Queueing Analysis of a Canonical Model of Real-Time Multiprocessors

C. M. Krishna
K. G. Shin

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

February 6, 1983

This work was supported by NASA Grant No. NAG 1-296. All correspondence should be addressed to Professor K. G. Shin. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the view of the funding agency.

Queueing Analysis of a Canonical Model of Real-Time Multiprocessors

ABSTRACT

Multiprocessors are beginning to be regarded increasingly favorably as candidates for controllers in critical real-time control applications such as aircraft. Their considerable tolerance of component failures together with their great potential for high throughput are contributory factors.

In this paper, we present first a logical classification of multiprocessor structures from the point of view of control applications. We point out that one important subclass has hitherto been neglected by the analysts. This is the class of systems with a common memory, minimal interprocessor communication and perfect processor symmetry.

The performance characteristic of the greatest importance in real-time applications is the response time distribution. Indeed, we have shown in a separate paper [2] how it is possible to characterize rigorously and objectively the performance of a real-time multiprocessor given the application and the multiprocessor response time distribution and component failure characteristics. We therefore present here a computation of the response time distribution for a canonical model of real-time multiprocessor.

To do so, we approximate the multiprocessor by a blocking model and present a means for efficient analysis. Two separate models are derived: one created from the system's point of view, and the other from the point of view of an incoming task. The former model is analyzed along largely conventional lines. For the latter model, an *artificial server* is used. and the system transformed into a queueing network. Validations show that the approximation is good.

February 6, 1983

handled by an immediate extension of this approach. To solve this problem, we employ *artificial servers*. Artificial or fictitious servers have been used in a number of models. In [6], they are employed in the analysis of an open queueing network with blocking. Our analysis, although superficially similar, is non-iterative and treats what amounts to an M/M/c queue with blocking. In such a setting, the iterative technique presented in [6] is not applicable and the artificial server approach must be combined with some means of determining the state of the system.

We begin by noting that the system as shown in Figure 2 can be approximated by that in Figure 3. The approximation is justified since the time spent by a task receiving service (as distinct from waiting in the queue) at the dispatcher is negligible compared to the memory service time. According to our approximate model, then, an incoming task waits in the Memory-Dispatcher (M.D.) queue, and, provided it is admitted (see below for definition), receives service first from the memory and then is assigned a processor for execution. Tasks are processed by the M.D. only when the number of tasks already admitted is less than the number of processors. A task is said to be *admitted* when it is either in the process of accessing the common memory or executing at a processor. The Memory-Dispatcher waits for a processor to become free, then transfers the relevant software to the target processor.

To analyze this model, we portray it from two complementary points of view. The first, which is the *system-oriented* model is as in Figure 3. On the other hand, Figure 4 depicts the system as seen by an incoming task, i.e., it is a *task-oriented* model. We consider each of the models in turn. The system oriented model will first be analyzed to provide parameter values that will then be used in the task-oriented model to obtain task response time distributions.

ORIGINAL PAGE IS OF POOR QUALITY

ences the architecture of the interconnection structure.

How well a multiprocessor controller performs clearly depends on the requirements of the application. We showed in [2] how controller performance could be rigorously and objectively evaluated, given the control application and multiprocessor response time distribution. Keeping this foundation in mind, we evaluate in this paper the performance of a canonical model of multiprocessors for real-time control.

This paper is organized as follows. In the following section, we classify real-time multiprocessors into three basic logical families. Then, in Section 3, we analyze an important subclass of multiprocessors, obtaining an approximate expression for its response time. We conclude in Section 4 by describing briefly some extensions of this work presently being undertaken, and their implications for real-time multiprocessor design.

2. Canonical Models of Real-Time Multiprocessors

The execution by a processor of any task involves, first, the transfer of all or part of the relevant software to the processor private memory or cache, and then, its execution. Because the entire set of programs that the control system will ever execute is well-defined, the designer has an option: the system can either have a common, or mass, memory that contains all the applications software -- thus requiring the transfer of the relevant software upon a task trigger[1] -- or every processor can hold in its private memory all the applications software it will ever need. The latter alternative eliminates the need to transfer software, thus reducing overall response time. Choosing this alternative presents the designer with two sub-alternatives: either provide each processor

1. The operating environment is modelled as the source of a stream of *task triggers*, i.e., environmental stimuli that cause a particular set of tasks to be performed by the multiprocessor.

with a private memory so large that the set of control programs required in the control of the process can be held in it in its entirety, or preallocate tasks to specific processors.

Based on the above observation, the following can be considered to be canonical logical models of multiprocessors used in real-time control.

Type 1: A Type 1 multiprocessor controller is one in which the processors do not specialize; in other words, tasks are not reserved for specific processors, and each processor in the system may be allocated any task. The process of task allocation is typically dynamic. This type is divided into two subclasses depending on the size of processors' private memory. In a *Type 1a* system, the private memory of the processors is large enough to hold both the applications and executive software in its entirety[2]. This system is ideal for applications in which a small number of relatively independent and small tasks form the job mix. One example of such Type 1a is the CM^{2}FCS [8] system of the United States Air Force.

In a *Type 1b* system on the other hand, the private memory is too small to hold all the applications software and requires the transfer of software with each task trigger. This transfer can be either single, or involve paging.

One can see the tradeoff between requirements of memory size and task execution time from this subclassification. As a general rule, the processors allocated a task execute it in its entirety without interruption unless some of the processors fail. There is thus little or no interaction between the individual processors in this type of system.

The best known implementation of the Type 1b structure is the Draper Laboratory's Fault-Tolerant Multiprocessor (FTMP) [5].

2. With memory densities rising and costs falling, this may be no longer as difficult as it once seemed.

ORIGINAL PAGE IS
OF POOR QUALITY

Type 2: In a Type 2 system, the processors are preallocated specific tasks (or subtasks) and the software related to these is loaded in their private memory. With the identification of specific tasks with particular processors comes the problem of reallocation of tasks on processor failure. Thus, the system of this type is inflexible to reconfigure itself upon failure but its reliability is obtained in general through physical redundancy in system components. In general, Type 2 is used when the individual tasks have significantly different time/safety criticality (e.g. flight control and navigation tasks in aircraft applications). Processor interaction can be considerable. The SIFT [9] system built by SRI International is an example.

Type 3: A Type 3 system is a composite of a Type 1 and a Type 2 system.

Figure 1 shows graphically the above classification of multiprocessors. Notice that the classification is *logical* and is different from most conventional ones that are usually based on physical interconnection structures.

A considerable body of literature has developed around the problem of analyzing multiprocessors. Almost invariably, the procedure is to use a Markov model of the system to solve for such parameters as throughput, reliability, availability, etc. Type 1a systems have been the focus of a great deal of attention. The tendency of almost all authors is to assume identical processors and an identical exponential service rate distribution for all job classes, upon which the system degenerates into an M/M/m queue. This analysis is then embedded in a determination of the multiprocessor performance. One good example is the work on closed-form estimations of performability by Meyer [3].

Type 2 systems can be modelled as queueing networks, and there is a large body of literature on this topic.

Type 1b, however, has been almost totally neglected. This is odd, considering that one of the few multiprocessors actually to be constructed (e.g. FTMP[5]) is an example of this type of system. Also, it is likely that Type 1b systems will grow in importance as time progresses since it is ideal when the job mix is composed of a large number of tasks, each called relatively ***infrequently***. Again, the analysis of Type 3 systems requires, as a prerequisite, the analysis of Type 1 systems. In this paper, we shall analyze an important subclass of the Type 1b system.

3. Response-Time Analysis of Type 1b System Without Paging

3.1. Description of the Real-Time Multiprocessor

The multiprocessor we analyze is shown in Figure 2. It has a dispatcher allocating tasks to c identical processors[3]. Service at the dispatcher is FCFS, and all the tasks place an identical (statistically speaking) demand upon the computational resources of the system.

When a processor is assigned a task, it first sends to the common mass memory for the relevant applications software. This is the only reference the processor makes to the common memory during a single execution. Tasks arrive at a Poisson rate of λ, and all service rates are exponential (in Section 3.5 we also consider non-exponential service rates): at the processors it is μ, and at the memory the software transfer rate (per task) is μ_m. Note that μ represents the actual execution rate for an individual task on a single processor; it does not take into account software transfer time.

A point tacitly made in almost all work in this area is that the input process is Poisson, and that all service rates are exponential. The principal reasons for such an assumption are that (i) while both input and service distributions are general in practice, they can usually be approximated well by Poisson and

3. Since processors may fail during a mission lifetime, c is a random variable.

exponential assumptions, and (ii) that analysis of a system with general arrival and service distributions is almost impossibly difficult.

The first point ensures that the exponential assumption leads to at least an approximate model of reality. It is important, however, to check this fact for each particular model by employing alternative approaches, eg., simulation. In this paper, we carry out this task by means of a simulation program that assumes Weibull service distributions. The exponential distribution is a special case of the Weibull. By varying the standard deviation of the service distribution while keeping the mean constant, we obtain an indication of the range of input intensities for which the exponential assumption is a good approximation.

It is readily seen that in this model, there is the problem of simultaneous possession of resources by the tasks. That is, there is a period when, immediately after being allocated a processor by the dispatcher, the processor queues up for service at the common memory. for the applications software. During this period, the processor is effectively locked out, i.e., it is forced to remain idle. Note that the present system is assumed to admit of no multiprogramming.

Since there is a period when a task is in possession of both a processor and the common memory, this multiprocessor does not fit into any of the well-behaved queueing models (such as M/M/m, G/M/m, etc.). There are two approaches to obtaining a solution for the response time distribution. The first is *iterative* and employs the method of surrogate servers as in [4]. The second, which we present here, also involves surrogate or artificial servers but is *non-iterative* in nature. It consists of approximating the multiprocessor by a model in which simultaneous possession of resources does not occur, but where *blocking* takes place. We have thus translated our problem into the context of a blocking model. An important consequence of this, which we shall discuss in Section 4, is that the analysis of the case where paging is allowed for can be handled by an immediate extension of this approach. To solve this problem, we employ *artificial servers*. Artificial or fictitious servers have been used in a number of models. In [6], they are employed in the analysis of an open queueing network with blocking. Our analysis, although superficially similar, is non-iterative and treats what amounts to an M/M/c queue with blocking. In such a setting, the iterative technique presented in [6] is not applicable and the artificial server approach must be combined with some means of determining the state of the system.

We begin by noting that the system as shown in Figure 2 can be approximated by that in Figure 3. The approximation is justified since the time spent by a task receiving service (as distinct from waiting in the queue) at the dispatcher is negligible compared to the memory service time. According to our approximate model, then, an incoming task waits in the Memory-Dispatcher (M.D.) queue, and, provided it is admitted (see below for definition), receives service first from the memory and then is assigned a processor for execution. Tasks are processed by the M.D. only when the number of tasks already admitted is less than the number of processors. A task is said to be *admitted* when it is either in the process of accessing the common memory or executing at a processor. The Memory-Dispatcher waits for a processor to become free, then transfers the relevant software to the target processor.

To analyze this model, we portray it from two complementary points of view. The first, which is the *system-oriented* model is as in Figure 3. On the other hand, Figure 4 depicts the system as seen by an incoming task, i.e., it is a *task-oriented* model. We consider each of the models in turn. The system oriented model will first be analyzed to provide parameter values that will then be used in the task-oriented model to obtain task response time distributions.

3.2. System-Oriented Model

The quantities that this model provides us with are $\{p_{i,j}\}$ where we define $p_{i,j}$ to be the probability that there are i tasks in the M.D. queue (including the one receiving service at memory if no blocking is taking place), and that j processors are executing (i.e., the applications software has been transferred to these j processors and actual execution is taking place). Keep in mind that blocking of incoming tasks begins to occur when $i+j=c$ and a new task arrives before any admitted task has been completed.

Note that unless otherwise stated, a variable is henceforth assumed to be zero if one or more of its subscript indices become negative. The state-transition diagram appears as Figure 5 and assuming steady-state, the following global balance equations can be written down:

$$\lambda p_{0,0} = \mu p_{0,1} \tag{1}$$

$$(\lambda+j\mu)p_{0,j} = (j+1)\mu p_{0,j+1} + \mu_m p_{1,j-1} \qquad \text{for } j<c \tag{2}$$

$$(\lambda+c\mu)p_{0,c} = \mu_m p_{1,c-1} \tag{3}$$

$$(\lambda+\mu_m)p_{i,0} = \lambda p_{i-1,0}+\mu p_{i,1} \qquad \text{for } i>0 \tag{4}$$

$$(\lambda+j\mu+\mu_m)p_{i,j} = \lambda p_{i-1,j}+\mu_m p_{i+1,j-1}+(j+1)\mu p_{i,j+1} \qquad \text{for } 0<j<c,\ i>0 \tag{5}$$

$$(\lambda+c\mu)p_{i,c} = \lambda p_{i-1,c}+\mu_m p_{i+1,c-1} \qquad \text{for } i>0 \tag{6}$$

The boundary condition is:

$$\sum_{i=0}^{\infty}\sum_{j=0}^{c} p_{i,j} = 1 \tag{7}$$

To solve for $p_{i,j}$, we use the method of generating functions.

Define

$$g_j(z) = \sum_{i=0}^{\infty} p_{i,j} z^i \tag{8}$$

Using generating functions and manipulating, we obtain the following recursion for $c>1$:

$$g_1(z) = \frac{\lambda(1-z)+\mu_m}{\mu} g_0(z) - \frac{\mu_m}{\mu} p_{0,0} \tag{9}$$

$$\begin{aligned} g_{j+1}(z) = {} & \frac{\lambda(1-z)+j\mu+\mu_m}{\mu(j+1)} g_j(z) - \frac{\mu_m}{z\mu(j+1)} g_{j-1}(z) \\ & - \frac{\mu_m}{\mu(j+1)} p_{0,j} + \frac{\mu_m}{z\mu(j+1)} p_{0,j-1} \qquad \text{for } 0<j<c \end{aligned} \tag{10}$$

$$g_c(z) = \frac{\mu_m}{z[\lambda(1-z)+c\mu]} g_{c-1}(z) - \frac{\mu_m}{z[\lambda(1-z)+c\mu]} p_{0,c-1} \tag{11}$$

From (9) and (10), we have

$$g_j(z) = A_j(z) g_0(z) + \sum_{i=0}^{j-1} B_{j,i}(z) p_{0,i} \qquad \text{for} 0\le j\le c \tag{12}$$

where the coefficients are defined by the following recursion for $1 \le j < c$ and $i < j-1$:

$$\begin{aligned} A_0(z) &= 1 \\ A_1(z) &= \frac{\lambda(1-z)+\mu_m}{\mu} \\ A_{j+1}(z) &= \frac{\lambda(1-z)+j\mu+\mu_m}{\mu(j+1)} A_j(z) - \frac{\mu_m}{z\mu(j+1)} A_{j-1}(z) \\ B_{j+1,i}(z) &= \frac{\lambda(1-z)+j\mu+\mu_m}{\mu(j+1)} B_{j,i}(z) - \frac{\mu_m}{z\mu(j+1)} B_{j-1,i}(z) \\ B_{j+1,j-1}(z) &= \frac{\lambda(1-z)+j\mu+\mu_m}{\mu(j+1)} B_{j,j-1}(z) + \frac{\mu_m}{z\mu(j+1)} \\ B_{j+1,j}(z) &= -\frac{\mu_m}{\mu(j+1)} \end{aligned} \tag{13}$$

In particular,

$$g_c(z) = A_c(z) g_0(z) + \sum_{i=0}^{c-1} B_{c,i} p_{0,i} \tag{14}$$

Equating the right-hand sides of (11) and (14), we have:

$$\begin{aligned} g_0(z) = {} & \left\{ D(z)A_{c-1}(z) - A_c(z) \right\}^{-1} \left[\sum_{i=0}^{c-2} \left\{ B_{c,i}(z) - D(z)B_{c-1,i}(z) \right\} p_{0,i} \right. \\ & \left. + \left\{ B_{c,c-1}(z) + D(z) \right\} p_{0,c-1} \right] \end{aligned} \tag{15}$$

where:

$$D(z) = \frac{\mu_m}{z[\lambda(1-z)+c\mu]} \tag{16}$$

The only remaining unknowns are the $p_{0,i}$ for $0 \le i \le c-1$. The equations required to solve for these are derived as follows.

The boundary condition (7) yields the relation:

$$\sum_{j=0}^{c} g_j(1) = 1 \tag{17}$$

Also, the generating functions $g_j(z)$ must converge in the unit disc $|z| \le 1$. All poles of the generating functions lying in the unit disc must therefore be cancelled out by corresponding zeros. The use of this condition yields further equations in the $p_{0,i}$ for $0 \le i \le c-1$ by (15) since the zeros of the generating function are functions of the values taken by the $p_{0,i}$. Note that no additional equations can be obtained from a further invocation of (10) since the apparent additional pole at the origin in (10) is cancelled out by a zero.

It can be shown that the above equations are sufficient to permit the computation of the $p_{0,i}$ values. The generating functions $g_j(z)$ are therefore completely determined. By inverting them (numerically, in most instances), the steady state probabilities $\{ p_{i,j} \}$ can be obtained.

Recall that the above analysis is for $c>1$. For the case when $c=1$, we have:

$$g_0(z) = \frac{z\mu_m\alpha(z) - \mu\mu_m}{z\alpha(z)\alpha_m(z) - \mu\mu_m} p_{0,0}$$

and

$$g_1(z) = \frac{\alpha_m(z)}{\mu} g_0(z) - \frac{\mu_m}{\mu} p_{0,0}$$

where

$$\alpha(z) = \lambda(1-z) + \mu$$
$$\alpha_m(z) = \lambda(1-z) + \mu_m$$

and

$$P_{0,0} = 1 - \left[\frac{\lambda}{\mu} + \frac{\lambda}{\mu_m}\right]$$

Remark: The dispatcher queue in physical implementations is clearly finite. By making it sufficiently large, say N, however, it may be considered effectively infinite with probability $\sum_{i=0}^{N} \sum_{j=0}^{n} p_{i,j}$.

3.3. Task-Oriented Model

In this model, we employ an *artificial server* to account for blocking delay. This yields an approximate solution for the response time distribution. This solution gives an upper bound for the true response time distribution, so that we err on the side of safety.

An incoming task views the system as expressed in Figure 4. With a probability α, it is blocked. The blocking is expressed in terms of an artificial server, to which the incoming task branches if it finds the system blocked, i.e., the service rate distribution of the artificial server is the same as the departure rate distribution for the tasks executing on the processors. When the service rate at the processors is exponentially distributed, the artificial server also has an exponential service rate distribution.

Implicit in this analysis is the assumption that steady-state exists at all times. This is patently not true. In most cases -- considering that controllers of this type are generally lightly loaded -- the probability of blocking is very small. However, once a blocking cycle begins, the probability of its continuing is greater than α for obvious reasons. Even so, it is unlikely that steady-state in either the blocking or nonblocking mode will be established perfectly. This is especially the case when the input intensity is neither very low nor near to driving the system into saturation. When in this intermediate range, one would expect the system to switch from the blocked to the unblocked state and vice versa with a fairly high frequency. Hence, any model that is based on the steady-steady state assumption would be expected to provide less accurate results in the intermediate range of input intensity than in the extreme ranges. (Indeed, the relative accuracy between analytical results at two different intensities might be used to obtain some indication of the frequency with which the system switches from the blocked to the unblocked state).

If we assume that steady-state exists at all times, the problem lends itself to a particularly simple solution and we obtain an upper bound to the true solution. It will be shown by simulation that this upper bound is good.

Under the assumption of steady-state, the system may be regarded as the queueing network shown in Figure 4. The artificial server represents the blocking delay. Notice that unlike in most analyses, incoming tasks have to wait in line to be served by this server. The interpretation of this last statement is as follows The task at the head-of-the-line (H.O.L.) position is waiting for the first of the tasks admitted to the system to leave so that it can be unblocked and begin to attempt to access the M.D. Tasks that have entered the queue. after the H.O.L. task have this latter task as an additional impediment to entry into the system proper. The system represented in Figure 4 follows immediately from the above argument. Notice that under the queueing rules for this model, there is no queueing for the processors once memory access is completed; a task is admitted into the memory only when there is a free processor ready and waiting for it.

It only remains for us, then, to compute the value of α and the artificial service rate, μ_{as}.

It is easy to see that

$$\alpha = 1 - \sum_{i+j \leq c} p_{i,j} \qquad (18)$$

and

$$\mu_{as} = \frac{\sum_{i+j\geq c} j\,\mu\,p_{i,j}}{\sum_{i+j\geq c} p_{i,j}} \qquad (19)$$

If $R_c(s)$ is the Laplace transform of the response time distribution for the tasks when the system has c processors functioning, we have:

$$R_c(s) = \left[\alpha\frac{\mu_a}{s+\mu_a}+1-\alpha\right]\cdot\frac{\mu_{mq}}{s+\mu_{mq}}\cdot\frac{\mu}{s+\mu} \qquad (20)$$

where

$$\mu_a = \mu_{as}-\alpha\lambda \qquad (21)$$

and

$$\mu_{mq} = \mu_m-\lambda$$

which on inversion yields the response time density function for c>0:

$$r_c(t) = Ae^{-\mu_a t} + Be^{-\mu_{mq} t} + Ce^{-\mu t}, \qquad (22)$$

where:

$$A = \frac{\alpha\mu\mu_a\mu_{mq}}{(\mu_{mq}-\mu_a)(\mu-\mu_a)}$$
$$B = \frac{\mu\mu_{mq}[\mu_a-\mu_{mq}(1-\alpha)]}{(\mu_a-\mu_{mq})(\mu-\mu_{mq})} \qquad (23)$$
$$C = \frac{\mu\mu_{mq}[\mu_a-\mu(1-\alpha)]}{(\mu_a-\mu)(\mu_{mq}-\mu)}$$

3.4. Validation

In an attempt to validate the analytical results obtained above, a multiprocessor with three processors was considered.

As a reference, a GPSS simulation model was developed for the system. The input intensity was varied over a wide range and results obtained through simulation compared with those obtained from analysis. In Table 1, the mean response times from the simulation and the analysis are compared.

Throughout the range of intensities studied, the analytical and simulation results were acceptably close (to within 10% in most cases). Notice that, as predicted in the preceding section, the analytical and simulation results for the lower and higher ranges of the input intensities agree much better than those for the intermediate region. In particular, the agreement at the upper and lower ends is better than 9%, while that in the intermediate range deteriorates to 15%. The most probable reason for this is that in this range of intensity, the system switches frequently from the blocked to the non-blocked state and vice versa. Steady state is therefore not achieved in the intermediate cases and one of the assumptions inherent to the task-oriented model is thereby violated. In support of this, note that the analytical results for this region indicate a consistently larger mean response time than the simulation results.

3.5. Non-exponential Service Rates

While the extreme complexity of analyzing systems with blocking and general service distributions forces the analyst to assume exponential service distributions, it is common knowledge that programs on many occasions have non-exponential service requirements. This can be modelled by equating the mean of the actual service time to a fictitious exponential parameter. In such instances, the sensitivity of the model to this inaccuracy is crucial[4].

We test the robustness of our model to non-exponential service rate distributions by assuming those to be Weibull, which is a more general distribution than the exponential. This distribution has the distribution function $F_{we}(t)=1-e^{-(\zeta t)^{\eta}}$. The mean is

$$\mu_{we} = \Gamma((\eta+1)/\eta)/\zeta$$

and the variance is

$$\sigma_{we}^2 = [\Gamma((\eta+2)/\eta)-\Gamma^2((\eta+1)/\eta)]/\zeta^2$$

4. Indeed, the success of such models as the central server is due in a large measure to a relative insensitivity to this inaccuracy.

where $\Gamma(\bullet)$ is the gamma function.

The exponential and the Rayleigh distributions are special cases of the Weibull, obtained by setting $\eta=1$, and $\eta=2$, respectively. When $\eta<1$, the variance is greater than the mean, when $\eta=1$, the variance equals the mean, and when $\eta>1$, the variance is less than the mean.

Our test for model robustness takes the form of plotting the ratio of the mean response time when η is 0.5 and 0.7 respectively to when $\eta=1$ (i.e., the exponential distribution). The mean response time stays close to the exponential value for a relatively large value of nominal input intensity. As expected, at high intensities, there is a marked divergence from the value predicted by the exponential distribution. However, in critical control applications, the utilization is almost always very low to allow sufficiently broad margins of safety. Note that the actual input intensity is greater than the nominal value which defined in Figure 6. We have performed computer simulation and obtained Figure 6.

4. Discussion

In this paper, we have presented a logical taxonomy for multiprocessor systems used in control applications, and analyzed one important class. We consider here methods of extending and using this analysis.

Extension to Type 1b systems with paging, although nontrivial, is not difficult. One would have here three classes of jobs: one in front of, or being served by, the memory-dispatcher, one undergoing service, and the third class consisting of jobs either being served by the memory after receiving a portion of their service at the processor or waiting to reaccess the processor. It is not difficult to think of a number of different systems based on variations of the above basic theme. For instance, one might obtain new models based on assigning priorities to the tasks. In *all* these cases, the artificial server approach can be used to advantage.

Also, the reader should bear in mind that we have analyzed a *logical* family of computer systems. This analysis is therefore valid for a wide range of *physical* implementations.

The response time distribution, once obtained, can be used in a number of ways. As already mentioned, we showed in [2] how response time distributions together with a probabilistic model of the computer and a full mathematical description of the control application can be processed to provide rigorous and objective means for evaluating the performance of a multiprocessor system *in the context of its application*.

Tradeoffs can also be studied and the multiprocessor system refined thereby. For example, one might consider trading off processor *number* against processor *power*[7].

5. References

[1] . "A Distributed Microprocessor System for Controlling and Managing Fighter Aircraft", *Proc. Distributed Data Acquisition, Computing, and Control Symposium*, Miami Beach, FL., December 1980, pp. 156-166.

[2] "Performance Measures for Multiprocessor Controllers," *Performance '83: Ninth Int'l Symp. on Computer Perf., Measurement and Evaluation.* (To be presented).

[3]. J. F. Meyer, "On Evaluating the Performability of Degrading Computer Systems," *IEEE Trans. Comput.*, Vol. C-29, No. 6, pp. 501-509, June 1980.

[4]. P. Jacobson and E. D. Lazowska, "The Method of Surrogate Delays: Simultaneous Possession in Analytic Models of Computer Systems," *ACM Sigmetrics Performance Evaluation Review*, Vol. 10, No. 3, pp. 165-174, September 1981.

[5]. 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.

[6]. A. A. Nilsson and T. Altiok, "Open Queueing Networks with Finite Capacity Queues," *Proc. 1981 Int'l Conf. Parallel Processing*, pp. 87-91, August 25-28, 1981.

[7]. P. J. B. King and I. Mitrani, "The Effect of Breakdown on the Performance of

ORIGINAL PAGE IS
OF POOR QUALITY

Multiprocessor Systems", *Performance'81: Proc. 8th Int'l Symp. Comp. Performance Model., Meas. and Eval.*, November 1981, pp. 201-211.

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

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

* Authors of the first two references are intentionally left out to keep anonymity as required in the review process but will be provided in the final form if accepted.

Acknowledgement

The authors thank Rick Butler and Milt Holt of NASA Langley Research Center for both technical and financial support.

ORIGINAL PAGE IS
OF POOR QUALITY

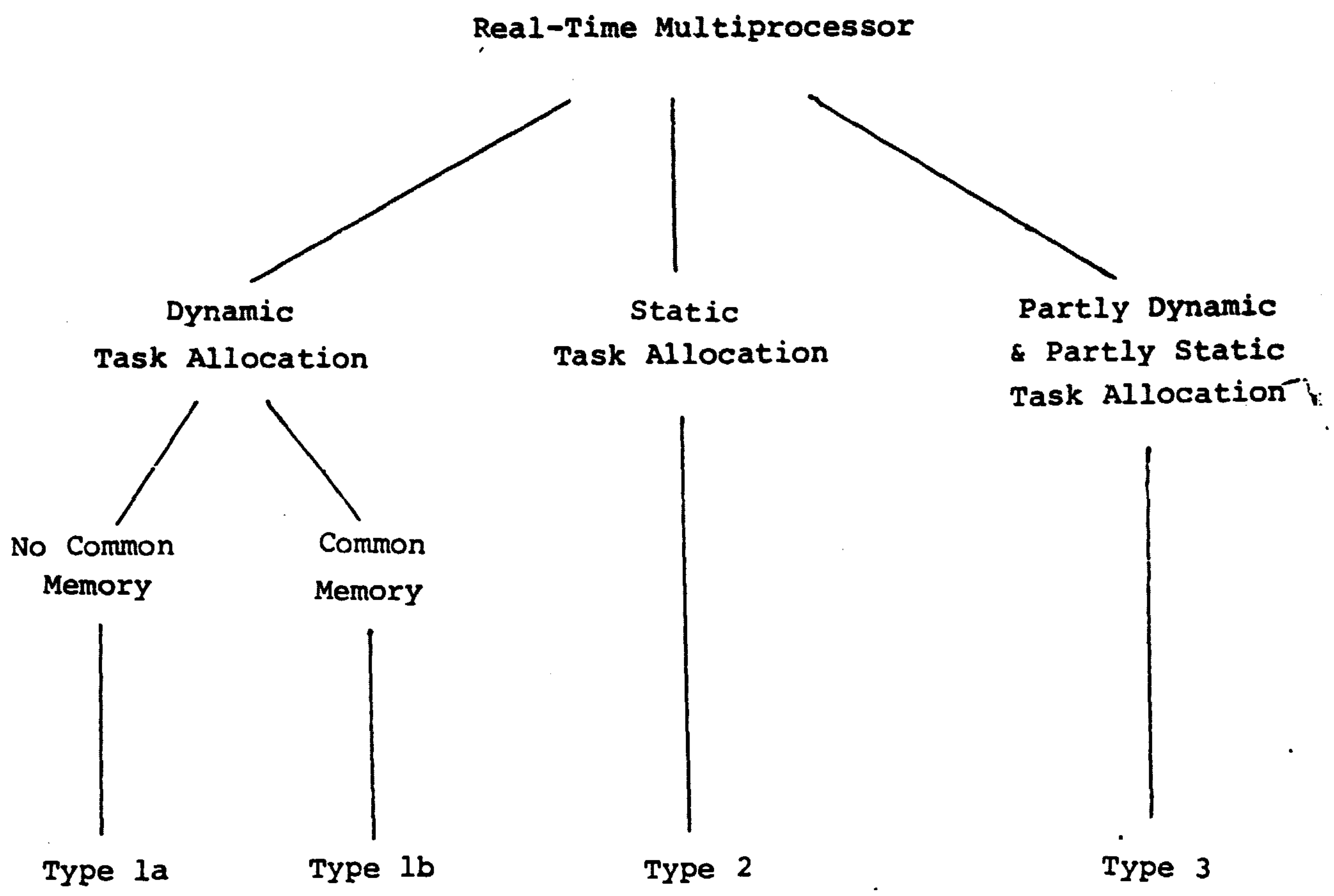


Figure 1 Logical Classification of Multiprocessor Controllers

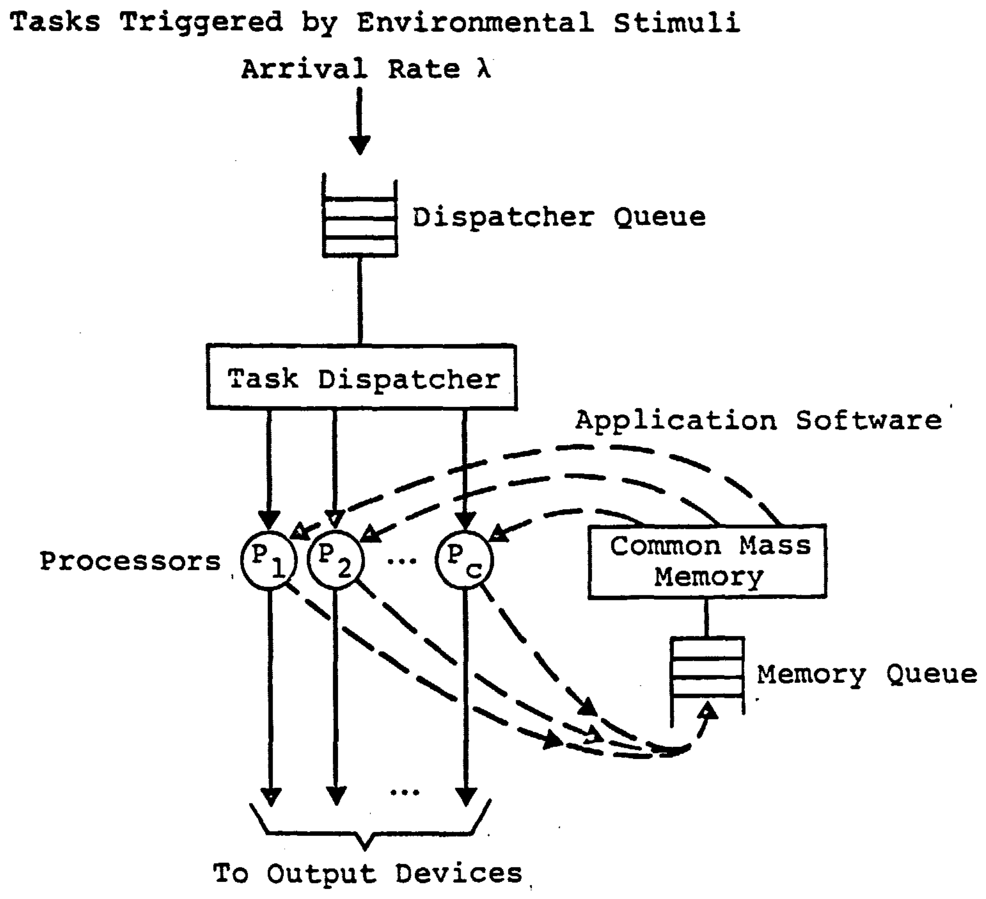


Figure 2 A Real-Time Multiprocessor Controller

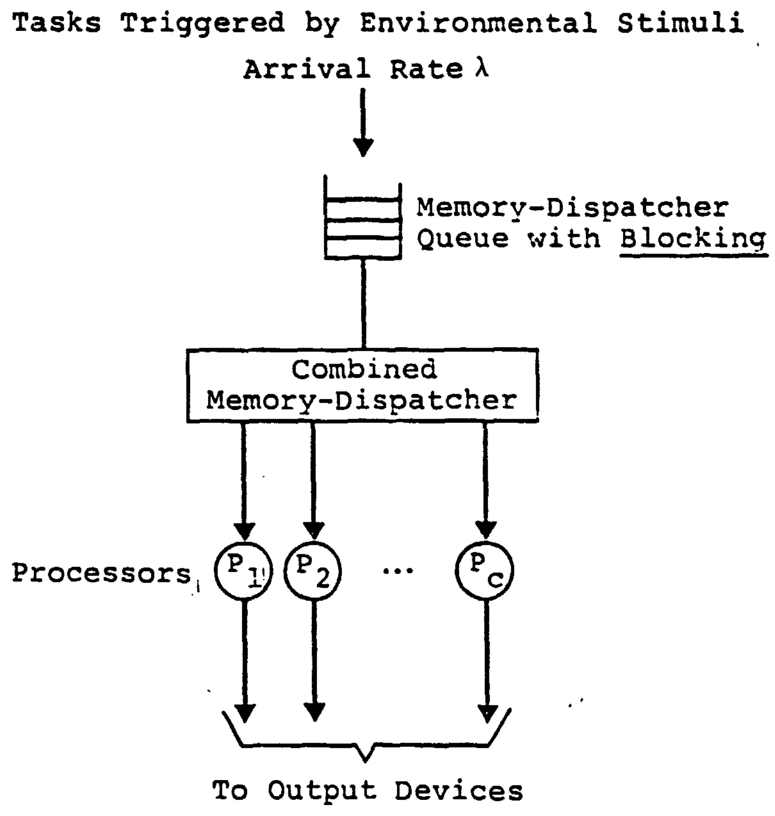

Tasks Triggered by Environmental Stimuli
Arrival Rate λ
Memory-Dispatcher Queue with Blocking
Combined Memory-Dispatcher
Processors
P1
P2
...
Pc
To Output Devices

Figure 3 Approximate Model of the Real-Time Multiprocessor Controller

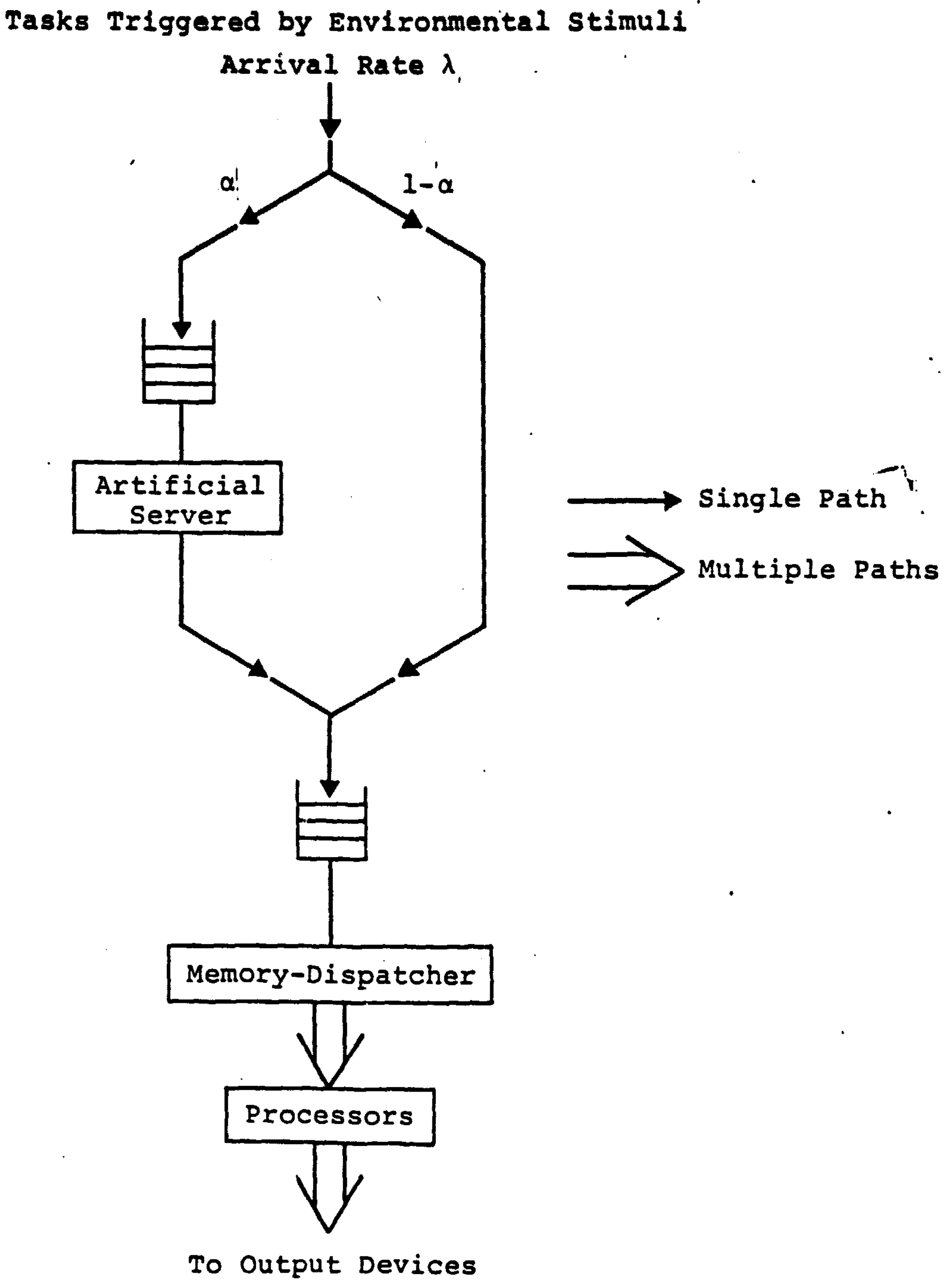


Figure 4 Task-Oriented Model

ORIGINAL PAGE IS OF POOR QUALITY

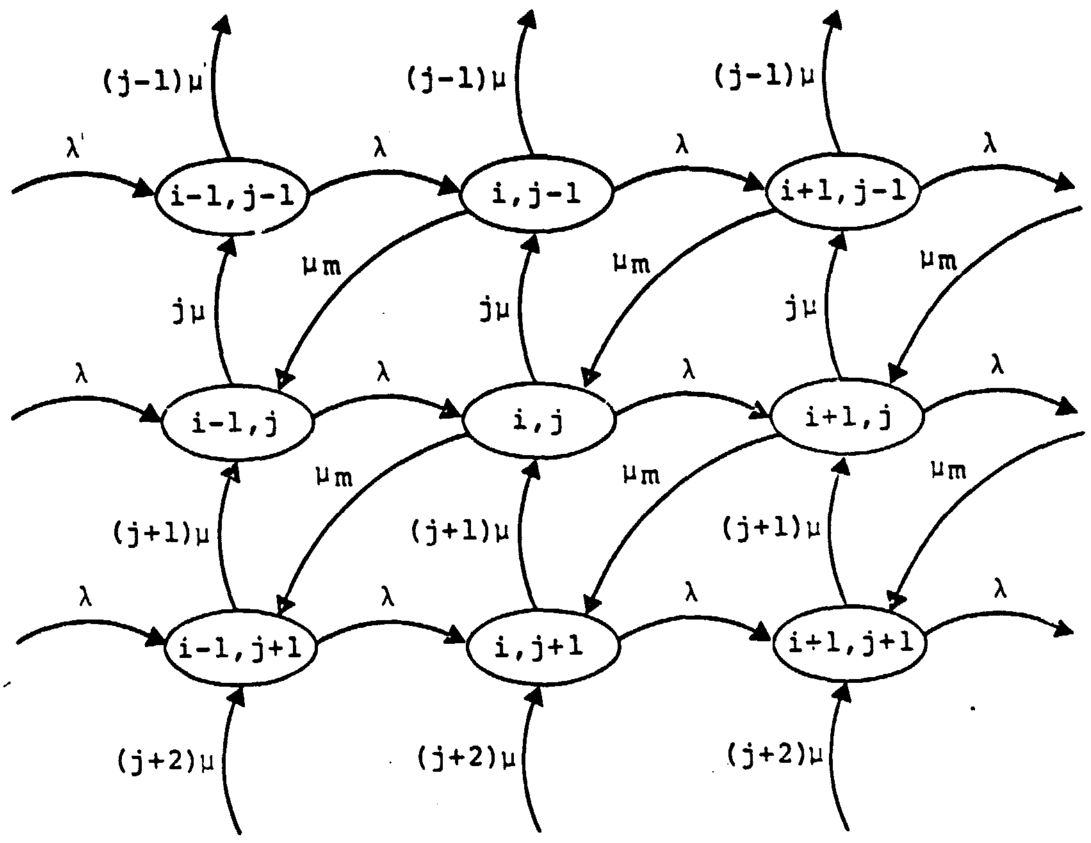
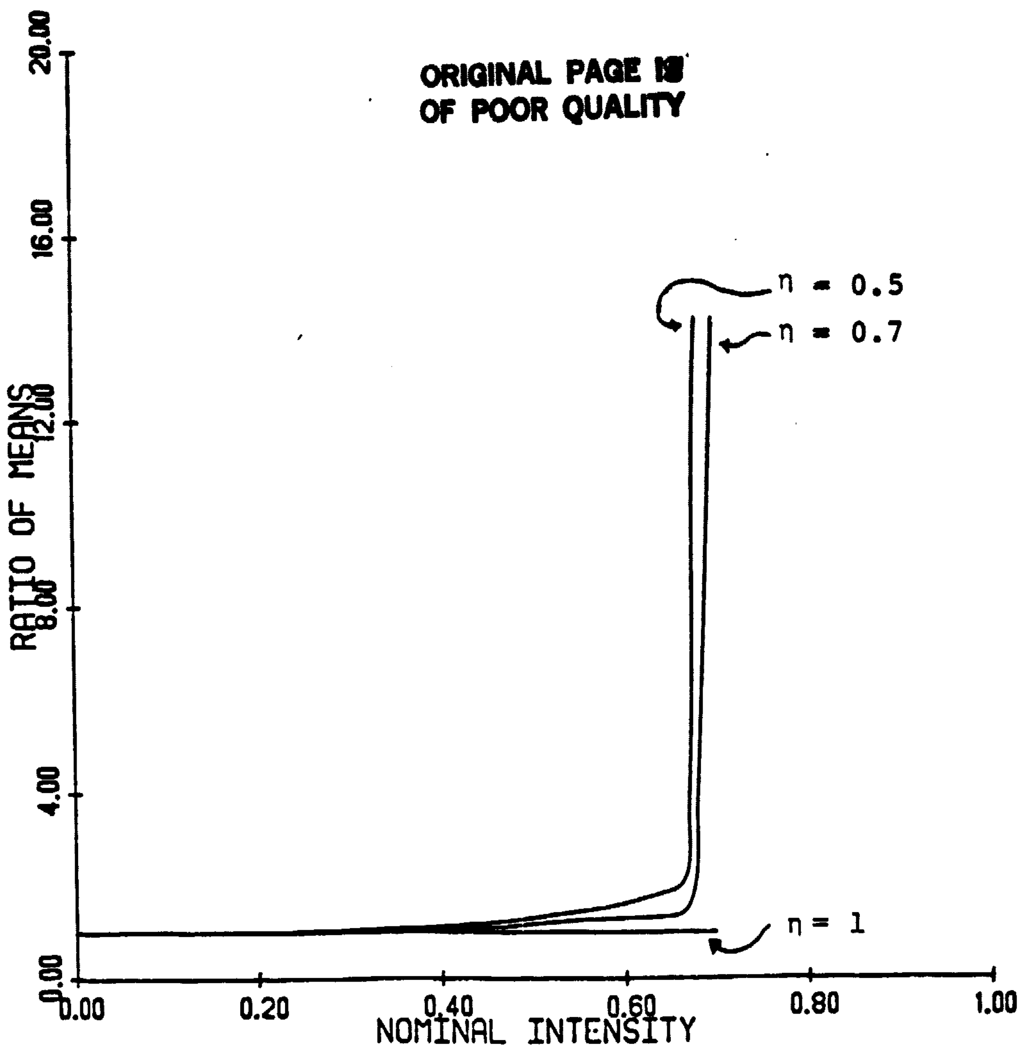


Figure 5 State Transition Diagram



$\mu_m = 5\mu$

$$\text{Ratio of Means} = \frac{\text{Mean Response Time when Weibull Parameter} = \eta}{\text{Mean Response Time when Weibull Parameter} = 1}$$

$$\text{Nominal Intensity} = \frac{\text{Mean Arrival Rate}}{\text{No. of Processors} \times \mu}$$

Figure 6 System Response with Weibull Service Distributions

ORIGINAL PAGE IS
OF POOR QUALITY

Table1. Comparison of Analytical and Simulation Mean Response Time

λ	Analytical	Simulation
2.0	73.4	71.7
4.0	76.6	72.3
6.0	79.4	73.3
8.0	82.0	74.4
10.0	84.9	75.6
12.0	88.0	77.9
14.0	91.3	80.8
16.0	95.2	83.6
18.0	99.8	86.5
20.0	105.3	92.9
22.0	112.1	98.4
24.0	120.6	108.2
26.0	131.9	121.4

$\mu = 20.0$, $\mu_m = 50.0$, $c=3$