PERFORMANCE MEASURES FOR MULTIPROCESSOR CONTROLLERS

C. M. Krishna
K. G. Shin

Department of Electrical and Computer Engineering
The University of Michigan
Ann Arbor, MI 48109

This work was supported by NASA Grant No. NAG 1-296. All correspondence should be addressed to Professor K. G. Shin.

PERFORMANCE MEASURES FOR MULTIPROCESSOR CONTROLLERS

ABSTRACT

Performance measures used in contemporary analyses suffer from a number of shortcomings when real-time multiprocessors are considered. Most of these measures do not take account of the needs of the application, except perhaps in a subjective manner.

In this paper, we consider some new performance measures to characterize fault-tolerant multiprocessors used in the control of critical processes. Our performance indices are based on controller response time. By relating this to the needs of the application, we have been able to derive indices that faithfully reflect the performance of the multiprocessor *in the context of the application*, that permit the objective comparison of rival computer systems, and that can either be definitively estimated or objectively measured.

An example of a controller in an idealized satellite application is provided.

Table of Notations

Name	Definition
A	Set of states in which failure is not certain (i.e., in which expected response time is finite).
B	Set of states in which failure is certain
$C_i(\xi)$	Cost accrued in executing a version of task i when multiprocessor response time is ξ.
$\tilde{C}_{i,a}$	Contribution to $\tilde{\Gamma}_i$ of a single version of task i with the multiprocessor in state $a \in A$.
c	Number of processors in multiprocessor (in Example)
$EXT(\Xi_{i,j}, \tau_{i,j}, n_{i,j}, t)$	Extant time of $\Xi_{i,j}$
g_i	Finite cost function of task i.
$H_a^T(t)$	Distribution function of sojourn time of the multiprocessor in state a when the mission lifetime is T (in Example.)
K	Cost Index
K_i	Cost Index for task i
$\tilde{K}$	Modified Cost Index
$\tilde{K}_i$	Modified Cost Index for task i
L	Distribution function for mission lifetime
M	Mean Cost
M_i	Mean Cost for task i
$\tilde{M}$	Mean Modified Cost

$\tilde{M}_i$	Mean Modified Cost for task i
n	Maximum number of working processors for which failure is certain (in Example)
$n_{i,j}$	State of the controller when version j of task is triggered.
$P_a(t)$	Probability of controller being in state a at time t
$\hat{P}_a^{(\xi)}(t)$	Probability of controller being in state $a \in A$ at time t, given that in a mission lifetime of ξ, it never enters a state in B.
p_{dyn}	Probability of dynamic failure
q	Number of buses in multiprocessor (in Example)
$q_i(t)$	Number of triggers of task i in the interval [0.t]
Q	Set of all states of multiprocessor
r	Total number of different tasks in controller software
RESP(i_j)	Response time associated with $\Xi_{i,j}$ (assuming execution is completed).
syst	$\sum_{i=1}^{r} tot(i)$
t_{di}	Hard deadline associated with task i, if critical
tot(i)	Value of $\tilde{M}_i$ over a mission lifetime
V	Variance Cost
V_i	Variance Cost associated with task i
$\tilde{V}$	Variance Modified Cost

$\tilde{V}_i$	Variance Modified Cost associated with task i
$w_{i,x}$	Response time probability density function when the multiprocessor is in state x (x>0) (in Example)
Γ_i	Cumulative cost function associated with task i
$\tilde{\Gamma}_i(t)$	$\sum_{j=1}^{q_i(t)} g_i(RESP(i_j))$
λ	$\sum_{i=1}^{r} \lambda_i$
λ_i	Poisson trigger rate for task i (in Example)
μ	Exponential task execution rate (in Example)
μ_D	Exponential despatcher failure rate (in Example)
μ_B	Exponential bus failure rate (in Example)
μ_P	Exponential processor failure rate (in Example)
$\tau_{i,j}$	Time at which $\Xi_{i,j}$ was triggered
$\Xi_{i,j}$	Version j of task i

1. Introduction

1.1. Motivation

Over the last few years, inexpensive, powerful, and reliable microprocessors have become available. At the same time, analytical, simulation and modelling techniques for use in computer communication networks have been developed. Multiprocessors are therefore becoming attractive. One special application to which they can contribute a great deal is reliable control.

Several reliable multiprocessors have been proposed and a smaller number built. Among the latter, one might mention the Fault Tolerant Multiprocessor (FTMP) [1] and the Software Implemented Fault Tolerance (SIFT) machine [2], both built under contract to NASA for the control of civilian aircraft of the next decade. Reliability requirements are stringent: the benchmark figure employed by NASA is that the probability of controller failure should not exceed 10^{-9} for a ten hour flight. Naturally, such a benchmark begs the question of how "failure" or the "performance" of a complex multiprocessor system is to be defined. In this paper, we present some measures that are appropriate in this context and by extension, in the context of other control applications -- nuclear reactors, life-support systems, etc. -- where controller failure can lead to catastrophic consequences.

1.2. The Nature of the Control Function

A computer controller consists of three communicating parts. These are the *data acquisition*, *data processing* and *output* sections. The data acquisition section consists of sensors, input panels and other associated equipment; the processing section consists of the computer (in our case the multiprocessor) and the output section consists of mechanical actuators, displays and other output devices. The system may logically be regarded as a three-stage pipe[1].

1. Indeed, the M^{2}FCS system [18] of the United States Air Force is physically configured in this manner.

The set of tasks to be executed by a control system is predetermined and the nature and behavior of the software known in advance -- at least in outline -- to the designer. The multiprocessor controller is therefore a rather specialized device; it might indeed be considered custom-built for a particular application. This fact makes it both easier and more necessary to obtain a reasonably good performance analysis of the system.

The controller software in the processing section consists of a set of *tasks*, each of which corresponds to some job to be performed in response to particular sets of *environmental stimuli*.

1.3. The Need For New Performance Indices

The determining characteristic of a computer controller's performance is a combination of reliability and high throughput. The throughput requirements arise from the need for quick system response to environmental stimuli. Speed is of the essence in a real-time controller since failure can occur not only through hardware failure in the system, but also on account of the system not responding fast enough to events in the environment. This fact imposes a premium on controller response time and leads naturally to the work presented in this paper.

As a result of these special performance requirements, performance measures used to characterize general-purpose uniprocessor systems are no longer appropriate for multiprocessor controllers. Conventional reliability, throughput and availability by themselves alone have little meaning in the context of control; a suitable combination of these is necessary. New performance measures are required; measures that are congruent to the application, permit the expression of specifications that reflect without contortion the true system characteristics and application requirements, in addition to allowing an objective comparison of rival systems for particular applications.

We cannot stress too heavily that it is meaningless to speak of the performance of a computer out of the context of its application. The form the performance measures take must reflect the needs of the application, and the computer system must be modelled within this context. The multiprocessor controller and the controlled process form a synergestic pair, and any effort to study the one must take account of the needs of the other.

It is important that performance measures should depend on parameters that can be definitively estimated or objectively measured. Much of the work published so far on characterizing the "goodness of fit" between the attributes offered by a computer system (reliability, throughput, etc.) and those required by the application depends on parameters obtained through essentially subjective analyses. (See, for example, [3]). It has been our policy in this paper, however, to always base performance indices on experimentally-measureable quantities, i.e., system response time for the various system tasks.

1.4. Organization of Paper

In Section 2, we discuss our new performance measures and in Section 3 means for practically obtaining these quantities. An example is presented in Section 4, and the paper concludes with Section 5.

2. Performance Measures

2.1. Survey

The concept of systems that are, through built-in redundancy, much more reliable than any of their components is not new. Some pioneering work was done by von Neumann [4] and by Moore and Shannon [5], both in 1956.

Work in modelling fault-tolerant multiprocessors was done by Bouricius *et. al*, [6], who took into account the impact of transient and permanent failures,

Borgerson and Frietas, [7], who studied the PRIME computer as a gracefully degrading system, and other researchers who, like their precursors, used Markov models in the representation of these systems [8]-[11].

With only one exception ([8]), the work referred to above assumed traditional performance measures. Other workers recognized the shortcomings of these and defined new ones. Among these, one might mention Beaudry [12] and Meyer [13].

The chief drawback of Beaudry's measures is that they express the capability of the system as a whole, not with respect to particular tasks. The computer is treated as a monolith in terms of the services it delivers. The implicit presupposition is that all tasks are qualitiatively of the same form and have practically the same "cost" as a function of the response time[2]. However, the tasks that a real-time computer controller executes are typically widely varying both in terms of the demands they make on various entities of the multiprocessor and in the fact that they are of unequal importance. The performance measures of Beaudry thus express the performance of the computer without reference to the particular needs of the application.

Meyer's concept of *performability* represents a useful advance in the search for appropriate performance measures. However, the actual performance measures used sometimes require further refinement. For example, in [13], Meyer employs as the performance measure, Y_s, the fraction of arrived tasks that are processed by a system S in an interval of time [0,t]. This, as in the case of Beaudry's performance measures allows for no discrimination between individual controller tasks.

To remove, at least partially, the limitations of the work cited above, we present here some new performance measures.

2. Or equivalently, that it does not matter if they do not.

2.2. Definitions and Basic Concepts

Because of the stochastic nature of the transient and/or permanent failures of components in a multiprocessor, the load-dependent blocking at shared resources, and conditional branches in the software, a probabilistic model is required to properly represent the behavior of a multiprocessor controller. A state-space model can be formulated, with the states representing the current operational capacity of the system[3]. Multiprocessor response time is a function of the state. (Response time is the time interval between the moment of task initiation and the actuator and/or display result that occurs.)

A task is *triggered* when some set of events in the operating environment initiates its execution. If the environment is stochastically stationary, there are intertrigger distributions.

Every time a task is triggered, a unique *version* of it is created. This version is called an *extant version* until its execution is complete. Versions are numbered in sequence of triggering: successive versions of task i being denoted by $\Xi_{i,1}$, $\Xi_{i,2}$, etc. The response time associated with a version $\Xi_{i,j}$ is denoted by RESP(i), under the assumption that the system continues operating until the version completes executing. (This is clearly only valid so far as the expected response time is finite). The *extant time* of a particular extant version of task i is the difference between the absolute (or system) time at present and that at the moment of triggering. When an extant version finishes executing, its extant time is frozen by definition. Thus, the extant time at t of a version $\Xi_{i,j}$ that was triggered at time $\tau_{i,j}$ with the system in state $n_{i,j}$ is defined by $EXT[\Xi_{i,j},\tau_{i,j},n_{i,j},t]=\min[t-\tau_{i,j}, RESP(i)]$.

A controller task may be critical or non-critical. A critical task has a *hard*

3. The actual definition of the states depends on the system that is being modelled. Choosing a suitable formulation to facilitate system analysis is often a challenging task.

deadline [15], the violation of which leads to ***dynamic failure***. The deadlines are generally random variables. Non-critical tasks have no such deadlines[4].

The ***mission lifetime*** is the duration of operation between successive service stages.

2.3. The Performance Measures

We define a cost function $C_i(\xi)$ associated with system response time ξ for task i. For ***critical tasks***, the cost function takes the following form:

$$C_i(\xi) = \begin{cases} g_i(\xi) & \text{if } \xi \leq t_{di} \\ \infty & \text{otherwise} \end{cases} \qquad (1)$$

where $g_i(\xi)$ is a suitable continuous function of ξ that increases monotonically in $[0, t_{di}]$, is zero for all $\xi > t_{di}$, and we recognize t_{di} as the hard deadline associated with critical task i.[5] For ***noncritical tasks***, $C_i(\xi)$ is continuous and therefore always bounded for a finite response time. For consistency, we assume that the costs accrue as execution proceeds. In other words, if task i was triggered τ units ago and has not yet been completed, its contribution to the cost so far accrued is $C_i(\tau)$.

The ***cumulative cost function*** associated with task i, $\Gamma_i(t)$, is defined as follows:

$$\Gamma_i(t) = \sum_{j=1}^{q_i(t)} C_i(\mathrm{EXT}(\Xi_{i,j}, \tau_{i,j}, n_{i,j}, t)) \qquad (2)$$

where there have been $q_i(t)$ triggers for task i in the interval $[0,t]$. The ***system cost function*** in a system with r different tasks is defined as:

---

4. Note that dynamic failure encompasses the traditional notion of catastrophic hardware failure as well as other causes (software crashes, system reconfiguration, electromagnetic interference on the communications network, etc.) of missing one or more hard deadline.
5. In practice, all that can be assured of the behavior of $g_i(\xi)$ is that it is monotonically nondecreasing in $[0, t_{di}]$. However, we shall need to invert the cost function, and so, we assume a strictly increasing cost function in $[0, t_{di}]$.

$$S(t) = \sum_{i=1}^{r} \Gamma_i(t). \tag{3}$$

Let L(t) = Prob { Mission Lifetime ≤ t }, Q the set of states of the controller, and A ⊂ Q the subset of states in which failure is not certain. Note that failure is certain when the expected response time goes to infinity. Define B = Q-A. Our performance measures are then[6]:

$$\text{Cost Index, } K(\chi) = \int_0^{\infty} \text{Prob}\{S(t) \le \chi\}\, dL(t) \tag{4}$$

$$\text{Probability of dynamic failure, } p_{dyn} = \int_0^{\infty} \text{Prob}\{S(t) = \infty\}\, dL(t) \tag{5}$$

$$\text{Mean Cost, } M = \int_0^{\infty} E\{S(t) \mid \text{system never enters state set B}\}\, dL(t) \tag{6}$$

$$\text{Variance Cost, } V = \int_0^{\infty} \text{Var}\{S(t) \mid \text{system never enters state set B}\}\, dL(t) \tag{7}$$

We define also the following auxiliary measures:

$$\text{Cost Index for Task i, } K_i(y) = \int_0^{\infty} \text{Prob}\{\Gamma_i(t) \le y\}\, dL(t) \tag{8}$$

$$\text{Mean Cost for Task i, } M_i = \int_0^{\infty} E\{\Gamma_i(t) \mid \text{system never enters state set B}\}\, dL(t) \tag{9}$$

$$\text{Variance Cost for Task i, } V_i = \int_0^{\infty} \text{Var}\{\Gamma_i(t) \mid \text{system never enters state set B}\}\, dL(t) \tag{10}$$

The auxiliary measures are principally for use during the design phase.

The mean and variance costs and the cost indices can be extremely tedious to compute. In their place, it is often useful to substitute the following *modified* parameters:

$$\text{Modified Cost Index, } \tilde{K}(\chi) = \begin{cases} \left\{ \int_0^{\infty} \text{Prob}\{\tilde{S}(t) \le \chi\, dL(t)\} \right\} \{1 - p_{dyn}\} & \text{for } \chi < \infty \\ p_{dyn} & \text{for } \chi = \infty \end{cases} \tag{11}$$

$$\text{Mean Modified Cost, } \tilde{M} = \int_0^{\infty} E\{\tilde{S}(t) \mid \text{system never enters state set B}\}\, dL(t) \tag{12}$$

$$\text{Variance Modified Cost, } \tilde{V} = \int_0^{\infty} \text{VAR}\{\tilde{S}(t) \mid \text{system never enters state set B}\}\, dL(t). \tag{13}$$

6. The Cost Index is generally only for use when rival multiprocessors are being compared. For the evaluation of a *single* system, the other three measures are sufficient.

and the auxiliary measures $\tilde{R}_i$, $\tilde{M}_i$, and $\tilde{V}_i$ are defined analogously, with

$$\tilde{r}_i(t) = \sum_{j=1}^{q_i(t)} g_i(RES_i(j)) \tag{14}$$

and $\tilde{S}(t) = \sum_{i=1}^{r} \tilde{r}_i(t)$. Also, from physical considerations, there exists a $T<\infty$ such that for all $t \geq T$, $L(t)=1$, so that the above integrals always converge.

For values of t much greater than either the mean task completion time or intertrigger interval, the modified costs are good approximations to their exact counterparts. This is of value, since it applies most mission lifetimes, (for aircraft, mission lifetime can range from 30 minutes to 10 hours, while for spacecraft, it can range up to several months) and the modified parameters are much easier to compute. In the examples presented in this paper, we evaluate modified values throughout.

Let $P_\nu(t)$ be the probability that at time t the system is in state $\nu \in Q$, and $P_a^{(\xi)}(t)$ the probability that the system is in state $a \in A$, given that in a mission lifetime of ξ, it never enters a state in B. Let $P_{arr(i)}(t)dt$ be the probability of arrival of one version of task i in $[t,t+dt]$, and $w_{i,a}$ and $W_{i,a}$ the density and distribution functions, respectively, of the controller response time for task i when in state a. Further, denote the distribution of the hard deadline for task i, if critical, by F_{di}. Then,

$$P_{dyn(i)} \approx \sum_{a \in A} \int_{\omega=0}^{\infty} \int_{\xi=0}^{\infty} \int_{t=0}^{\xi} [1-W_{i,a}(\omega)] P_a^{(\xi)}(t) P_{arr(i)}(t)\, dt dL(\xi) dF_{di}(\omega) + \sum_{b \in B} \int_0^{\infty} P_b(t)dt \tag{15}$$

$$P_{dyn} \approx \sum_{i=1}^{r} P_{dyn(i)} - (r-1) \sum_{b \in B} \int_0^{\infty} P_b(t)dt \tag{16}$$

where the approximations hold whenever (as is almost invariably the case in practice) each of the integrals is much less than one.

Let $\tilde{C}_{i,a}$ be a random variable denoting the contribution to $\tilde{r}_i$ of a single version of task i, with the system in state $a \in A$, $tot(i)$ the value of $\tilde{r}_i$ over a mission

lifetime given that the system remains in state-set A throughout, and $\text{syst} = \sum_{i=1}^{r} \text{tot}(i)$. Let $\Pi_{\text{arr}(i)}(u,\vartheta)$ be the probability of u arrivals of versions of task i in an interval of length ϑ, and H_ξ^ν the distribution for the sojourn time of the system in state ν in a mission of length ξ.

Then, define the following characteristic functions:

$$\varphi_{\hat{C}_{i,a}}(s) = \begin{cases} \int_{t=0}^{\infty} \int_{q=0}^{g_i(t)} e^{-sq} w_{i,a}(g_i^{-1}(q))dq\, dF_{di}(t) & \text{if task i is critical} \\ \int_{q=0}^{\infty} e^{-sq} w_{i,a}(g_i^{-1}(q))dq & \text{otherwise} \end{cases} \tag{17}$$

for all $a \in A$,

$$\varphi_{\text{tot}(i)}(s) = \sum_{a \in A} \sum_{u=0}^{\infty} \int_{t=0}^{\infty} \int_{0}^{t} [\varphi_{\hat{C}_{i,a}}(s)]^u \Pi_{\text{arr}(i)}(u,\vartheta)\, dH_a^t(\vartheta)\, dL(t) \tag{18}$$

and

$$\varphi_{\text{syst}}(s) = \prod_{i=1}^{r} \varphi_{\text{tot}(i)}(s) \tag{19}$$

The reader will have noticed two implicit assumptions: (a) that the service time for a task is much less than the sojourn time of the system at any one state,[7] and (b) that costs incurred by each task-version are independent.

The Cost Indices can now be determined by inverting $\varphi_{\text{tot}(i)}(s)$ and $\varphi_{\text{syst}}(s)$, and weighting with the appropriate probability of dynamic failure.

2.4. Application of the Performance Measures

The mean cost, variance cost and the probability of dynamic failure can all be used in the design and evaluation of individual systems. Every such design is the result of a multiplicity of decisions regarding scheduling strategy, individual component redundancies, speed differentials, etc. The performance measures

7. State changes occur when hardware or software failures occur in the system. The rate at which these occur is far smaller than the mean task completion time. Also, of course, g_i is invertible in $[0, t_{di}]$.

can be used as optimization criteria in this context.

It is suggested that the measures be used in a two-stage process. First, the probability of dynamic failure of the designed system is computed and this compared with that required by the specifications. In the event that the system meets the dynamic failure requirements, expressions for the mean and variance costs are derived, and fine-tuning of the design carried out by studying the sensitivity of these measures to changes in system structure, speed, etc. If the system does not meet the dynamic failure requirements, the mean and variance costs are not computed. Instead, the system is redesigned to the point where the failure requirements are met before mean and variance costs are considered. Figure 1a summarizes the approach.

The System and the Task Cost Indices may be regarded as the distribution functions of random variables that we shall call the *system capability* and the *task capability* respectively. The system capability can be used in the comparison of two or more rival computer systems. Figure 1b explains the procedure. The task capabilities are used more subtly. See Section 3.3.

3. Remarks on Determining the Performance Measures

In what follows, the system consists of the *controlled process* (often referred to simply as the "process") and the *controller* (i.e., the multiprocessor).

Four items need to be determined, prior to computing the performance indices. These are the distribution of the hard deadlines t_{di} for each critical task i, the finite cost function g_j for each task j, the multiprocessor response time distribution as a function of its state, and the $P_\nu(t)$ for all $\nu \in Q$. We concentrate here on the first two, referring the reader to the example presented in Section 4 and the queueing theory and probability literature for the last two.

3.1. Derivation of the Hard Deadlines

Typically, critical tasks are associated with life-critical activity and so one is especially concerned in finding practical means for accurately estimating p_{dyn}. In order to do so, the hard deadlines must first be derived. In most cases, since the environment is only stochastically known, this takes the form of probability distributions.

To explain the derivation of the distribution of t_{dl}, a state-variable approach is useful. A general process may be described as

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t), t) \tag{20}$$

where $\mathbf{x}$ is the state vector, $\mathbf{u}$ is the input vector, and t represents time. The input vector can be partitioned thus:

$$\mathbf{u}^T = [\mathbf{u}_E^T \mid \mathbf{u}_C^T] \tag{21}$$

where $\mathbf{u}_E$ is the environment input subvector[8]. It represents the effects of the environment on the controlled process. For example, a gust of air is an environmental input when the controlled process is an aircraft.

$\mathbf{u}_C$ is the control input subvector. It represents the input delivered under controller command in response to an environmental input. For obvious reasons, it is always bounded.

The process is required to perform within a certain subset of the state-space. Let the admissible set of the state vector be bounded and denoted by $\mathbf{X}_A$. Note that the admissible state-space is not necessarily static.

The control $\mathbf{u}_C$ is employed to keep the process in $\mathbf{X}_A$. The control is clearly a function of $\mathbf{u}_E$. Since $\mathbf{X}_A$ is bounded, there is a bound on the controller response time allowed. This bound is the hard deadline. Since the process dynamics and the distribution of $\mathbf{u}_E$ are known, the distribution of each hard

8. By definition, an input from an I/O panel is also an environmental input.

deadline can in theory be determined.

For clarity, we restate the above. The interval between a trigger and the termination of the resulting controller response can be divided into two portions: controller think-time (i.e., the response time) and the actuation time during which the process reacts to the environmental trigger under controller directives. The hard deadline t_{di} associated with a control task i which is triggered by an environmental input is the maximum controller think time permitted, consistent with environmental conditions, the process dynamics and the requirement to keep the system within the admissible state-space, X_A.

Clearly, in order to derive the hard deadlines, a precise formulation of the process dynamics is required. Since such a formulation is a required part in the design of a critical process, no additional requirements are imposed on the system designer.

For an example in determining hard deadlines, the reader is directed to Section 4.

3.2. Derivation of Finite Cost Functions

Very little work has been published in this area. Cost functions of this type are usually -- and if at all -- specified in an ad-hoc manner. The main problem here is that the cost functions must be linked to the controlled process to have any concrete meaning. Contemporary workers (e.g., [3]) have skirted the problem -- with less than convincing results -- by ascribing qualitatively-defined "weights" to controller attributes (conventional reliability, throughput, etc.) and attempting to match these to corresponding weights for the application (also qualitatively obtained).

Controlled processes have performance measures that are functionals of system state and input, and which express the cost of running the process [17],

[18]. The traditional formulation is:

$$J(t) = f_o(x(t), u(t), t) \tag{22}$$

where J(t) is the instantaneous performance measure , f_o the functional and the other symbols have their usual meanings.

The cost of running the process over, say, an interval $[t_o, t_f]$.

$$\Theta = \int_{t_0}^{t_f} J(t)dt \tag{23}$$

We claim that a good representation for $g_i(\xi)$ is given by[9]

$$g_i(\xi) = \begin{cases} \Omega(\xi) - \Omega(0) & \text{for } 0 \le \xi \le t_{di} \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

where:

$\Omega(\eta)$ = Expected contribution of u_{ci} to Θ if RESP(i)=η, and

u_{ci} = control input subvector associated with task i.

See Section 4 for an example.

3.3. Discussion

By connecting the activity of the control system with that of the controlled process, a proper foundation has been provided for the definition of controller performance. All quantitites relating to the performance measures can be calculated from quantities that may be estimated or measured. It is true that our methods presuppose a knowledge of the controlled process and its dynamics that may not at first be accurately available. However -- and this is the crucial point -- an understanding of the dynamics of the process will surely increase with experience (say with a mathematical model or a prototype). There is

9. It is easy to see that many other suitable representations of the cost function can exist. For instance, one can modify $\Omega(\eta)$ to include higher moments of the contribution to Θ. However, for most practical purposes, the above measure should suffice. It is usually only when the intertrigger or service time distributions exhibit strongly non-Markovian behavior that such a modification needs to be considered. Also, g_i is 0 for values of response time greater than the associated hard deadline since to speak of finite cost after failure has occurred is meaningless.

therefore a learning curve associated with process operation and hence by induction another for the performance measures: Note that we are here considering *critical* processes such as nuclear reactors or aircraft and that such processes call for extensive testing and analysis before release or installation. Also, the operating environment for these critical processes is usually well known, so that a good model generally exists for the intertrigger distributions.

One can express uncertainty about the accuracy of the individual cost functions in terms of a confidence measure. The sensitivity of the system capability to changes in the cost functions is given indirectly by the individual task capabilities . The latter can therefore be used with confidence intervals for the cost functions to obtain a confidence interval for the system capability.

4. Example

In this section, we present an example for the determination of hard deadline distributions, mean and variance costs, and the analysis of a multiprocessor system. Both the example process and the multiprocessor are somewhat idealized, the principal purpose being to illustrate what has gone before.

4.1. The Process

4.1.1. Description

The controlled process is a communications satellite of mass m, floating in free space. One critical controller task (task 1) is to keep it within a sphere of radius R centred on a "rest position", $\mathbf{x}_0$. The environment is characterized by a series of impulses, deriving from meteorite impacts, and arriving along random directions. The meteorites arrive according to a Poisson distribution with parameter λ. The satellite must be restored to rest at $\mathbf{x}_0$ before the next meteorite comes in. The energy the meteorites impart to the satellite is assumed to be

constant at k units per impact.

The controller employs rockets to respond to the impulses and can impose a maximum thrust of α units of force in any direction. The rockets must be adjusted to the right direction, and this deployment takes τ_a units of time.

The performance index with respect to this one task is θ = total energy expended by the rockets. The problem is to determine the distribution of t_{d1} and the finite cost function, $g_1(t)$.

4.1.2. Derivation of t_{d1} Distribution

Catastrophic failure will occur either if the satellite is not restored to rest at x_0 by the time the next impulse comes along, or if the satellite leaves the admissible state space.

t_{d1} is the maximum think-time, and finding it is equivalent to solving for the minimum actuation time. We clearly have a bang-bang control solution. Using standard methods from optimal control theory, we conclude that

$$t_{d1} = \max \{ \min[\Lambda(\tau), t_1], 0 \} \tag{25}$$

where

τ is the interval between successive meteorite arrivals,

$$\Lambda(\tau) = \tau - \tau_a + \frac{\sqrt{2km}}{\alpha} - 2\sqrt{\frac{km}{\alpha^2} + \frac{\tau\sqrt{2km}}{\alpha}} \tag{26}$$

$$t_1 = \sqrt{\frac{m}{2k}} \cdot \left[R - \frac{k}{\alpha} \right] \tag{27}$$

The distribution function for t_{d1} is given by:

$$F_{d1}(\zeta) = \begin{cases} 0 & \text{if } \zeta < 0 \\ 1 - e^{-\lambda\Lambda^{-1}(\zeta)} & \text{if } 0 \le \zeta \le t_1 \\ 1 & \text{otherwise} \end{cases} \tag{28}$$

4.1.3. Derivation of g_1

Notice that since the environment is stochastic, the controller will always

order maximum thrust. The duration for which the thrust is maintained will depend on the response time, ξ. All computations are made under the condition that $\xi \leq t_{d1}$. Using the laws of mechanics, we arrive at the following result, which the reader may easily verify:

Expected contribution of u_{o1} to θ if response time = ξ,

$$\Omega_1(\xi) = \Omega_1(0) + \sqrt{\frac{2k}{m}}\, a\xi \tag{29}$$

We can therefore now write:

$$g_1(\xi) = \sqrt{\frac{2k}{m}}\, a\xi \quad \text{for } 0 \leq \xi \leq t_{d1} \tag{30}$$

The cost function $C_1(\xi)$ is now completely determined. It only remains for us to obtain the controller response time distribution and state probability functions.

> **Remark: Notice the tradeoff between t_{d1} and g_1. As the rocket thrust, a, is increased, the expected value of t_{d1} increases, meaning that more time is available for the controller, and the chances of catastrophic failure are reduced. This is paid for in the form of increased operating cost, i.e., g_1 rises faster.**

4.2. The Multiprocessor

4.2.1. Description

The multiprocessor is configured as shown in Figure 2. It consists of a despatcher with an infinite buffer which dynamically assigns tasks to processors as they come in and the processors become free. Input rate is Poisson, with parameter λ_i for task i. There are r tasks in the system, and all are critical. All tasks have an identical service time distribution[10]. There is no common memory in the system, each processor is assumed to have all the system applications software stored in its private memory[11]. The despatcher schedule is FIFO.

10. This assumption (used by many authors eg., [14])is removed by us in [20].
11. This assumption is not as unrealistic as it might seem. With memory densities rising, and costs falling, there are emerging designs based on this idea (eg., CM*FCS system [29]). The removal of this assumption entails analysis of a blocking problem. For a study of this type of system, see [22].

Processors fail at an exponential rate of μ_P; the despatcher (which is assumed to have internal redundancy) with rate μ_D, and each of the q redundant buses (of which only one is active at any one time) with rate μ_B. Components fail independently of each other, and coverage is 100%. Queueing delays at buses are small enough to be ignored[12].

4.2.2. Determining Task Execution Rate

The time taken to execute a task on a processor is a random variable whose distribution is affected by operating conditions and system parameters. The distribution is determined by operating system characteristics, processor failure rates (both transient and permanent) and the probability of incorrect transmission over the intercommunication medium together with the conditional branches within the executed code.

In any reliable system, the effects of the pertubations due to hardware failures or electromagnetic interference are very small due to their low probability of occurrence. It is generally assumed in analyses of this type that the software execution time assuming no pertubation due to hardware failure or interference is exponentially distributed. Since the pertubations are small, we approximate them by assuming the service rate still to be exponential, but shifting its parameter slightly to allow for the pertubations. Let this parameter be μ. To determine μ, we use the probability model shown in Figure 3. π_0 is the density function for the execution time of the software assuming no pertubations due to failure, incorrect transmissions, etc. This quantity can be derived through experiment, and naturally depends on the nature of the code. π_j is the density function of the j-th pertubation and α_j is the probability that this pertubation occurs during one execution of the code. Both these quantities can be

12. This is characteristic of this type of system. Messages transmitted are invariably either control messages or very small packets of data.

determined by experiment and/or analysis. Assuming, for analytical simplicity, that the pertubations are mutually independent, the characteristic function of the service time pdf for the code, Φ_q, can be written down by inspection as:

$$\Phi_q(s) = \Phi_0(s) \cdot \prod_j \left\{ \alpha_j \Phi_j(s) + 1 - \alpha_j \right\} \tag{31}$$

where

$\Phi_j(s)$ is the Laplace transform of $\pi_j(t)$.

The mean and variance of the execution time can be determined from $\Phi_q(s)$. Let them be M and V, respectively. Then, the value of μ is taken to be $\min\left\{1/M, 1/V\right\}$. This is generally a rather conservative estimate.

4.2.3. Analysis of the Multiprocessor

(a) Response Time Density Function: System failure occurs when the despatcher fails, all buses fail, or when there are $n = \lfloor \frac{\sum_{i=1}^{r} \lambda_i}{\mu} \rfloor$ processors or less functioning. The state description is: $\gamma=0$ when there is system failure, and $\gamma=n+1, \ldots, c$, when there is no system failure, and there are $n+1, \ldots, c$, processors respectively functioning in the system. Clearly, $A=\{n+1, \ldots, c\}$, and $B=\{0\}$.

Let $\lambda = \sum_{i=1}^{r} \lambda_i$. The system at state x (x>0) is essentially an M/M/x queue when the (small) despatcher service time is ignored. Therefore, the pdf of the response time of the system in state x (x>0) for each task i = 1, ..., r is given by [23]:

$$w_{i,x}(t) = \frac{\mu e^{-\mu t}[\lambda - x\mu + \mu \bar{W}_x(0)] - [1-\bar{W}_x(0)][\lambda - x\mu]\mu e^{-(x\mu-\lambda)t}}{\lambda - (x-1)\mu}, \tag{32}$$

where:

$$\bar{W}_x(0) = 1 - \frac{x(\lambda/\mu)^x}{x!(x-\lambda/\mu)} \left\{ \sum_{j=1}^{x-1} \frac{1}{j!} \left[\frac{\lambda}{\mu}\right]^j + \frac{1}{x!} \left(\frac{\lambda}{\mu}\right)^x \left[\frac{x\mu}{x\mu-\lambda}\right] \right\}^{-1} \tag{33}$$

(b) State Probability Functions:

Let $p_P(t)$, $p_D(t)$, and $p_B(t)$ be the probability of failure of a processor, despatcher and bus by time t respectively, where:

$$p_P(t) = 1 - e^{-\mu_P t} \tag{34}$$

$$p_B(t) = 1 - e^{-\mu_B t} \tag{35}$$

$$p_D(t) = 1 - e^{-\mu_D t} \tag{36}$$

Then, the probability of the system being in state 0 at time t,

$$P_0(t) \approx \sum_{i=0}^{n} \binom{c}{i} \{p_P(t)\}^{c-i}[1-p_P(t)]^i + [p_B(t)]^q + p_D(t) \tag{37}$$

and, the corresponding probability for state a, for a=n+1, ..., c,

$$P_a(t) \approx [1 - \{[p_B(t)]^q + p_D(t)\}]\left[\binom{c}{a}[p_P(t)]^{c-a}[1-p_P(t)]^a\right], \quad \text{and} \tag{38}$$

$$P_a^{(\ell)}(t) = \frac{P_a(t)}{1 - \sum_{b \in B} P_b(\ell)} \tag{39}$$

(c) Sojourn Time Density Function, $H_a^T(t)$: The derivation is conceptually straightforward. We therefore present only the result.

For a=n+1, ..., c-1,

$$H_a^T(t) = \begin{cases} 0 & \text{for } t<0 \\ 1-F_{\sigma_a}(T-t)\{1-F_{\pi-\sigma_a}(t)\}\{1-F_{\tilde{t}_a}(t)\} & \text{for } 0 \le t < T \\ 1 & \text{for } t \ge T \end{cases} \tag{40}$$

where

$$F_{\sigma_a}(t) = \begin{cases} 0 & \text{for } t<0 \\ \sum_{j=1}^{c-a} A_j^{(a)}\{1-e^{-(a+j)\mu_P t}\} & \text{for } 0 \le t < T \\ 1 & \text{for } t>T \end{cases} \tag{41}$$

$$A_j^{(a)} = \frac{(-1)^{j-1}c!}{(j-1)!(c-a-j)!a!(a+j)} \tag{42}$$

$$F_{\tilde{t}_a}(t) = \begin{cases} 0 & \text{for } t<0 \\ 1-e^{-a\mu_P t} & \text{for } t \ge 0 \end{cases} \tag{43}$$

$$F_{\pi-\sigma_a}(t) = 1 - \int_{k=t}^{\infty} F_{\sigma_a}(k-t)\,dF_\pi(k) \tag{44}$$

$$F_\pi(t) = e^{-\mu_D t}(1-e^{-\mu_B t})^q + (1-e^{-\mu_D t})(1-\{1-e^{-\mu_B t}\}^q) + (1-e^{-\mu_D t})(1-e^{-\mu_D t})^q \tag{45}$$

Also,

$$H_d^T(t) = \begin{cases} 0 & \text{for } t<0 \\ F_{\tau_{ij}}(t)+F_R(t)-F_{\tau_{ij}}(t)F_R(t) & \text{for } 0\le t<T \\ 1 & \text{for } t\ge T \end{cases} \qquad (46)$$

where

$$F_{\tau_{ij}}(t) = 1 - e^{-a\mu_P t} \qquad (47)$$

The expressions follow from a realization that when the mission lifetime is given, the sojourn times are no longer independent random variables.

4.3. Numerical Results

The parameters used for the process are r=78, λ_i=.5 for all tasks, μ=20 k=1, m=2, α=1, and R=1.75. The controller parameters are $\mu_P = 10^{-5}$, $\mu_B = 10^{-6}$, and $\mu_D = 10^{-6}$ All tasks are assumed to be critical with the cost function of the task in the example in Section 4.1.

For a system of this type, the only design variables are the number of processors the number of buses, and the speed of the hardware. The graphs that make up Figure 4 show the marginal benefit to be gained from the addition of processors. Addition of buses, it was found, has practically no impact on the probability of dynamic failure or on the mean and variance costs. (Mean and variance costs are equal in this system).

The marginal benefit to be gained from raising the processor strength from 2 to 3 is considerable; above a processor count of 4, the marginal benefits are practically non-existent for probability of dynamic failure. Similarly, above a processor count of 5, there is virtually no benefit to be accrued in adding processors. (Keep in mind that these re narks hold good only for mission lifetimes in the range considered.) The failure rate at the asymptotic level of infinite processors is principally due to the probability of working processors exceeding the hard deadline, and the mean and variance costs become the cost associated

with mean processor service time. If better performance than that provided by the asymptotic level is required, it can only be obtained by the introduction of faster hardware.

5. Discussion

In this paper, we have presented performance indices that are objective in the way they measure performance. Due to this objectivity, they can be used to advantage in both the design and evaluation phases of the development of a controller system.

As for design, the measures should help identify quasi-optimal architectures and operating systems. In the former category are included decisions regarding the interconnection structure and component speed differentials. In the latter category, we include the choice of schedule in the access of shared resources, the design of failure recovery procedures --especially the optimal placement of recovery points-- and algorithms controlling the allocation and reallocation of tasks to the individual processors. The lack of objective indices so far has forced contemporary workers to employ overly simplistic performance indices, most notably in solving the task allocation problem (such as ascribing unit cost to each transaction on the interconnection network [24].)

In the sphere of evaluation, the measures can be used to provide either an absolute or relative index to controller performance. In the former case, the performance functional, f_o, has to be defined with particular care, and the indices provide a measure of the overhead cost incurred in control. A second important application is the comparison of rival controller systems for particular applications.

Acknowledgements

The authors wish to thank Ricky Butler and Milton Holt of the NASA Langley

Research Center and Y. H. Lee of The University of Michigan for technical discussions.

References[13]

1. A. L. Hopkins, *et. al*, "FTMP -- A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft", *Proc. IEEE*, Vol. 66, No. 10, pp. 1221-1239, October 1978.

2. J. H. Wenseley, *et. al*, "SIFT -- Software Implemented Fault Tolerance," *Proc. IEEE*, Vol. 66, No. 10, pp. 1240-1256, October 1978.

3. M. J. Gonzalez and B. W. Jordan, "A Framework for the Quantitative Evaluation of Distributed Computer Systems", *IEEE Trans. Comput.*, Vol C-29, No. 12, Dec. 1980, pp. 1087-1094.

4. J. von Neumann, "Probabilistic logics and the Synthesis of Reliable Organisms from Unreliable Components", in *Automata Studies*, pp. 43-98, Princeton University Press, Princeton, NJ 1956.

5. E. E. Moore and C. E. Shannon, "Reliable Circuits Using Less Reliable Relays", *J. Franklin Inst.*, Pt. I, Vol. 262, pp. 191-208, and Pt. II, pp. 281-297, 1956.

6. W. G. Bouricius, *et. al*, "Reliability Modelling Techniques for Self-Repairing Computer Systems," *Proc. ACM 1969 Annual Conf.*, pp. 295-309, August 1969.

7. R. R. Borgerson and R. F. Frietas, "A Reliability Model for Gracefully Degrading and Standby-Sparing Systems," *IEEE Trans. Comput.*, Vol. C-24, pp. 517-525, May 1975.

8. G. N. Cherkesov, "Semi-Markovian Models of Reliability of Multichannel Systems with Unreplenishable Reserve of Time," *Engineering Cybernetics*, Vol. 18, pp. 65-78, Mar/April 1980.

9. J. Losq, "Effects of Failures on Gracefully Degradable Systems," *Seventh Annu. Int'l Conf. on Fault-Tolerant Computing*, Los Angeles, CA, pp. 29-34, March 1977.

10. R. Troy, "Dynamic Reconfiguration: An Algorithm and its Efficiency Evaluation," *op. cit*, pp. 44-49.

11. J. F. Meyer, *et. al*, "Performability Evaluation of the SIFT Computer," *IEEE Trans. Comput.*, Vol. C-29, No. 6, pp. 501-509, June 1980.

12. M. D. Beaudry, "Performance-Related Reliability Measures for Computing Systems," *IEEE Trans. Comput.*, Vol. C-27, No. 6, pp. 540-547, June 1978.

13. J. F. Meyer, "On Evaluating the Performability of Degrading Computer Systems," *IEEE Trans. Comput.*, Vol. C-29, pp. 720-731, August 1980.

---

13. To maintain anonymity during the review process, references 20 and 22 are not included in this list.

14. J. F. Meyer, "Closed-Form Solutions of Performability," *IEEE Trans. Comput.*, Vol. C-31, No. 7, pp. 648-657, July 1982.

15. G. K. Manacher, "Production and Stabilization of Real-Time Task Schedules," *J. ACM*, Vol. 14, No. 3, July 1967, pp. 439-465.

16. J. A. White, *et. al*, "Multimicroprocessor Flight Control: System Architectural Concepts," *Proc. AIAA Comp. in Aerosp. Conf.*, pp. 87-92, October 1979.

17. D. E. Kirk, *Optimal Control Theory*, Prentice Hall, Englewood Cliffs, NJ, 1970.

18. A. P. Sage, *Optimum Systems Control*, Prentice Hall, Englewood Cliffs, NJ, 1977.

19. M. Reiser and K. M. Chandy, eds., *Computer Performance*, North-Holland Publishing Co., Amsterdam, 1977.

20, 22: (Our forthcoming papers: see Footnote 13.)

21. S. J. Larimer and S. K. Maher, " The Continuously Reconfiguring Multiprocessor," *NATO-AGARD Meeting on Tactical Airborne Computing*, Roros, Norway, 1981.

23. D. Gross and C. M. Harris, *Fundamentals of Queueing Theory*, John Wiley, New York, 1974.

24. W. W. Chu, *et. al*, "Task Allocation in Distributed Data Processing," *Computer*, Vol. 13, No. 11, pp. 57-70, Nov 1980.

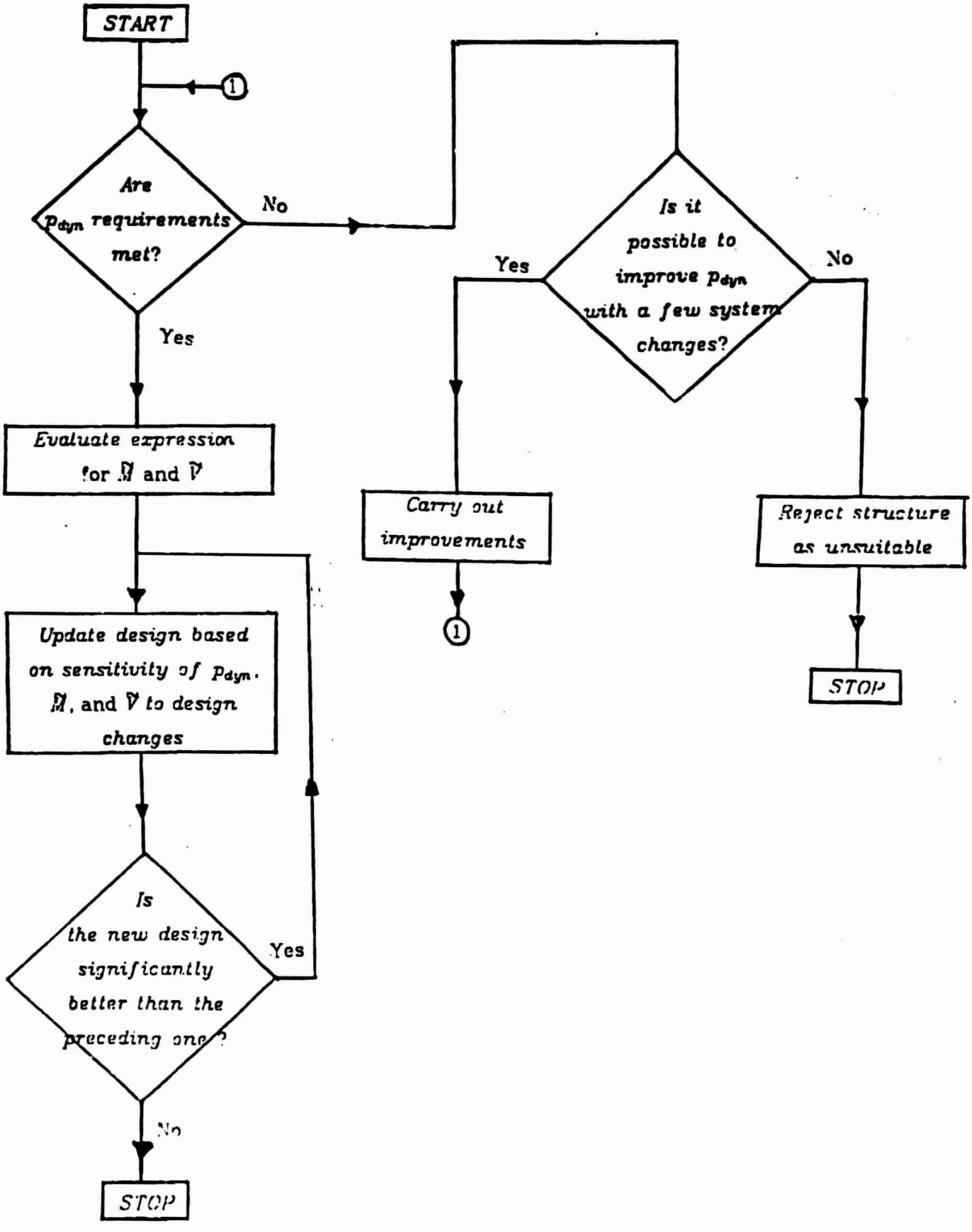


Fig 1a: System Refinement

```
PROCEDURE: COMPARISON(A)

/* A={A1, ..., Am} are the systems to be compared. */
        for i=1 to m do
            begin
                CALL SELECTION (A)
                RANK(i) ← SELECT
                A = A - {SELECT}
            end
        end COMPARISON;

PROCEDURE: SELECTION (A)

/*SCi is the system capability of system Ai.
A = {A1, . . . , An} are the systems to be compared. */
        CHOSEN ← 0
        for i= 1 to n do
            begin
                define MIN (i) = min_{j=1,...,i-1,i+1,...,n} {SCj}
                define PROB(i) = Prob { SCi ≤ MIN(i) }
                if PROB(i) > CHOSEN then
                    begin
                        CHOSEN ← PROB(i)
                        SELECT ← i
                    end
            end
        end SELECTION;
```

Fig 1b: Comparing Rival Systems

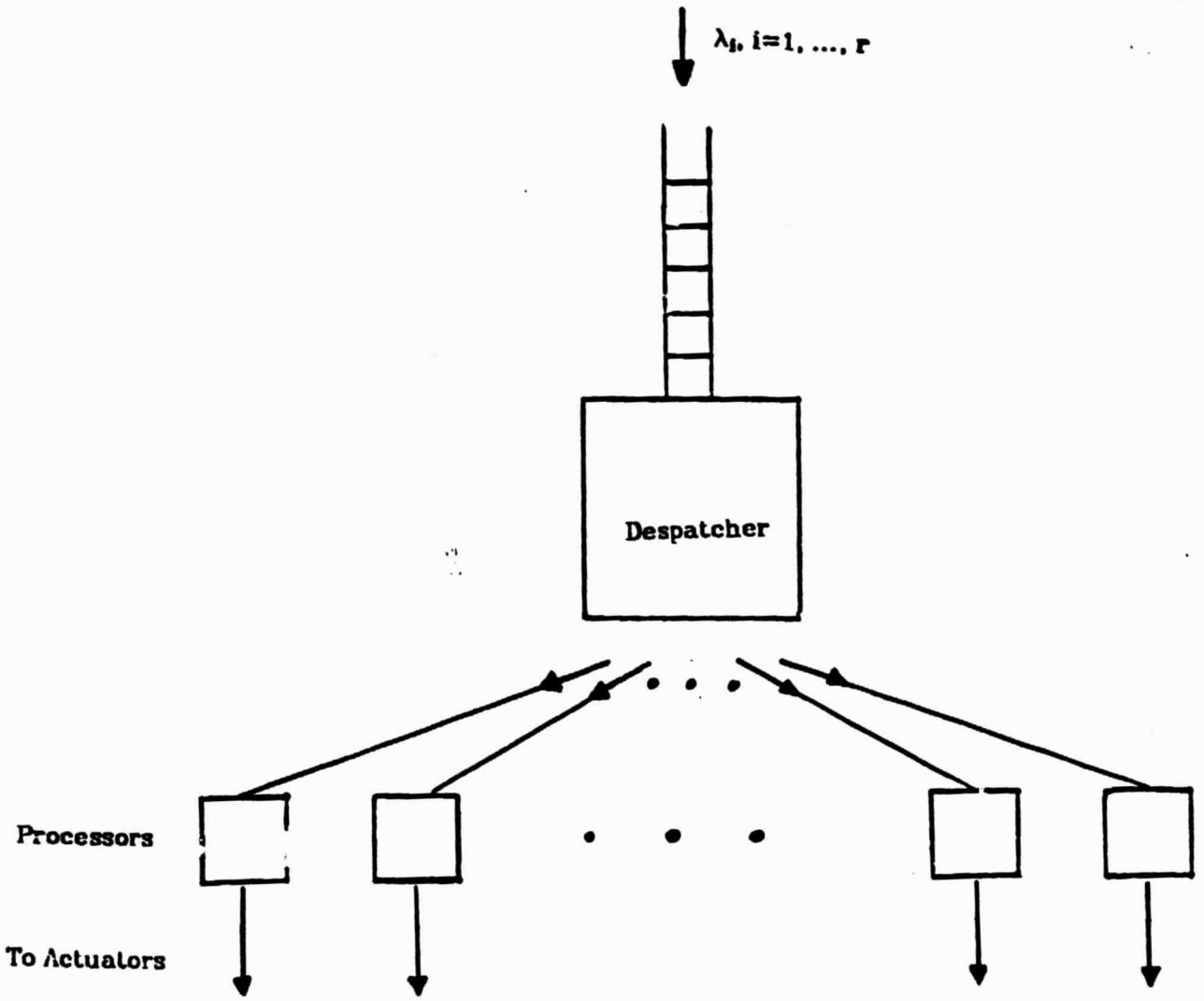
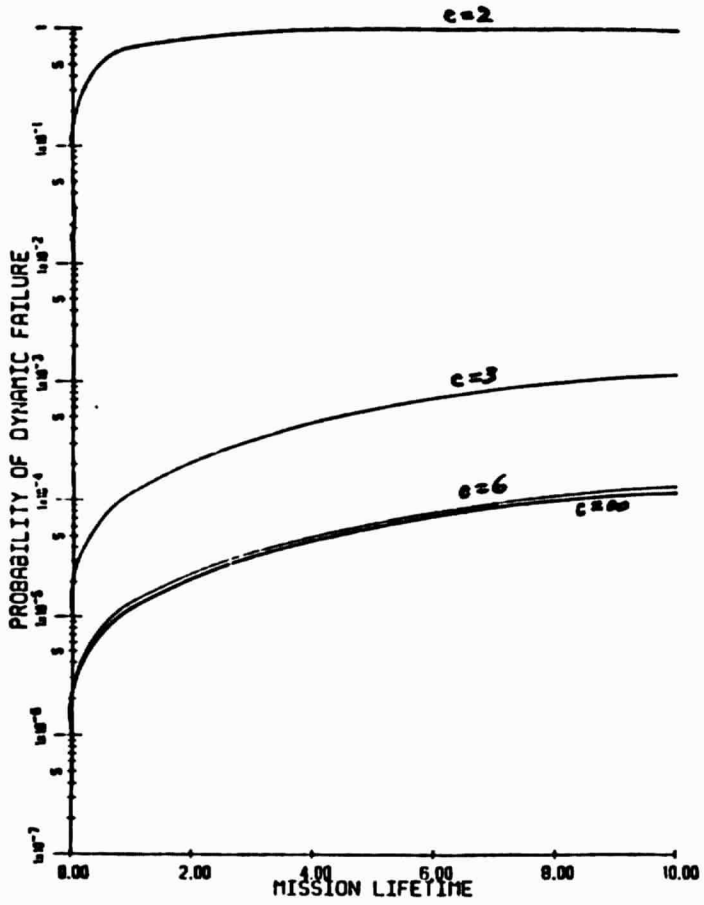


Fig 2: The Multiprocessor

Fig 3: Determining Task Response Time

ORIGINAL PAGE IS
OF POOR QUALITY



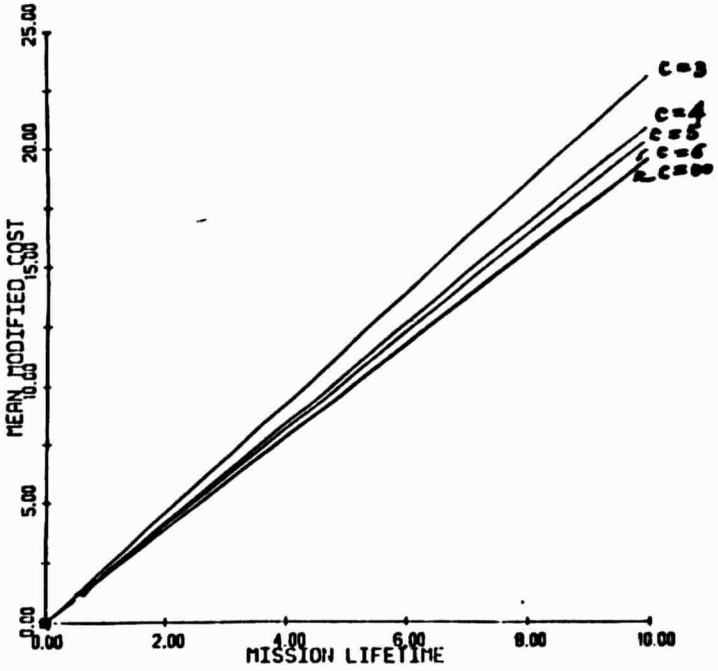


Fig 4: Numerical Results