\pi_m^*$.

**Algorithm for Optimal Sub-Strategy:**

1. Select an arbitrary strategy

   $$\pi_m = \{a(i,k) \mid i \in \Phi, \ k = 0, 1, ..., K - 1, \ a(i,k) \in A_{i,k\delta}\}.$$

2. For each $i = 1, 2,..., n$, calculate the state transition probability and reward between periods $mT$ and $(m + 1)T$. This can be done by solving Eqns. (5) and (6) iteratively, with the initial conditions $q_{ij}(K) = 0$ if $j \neq i$, 1 otherwise, and $w(i,K) = 0$. Then, $q_{ij}(\pi_m)$ and $w(i, \pi_m)$ are set to $q_{ij}(0)$ and $w(i, 0)$, respectively.

3. We must solve the set of linear equations (4), which describes the embedded Markov chain between successive periods. Note that, since $[q_{ij}(\pi_m)]$ is a transition matrix, the set of equations (4) is dependent, and thus, $\tilde{h}(i, \pi_m)$ cannot be uniquely determined. This is not a limiting factor in practice since it is easy to show that only the relative, and not the absolute, values of $\tilde{h}(i, \pi_m)$ affect our search for the optimal actions. We can set one of the $\tilde{h}(i, \pi_m)$ to some value, and solve for the rest.

4. With $\hat{h}(i,K) = \tilde{h}(i, \pi_m)$, we search for those actions $\hat{a}(i,k)$ which maximize $\hat{h}(i,k)$ of Eq. (7) for $k = K - 1$ to 0, and $i \in \Phi$ .

5. Let $\pi_m' = \{\hat{a}(i,k) \mid i \in \Phi,\ k = 0, 1,...,K-1\}$. If $\pi_m' = \pi_m$, then stop the algorithm with $\pi_m^* = \pi_m'$. Otherwise, set $\pi_m = \pi_m'$, and go to step 2.

For finite $K$, this algorithm yields nearly-optimizing, but not necessarily optimal, actions, since the actions are taken at specific epochs in the operating interval (namely, at multiples of $\delta$). It is easy to show, from the fundamentals of dynamic programming, that the algorithm tends to be optimal as $K \to \infty$ [10,11]. 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 $K = k_2$ yields 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, reconfiguration of a computer system is too expensive (in terms of time overhead) to be performed frequently. $\delta$ 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:** The above algorithm 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 $\pi_a$ in the periodic system, there is a corresponding action $a_a \in A_h$ such that $q_{ij}^{(p)}(\pi_a) = q_{ij}^{(h)}(a_a)$,
- the expected reward per time $T$ accruing as a result of taking action $a_a$ with $S_h$ in state $h$ is equal to the reward per period (of length $T$) due to policy $\pi_a$ 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.

◊

## 4.0 Optimal Operation of an On-line Transaction Processing System

Consider a transaction processing system which has a periodic job arrival rate $\lambda(t)$ with a period of 24 hours. The job arrival rate is shown in Figure 1: it is low until about 7:00 in the morning, and then rises to its 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.

The system consists of ten processors which can be grouped into several operation units which are either dyads or triads. Thus, a single processor failure can be detected if dyads are used or masked immediately if triads are used. We further assume that the system must be configured entirely in triads or entirely in dyads. The reward rate the system receives is assumed to be the product of the arrival rate and the number of operation units. Given there are $i$ processors available, the reward rate is $\lambda(t)\lfloor i/2 \rfloor$ or $\lambda(t)\lfloor i/3 \rfloor$ at time $t$ when the system is configured in dyads or triads, respectively.

When a processor in a dyad fails, there is a penalty incurred since the job that is interrupted must be restarted. We express this through a penalty multiplier which sets the penalty for failure equal to the arrival rate at the failure moment times the multiplier. However, if a processor in a triad fails, errors can be masked by voting, and no job recovery is necessary. The penalty for processor failure is zero in this configuration.

There is a repairman on call. We have the choice of summoning him if he is away, and of keeping him or sending him away if he is on-site. The cost rate of keeping a repairman per unit time is also assumed to be periodic with a period of 24 hours. The rate is shown in Figure 2. As one might imagine, this rate is greatly magnified when a repairman is called in after normal business hours.

The state of the system expresses two things: whether the repairman is present or absent, and how many processors are functional. That is, $\Phi = \{(i, \zeta) \mid i \in \{01,2,3,...,10\},\ \zeta \in \{\text{present, absent}\}\}$. The action taken by the system as it enters each state is to decide what to do with the repairman (whether to call, keep, or send him away), and how to configure the system (into dyads or triads). There is a cost associated with summoning the repairman and a cost per unit time for keeping him. As explained above, there is a penalty incurred when a dyad fails. The system receives a reward per unit time equal to the number of groups (dyads or triads) functioning minus any costs incurred due to the action at that state.

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

$\mu_r$: repair rate.

$\mu_f$: failure rate.

$r_k$: cost per hour of keeping the repairman.

$r_s$: cost per hour of summoning the repairman.

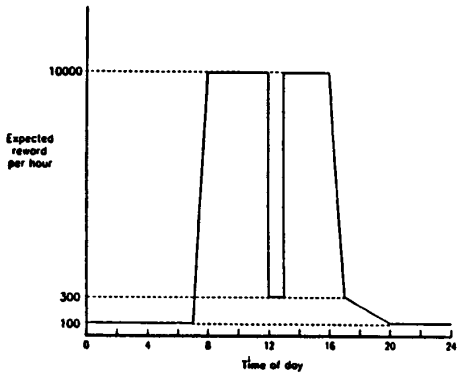



**Figure 1. Reward Function**

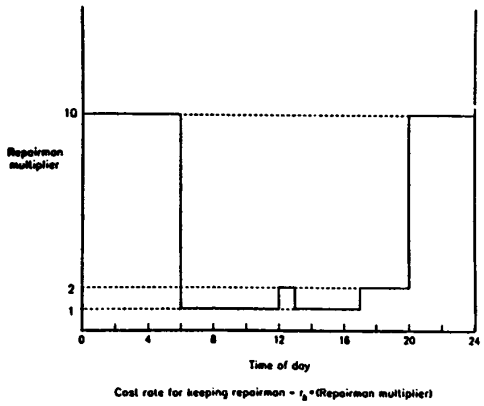
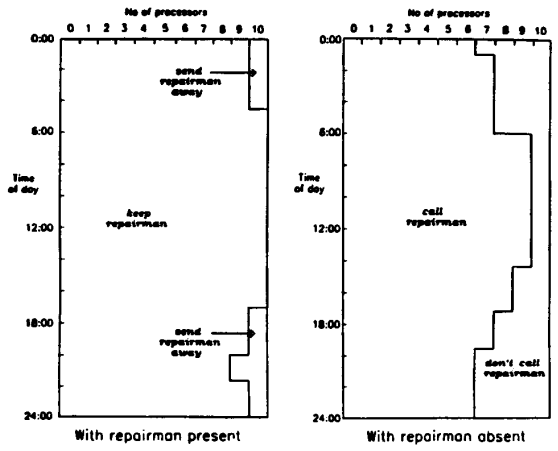


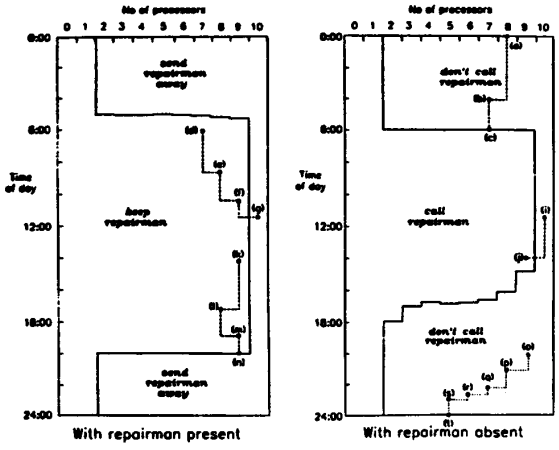

**Figure 2. Cost of Keeping the Repairman**

We have set $K = 480$, i.e., the day is divided down into 480 3-minute segments. We also assume that if the repairman is called at time $k\delta$, he arrives at time $(k + 1)\delta$.

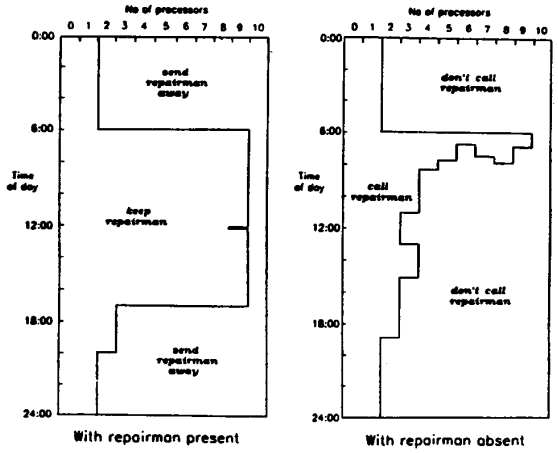
Numerical results are presented in Figures 3 to 7. Figure 3 deals with the effect of $r_k$, i.e., the cost of having the repairman on-site, on the optimal action. As expected, the system tends to use him less when he is more expensive. As $r_k$ increases, it becomes better to send the repairman away during particularly expensive periods (e.g., during the lunch hour or in the evening), and to accept the additional cost, $r_s$, of summoning him later. A sample trajectory is plotted in Figure 3b. The system starts the day (0:00 hours) 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




**3a. $r_k = 1000$**




**3b. $r_k = 5000$**




**3c. $r_k = 25000$**

**Figure 3. Effect of Changing $r_k$**
**$r_s = 30000$, $\mu_f = 0.01$, $\mu_r = 1.0$, penalty multiplier=300**

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.

Figure 4 considers the effect of the penalty incurred on 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 when there are 9 functioning 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.

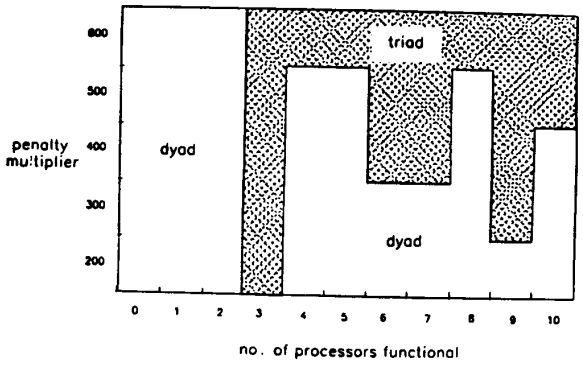



**Figure 4. Effect of Failure Penalty on Configuration**
**$r_k = 1000$, $r_r = 30000$, $\mu_f = 0.01$, $\mu_r = 1.0$**

The change in penalty of failure also affects how the repairman is handled. In Figure 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 Figure 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 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.

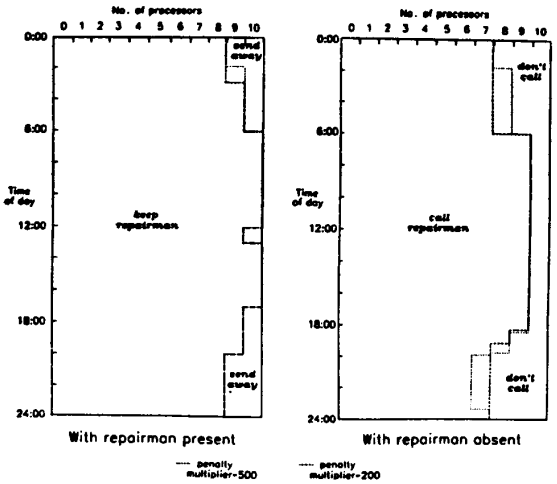



**Figure 5. Effect of Failure Penalty on Repairman**
**$r_k = 1000$, $r_r = 30000$, $\mu_f = 0.01$, $\mu_r = 1.0$**

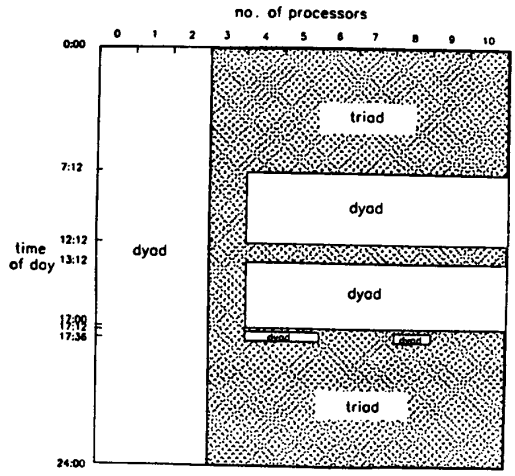



**Figure 6. Effect of Time of Day on Configuration with a Constant Penalty of 10000.**

Figure 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 repairmen under such circumstances. When $r_s = 10^6$ , 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 don't 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 obtained through intuition alone, and do require an algorithm such as this one.

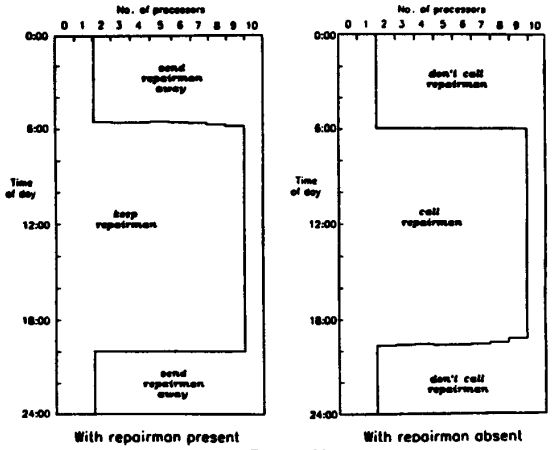
## 5.0 Discussion

In this paper, we have studied the problem of optimally controlling resources and system operations in periodic real-time environments. The problem is of great practical significance because gracefully-degrading systems are being used more and more in such commercial fields as banking, communication and process control. Such a environment often has a periodic job arrival or workload. It is also possible to estimate more or less accurately, running costs, the benefits from having a certain throughput at various times, repair costs, etc. We have proposed an algorithm which uses an embedded Markov decision process to model system behavior over a period. Optimal control of the kind described here is the key to achieving good performance and availability.

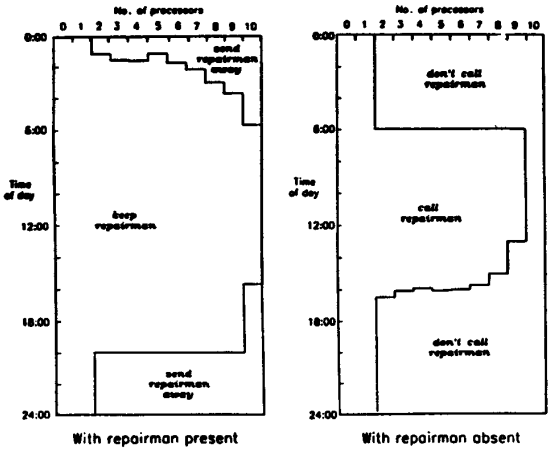
In this paper, we discussed on-line transaction processing systems. However, this algorithm should be useful wherever a multiprocessor system has to cope with a job load that has a roughly periodic structure (no job loading is ever exactly periodic). Examples are real-time embedded systems, local-area networks, etc.


## Acknowledgement

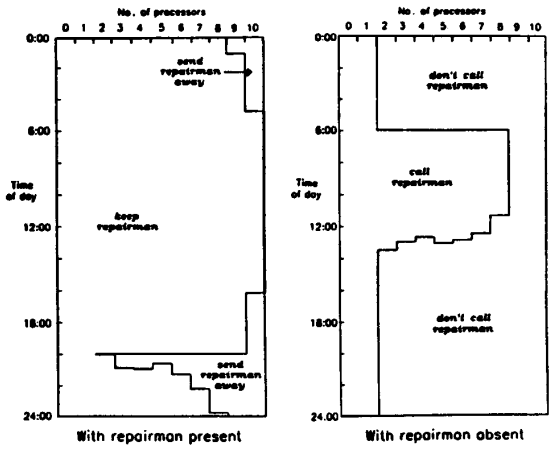
The authors would like to thank the referees for many helpful comments that improve the paper.


7a. $r_s = 5000$




7b. $r_s = 50000$




7c. $r_s = 100000$

**Figure 7. Effect of Summoning-Cost on Repairman**
**$r_s = 3000$, $\mu_f = 0.01$, $\mu_r = 1.0$, penalty multiplier = 300**

## References


[1] J. F. Meyer, "Closed-form Solutions of Performability," *IEEE Trans. Computers,* 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. Computers,* 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," *Research Report RC-10325* IBM Thomas J. Watson Research Center, Yorktown Heights, NY, Jan. 1984, (to appear in *Journal of the ACM* ).

[4] B. R. Iyer, L. Donatiello, and P. Heidelberger, "Analysis of Performability for Stochastic Models of Fault-Tolerant Systems," *Research Report RC-10719* IBM Thomas J. Watson Research Center, Yorktown Heights, NY, Sep. 1984, (to appear in *IEEE Trans. Computers).* ).

[5] Y.-H. Lee and K. G. Shin, "Optimal Reconfiguration Strategy for a Degradable Multi-Mudule computing System," *Journal of ACM* Vol. 34, No. 2, April 1987, pp. 326-348.

[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,* Academic Press, 1970.

[8] S. M. Ross, *Applied Probability Models with Optimization Applications,* Holden-Day, 1970.

[9] E. B. Dynkin and A. A. Yushkevich, *Controlled Markov Processes,* Springer-Verlag, 1979.

[10] A. Federgruen and H. C. Tijms, "The Optimality Equation in Average Cost Denumerable State Semi-Markov Decision Problems, Recurrency Conditions and Algorithms," *J. Applied Probability,* Vol. 15, 1978, pp. 356-373.

[11] R. Bellman, "Functional Equations in the Theory of Dynamic Programming -- IV, A Direct Convergence Proof," *Annals Math. Vol. 65, pp. 215-223, March 1957.*