Processor Tradeoffs in Distributed Real-Time Systems

C. M. KRISHNA, MEMBER, IEEE, KANG G. SHIN, SENIOR MEMBER, IEEE, AND INDERPAL S. BHANDARI

Abstract—**Optimizing the design of real-time distributed systems is important since the systems are frequently critical to life. This optimization is a difficult problem, and heuristics and designer judgment are called for in the process. The chief cause of the difficulty is the large number of parameters under the designer's control which impact performance and life-cycle cost. We study the interplay between the more important parameters in this paper using two objective measures, i.e., the *mean cost* and the *probability of dynamic failure* in [6], [10]. Among these are the processor burn-in time and processor replacement policy. A central feature of this work is a look at how the application requirements affect the optimality of the distributed systems; indeed, the application requirements are an integral part of the analysis.**

Index Terms—**Burn-in time, hazard rate, mean cost, probability of dynamic failure, processor tradeoffs, real-time multiprocessor and control, replacement policy.**

I. Introduction

The objective of all computer design is to optimize a performance criterion subject to constraints imposed on the life-cycle cost, the application, and the performance characteristics of the available components. Unfortunately, it is impossible to solve this optimization problem because 1) it usually results in a mixed integer-and-noninteger nonlinear programming problem for which no efficient solutions exist, and 2) in most instances it is impossible to specify a single performance metric that captures fully the various facets of "performance." So, designers typically have to proceed heuristically, using sensitivity analyses or branch-and-bound techniques and rule-of-thumb performance criteria. In any case, approximate (but robust) performance figures are all that is usually required for general-purpose systems. Things are very different with distributed computers for real-time control applications such as nuclear reactors and aircraft. (We term these *real-time computers.*) Such machines are becoming increasingly important in practice.

Real-time computers work in the feedback loop of the system they control, i.e., *controlled system.* They derive their input from sensors and the operator(s), and their output is used to control actuators or to update displays. Because the control function is well-defined, so is the set of tasks that the computer has to run. These tasks may be run cyclically in an indefinite loop, or may be triggered by some event in the controlled system or the environment. A task trigger can thus be produced by the operator, the environment, the controlled system, or any combination of these.

The differences between computers for such applications and general-purpose machines include the following.

- Reliability specifications are stringent and so an approximation to the probability of failure is acceptable only if it can be shown to be an upper bound.
- All executed tasks belong to predefined task classes, and the interarrival time distribution and service requirement of these classes are known to a fair accuracy.
- There are now performance measures available for real-time computers which are specific to the application at hand, and which express performance in application-specific terms [6], [10].

The real-time computer is a "black box" from the point of view of the controlled system. It is exemplified entirely by the speed and precision of its output. Thus, the computer response time can be used as a parameter with which to characterize computer performance.[1] If the response time is greater than some given limit, called the *hard deadline,* catastrophic system failure can occur. A real-time task is said to be *critical* if there is a finite hard deadline associated with it. The hard deadline, denoted by t_{di} for task class i, is a maximum controller (computer) think time which depends on the dynamics of the controlled system and its environment.[2] Since the operating environment is stochastic, the hard deadline is a random variable. (See [10] for a detailed account of this.) We denote the probability distribution function of t_{di} by $F_{di}(\cdot)$. When system failure occurs due to a deadline being missed, *dynamic failure* is said to have happened. The measure of reliability is called the *probability of dynamic failure,* and is denoted by p_{dyn}.

In [6], we define the "cost" associated with a computer response time of ξ to a control task i to be some function $f_i(\xi)$,[3] if the response time is less than the hard deadline, and infinity otherwise. Naturally, we assume that the functions $f_i(\cdot)$ are nondecreasing and continuous. All this is formalized by defining a *cost function.*

$$C_i(\xi)=\begin{cases} f_i(\xi) & \text{if } 0<\xi\le t_{di} \\ \infty & \text{otherwise.}\end{cases}$$

Manuscript received August 25, 1985; revised December 17, 1986. This work was supported in part by NASA under the Grant 1-296 and the ONR under Contract N0014-85-K-0122. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the funding agencies.

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

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

I. S. Bhandari is with the Department of Electrical and Computer Engineering, Carnegie-Mellon University, Pittsburgh, PA 15213.

IEEE Log Number 8715299.

[1] Response time is infinite by definition if an incorrect value is put out by the real-time computer.

[2] To avoid needless duplication, we allow noncritical tasks to have t_{di} defined for them, only these are infinite.

[3] We showed in [10] how to derive this function.

The cost function expresses the fact that since a positive computer delay aggravates latent system instability, the magnitude of the control called for will rise with the computer response delay. The cost function is an expression of cost in *physical* terms such as energy, fuel, etc., and is the extra cost of control—in these terms—that is incurred because of the nonzero nature of the computer response delay. Costs incurred add up; if we denote by Γ_{ij} the cost incurred in relation to version V_{ij} of task i, the total cost of control is $\Sigma_i \Sigma_j \Gamma_{ij}$. The *mean cost* (MC) is the mean total cost of control incurred over a mission lifetime, given that the dynamic failure has not occurred. See the Appendix for a list of terms associated with real-time control systems. In [6], we discuss the cost function in detail, and in [10], we provide a detailed case study of its determination and use aboard aircraft when the task is to control the deflection of the elevator during landing. So far as this present discussion is concerned, there are only two quantities of interest: the probability of dynamic failure and the mean cost. In other words, we shall analyze certain processor tradeoffs using these two objective measures.

We can now restate our optimization problem to make it specific to real-time computers: find a design which minimizes the mean cost for a particular application over a total working lifespan of a given number of missions, subject to the constraints of a) life-cycle cost, b) the interarrival and service time distributions for the various classes of tasks to be executed, c) the characteristics of the available components, and d) the maximum acceptable probability of dynamic failure.

While this problem is now more definite and formal, it is just as impossible to solve exactly and efficiently as the one with which we began. So we must study tradeoffs between the various aspects of the problem, and use human intuition and judgment to arrive at a quasi-optimal design. In this paper, we study several important tradeoffs which occur in one potentially important type of distributed real-time computer.

We study the impact of the application requirements (expressed through the task loading, the finite cost functions, and hard deadline distributions for the various classes of tasks) on computer performance (expressed through p_{dyn} and MC). And we consider the impact of the processor replacement policy and processor burn-in times (i.e., the time for which a processor is made to execute in a test setting) on computer performance. Finally, we calculate the number of processors that must be replaced under any replacement policy and burn-in time, thus computing an important factor in the life-cycle cost.

In order to be realistic, exponentially distributed service times are *not* assumed in our analysis, and an exact analytical solution to the models we study is as yet unknown. Analytical methods will be used to obtain upper and lower bounds to the reliability: to do this using simulation is impractical since p_{dyn} for critical real-time applications must be of the order of 10^{-5} or (usually) much less.

This paper is organized as follows. In Section II, we describe the architecture studied, and our modeling assumptions. Section III contains analyses such as of the impact of processor replacement strategies and the burn-in time on the probability of dynamic failure and mean cost. Section IV has some detailed numerical examples. The solution is numerically obtained, tabulated, and interpreted in the context of real-time application. We conclude with Section V. The Appendix contains a list of definitions.

II. System Models and Assumptions

A. System Model

We consider a potentially very popular class of computer architectures, which is similar to the continuously reconfigurable multimicroprocessor flight control system (CM^{2}FCS) structure proposed by the United States Air Force [8]. (See Fig. 1 for an architectural block diagram.)

This class of computers shares the following characteristics. Processors work together in triads, and are synchronized. Faults are masked by voting when a failure is located, and the injured triad is either purged of its faulty element and brought up to strength with the introduction of a spare (if a spare exists), or the triad is disbanded and its working processors classified as spares.

The processors each have their own private memory. In this class of architectures, each private memory contains a copy of the operating system and all the applications software. In certain applications, where there are only a small number of programs to be executed, this is possible. This can be justified by the fact that on-line loading of both program code and data requires too much time to meet real-time constraints, and continuous increase in capacity and drop in cost of memory make such an arrangement economically feasible.

Tasks enter the system and must be scheduled. The job of scheduler is taken up by one of the triads.

When a task arrives, it is queued at the scheduler until a processor triad becomes free. It is then allocated, on a first-come-first-served basis, to that triad. When a free triad in the system receives a task, it can begin executing it immediately.

B. Assumptions

We make the following assumptions:

A1. All processors are identical.

A2. Tasks belong to well-defined task classes. There is no restriction on the distribution of the task service time, except that service times are stochastically independent.

A3. When a task is completed, it is voted on and the results are used to control an actuator or update a display.

A4. There is no intertask communication during task execution. That is, tasks communicate with each other at the beginning for input and at the end of execution for output, but not during execution.

A5. Tasks arrive according to a Poisson process.

A6. The computer system operates in *missions.* A mission is a continuous interval of time during which the computer performs its function. We treat the case where no repair is possible during a mission; repairs must be conducted between missions.

All these assumptions are realistic for computers in charge of real-time control. By A4, the influence of the interconnec-

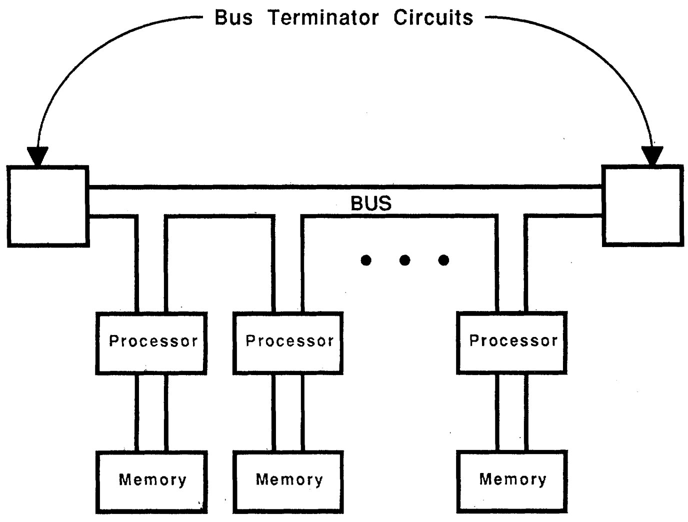


Fig. 1. Schematic of system architecture.

tion network on system reliability can easily be computed using combinatorial arguments and the result amalgamated with a study of the influence of processor reliability, and so we confine ourselves here to studying processor-related performance issues.

In this work, we make none of the assumptions of exponentially distributed service and failure times which, while permitting an easy analysis, would limit the applicability of the work. If either of these parameters is exponentially distributed, then the analysis is greatly simplified, as we shall show.

III. Tradeoff Analysis

In this section, we derive formulas for the sensitivity of computer performance to the hard deadlines, the cost functions, the amount of component burn-in (if any) used, and the processor replacement policy.

It is not difficult to qualitatively describe the nature of these sensitivities. If the hard deadlines are smaller, meaning that there is less time allowed for the computer to complete execution, an increased premium is placed on processor speed, and the balance between failure caused by massive hardware failure (*static failure*) and that due to missing a deadline for other causes (*nonstatic failure*) shifts toward making the latter the dominant component. Similarly, if the finite cost functions are great, the importance of processor speed increases.

Component burn-in is important because in many cases the hazard rate follows a bathtub curve: a high rate at first, then dropping to a quasi-steady-state value, and finally rising again. The first and last segments reflect, respectively, the latent faults present upon processor manufacture and the effects of aging. The sensitivity of mean cost and the probability of dynamic failure on burn-in is determined by the slope of the bathtub curve, and the extent to which they should be applied depends upon this slope as well as on the contribution of burn-in to the life-cycle cost. In cases where the hazard rate is constant, denoting an exponential failure law, the optimal burn-in period is clearly zero.

The processor replacement policy followed is also a function of the failure process. If the processors follow a more-or-less exponential failure law, the optimal policy is to replace a processor only after it has failed. If the failure law is highly nonexponential, it is useful to replace processors after they have reached a certain age. We will denote by P_ξ the policy under which, at the end of a mission, processors older than ξ and those which have failed, are replaced.

Our analysis will proceed as follows. Both the probability of dynamic failure and the mean cost accrued over a mission are calculated as functions of the response time distribution for the various tasks of the system, the distribution of the sojourn time of the system in its various states (i.e., number of functional triads), together with the cost functions and hard deadlines for the given set of task classes. The response time distribution in turn depends on the task interarrival and service time distributions. The latter distributions are given as part of the application description. The sojourn time distribution is calculated as a function of the processor replacement policy, the processor failure law, and the burn-in period.

In Section III-A, we obtain formulas for p_{dyn} and MC as a function of the state transition epoch and task waiting time distributions in any given mission. In Section III-B, an approximation to the waiting time distribution is presented. In Section III-C, we derive an expression for the state transition epoch distribution as a function of the distribution of the age of the processors at the start of any given mission. In Section III-D this age distribution is derived as a function of the burn-in time, t_{burn}, and the processor replacement policy.

A. Calculating p_{dyn} and MC

Both p_{dyn} and MC are calculated with respect to the mission number, say l. All times are relative to the start of the mission.

Let $\pi_{\text{fail}}(t, \eta_{n_1}, \eta_{n_1-1}, \cdots, \eta_1)$ be the probability of dynamic failure by time t if the system leaves state i at time η_i during mission l. Define $\eta_0 \triangleq m$, where m is the mission lifetime. Define a state transition function

$$\psi(t, \eta_{n_1}, \eta_{n_1-1}, \cdots, \eta_1) \triangleq \begin{cases} n_1 & \text{if } t \le \eta_{n_1} \\ i & \text{if } \eta_{i+1} < t \le \eta_i, \\ 0 & \text{if } t > \eta_1 \end{cases}$$

and let $W(\cdot|r)$ be the steady-state waiting time distribution if the system is in a state of r triads. Let $F_{wd}(\cdot)$ be the probability distribution function of the *waiting-time deadline* which is the maximum waiting time permitted if the task is to finish service by its hard deadline. If w_i, x_i, and t_{di} represent the waiting time, service time, and deadline of some task in class i, clearly, $w_i \le t_{di} - x_i$ if that task is not to miss its deadline. From this, we easily derive the following result.

$$F_{wd}(w) = \sum_{i=1}^{r} \Pr\{w \le t_{di} - x_i | \text{task is of class } i\} \cdot \Pr\{\text{task is of class } i\}$$

$$= \sum_{i=1}^{r} \frac{\lambda_i}{\lambda} \int_0^\infty F_{di}(x_i + \omega)\, dB_i(x),$$

where $B_i(\cdot)$ and $F_{di}(\cdot)$ are the service time and hard deadline distributions, respectively, for class i.

The waiting-time deadline distribution is as simple as it is because the arrival process is Poisson. $F_{wd}(\cdot)$ is independent of the current system state because the service time and hard deadline are independent of the current state. Let

$$P_{\text{fail}}(t, \eta_{n_1}, \cdots, \eta_1) = \int_0^\infty \lambda\{1 - W(w|\psi - 1)\}\, dF_{wd}(w),$$

where we suppress the arguments of ψ for notational convenience. $P_{\text{fail}}\, dt$ is the probability in $[t, t + dt)$ of missing a hard deadline if the system leaves state i after η_i time units into the mission. In the event that $W(w|\psi - 1)$ does not exist (which will occur when the system is oversaturated at state $\psi - 1$), set $W(w|\psi - 1) = 0$ for all w. (The "-1" arises because one triad acts only as a scheduler.)

Clearly,

$$\dot{\pi}_{\text{fail}}(t, \eta_{n_1}, \eta_{n_1-1}, \cdots, \eta_1) \approx \{1 - \pi_{\text{fail}}(t, \eta_{n_1}, \cdots, \eta_1)\} P_{\text{fail}}(t, \eta_{n_1}, \cdots, \eta_1).$$

So, the probability of failure during mission l is given (approximately) by

$$\int_{m=0}^\infty \int_{\eta_{n_1}=0}^m \int_{\eta_{n_1-1}=\eta_{n_1}}^m \cdots \int_{\eta_1=\eta_2}^m \cdot\, \pi_{\text{fail}}(m, \eta_{n_1}, \eta_{n_1-1}, \cdots, \eta_1) g_l(\eta_{n_1}, \eta_{n_1-1}, \cdots, \eta_1)\, d\eta_1\, d\eta_2 \cdots d\eta_{n_1}\, dM(m)$$

where $g_l(\eta_{n_1}, \eta_{n_1-1}, \cdots, \eta_1)$ represents the joint density function of time η_i, $i = 1, \cdots, n_1$ and $M(\cdot)$ is the distribution of mission lifetimes. The approximations inherent in this derivation are a) the steady-state values of waiting-time distribution are used, and b) it is assumed that the number of operational system triads (i.e., the system state) does not change between the time a task enters the scheduler queue, and when it achieves service.

In the event that the assumption in b) above is not acceptable, we can obtain an upper bound to the failure probability as follows. Fig. 2 illustrates an example mission profile. The system starts with four triads, and is down to two triads by the time the mission ends. The system is in state 4 until η_4, in state 3 in $[\eta_4, \eta_3)$, and in state 2 beyond that.

Let α_4 denote arrival time of the first task which arrived when the system state was four triads, but which arrived at the server to find the system state changed. Similarly for α_3. (We are assuming that there are such tasks; if there are not, the result will still be an upper bound of the failure probability.) Let $\chi_4(\cdot)$ denote the distribution of $t_4 \triangleq \eta_4 - \alpha_4$. Clearly, t_4 must be less than or equal to the waiting time of the given task, or it will not arrive at the server to find the system state changed. So, $\chi_4(t) \geq W(t|4)$, $\forall t > 0$, i.e., t_4 is *stochastically* less than or equal to a random variable distributed according to $W(\cdot|4)$ [9]. Similarly, t_3 is stochastically less than or equal to a random variable distributed according to $W(\cdot|3)$.

Therefore, if we generalized and transform the mission, replacing η_i by $\hat{\eta}_i = \max\{0, \hat{\eta}_{i+1}, \eta_i - \xi_i\}$, where ξ_i is distributed according to $W(\cdot|i)$, $\pi_{\text{fail}}(m, \hat{\eta}_{n_1}, \hat{\eta}_{n_1-1}, \cdots, \hat{\eta}_1)$ is an *upper bound* to the probability of failure when the system leaves state i at η_i.

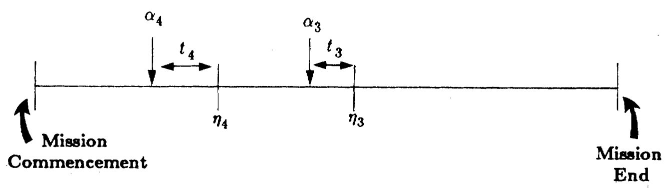


Fig. 2. Example mission profile.

The joint density function of the $\hat{\eta}_1$, and density function, $\hat{g}_l$, of the transformed mission must now be computed.

An examination of the formula defining $\hat{\eta}_i$ shows it depends on $\hat{\eta}_{i+1}$, and η_i. We can, therefore, define function $\hat{h}$ such that

$$\hat{g}_l(\hat{\eta}_{n_1}, \hat{\eta}_{n_1-1}, \cdots, \hat{\eta}_1 | \eta_{n_1}, \eta_{n_1-1}, \cdots, \eta_1) \triangleq \hat{h}_{n_1}(\hat{\eta}_{n_1}|\eta_{n_1}) \cdot \hat{h}_{n_1-1}(\hat{\eta}_{n_1-1}|\hat{\eta}_{n_1}, \eta_{n_1-1}) \cdots \hat{h}_1(\hat{\eta}_1|\hat{\eta}_2, \eta_1).$$

From the definition of $\hat{\eta}_1$, it follows that

$$\hat{h}_{n_1}(\hat{\eta}_{n_1}|\eta_{n_1}) = \begin{cases} \int_{\eta_{n_1}^-}^\infty dW(\xi|n_1) & \text{if } \hat{\eta}_{n_1} = 0 \\ w(\eta_{n_1} - \hat{\eta}_{n_1}|n_1) & \text{if } \hat{\eta}_{n_1} > 0 \end{cases}$$

where $w(\cdot|i) = W'(\cdot|i)$. We assume W is differentiable. If not, minor changes are required to be made: the integral will become a summation, and w will be the probability mass function. The same remark applies to the formulas below. If $\hat{\eta}_1 > 0$, then for $i < n_1$,

$$\hat{h}_i(\hat{\eta}_i|\hat{\eta}_{i+1}, \eta_i) = \begin{cases} 0 & \text{if } \hat{\eta}_i < \hat{\eta}_{i+1} \\ \int_{(\eta_i - \hat{\eta}_{i+1})^-}^\infty w(\xi|i)\, d\xi & \text{if } \hat{\eta}_i = \hat{\eta}_{i+1} \\ w(\eta_i - \hat{\eta}_i|i) & \text{if } \hat{\eta}_i > \hat{\eta}_{i+1} \end{cases}$$

and

$$\hat{h}_i(0|\hat{\eta}_{i+1}, \eta_i) = \begin{cases} 0 & \text{if } \hat{\eta}_{i+1} > 0 \\ \int_{\eta_i^-}^\infty w(\xi|i)\, d\xi & \text{if } \hat{\eta}_{i+1} = 0. \end{cases}$$

We can now uncondition $\hat{g}_l$ on the η_i's and condition on m, the lifetime of the lth mission. Denote this function by $\tilde{g}_l(\cdot|m)$. (Recall that all our calculations are w.r.t. the lth mission.)

$$\tilde{g}_l(\hat{\eta}_{n_1}, \hat{\eta}_{n_1-1}, \cdots, \hat{\eta}_1|m) = \int_{\eta_{n_1}=0}^m \int_{\eta_{n_1-1}=\eta_{n_1}}^m \cdots \int_{\eta_1=\eta_2}^m \cdot\, \hat{g}_l(\hat{\eta}_{n_1}, \hat{\eta}_{n_1-1}, \cdots, \hat{\eta}_1|\eta_{n_1}, \eta_{n_1-1}, \cdots, \eta_1) \cdot g_l(\eta_{n_1}, \eta_{n_1-1}, \cdots, \eta_1) \cdot d\eta_1\, d\eta_2 \cdots d\eta_{n_1},$$

where $g_l(\cdot)$ is derived in Section III-C below. The probability of dynamic failure over mission l is then upper bounded by

$$\int_{m=0}^{\infty} \int_{\hat{\eta}_{n_1}=0}^{m} \int_{\hat{\eta}_{n_1-1}=\hat{\eta}_{n_1}}^{m} \cdots \int_{\hat{\eta}_1=\hat{\eta}_2}^{m} \cdot \pi_{\text{fail}}(m, \hat{\eta}_{n_1}, \hat{\eta}_{n_1-1}, \cdots, \hat{\eta}_1) \cdot g_l(\hat{\eta}_{n_1}, \hat{\eta}_{n_1-1}, \cdots, \hat{\eta}_1 | m) \cdot d\hat{\eta}_1, d\hat{\eta}_2 \cdots d\hat{\eta}_{n_1}\, dM(m),$$

where, as before, $M(\cdot)$ is the distribution function for the length of a mission.

Deriving the mean cost accrued over mission l is much easier. Since the probability of dynamic failure over a mission must be kept small, (typically much less than 10^{-5}), we can simply add up the contribution of each task without accounting for the possibility that the system may have failed prior to its arrival.

The contribution to the mean cost of an arrival in $[\hat{\eta}_j, \hat{\eta}_{j-1}]$ is upper bounded by

$$C(j-1) = \sum_{i=1}^{r} \frac{\lambda_i}{\lambda} \int_{t=0}^{\infty} \Theta_i(t) f_i(t) \{1 - F_{di}(t)\}\, dt$$

where $\Theta_i(t) = \int_{\alpha=0}^{t} w(\alpha) dB_i(t-\alpha)$ and we may recall that $f_i(t)$ is the (finite) cost function for task class i and $B_i(\cdot)$ is the service time distribution.

Therefore, the contribution to the mean cost of an arrival at time t into mission l, $\Psi_l(t) = \sum_{j=1}^{n_1} C(j) u_l(j, t|m)$, where $u_l(j, t|m)$ is the conditional probability that $t \in [\hat{\eta}_j, \hat{\eta}_{j-1})$ if the lifetime of the lth mission is m. u can be computed directly from the sojourn time distribution $\tilde{g}(\cdot|m)$.

The mean cost of the mission is, therefore, upper bounded by

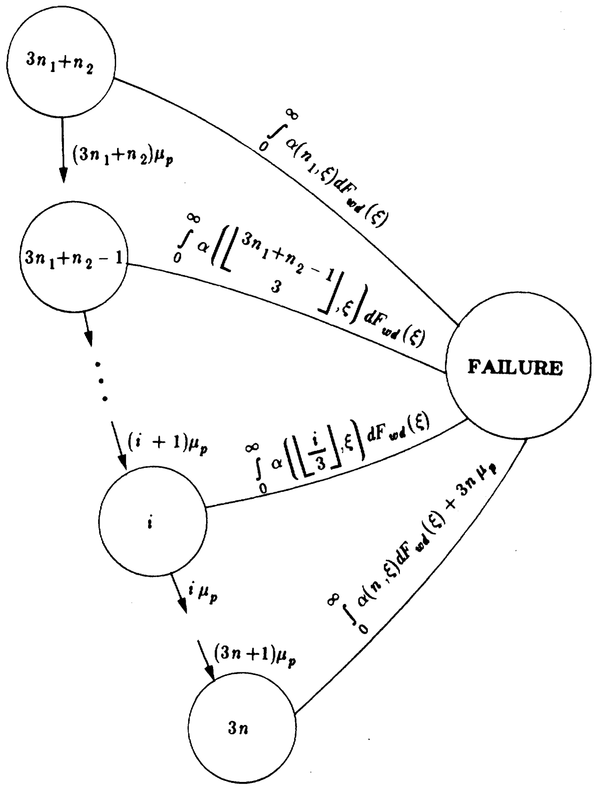
$$\lambda \int_{m=0}^{\infty} \int_{t=0}^{m} \Psi(t)\, dt\, dM(m).$$

The probability of dynamic failure and mean cost over the entire life-cycle of, say, v missions, can now be easily calculated.

Special Case: Exponential Failure Laws: This is a very important special case, and it lends itself particularly nicely to analysis. If we assume that the processors fail with rate μ_p, it is easy to see that an approximate value for p_{dyn} can be obtained using the Markov model in Fig. 3. Let $n - 2$ be the state at which the system is saturated. Then, defining $\alpha(j, t) \triangleq \lambda\{1 - W(t|j-1)\}$, the probability of having i processors functioning at time t by $\pi_i(t)$, the following equations can be written.[4]

$$\dot{\pi}_{3n_1+n_2}(t) = -\left\{(3n_1+n_2)\mu_p + \int_0^{\infty} \alpha(n_1, \xi)\, dF_{wd}(\xi)\right\} \pi_{3n_1+n_2}(t)$$

[4] Recall that one triad is used only as a scheduler.



$3n_1+n_2$
$(3n_1+n_2)\mu_p$
$\int_0^\infty \alpha(n_1,\xi)dF_{wd}(\xi)$
$3n_1+n_2-1$
$\int_0^\infty \alpha\left(\left\lfloor \frac{3n_1+n_2-1}{3} \right\rfloor, \xi\right) dF_{wd}(\xi)$
FAILURE
$(i+1)\mu_p$
$\int_0^\infty \alpha\left(\left\lfloor \frac{i}{3} \right\rfloor, \xi\right) dF_{wd}(\xi)$
i
$i\mu_p$
$\int_0^\infty \alpha(n,\xi)dF_{wd}(\xi) + 3n\mu_p$
$(3n+1)\mu_p$
$3n$

Fig. 3. Markov model for special case.

$$\dot{\pi}_i(t) = -\left\{i\mu_p + \int_0^{\infty} \alpha\left(\left\lfloor \frac{i}{3} \right\rfloor, \xi\right) dF_{wd}(\xi)\right\} \pi_i(t) + (i+1)\mu_p \pi_{i+1}(t), \qquad 3n \le i < 3n_1 + n_2$$

$$\dot{\pi}_{\text{fail}}(t) = \sum_{i=3n}^{3n_1+n_2} \left\{\int_0^{\infty} \alpha\left(\left\lfloor \frac{i}{3} \right\rfloor, \xi\right) dF_{wd}(\xi)\right\} \cdot \pi_i(t) + n\mu_p \pi_n(t),$$

where $\pi_{\text{fail}}(t)$ = probability of having failed by time t, and $\pi_{3n_1+n_2}(0) = 1$, $\pi_i(0) = 0$, $\forall i \ne 3n_1 + n_2$. Then, the probability of dynamic failure over mission number l is given by $p_{\text{dyn}} \approx \int_0^{\infty} \pi_{\text{fail}}(t) dM(t)$.

Implicit in these calculations is the assumption that the hard deadline is much smaller than the mission lifetime or the sojourn time in the various states. This is almost invariably the case in practice.

The mean cost is similarly easy to derive. The contribution to the mean cost of an arrival which finds the system with j working triads is approximately $C(\lfloor j/3 \rfloor)$, and we, therefore, have

$$\text{Mean Cost (MC)} \approx \sum_{i=3n}^{3n_1+n_2} \int_0^{\infty} \int_{t=0}^{m} \lambda \pi_i(t) \cdot C\left(\left\lfloor \frac{i}{3} \right\rfloor\right) dt\, dM(m).$$

B. Steady-State Waiting Time Distribution

If there are n triads, the system can be modeled as an M/G/n queue. The waiting time distribution of such queues is as yet unknown, so approximations must be sought.

We employ the approach of Arjas and Lehtonen [1]. The M/G/n queue is approximated by two E_n/G/1 queues: one yields the upper bound, and the other the lower bound of the response time distribution.

The idea is simple and its proof in [1] is elegant. Divide the arriving tasks, in their order of arrival, into groups of size n. Represent each such group by a single task, whose service time requirement is the maximum of all service times in its group. Replace the M/G/n system with an E_n/G/1 system whose single server is identical to any of the n identical servers in the M/G/n queue. The response time distribution of the E_n/G/1 system is the distribution of an upper bound to the response time of the M/G/n system.

The lower bound is similarly obtained: instead of setting the service time to the group maximum, set it to the group minimum.

We now introduce some additional notation. Let C_l denote the lth task to arrive, t_{l+1} the time between arrivals of C_l and C_{l+1}, x_m the service time of C_m, and $u_m = x_m - t_{m+1}$. Denote the probability distribution of u_m by L_m.

Because of the assumption of Poisson arrivals, the waiting time distribution is independent of the class that C_l belongs to, and the service time distribution of C_l is independent of the arrival process. C_l belongs to class i with probability λ_i/λ, where $\lambda = \Sigma_{j=1}^{r} \lambda_j$. Define $B(\cdot) \triangleq \Sigma_{i=1}^{r} (\lambda_i/\lambda)B_i(\cdot)$. Let b_1, $b_2, \cdots, b_n$ be a set of independently identically distributed (IID) random variables with distribution $B(\cdot)$. Let $B^{(u)}(\cdot)$ and $B^{(l)}(\cdot)$ denote the distribution of $b^{(u)} = \max\{b_1, \cdots, b_n\}$ and $b^{(l)} = \min\{b_1, \cdots, b_n\}$, respectively.

Let $A(t) = \lambda e^{-\lambda t}((\lambda t)^{n-1}/(n-1)!)$ which represents the distribution of $\Sigma_{i=1}^{n} t_i$. Then, by the result of Arjas and Lehtonen [1], the waiting time distribution of tasks in the original M/G/n queue which represents the computer, is upper-bounded by an E_n/G/1 queue with arrival and service distributions $A(\cdot)$ and $B^{(u)}(\cdot)$, respectively, and lower bounded by another E_n/G/1 queue with arrival and service distributions $A(\cdot)$ and $B^{(l)}(\cdot)$, respectively.

Recalling that $L_m(\cdot)$ is the distribution of u_m, Lindley's equation [4] immediately yields

$$W_{m+1}(y) = \int_0^\infty L_m(y-w)\, dW_m(w),$$

where $W_1(w) = 1 \ \forall\ w \geq 0$. If $E[u_m] < 1 \ \forall\ m$, there exists a limit such that $\lim_{m\to\infty} W_m(y) = W(y)$.

Implementing this iteration directly on a computer is, in general, fraught with formidable difficulties. If the Laplace transform of L_m is rational, spectral factorization followed by a Laplace inversion [4] can be used to obtain an exact solution. If it is not rational, then we must seek approximate methods.

One such method is due to Kingman [5]. Let $L^*(s)$ denote the Laplace transform of L. Then, defining $s_0 = \max\{s > 0 | L^*(-s) \leq 1\}$, we have the following upper bound to the waiting time distribution:

$$W(y|n) = 1 - e^{-s_0 y}.$$

Since the service time distribution of each task is known, it is now a simple matter to calculate the response time distribution by using the service time with the waiting time distributions.

C. Probability Density Function of η_i

In this subsection, we compute the joint density of the η_i under the following assumptions.

1) Processor failures occur independently, and a processor *ages* only during a mission, not between missions.

2) Mission lifetimes are IID.

3) Processors are replaced upon failure at the end of the current mission.

It is worth considering assumption 1 for a moment. By assuming that a processor ages only during a mission, and that between missions (when, presumably, the system is powered down) processors do not fail, we have made it possible to handle the missions as if they were back-to-back, and that from the point of view of system failure, it does not matter how long the actual idle period between missions is. This assumption is valid in most instances since an unpowered unit tends to age very slowly.

When we say that a processor's age is τ, we mean, therefore, that the processor has seen τ seconds of service. Let $\eta_i^{(l)}$ be the epoch in the lth mission that the system leaves state i (note that all these times are relative to the time of starting the mission). For convenience, we shall suppress the superscript. Clearly, we can write

$$g_l(\eta_{n_1}, \eta_{n_1-1}, \cdots, \eta_1) = h_{n_1}^{(l)}(\eta_{n_1}) \cdot h_{n_1-1}^{(l)}(\eta_{n_1-1}|\eta_{n_1}) \cdots h_1^{(l)}(\eta_1|\eta_2),$$

where $h_a^{(l)}(\eta_a|\eta_{a+1})\, d\eta$ is the probability that the system leaves state a in $[\eta_a, \eta_a + d\eta)$, given that it entered state a at η_{a+1} (in mission l). For $a < n_1$ we get $h_a^{(l)}(\eta_a|\eta_{a+1})\, d\eta$ = Pr {sojourn time in state a lies in the interval $[\eta_a - \eta_{a+1}, \eta_a - n_{a+1} + d\eta_a)|Z\}$

$$\triangleq \Pr\{X \text{ and } Y|Z\}$$

$$= \binom{3a+2}{2}\{\delta_l(\eta_{a+1}, \eta_a - \eta_{a+1})\}^2\{1-\delta_l(\eta_{a+1}, \eta_a - \eta_{a+1})\}^{3a} \cdot \binom{3a}{1}\frac{d\beta_l(\eta_a)}{1-\beta_l(\eta_a)}$$

and

$$h_{n_1}^{(l)}(\eta_{n_1}) = \binom{3n_1+n_2}{2}\{\delta_l(0, \eta_1)\}^2 \cdot \{1-\delta_l(0, \eta_1)\}^{3n_1}\binom{3n_1}{1}\frac{d\beta_l(\eta_{n_1})}{1-\beta_l(\eta_{n_1})},$$

where

X = two out of the $3a + 2$ processors fail in less than $\eta_a - \eta_{a+1}$ seconds after state a was entered,

Y = one further failure occurs in $[\eta_a - \eta_{a+1}, \eta_a - \eta_{a+1} + d\eta)$ after state a was entered,

Z = state a was entered at η_{a+1},

$$\beta_l(t)=\int_0^\infty P(\chi, t)\ \text{age}_{l,\xi}(\chi)\ d\xi$$

$$\delta_l(\eta, t)=\frac{\beta_l(\eta+t)-\beta_l(\eta)}{1-\beta_l(\eta)}.$$

$$P(\xi_1, \xi_2)=\begin{cases}\dfrac{p(\xi_1+t_{\text{burn}}, \xi_2)}{1-p(0, \xi_1+t_{\text{burn}})} & \text{if } \xi_1+t_{\text{burn}}\le\xi_2\\ 0 & \text{otherwise,}\end{cases}$$

where $p(t, \tau)$ is the probability of a processor failing after operating (or burning-in) for $\zeta \in [t, t+\tau)$. The function $g_l(\eta_{n_1}, \eta_{n_1-1}, \cdots, \eta_1)$ is only valid for $\eta_i \le T_l$, the lifetime of mission l.

Special Case: Processor Failure Time Exponentially Distributed: Let the failure rate be μ_p, then $P(t, \tau) = 1 - e^{-\mu_p\tau}$. The equations for g_l and $h_i^{(l)}$ will still remain valid, with the simplification that

$$\delta_l(\eta, t)=\beta_l(t)=1-e^{-\mu_p t}.$$

D. Calculating the Age Density Function

Let $\text{age}_{i,\xi}(l|t_1, \cdots, t_{i-1})$ be the density function of the processor age under policy P_ξ at the beginning of mission i, and conditioned on mission j ending at t_j, $j = 1, \cdots, i-1$. Then $\text{age}_{i,\xi}$ can be derived recursively as follows.

$$\begin{aligned}&\text{age}_{i+1,\xi}(0|t_1, \cdots, t_i)\\ &=\Pr\{\text{age of processor at } t_i>\xi, \text{ or processor has failed in } [t_{i-1}, t_i]\}\\ &=\int_0^\xi p(l, T_i)\ \text{age}_{i,\xi}(l|t_1, \cdots, t_{i-1})\ dl\\ &\quad+\int_\xi^\infty \{1-p(l-T_i, T_i)\}\\ &\quad\cdot \text{age}_{i,\xi}(l-T_i|t_1, \cdots, t_{i-1})\ dl,\end{aligned}$$

where $T_i = t_i - t_{i-1}$. The first term is the probability that the processor fails during mission i, and the second term is the probability that it does not fail during mission i, but has seen more than ξ seconds of service and must, therefore, be replaced.

If $l > 0$, we get

$$\text{age}_{i+1,\xi}(l|t_1, \cdots, t_i)=\begin{cases} l-p(l-T_i, T_i)\ \text{age}_{i,\xi}(l-T_i|t_1, \cdots, t_{i-1}) & \text{if } T_i\le l<\xi\\ 0 & \text{otherwise.}\end{cases}$$

The initial condition for the recursion is trivially obtained. Take $\text{age}_{0,\xi}(0) = 1$, $\text{age}_{0,\xi}(t) = 0\ \forall\ t > 0$. We create a dummy mission number zero which ends at time zero with all processors new.

To find the unconditional density function of the age, we need the following integration.

$$\begin{aligned}a_{i,\xi}(l)=&\int_{x_1=0}^\infty\int_{x_2=x_1}^\infty\cdots\int_{x_{i-1}=x_{i-2}}^\infty\\ &\cdot \text{age}_{i,\xi}(l|x_1, x_1+x_2, \cdots, x_1+x_2+\cdots+x_{i-1})\ dM(x_{i-1})\\ &\cdots dM(x_1).\end{aligned}$$

IV. Numerical Example

In this section, we present examples of the dependencies and tradeoffs among the various parameters under the control of the designer, and consider the effect of the application requirements on the performance of the computer system. To ensure clarity, we consider the individual tradeoffs and dependencies separately.

In [11] and [12], we have considered the impact of the application requirements on the system performance in detail. The influence of the application is felt through the task loading, the cost functions, and the hard deadline distributions. To illustrate this, we have treated in [11] and [12] an example sufficiently idealized to remove any extraneous factors. This is the number-power tradeoff, first considered—in a different context—by King and Mitrani [3]. Assuming that the product of the number of triads and the power[5] of the individual triads (which is termed the *number-power product*) is constrained to be a constant, and considering the effects of varying the processor number, one can gain useful insights into the impact of the application requirements on the optimal structure. Especially, one can analyze the tradeoffs between the processor speed and processor redundancy in distributed real-time systems.

Since processors can only be replaced at the end of missions, the mission length greatly affects the probability of dynamic failure and the mean cost. We illustrate this in Section IV-A by an example with two task classes, each of deterministic service time requirements.

When processor failure laws are not memoryless, burn-in and replacement policy are important. We give an example of the influence of both of these on the probability of dynamic failure, the mean cost, and the expected number of processors to be replaced over the system life, in Section IV-B.

A. Influence of Mission Lifetime

The overall useful life of the system in this example is 960 h. We study the effects of breaking this period up into n missions, each of length $960/n$, for $n = 2, \cdots, 16$. The system parameters are: $n_1 = 11$, $n_2 = 0$, task arrival rate is 300, and there are two tasks, 1 and 2, with $\lambda_1 = \lambda_2 = 150$, and the service time being 0.001 and 0.002, respectively. The hard deadlines are 0.099 and 0.098, respectively. For the purposes of our example, the finite cost function is taken as equal to the waiting time. The processor hazard rate is shown in Fig. 4.

As might be expected, the probability of dynamic failure drops as the number of missions increases, and the missions become correspondingly shorter. This effectively measures the sensitivity of the system to the interrepair periods. One has to pay for this in terms of a greater number of processors

[5] Power is the instruction processing rate.

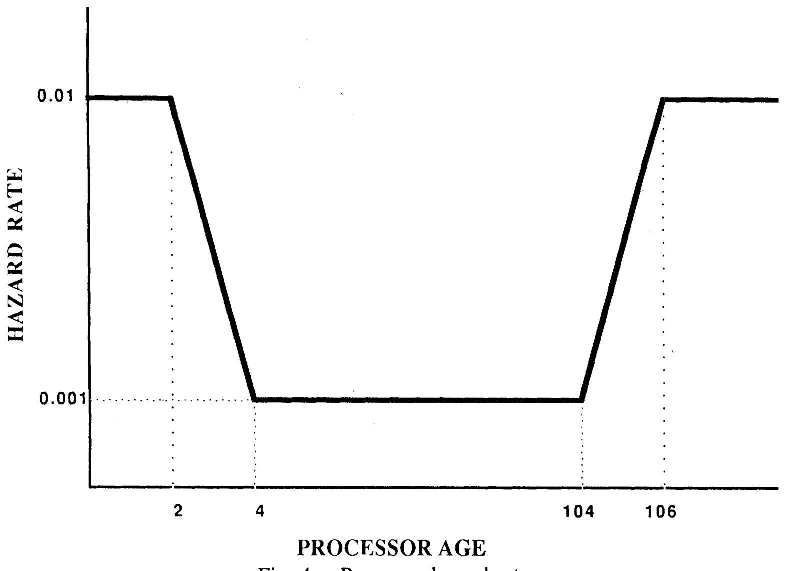


Fig. 4. Processor hazard rate.

TABLE I
INFLUENCE OF MISSION LIFETIME

No. of Missions	Mission Length	P_{dyn}	M.C.	Mean No. of Processors Replaced
2	480.0	9.99×10^{-1}	25.39	32.33
3	320.0	9.94×10^{-1}	49.38	59.49
4	240.0	7.86×10^{-1}	53.81	78.54
5	192.0	2.81×10^{-1}	40.60	90.73
6	160.0	5.98×10^{-2}	31.31	98.31
7	137.143	1.39×10^{-2}	27.02	103.13
8	120.0	4.94×10^{-3}	25.06	106.28
9	106.67	2.71×10^{-3}	24.05	108.49
10	96.0	6.39×10^{-4}	22.30	112.10
11	87.27	3.68×10^{-5}	20.36	117.27
12	80.0	1.67×10^{-6}	19.16	121.01
13	73.85	1.52×10^{-7}	18.38	124.14
14	68.5	5.82×10^{-8}	17.86	126.49
15	64.0	4.16×10^{-8}	17.51	128.22
16	60.0	3.48×10^{-8}	17.28	129.77

replaced. This is captured in the third and fourth columns of Table I, which are the ratio of the marginal decrease in the probability of dynamic failure and the mean cost, respectively, to the marginal increase in the mean number of processors replaced.

Notice that the mean cost for the system with two missions is very low. The reason is that the mean cost is computed on condition that the computer does not fail, and when the failure probability becomes sufficiently close to one, this causes an actual decrease in the mean cost.

B. Influence of Burn-In Time

The task mix used here is the same as in the previous one. $\lambda_1 = \lambda_2 = 150$, the length of each mission is 170 h, the deadlines and service times are as before, the replacement strategy is P_0, and the processor hazard rate is given by

$$\mu_p(t) = \begin{cases} 0.02e^{-0.02995t} & \text{if } 0 \le t < 100 \\ 0.001 & \text{if } 100 \le t \le 2100 \\ 0.02e^{0.02995(t-2200)} & \text{if } 2100 < t < 2200 \\ 0.02 & \text{if } t \ge 2200. \end{cases}$$

We consider the failure probabilities over a period of 15 missions, each of length 170 h.

Table II contains the numerical results of the dependence on t_{burn}. As the burn-in period rises, p_{dyn} initially drops, as does the mean cost. Above a burn-in period of 35, however, an increase in t_{burn} causes an increase in p_{dyn}, i.e., deterioration of

TABLE II
INFLUENCE OF BURN-IN TIME

Burn-In Time	p_{dyn}	M.C.	Mean No. of Processors Replaced	A	B
0	1.93×10^{-8}	50.38	127.78	-	-
5	3.38×10^{-9}	47.93	118.42	0.32×10^{-8}	1.87
10	1.03×10^{-9}	46.17	110.84	0.47×10^{-9}	1.51
15	3.75×10^{-10}	44.87	104.52	0.13×10^{-9}	1.26
20	1.56×10^{-10}	43.91	99.29	0.44×10^{-10}	1.05
26	7.36×10^{-11}	43.05	94.21	0.14×10^{-10}	1.02
30	6.23×10^{-11}	42.61	91.40	0.28×10^{-11}	0.56
35	7.75×10^{-11}	42.17	88.44	-0.30×10^{-11}	0.59
40	1.28×10^{-10}	41.83	85.99	-0.10×10^{-10}	0.49
45	2.37×10^{-10}	41.57	83.96	-0.22×10^{-10}	0.41
50	4.51×10^{-10}	41.38	82.28	-0.43×10^{-10}	0.34
55	8.65×10^{-9}	41.23	80.93	-0.16×10^{-8}	0.27

$$A = \frac{p_{dyn} \text{ of previous row} - p_{dyn} \text{ of present row}}{t_{burn} \text{ of present row} - t_{burn} \text{ of previous row}}$$

$$B = \frac{MC \text{ of previous row} - MC \text{ of present row}}{t_{burn} \text{ of present row} - t_{burn} \text{ of previous row}}$$

reliability. This is because the burn-in time is large enough to bring the aging-caused rise of the p_{dyn} within the useful life of the system. The rightmost two columns of Table II give the ratio of the improvement in p_{dyn} and the mean cost, respectively, due to burn-in, and the corresponding value of the t_{burn} used. Such ratios, which indicate a return for the burn-in, can be used in the system optimization mentioned in Section II.

C. Influence of Processor Replacement Strategy

We assume $t_{burn} = 4$, the processor hazard rate as in Fig. 4, the task mix as in Section IV-B, and 16 missions of length 60 h each. Our numerical results are contained in Table III. Rather than indicating replacement policy by processor age, we have chosen the alternative of indicating it by the number of missions undergone. Since the mission lifetimes in this example are deterministic, this is just another way of specifying processor age. As before, we provide sensitivity ratios for both the p_{dyn} and the mean cost.

V. Conclusion

It has been argued [7] that computer performance is only meaningful in the context of its application. When computers are designed for specific applications, the needs of the application can be formally embedded within the computer performance analysis. This results, as we have shown in this paper, in precise and quantitative tradeoffs, indicating how changes in computer parameters affect the capability of the computer to satisfy the demands of the application.

Many extensions of this work are possible: the bottleneck is likely to be the queueing analysis. Similar analyses can be carried out for other architectures. One obvious candidate is the FTMP-type structure [2]. Analyzing this would require an approximation to the waiting time distribution in G/G/n queues with blocking, and this is a difficult problem.

Appendix

Definitions

A. Terminology Used for Real-Time Control Systems

Given below are the definitions of frequently used terms.

- *Mission:* A continuous interval of time during which the computer performs its function. We treat the case where no repair is possible during the mission: it is only after a mission has been completed that the system can be repaired and processors replaced.
- *Task Trigger:* The initiation of a task. A task trigger can be produced by a timer, a prespecified combination of controlled system states, by the operator, or any combination of the above. Triggers produced by timers are *open-loop* triggers, those produced by the controlled system state combinations are *close-loop.*
- *Critical Task:* The system response time for any version of a *critical task* must be less than a preset finite deadline if catastrophic failure is to be averted.
- *Hard Deadline:* The hard deadline is a maximum computer think time allowed to keep the controlled system within a "safe" region (see [10] for more on this). The hard deadlines of critical tasks of class i are denoted by t_{di}. Hard deadlines are generally random variables, characterized by a distribution function $F_{di}(\cdot)$.
- *Static Failure:* When so massive a set of *permanent*

TABLE III
INFLUENCE OF PROCESSOR REPLACEMENT STRATEGY

Max. Age (in Missions)	P_{dyn}	M.C.	Mean No. of Processors Replaced	A	B
1*	0.00	14.66	528.00	-	-
2*	0.00	14.66	267.13	0	0
3	1.29×10^{-12}	15.16	176.68	1.29×10^{-12}	0.50
4	3.24×10^{-9}	16.46	142.85	3.24×10^{-9}	1.30
5	3.24×10^{-9}	16.98	127.02	0	0.52
6	3.24×10^{-9}	17.21	121.01	0	0.23
7	3.24×10^{-9}	17.33	117.82	0	0.12
8	3.25×10^{-9}	17.39	116.37	1×10^{-11}	0.06
9	3.25×10^{-9}	17.42	115.53	0	0.03
10	3.25×10^{-9}	17.43	115.13	0	0.01
11	3.25×10^{-9}	17.44	114.96	0	0.01
12	3.25×10^{-9}	17.45	114.87	0	0.01
13	3.25×10^{-9}	17.45	114.81	0	0.00
14	3.25×10^{-9}	17.45	114.78	0	0.00
15	3.25×10^{-9}	17.45	114.78	0	0.00
16	3.25×10^{-9}	17.45	114.78	0	0.00
17	3.25×10^{-9}	17.45	114.77	0	0.00

* too low to calculate under the precision used.

A = p_{dyn} of present row - p_{dyn} of previous row

B = MC of present row - MC of previous row

hardware failures have occurred that it is impossible for the computer to perform its duties, *static failure* is said to result. The onset of static failure is typified by a utilization demand of greater than unity.

- *Dynamic Failure:* When the deadlines for one or more critical tasks have been violated, *dynamic failure* is said to have occurred. Note that dynamic failure *subsumes* static failure. The *probability of dynamic failure* is denoted by p_{dyn}.
- *Burn-in Time:* The time for which processors are "burnt-in" by being made to execute in a test setting. Burn-in tends to remove latent manufacturing defects.
- *Processor Replacement Strategy:* The processor replacement strategy P_ξ is the policy according to which a processor is replaced at the end of a mission during which it has failed or has seen more than ξ hours of service, whichever is the lesser.

B. List of Symbols Used

$\text{age}_{l,\xi}(\cdot)$	Density function of the age of a processor at the beginning of mission l when replacement policy P_ξ is used.
$B_i(\cdot)$	Service time distribution for tasks of class i.
f_i	Finite cost function for tasks of class i.
F_{di}	Distribution of hard deadlines for tasks of class i.
$F_{wd}(\cdot)$	Distribution of waiting time deadline.
η_i	Epoch when the system leaves state i (i.e., degrades to $i - 1$ working triads) during mission l. Actually, it has to be η_i^l to indicate its dependence on the mission l, but the superscript l is dropped for convenience.
$g_l(\eta_{n_1}, \cdots, \eta_1)$	Joint density function of time η_i, which is when the system leaves state n_i, $i = 1, \cdots, n_1$, for mission l.
$M(\cdot)$	Distribution function for the length of a mission.
$M^{(i)}$	i-fold convolution of M.
n_1	Number of triads in the system at the beginning of a mission.
n_2	Number of spares in the system at the beginning of a mission.
$p(t, \tau)$	Probability of a processor failing after operating (or burning-in) for $\zeta \in [t, t + \tau)$.
q_i	Probability of a processor having to be replaced at the end of the ith mission.
r	Number of task classes.
T_i	Random variable denoting the length of the ith mission.
t_i	$\Sigma_{j=1}^{i} T_j$.
$W(\cdot \mid j)$	Distribution function of the steady-state waiting

time of a task when the system has j triads operational.

λ_i Input intensity for tasks of class i, and $\lambda \triangleq \Sigma_{i=1}^{r} \lambda_i$.

$\pi_{\text{fail}}(t, \eta_{n_1}, \cdots, \eta_1)$ Probability of the system failing t units into the current mission when it leaves state i at η_i.

References

[1] E. Arjas and T. Lehtonen, "Approximating many server queues by means of single server queues," *Math. Oper. Res.*, vol. 3, p. 205, 1978.

[2] A. L. Hopkins *et al.*, "FTMP—A highly reliable fault-tolerant multiprocessor for aircraft," *Proc. IEEE*, vol. 66, pp. 1221-1239, Oct. 1978.

[3] P. J. B. King and I. Mitrani, "The effect of breakdown on the performance of multiprocessor systems," in *Proc. Performance '81*. Amsterdam, The Netherlands: North-Holland, 1981, pp. 201-211.

[4] L. Kleinrock, *Queueing Systems: Vol. I*. New York: Wiley, 1975.

[5] ——, *Queueing Systems: Vol. II*. New York: Wiley, 1976.

[6] C. M. Krishna and K. G. Shin, "Performance measures for multiprocessor controllers," in *Performance '83*. Amsterdam, The Netherlands: North-Holland, 1983, pp. 229-250.

[7] C. M. Krishna, K. G. Shin, and Y.-H. Lee, "Optimization criteria for checkpoint placement," *Commun. ACM*, vol. 27, pp. 1008-1012, Oct. 1984.

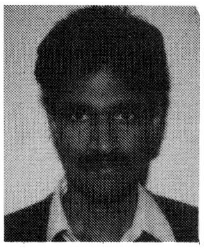
[8] S. L. Maher and S. J. Larimer, "Continuous reconfiguration in a multimicroprocessor flight control system," in *Proc. NATO AGARD Conf. Tactical Airborne Distributed Comput. Networks*, Roros, Norway, 1981.

[9] S. M. Ross, *Stochastic Processes*. New York: Wiley, 1983.

[10] K. G. Shin, C. M. Krishna, and Y.-H. Lee, "A unified method for evaluating real-time computer controllers and its application," *IEEE Trans. Automat. Contr.*, vol. AC-30, pp. 357-366, Apr. 1985.

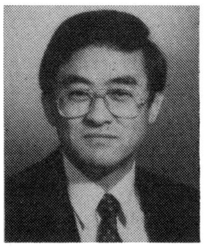
[11] K. G. Shin and C. M. Krishna, "The processor number-power tradeoff in a class of multiprocessors," in *Proc. 5th Int. Conf. Distributed Comput. Syst.*, Denver, CO, May 1985, pp. 321-328.

[12] ——, "New performance measures for design and evaluation of real-time multiprocessors," *Comput. Syst. Sci. Eng.*, vol. 1, pp. 179-191, Oct. 1986.



C. M. Krishna (S'78-M'84) received the B.Tech. degree from the Indian Institute of Technology, Delhi, in 1979, the M.S. degree from Rensselaer Polytechnic Institute, Troy, NY, in 1980, and the Ph.D. degree from the University of Michigan, Ann Arbor, in 1984, all in electrical engineering.

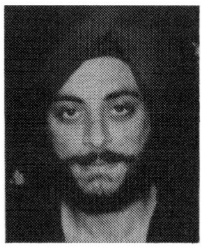
Since September 1984, he has been on the faculty of the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst. His research interests include reliability modeling, queueing and scheduling theory, and distributed architectures and operating systems.



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.

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. Since September 1982, he has been with the Department of Electrical Engineering and Computer Science at The University of Michigan, Ann Arbor, MI, where he is currently a Professor. He has been very active and authored/coauthored over 100 technical papers in the areas of distributed fault-tolerant real-time computing, computer architecture, and robotics and automation. As an initial phase of validation of architectures and analytic results, he and his students are currently building a 19-node hexagonal mesh real-time system at the Real-Time Computing Laboratory (RTCL), The University of Michigan.

Dr. Shin is a member of the Association for Computing Machinery, Sigma Xi, and Phi Kappa Phi. He was the Program Chairman of the 1986 IEEE Real-Time Systems Symposium and has served as the Guest Editor of the special issue of IEEE TRANSACTIONS ON COMPUTERS on Real-Time Systems, August 1987.



Inderpal S. Bhandari received the M.S. degree in electrical and computer engineering from the University of Massachusetts, Amherst, in 1985 and the B.Tech. degree in electrical and electronics engineering from the Birla Institute of Technology and Science, Pilani, India.

He is currently a doctoral candidate in electrical and computer engineering, Carnegie-Mellon University, Pittsburgh, PA. His research areas are artificial intelligence, computer-aided design, and distributed systems.