Performance Modeling and Measurement of Real-Time Multiprocessors with Time-Shared Buses

MICHAEL H. WOODBURY, STUDENT MEMBER, IEEE, AND KANG G. SHIN, SENIOR MEMBER, IEEE

Abstract—**A closed queueing network model is constructed to address workload effects on computer performance for a highly reliable unibus multiprocessor used in real-time control. The queueing model consists of multiserver nodes and a nonpreemptive priority queue. Use of this model requires partitioning the workload into *task classes*. The time average steady-state solution of the queueing model directly produces useful results that are necessary in performance evaluation.**

The model is experimentally justified with the Fault-Tolerant Multiprocessor (FTMP) located at the NASA AIRLAB. Extensive experiments are performed on FTMP with a synthetic workload generator (SWG) to directly measure performance parameters, such as processor idle time, system bus contention, and task processing times. These measurements determine values for parameters in the queueing model. Experimental and analytic results are then compared.

Index Terms—**Closed queueing network, experimental measurement, fault-tolerant and real-time systems, performance modeling, synthetic workload, task classes, workload.**

I. Introduction

WORKLOAD representation is an important factor in the study of computer system performance, because system performance is directly related to the type of workload handled. Based on the level of abstraction, the *workload* of a computing system is the collection of data processing requirements presented to the system during a specified period of time. This paper presents the analytic development and experimental justification of a model to study workload effects on performance for a highly reliable multiprocessor with a time-shared (redundant) system bus used in real-time control.

Despite the growing use and importance of real-time systems, their exclusive analysis is an approach that has not been largely addressed in the literature relating to the evaluation of multiprocessor performance. Usually, a general-purpose multiprocessor is discussed, as in [1]–[3]. When describing workload effects on performance, this type of system becomes unreasonably complex. It appears that significant results can be obtained only if the analysis is narrowed to the structure of a real-time system and its workload.

Manuscript received October 21, 1985; revised May 12, 1987. This work was supported in part by NASA Grant 1-296 and NASA Training Grant NGT 23-005-801. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the view of NASA.

The authors are with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122.

IEEE Log Number 8716227.

Most other works deal with a general class of systems, rather than an actual system. This introduces an inexactness to the analysis and conclusions. Alternatively, we will emphasize a specific system, i.e., computers used for real-time control, and derive results using both analytic and experimental methods. Analytic modeling descriptions will necessarily be different than those used for simulation approaches.

A real-time computer system can be viewed as the combination of two dependent components: the *controlled process* and the *controlling computer* [4]. Thus, the development and justification of a performance model for this system should rely on the workload being modeled as well as the structure of the system handling the workload. Detailed analysis of real-time systems is also desired because of their increasing amount of critical applications, e.g., aircraft, spacecraft, and nuclear reactor control, where controlling computer failure would result in catastrophic losses. A failure could be the result of a physical malfunction or the system not reacting quickly enough. The latter subsumes the former and is termed *dynamic failure* in [4].

Many authors have presented synthetic workload designs for performance modeling, and usually rely on heuristic methods to provide an adequate workload description for a general class of computing systems [5]–[8]. Ferrari [9] has emphasized that a more systematic method is necessary, because of the fundamental correlation between workload and performance modeling. In this paper, a model to represent the workload effects on a specific real-time system is proposed that is amenable to the type of performance analysis desired for real-time applications.

The problem is not oversimplified by restricting it to a particular system. Because the analysis of a general-purpose system is necessarily inexplicit, one fails to obtain an in-depth understanding of a computer's operation. Focusing on a specific system, we are able to directly address areas relevant to that system and derive results that might otherwise be overlooked. Also, the architecture and operation of the system analyzed is typical of those used in many real-time applications.

The performance model developed is a closed queueing network representing the different states of the processors in a multiprocessor. It consists of multiserver nodes and a nonpreemptive priority queue. Vital factors can be directly determined, such as processor idle time, contention for the single bus, and which tasks most significantly effect system performance.

0018-9340/88/0200-0214$01.00

The structure and use of the queueing model is justified through extensive experiments on the Fault-Tolerant Multiprocessor (FTMP) [10], [11]. A synthetic workload generator (SWG) is used to create an environment where a real-time workload can be readily simulated, and hardware and software measurements can be conducted. The results of these measurements determine values for the parameters in the queueing model. The experiments directly measure the performance elements mentioned above and demonstrate the practicality of the queueing model. We then compare experimentally derived performance values with analytic results from the queueing model. Other results derived from the experiments, but not directly related to the model, are presented for completeness.

The rest of the paper is organized as follows. In Section II the specific architecture being addressed is elaborated, and the basic principles of operation are outlined. In Section III, the queueing model is discussed in detail. Analytic results of the model are presented in Section IV. Section V explains the structure and operation of FTMP and the use of the SWG. The experimental results are presented in Section VI, and their use to justify the queueing model is discussed in Section VII. The paper concludes with Section VIII.

II. System Architecture and Operation

The computer system addressed is a highly reliable unibus multiprocessor, typical of those used for critical real-time applications. Reliability is attained through redundancy at the component level. The general structure of such a system consists of four major components: *processing clusters, input/output links,* a *time-shared system bus,* and *system memory* (see Fig. 1).

A *processing cluster* operates on one task at a time and consists of one or more pairs of a processing unit and its local memory. Component redundancy is considered immaterial to the performance of the cluster, but does affect reliability and configuration aspects of system operation. It is assumed that all clusters are identical, i.e., they are constructed with the same elements and contain the same number of processor–memory pairs.

An *input/output link* enables data transmission between the system and external devices, e.g., sensors, actuators, displays, terminals, or other similar devices.

The *time-shared system bus* is for exchanging all data and control signals and interconnects the processing clusters, I/O links, and system memory. The bus may be redundant for reliability reasons, but only one cluster at a time *controls* the bus. Thus, a redundant system bus logically acts as a unibus.

Finally, there exists a single *system memory*, consisting of RAM's, accessible via the system bus. The system memory may be redundant with the restriction that only one memory location may be addressed at a time.

The control computer is analyzed at the system level, where the elements of concern are the components listed above and system tasks. Typically, a real-time system workload is a fixed group of tasks that are repeatedly executed at specific intervals. There is usually a group of short, frequently initiated tasks that monitor internal and external conditions. There are also tasks initiated less frequently that require more computation time. Tasks are partitioned into n disjoint sets called *task classes* and stored in system memory. A task class contains tasks with similar execution times which are required to repeatedly execute at the same frequency. More specifically, tasks from task class i are executed every r_i seconds. A *major frame* is defined as $\max_i r_i$. When a cluster is idle, i.e., performing no useful computations, it is considered to be executing an "idle task." This "task" is not considered part of the system workload.

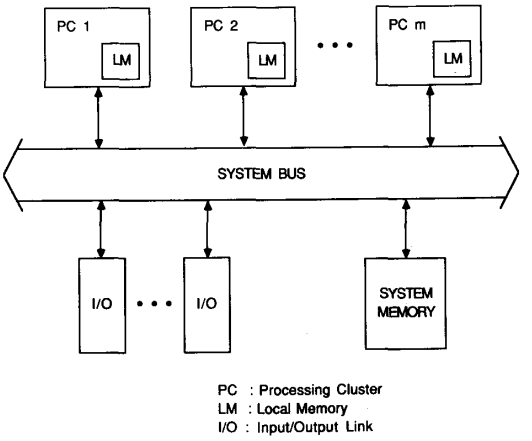


Fig. 1. System architecture.

The operating principles of this multiprocessor system can be described as follows. Each task class is assigned a priority to determine which processing cluster may access the system bus when there is contention for bus control. A cluster executing a task from class i has priority over another cluster, if the other cluster is working on a task from class j, where $1 \leq i < j \leq n$. Priority of clusters executing tasks from the same task class is determined by a first come first served (FCFS) policy. Task queues stored in system memory are kept for each task class.

An idle cluster preparing to process a task from class i must first gain bus control by waiting for bus inactivity, and proceeding to participate in a *polling sequence.* A polling sequence is a decentralized decision process where each cluster transmits its priority number over the system bus and individually determines if it has the highest priority. The polling sequence concludes with the highest priority cluster receiving bus control. (See [10] for a detailed example.) The controlling cluster next reads the task queue for class i and determines which task to execute by selecting the first task in the queue not assigned to a cluster. There are other mechanisms such as counters, queues, and interrupt timers to aid a cluster in determining which task class to request. It then reads the task code and *all* data necessary to internally execute the task, updates the task queue, and releases the bus. When a cluster completes a task, it will again request bus control, transmit results to relevant addresses, determine which task class to work on next, and proceed as before. To increase throughput by having the maximum number of clusters executing tasks, an idle cluster is given priority over nonidle clusters in obtaining bus control.

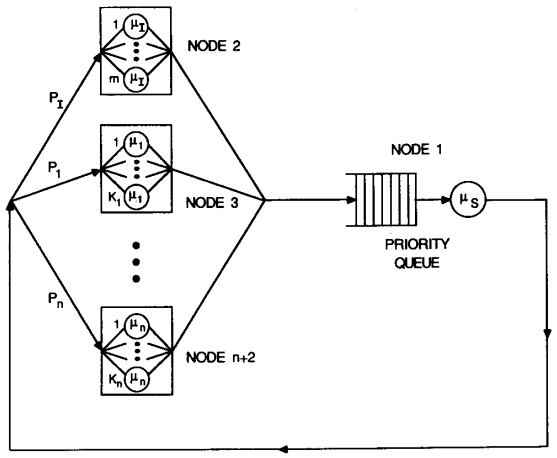


Fig. 2. Queueing model.

It should be stressed that while a cluster is internally executing a task it does not communicate with any other task or cluster. In other words, task communication is only at the beginning (for input) and at the end (for output) of execution. This operation is typical of real-time applications and is not a limitation of this work.

For a general-purpose computing system's workload, the best number of task classes and distribution of the tasks among these classes is difficult to determine [8]. The major problems of representing the workload in a general-purpose multiprocessor system model are 1) showing task interdependencies, 2) a nonstationary workload, i.e., tasks of one type might occur at different rates at different times, 3) the possibly unlimited number of tasks, and 4) component contention for concurrently executing tasks. Providing a model capable of representing all these features would be extremely difficult, if not impossible. Fortunately, when real-time applications on a unibus multiprocessor system are considered, these problems become relatively easier to address.

The workload of a real-time system is usually a fixed set of tasks to be executed in a prescribed order at regular intervals. As a result, the physical and logical interdependencies are more tractable. It also implies a stationarity among the relative frequencies of different tasks. Therefore, natural task classes can be formed and parameterized based on the frequency and internal processing requirements of each task.

III. Queueing Model Description

The computer system state is defined by the relative states of the processing clusters. Cluster states most relevant to system performance are when a cluster is 1) competing in a polling sequence, 2) transmitting or waiting to transmit on the system bus, 3) processing a task from task class i, or 4) idle. Fig. 2 is a closed queueing network relating these states. The actions of bus contention and the polling sequence are reduced to a single *nonpreemptive priority queue.* In a nonpreemptive priority queue, arriving customers move ahead of customers with lower priorities and behind those of equal or higher priority. In this manner, customers of the highest priority are served first on a FCFS basis. When a customer begins service, it is able to complete that service regardless of the priority of customers in the queue. These actions are exactly those performed by the polling sequence and system bus. The different task classes are explicitly parameterized by nodes 3 through $n + 2$.

Before describing the details of the model, it should be clarified that the parameters and node representations of this model *differ* from those of most conventional queueing models. Typically, the nodes of a queueing model represent servers of some type, e.g., processors or workers, and the tokens or markings moving about the model represent customers that desire service, e.g., programs or jobs. The model described here *reverses* the conventional meanings of node and token. In this model, a node represents a customer that needs service, and the associated exponential service rate describes the time required to complete that service. The tokens on the other hand represent servers, where all the servers are identical. Therefore, this model represents servers moving from customer to customer and performing the service requested by that customer. This is analogous to the concept of a "traveling serviceman." This unorthodox representation is used because it 1) simplifies the model, and 2) explicitly shows the state of each processing cluster.

It is the goal to determine the *time average steady-state probabilities* for the distribution of clusters among the different system states. Each of these probabilities is the long run probability averaged over time of an outside observer finding the system in a particular state. A favorable feature of highly reliable systems is that the time between cluster failures is much longer than the time it takes the system to reach steady state. Typical values for the mean time between failures (MTBF) of computing clusters are on the order of 10^3–10^4 h, whereas, steady state can be reached in a matter of minutes at most. Consequently, it is justifiable to assume that the system will reach steady state before a cluster fails. Once steady state is reached, a cluster may fail. At that point, the system with one less cluster reconfigures. It is reasonable to assume that *this* system will reach steady state before another failure occurs. Therefore, in the following analysis, we will assume that no cluster fails, and the number of clusters remains constant.

A token in the queueing model represents a single cluster. Let m equal the number of homogeneous clusters in the system. There are $n + 2$ nodes, where n is the number of task classes. The number of tasks in task class i is denoted by n_i. Therefore, the maximum number of cluster executing tasks from class i is $K_i = \min(m, n_i)$. As mentioned earlier, tasks with similar traits are grouped into the same task class. Thus, all tasks in a class are assumed to have the same distribution of internal processing time. The processing time distribution is an exponentially distributed random variable, because execution times are usually data dependent and random in nature.

Each node is described below.

NODE 1 represents transmission activity and contention of the system bus, where transmission activity includes system memory and I/O link access time. It consists

of a nonpreemptive priority queue and a transmission server. A token at this node represents a cluster that is either waiting to transmit on the system bus or actively transmitting. The parameter μ_s describes the exponential transmission rate of a cluster, i.e., $1/\mu_s$ is the average transmission duration.

NODE 2 represents clusters that are idle, performing no useful computations. It is a multiserver node with m servers. The sojourn time in this state is assumed to be exponentially distributed with rate μ_I. Tokens leave the node at a rate of $k\mu_I$, where k is the number of tokens being served by the node.

NODES 3 through $n + 2$ represent the n different task classes. Node $i + 2$ is a multiserver node with K_i servers corresponding to the processing activity of tasks in task class i. The parameter μ_i is the processing rate of a task of class i. Typically, $\mu_i \geq \mu_j$ when $i < j$. Tokens leave these nodes at a rate of $k\mu_i$, where k is the number of tokens being served by the particular node.

The remaining model parameters are the branch probabilities. When a cluster completes a transmission, it either drops into the idle state or continues processing. P_I, the probability that the next state is the idle state, is determined by the triggering frequency of all tasks and their execution times, and is equivalent to the expected proportion of idle time during a major frame. A cluster enters the processing state of task class i with probability P_i, where $P_I + \Sigma_{i=1}^{n} P_i = 1$. Typically, $P_i \geq P_j$ when $i < j$. The derivation of the P_i's requires information on the number of tasks in each task class and their relative frequencies. Task classes with high frequencies or a large number of tasks with respect to the other task classes will have a higher probability of occurring.

IV. Queueing Model Solutions

A common method for determining time average steady-state probabilities of a queueing model is to convert the model to a continuous parameter Markov chain [12]. The following definitions are for the construction of a Markov chain.

Definition 1: A *cluster state* is a pair (c_i, n_i), where $c_i \in \{1, 2, \cdots, m\}$ labels a particular processing cluster, and $n_i \in \{1, 2, \cdots, n + 2\}$ is the node where the token representing the cluster is located. There are $m(n + 2)$ cluster states.

Definition 2: A *system state* is an m-tuple $(s_1, s_2, \cdots, s_m) \in S_1 \times S_2 \times \cdots \times S_m$, where S_i is the set of cluster states whose first element is c_i. There is a maximum of $(n + 2)^m$ system states.

A system state example for a system with three clusters and three task classes is ((1, 1), (2, 3), (3, 1)). This represents the configuration when clusters 1 and 3 are waiting for the system bus or currently transmitting, and cluster 2 is processing a task from task class 1.

A system state contains more information than necessary. Since the clusters are homogeneous, the number of tokens present at a node determines how fast tasks will be completed or delayed. This motivates the following definition.

Definition 3: A *reduced system state* is the $(n + 2)$-tuple $(a_1, a_2, \cdots, a_{n+2})$, where $a_i \in \{0, 1, \cdots, m\}$ is the number of tokens at node i. There are J reduced system states, where J is defined below.

First note that a mapping Φ from a system state to a reduced system state can be defined as follows: $\Phi(s_1, s_2, \cdots, s_m) = (a_1, a_2, \cdots, a_{n+2})$, where a_i = number of s_j's ($j = 1, \cdots, m$) whose second component is i. Referring to the example above, the system state ((1, 1), (2, 3), (3, 1)) is represented by the reduced system state (2, 0, 1, 0, 0). It should also be noted that system states ((1, 1), (2, 3), (3, 1)), ((1, 1), (2, 1), (3, 3)), and ((1, 3), (2, 1), (3, 1)) are all represented by the same reduced system state.

The reduced system states are used as the Markov chain states, where state transitions are defined by the relevant service rates of each of the nodes in the closed queueing network. As stated in [13], a closed queueing model, where all the nodes are homogeneous with K customers and N nodes has $J = \binom{N+K-1}{N-1}$ states in its Markov chain representation. For our model, $K = m$ and $N = n + 2$. Therefore, the maximum number of reduced system states in the closed queueing model is $J = \binom{n+m+1}{n+1}$. The minimum number of reduced system states occurs when $K_i = 1$, $i = 1, \cdots, n$. In this case, $J = \Sigma_{i=0}^{L} \binom{n}{i}(m - i + 1)$, where $L = \min(m, n)$. Various maximum and minimum values when $m = 3$ are shown in Table I.

From the Markov chain, a $J \times J$ transition rate matrix A can be formed, whose structure is the coefficient matrix for the set of linear equations

$$P(k_1, k_2, \cdots, k_{n+2}) \sum_{i=1}^{n+2} \delta_{k_i - 1}\alpha_i(k_i)\mu_i$$

$$= \sum_{i=1}^{n+2}\sum_{j=1}^{n+2} \delta_{k_j - 1}\alpha_i(k_i + 1)\mu_i \cdot r_{ij}(k_1, \cdots, k_j - 1, \cdots, k_i + 1, \cdots, k_{n+2}) \cdot P(k_1, \cdots, k_j - 1, \cdots, k_i + 1, \cdots, k_{n+2}),$$

where

k_i = number of tokens at node i,
s_i = number of servers at node i,
μ_i = service rate at node i,

$$\delta_k = \begin{cases} 1 & k \geq 0 \\ 0 & k < 0 \end{cases},$$

$$\alpha_i(k_i) = \begin{cases} k_i & k_i \leq s_i \\ s_i & k_i \geq s_i \end{cases}, \text{ and}$$

$r_{ij}(k_1, k_2, \cdots, k_{n+2})$

= Prob[when in state $(k_1, k_2, \cdots, k_{n+2})$ a token that completes service at node i will next enter node j].

Solving the matrix equation $\boldsymbol{Ax} = \boldsymbol{0}$ determines the time average steady-state probabilities for each state in the Markov chain, where $\boldsymbol{x} = (x_1, x_2, \cdots, x_J)^T$, and x_i represents the time average steady-state probability of the system being in state i. A nontrivial solution results when the probability constraint

TABLE I
NUMBER OF MARKOV CHAIN STATES J FOR THREE CLUSTERS

No. of Task Classes (n)	Number of States	
	Maximum	Minimum
1	10	7
2	20	12
3	35	20
4	56	32
5	84	49
6	120	72
7	165	102
8	220	140
9	286	187
10	364	244

Maximum number when $K_i = m$, $i=1,\ldots,n$.
Minimum number when $K_i = 1$, $i=1,\ldots,n$.

$\Sigma_{i=1}^{J} x_i = 1$ is considered. The existence of such a solution is guaranteed because we have constructed a finite state, irreducible, and recurrent, i.e., ergodic, Markov chain.[1] Since a token can move from one node to any other node directly or through some intermediate nodes, *and* there exists a nonzero probability that a token leaving a node will return to that node, the Markov chain is indeed irreducible and recurrent.

Two useful results can be immediately obtained from the steady-state probabilities. The probability that a cluster is idle is the sum of the probabilities for each of the Markov chain states that represent having one or more clusters at node 2. When more than one cluster is at node 1, there is a cluster waiting to obtain bus control. Thus, the probability of having bus contention is the sum or the probabilities for each of the states that represent having more than one cluster at node 1. These two results are necessary to produce a performance measure of any type.

A third result is how long a cluster executing a task from task class i would have to wait, on the average, if there is contention for the system bus. This information is necessary to determine the probability of a task meeting its deadline. It is shown in [14] that the average queueing time for customers of priority class i in a nonpreemptive priority queue is

$$W_i = \frac{\frac{1}{2}\sum_{j=1}^{k} \lambda_j y_j}{\left(1-\sum_{j=1}^{i-1}\frac{\lambda_j}{\mu_j}\right)\left(1-\sum_{j=1}^{i}\frac{\lambda_j}{\mu_j}\right)},$$

where

k = the number of priority classes,
λ_j = the mean arrival rate of a customer of class j,
μ_j = the mean service rate of a customer of class j, and
y_j = the second moment of the service-time distribution for customers of class j.

The mean queueing time of all customers is $W_q = \Sigma_{i=1}^{k} \alpha_i W_i$, where $\alpha_i = \lambda_i / \Sigma_{j=1}^{k} \lambda_j$.

To simplify notation, we classify idle clusters as executing priority class 0 "idle tasks," since idle clusters are given priority to gain bus control over nonidle clusters. Tokens requesting service at node 1 are assumed to have the same distribution of service time. Therefore, $k = n$, $\mu_i = \mu_S$ for all i, and $y_i = 1/\mu_S^2$ for all i. The average queueing time for a cluster working on a task from task class i (note the inclusion of priority class 0) now becomes

$$W_i = \frac{\sum_{j=0}^{n} \lambda_j}{\left(\mu_S - \sum_{j=0}^{i-1}\lambda_j\right)\left(\mu_S - \sum_{j=0}^{i}\lambda_j\right)}, \quad i=0, 1, \cdots, n.$$

W_i is the average *queueing time* only. The total average waiting time of a cluster working on a task of task class i, W_i^{tot}, is the sum of the average queueing time *and* the service time, i.e.,

$$W_i^{\text{tot}} = W_i + \frac{1}{\mu_S}.$$

Deriving W_i requires values for each of the λ_i's. Let $p(s)$ equal the time average steady-state probability of being in state s of the Markov chain and S_{ij} be the set of states representing j clusters at node i. The rest of the clusters, if any, may be at any of the remaining nodes. Then, $\lambda_i = \mu_i \Sigma_{j=1}^{m} \Sigma_{s \in S_{i+2,j}} j \cdot p(s)$.

V. Experimental System Description

To illustrate the application of the performance model, FTMP is analyzed. FTMP is a real-time multiprocessor with a hardware and software structure similar to that assumed for the queueing model. With parameter values derived through experiments on FTMP, the effects of varying the workload structure are illustrated and analyzed.

FTMP is a highly reliable multiprocessor installed at the NASA AIRLAB intended for real-time control of commercial aircraft of the next decade. Because disastrous effects could occur if this computer should fail while in operation, NASA specified the probability of system failure to be less than 10^{-9} for a 10 h flight. This obviously calls for extremely rigid performance criteria.

The FTMP architecture, from a programmer's view, consists of three triads, system memory, input/output links, system clock, system control registers, and a single time-shared system bus [10]. A triad consists of three pairs of a processor and its local memory. Every component of the system is redundant and is either an active, standby, or shadow component. The three processors in a triad are operating in tight synchrony and should receive identical data under fault-free conditions. When there is a disagreement, an error is considered to have occurred, but masked, and task execution continues. The error is recorded in an error latch for later identification of the faulty component. The interested reader is referred to [10] for a complete architectural description of FTMP.

The operating workload for FTMP is the Executive Software and Applications Software [11]. Most workload tasks are dispatched at regular intervals to handle repetitive applications such as flight control, configuration control, fault detection,

[1] A unique time average steady-state solution exists for this type of Markov chain.

recovery, and system displays. Based on the application, FTMP developers determined that tasks should be executed at three different frequencies. They termed the three rate groups $R1$, $R3$, and $R4$ with respective nominal frequencies of 3.125, 12.5, and 25 Hz. Tasks executing at a particular frequency are given priority to access system components over tasks initiated at lower frequencies, implying $R4$ rate group tasks have priority for bus access over $R3$ tasks, etc.

Extensive experiments were conducted on FTMP with a synthetic workload generator (SWG) developed by researchers at Carnegie-Mellon University [15]. The SWG provides an FTMP experimenter with a variety of workloads. By constructing different synthetic workloads, performance characteristics of FTMP can be analyzed. An experimenter is able to construct a workload consisting of the executive software, existing application tasks, i.e., those delivered with FTMP, and user-generated synthetic tasks. The executive software controls the hardware and software resources. It is responsible for presenting the user with a virtual machine such that hardware redundancy, redundancy management, and the timely execution of application tasks are transparent to the user. Therefore, the executive software must be part of all synthetic workloads.

There are four application tasks on the system and up to nine synthetic tasks can be added, limited by the availability of system memory. The existing application tasks are 1) TIME, which updates the register holding the current time, 2) DISPLAY, which updates the display terminal indicating the status of FTMP, 3) READALL, which reads and interprets the fault latches, and 4) SCC, the system configuration controller. During fault-free behavior, none of these tasks are essential and may be excluded from the workload. Therefore, a workload consisting exclusively of user-generated synthetic tasks and the executive software can be constructed.

All synthetic tasks have the structure shown below.

```
TASK_i
Begin
  Read(P_i, Q_i, R_i, S_i, T_i);
  Store(Time);
  For X = 1 to P_i do
    Read External Sensor Data;
  Store(Time);
  For X = 1 to Q_i do
    Read Internal Data;
  Store(Time);
  For X = 1 to R_i do
    Process Data (A = B + C);
  Store(Time);
  For X = 1 to S_i do
    Write External Actuator Commands;
  Store(Time);
  For X = 1 to T_i do
    Write Internal Data Results;
  Store(Time);
End;
```

The parameters P_i, Q_i, R_i, S_i, and T_i are user-supplied values. Even though they are read each time the task is executed, they are fixed when the workload is constructed. Each task may have different parameter values, but the values for a particular task remain the same for all iterations.

This structure is typical of most real-time application tasks, where I/O is permitted only at the beginning and end of a task. The read and write instructions in the synthetic tasks are dummy system operations to fixed external and internal addresses. Therefore, no relevant data are transmitted. It is the action of the operation that is being characterized. By adjusting the parameters P_i, Q_i, S_i, and T_i, the I/O requirements of any task can be closely modeled.

The modeling of the data processing portion of a task (i.e., the portion characterized by the parameter R_i above) is less representative of an actual task. Generally, a specific application task may have different computational requirements each time it is invoked, because different inputs place different demands on the task. The synthetic task shown above is not data dependent. The processing portion consists of executing a representative instruction (i.e., the add instruction) R_i times. Thus, the computation time required is constant for all iterations of the task. However, this is an adequate representation when a worst case analysis is desired.

In each synthetic task, system clock values are stored for timing analysis.[2] The time between specific task events can be measured. For example, 1) the total task execution time, 2) the time between iterations of a task, 3) task switching time, 4) task startup time, 5) the time to perform I/O, and 6) the idle time of processors. See [15] for the justification of these application level measurements and their calibration.

VI. Experimental Results

Synthetic workloads were constructed to measure the performance of FTMP. With FTMP executing these workloads, hardware and software measurements were made, demonstrating that all the parameters of the queueing model can be experimentally derived.

A. Experimental Workloads

Two synthetic workloads, consisting of the executive software and synthetic tasks only, are used for the experiments. Synthetic tasks are the only tasks where software timing measurements can be made.

The workloads are outlined in Table II. The number of tasks in each rate group are shown along with the parameter used for all tasks in that group. The number of tasks for each group is intended to show that higher frequency rate groups usually have more tasks to execute. Because a maximum of three synthetic tasks can be added to each rate group, this is the most variation that can be demonstrated. When constructing the workload, all the parameters of a task were set to the same value, i.e., $P_i = Q_i = R_i = S_i = T_i =$ Parameter value in the table. Making them all equal was an arbitrary choice and does not affect the complexity of the problem. We could have equivalently chosen values that were all different.

The parameter values were selected to reflect the relative

[2] System clock reads require the same action as memory read. This violates our assumption that I/O can only be done at the beginning and end of a task and will be taken into account when analyzing the experimental results.

TABLE II
SYNTHETIC WORKLOAD PARAMETERS

	R1 Tasks		R3 Tasks		R4 Tasks	
	No.	Parameter	No.	Parameter	No.	Parameter
Workload 1	1	8	2	4	3	1
Workload 2	1	80	2	40	3	10

TABLE III
EXPERIMENTALLY DERIVED PROBABILITIES

	Workload 1		Workload 2		Existing System Workload	
Prob. Busy Bus at Poll Request	0.51	(2033)	0.49	(2000)	0.43	(3000)
Prob. Free Bus at Poll Request	0.49	(2033)	0.51	(2000)	0.57	(3000)
Prob. Succ. First Poll	0.95	(1789)	0.95	(1805)	0.95	(2794)
Prob. Fail First Poll	0.05	(1789)	0.05	(1085)	0.05	(2794)
Prob. Succ. Second Poll	0.99	(87)	1.0	(86)	1.0	(140)
Prob. Fail Second Poll	0.01	(87)	0.0	(86)	0.0	(140)

frequencies of the rate groups. For every eight executions of an $R4$ task, an $R1$ task is executed once. Therefore, the parameters of $R1$ tasks are set to eight times the parameters of $R4$ tasks. The $R3$ tasks are set in a similar manner. These values also reflect how lower frequency tasks have longer execution times.

For all experiments and workloads, FTMP was configured to have three fault-free triads operating. This is the initial state of FTMP and demonstrates the maximum contention for system components.

B. Hardware Measurements

Using a Tektronics DAS 9100 logic analyzer, hardware measurements were made for both synthetic workloads and the existing workload. Measurements were performed by sampling five pin locations within one of the processing regions: 1) poll request line, 2) idle bus indicator line, 3) polling sequence transmit line, 4) polling sequence receive line, and 5) the 1 MHz system clock used for polling sequence transmission. Results are shown in Tables III and IV. The numbers in parenthesis after the values in Table III are the number of samples used to derive the value. For Table IV values, one unit is 0.5 μs. The parenthesized numbers in this table, and all succeeding tables with timing measurements, is the 99 percent confidence interval based on the number of samples taken for each entry. This means that with 99 percent confidence, the true mean is in the range of the measured mean plus or minus the value given.

The average bus transaction duration is one fifth the time between bus requests, but the system bus is busy almost 50 percent of the time when a bus request is made. This is explained by the fact that three triads are operating simultaneously, and there is a variation in both the transaction duration and time between requests. Another point of interest concerns the probability of succeeding in a second poll, if a triad fails the first poll. It was never observed that a triad lost two successive polling sequences, except for one case noted in Workload 1.

TABLE IV
EXPERIMENTALLY DERIVED TIME MEASUREMENTS

	Workload 1	Workload 2	Existing System Workload
Time Between Bus Requests	Ave. 337.47 (±0.06) Var. 1548.33 SD. 39.35	336.50 (±0.04) 746.59 27.32	337.67 (±0.04) 1935.80 44.00
Time From Bus Req. To Start Of Poll (Idle Bus)	Ave. 4.83 (±0.00) Var. 0.56 SD. 0.75	4.57 (±0.00) 0.42 0.65	4.75 (±0.00) 0.54 0.74
Time From Bus Req. To Start Of Poll (Busy Bus)	Ave. 48.33 (±0.16) Var. 6104.90 SD. 78.13	46.74 (±0.15) 6711.92 81.93	44.00 (±0.19) 6562.20 81.01
Idle Time If Lost First Poll	Ave. 64.62 (±0.91) Var. 952.95 SD. 30.87	58.47 (±0.94) 979.50 31.30	76.75 (±1.07) 2960.22 54.41
Duration of Bus Transaction (Incl. Poll)	Ave. 69.53 (±0.14) Var. 6867.14 SD. 82.87	57.82 (±0.10) 3619.77 60.16	62.30 (±0.08) 5184.05 72.00

C. Software Measurements

For the purpose of justifying our queueing model, two software timing measurements are necessary: triad (processor) idle time and task execution time. When a triad becomes idle in FTMP, it enters an infinite loop performing null operations. The loop is exited by a timer interrupt triggering the beginning of a *time frame* for a rate group. All tasks in a rate group are executed exactly once in their respective time frame. The major frame for FTMP is 0.32 s, the time frame for the $R1$ rate group.

During a major frame, exactly one iteration of each $R1$ task, four iterations of each $R3$ task, and eight iterations of each $R4$ task are executed. Because a triad can leave the idle state at any time, it is difficult to directly measure the length of time a triad is idle. However, this can be measured indirectly. The effective length of a major frame is equivalent to the length of eight $R4$ time frames. The beginning of each $R4$ time frame is recorded by the first $R4$ task executed by that triad in that frame. Using these clock values, the time when each major frame begins and ends can be measured. The triad idle time is determined by subtracting the execution times of all the tasks executed by that triad during the major frame.

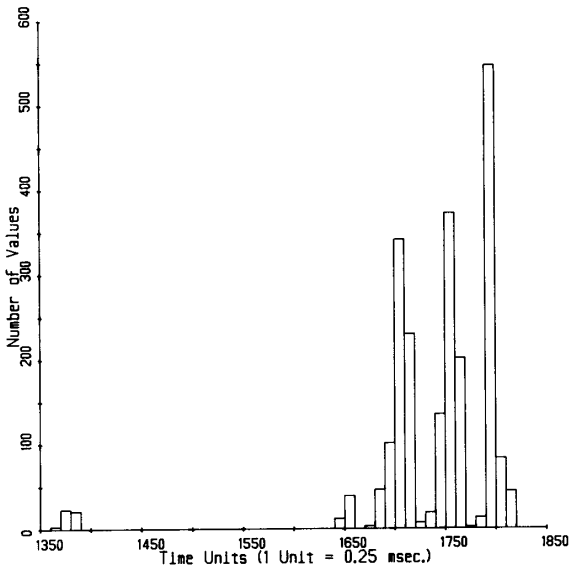
Using this approach, the idle times of each triad were measured. The results are shown in Table V and Fig. 3 for each of the synthetic workloads.[3] One unit is 0.25 ms in Table V. There is a significant decrease in idle time, 30.5 percent, when a more intensive workload is introduced, as expected. It was observed that the variation in idle time is greater for Workload 2, because the range of task execution times for this workload is greater.

The second software measurement was the execution times of synthetic tasks. FTMP was configured into a single operating triad to ensure no contention for the system bus. The results of the task execution times are shown in Table VI, where task Tn is the synthetic task with parameter n. Again, $P_i = Q_i = R_i = S_i = T_i =$ Parameter value and one unit is

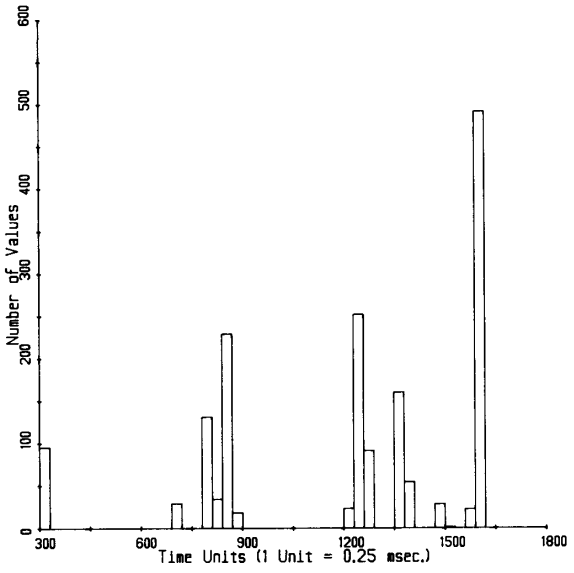
[3] Obviously, these values cannot be determined for the existing system workload.

TABLE V
EXPERIMENTALLY DERIVED IDLE TIMES FOR CLUSTERS

		Workload 1	Workload 2
Idle Time Of A Cluster	Ave.	1741.53 (±0.08)	1210.09 (±0.60)
	Var.	4431.76	141450.42
	SD.	66.57	376.10



(a)



(b)

Fig. 3. (a) Triad idle time for Workload 1. (b) Triad idle time for Workload 2.

0.25 ms. The consistency of the values, indicated by the low variance, demonstrates that the synthetic tasks do have constant execution times. The execution time for a task is found to be a linear function of the parameters. A linear regression analysis on the mean execution times for tasks $T0$, $T1$, $\cdots$, $T10$ produces

$$\text{Task Execution Time} = 0.67 \times \text{Parameter} + 0.96 \text{ ms.}$$

Similarly, for tasks $T0$, $T10$, $\cdots$, $T100$ we find

$$\text{Task Execution Time} = 0.67 \times \text{Parameter} + 0.97 \text{ ms.}$$

Therefore, we are able to predict the execution times for any synthetic task where all the parameters are equal.

TABLE VI
EXPERIMENTALLY DERIVED SYNTHETIC TASK EXECUTION TIMES

Task	Parameter	Average	Variance	Std. Dev.
T0	0	3.85 (±0)	0.13	0.36
T1	1	6.52 (±0)	0.25	0.50
T2	2	9.24 (±0)	0.18	0.42
T3	3	11.91 (±0)	0.08	0.29
T4	4	14.56 (±0)	0.25	0.50
T5	5	17.26 (±0)	0.19	0.44
T6	6	19.97 (±0)	0.02	0.16
T7	7	22.68 (±0)	0.22	0.47
T8	8	25.34 (±0)	0.22	0.47
T9	9	28.05 (±0)	0.05	0.22
T10	10	30.75 (±0)	0.19	0.43
T20	20	57.71 (±0)	0.21	0.46
T30	30	84.52 (±0)	0.25	0.50
T40	40	111.41 (±0)	0.24	0.49
T50	50	138.34 (±0)	0.22	0.47
T60	60	165.24 (±0)	0.18	0.43
T70	70	192.60 (±0.02)	12.59	3.55
T80	80	219.04 (±0)	0.35	0.59
T90	90	245.93 (±0)	0.11	0.33
T100	100	272.84 (±0)	0.40	0.63

VII. Comparison of Analytic and Experimental Results

The architecture and system operation of FTMP is a structure that can be modeled by our queueing network. The triads in FTMP can be represented as three clusters and each of the rate groups as a task class. Task class 1 is rate group $R4$, because of the relative priorities of the rate groups and task classes. Likewise, task class 2 is $R3$, and task class 3 is $R1$. Therefore, in the queueing model representation, there are five nodes and three tokens representing clusters, i.e., $n = 3$ and $m = 3$. There is some dependence when tasks from a rate group are executed based on the state of tasks in a higher priority rate group. However, the model can handle this by increasing the number of task classes. For the purpose of illustration, these dependencies are assumed to be negligible.

Both synthetic workloads have three tasks in task class 1, two in class 2, and one in class 1. Therefore, there are $J = 29$ states in the Markov chain representation of FTMP. These states and their respective reduced system states are given in Table VII. Experimental values for parameters were determined from the measurements described in Section VI and are outlined in Table VIII. The values for μ_S and μ_I are the inverses of the mean times measured. The experimental values for μ_1, μ_2, and μ_3 have been adjusted, as discussed below.

In the construction of our queueing model we assumed that all the data input for the task was done once at the beginning of the task, and output was done once at the end of the task. Because the SWG was designed for other types of experiments, we did not have the luxury of constructing tasks exactly of this type. As a result, synthetic tasks had many more I/O operations, e.g., task $T1$ had 15 read/write operations and task $T80$ had 252. To compensate for this, the internal data

TABLE VII
MARKOV STATE DESCRIPTIONS AND STEADY-STATE PROBABILITIES

Markov States		Computed Steady State Prob.	
State	Reduced System State	Workload 1	Workload 2
1	(3, 0, 0, 0, 0)	0.152	0.152
2	(2, 1, 0, 0, 0)	0.286	0.241
3	(2, 0, 1, 0, 0)	0.014	0.045
4	(2, 0, 0, 1, 0)	0.010	0.023
5	(2, 0, 0, 0, 1)	0.002	0.003
6	(1, 2, 0, 0, 0)	0.269	0.190
7	(1, 1, 1, 0, 0)	0.027	0.071
8	(1, 1, 0, 1, 0)	0.019	0.037
9	(1, 1, 0, 0, 1)	0.003	0.005
10	(1, 0, 2, 0, 0)	0.001	0.006
11	(1, 0, 1, 1, 0)	0.001	0.008
12	(1, 0, 1, 0, 1)	0	0.001
13	(1, 0, 0, 2, 0)	0	0.001
14	(1, 0, 0, 1, 1)	0	0.001
15	(0, 3, 0, 0, 0)	0.168	0.099
16	(0, 2, 1, 0, 0)	0.025	0.057
17	(0, 2, 0, 1, 0)	0.017	0.028
18	(0, 2, 0, 0, 1)	0.003	0.004
19	(0, 1, 2, 0, 0)	0.001	0.009
20	(0, 1, 1, 1, 0)	0.002	0.013
21	(0, 1, 1, 0, 1)	0	0.002
22	(0, 1, 0, 2, 0)	0	0.002
23	(0, 1, 0, 1, 1)	0	0.001
24	(0, 0, 3, 0, 0)	0	0
25	(0, 0, 2, 1, 0)	0	0.001
26	(0, 0, 2, 0, 1)	0	0
27	(0, 0, 1, 2, 0)	0	0
28	(0, 0, 1, 1, 1)	0	0
29	(0, 0, 0, 2, 1)	0	0

TABLE VIII
EXPERIMENTALLY DERIVED QUEUEING MODEL PARAMETERS

$\mu_S = 28765$	$\mu_I = 2.2968$	$P_I = 0.00015$
$\mu_1 = 224140$	$\mu_2 = 107760$	$\mu_3 = 88362$

(a) Workload 1

$\mu_S = 34590$	$\mu_I = 3.3055$	$P_I = 0.00015$
$\mu_1 = 84483$	$\mu_2 = 56035$	$\mu_3 = 53879$

(b) Workload 2

processing time of a task (i.e., the step parameterized by R_i) was divided into segments to create tasks with only two I/O operations. Since task $T1$ had 13 more I/O operations than modeled, the internal processing time was divided by 13. Task $T1$ had one execution of the step $A = B + C$ to represent the internal processing which takes 0.058 ms to execute.[4] Therefore, we use the value $(0.058 \text{ ms}/13)^{-1}$ for μ_1 in the analytic representation of Workload 1. Similarly we use the value of $(80 \cdot 0.058 \text{ ms}/250)^{-1}$ for μ_3 in the representation of Workload 2. The other task execution rates were similarly adjusted. These adjustments account for the low P_I value, because we are essentially creating more tasks and the chance of a cluster moving into an idle state is proportionally reduced.

The branch probabilities, P_i's, were determined using the nominal frequencies for the rate groups expressed earlier and the task class sizes (see Table IX). The computed time average steady-state probabilities for the states in the Markov chain using these parameter values are shown in columns 3 and 4 of Table VII. With these values, analytic results can be compared to experimentally derived values.

[4] This value was measured in [15].

TABLE IX
MARKING DEPENDENT BRANCH PROBABILITIES

Markov States		Branch Probabilities, $P_i(s)$ $P_i(s)=(1-P_I)\cdot Q_i(s)$		
s	Reduced System State	$Q_1(s)$	$Q_2(s)$	$Q_3(s)$
1	(3, 0, 0, 0, 0)	0.72728	0.24242	0.03030
2	(2, 1, 0, 0, 0)	0.72728	0.24242	0.03030
3	(2, 0, 1, 0, 0)	0.64000	0.32000	0.04000
4	(2, 0, 0, 1, 0)	0.82759	0.13793	0.03448
5	(2, 0, 0, 0, 1)	0.75000	0.25000	0
6	(1, 2, 0, 0, 0)	0.72728	0.24242	0.03030
7	(1, 1, 1, 0, 0)	0.64000	0.32000	0.04000
8	(1, 1, 0, 1, 0)	0.82759	0.13793	0.03448
9	(1, 1, 0, 0, 1)	0.75000	0.25000	0
10	(1, 0, 2, 0, 0)	0.47059	0.47059	0.05882
11	(1, 0, 1, 1, 0)	0.76190	0.19048	0.04762
12	(1, 0, 1, 0, 1)	0.66667	0.33333	0
13	(1, 0, 0, 2, 0)	0.96000	0	0.04000
14	(1, 0, 0, 1, 1)	0.85714	0.14286	0
15	(0, 3, 0, 0, 0)	x	x	x
16	(0, 2, 1, 0, 0)	x	x	x
17	(0, 2, 0, 1, 0)	x	x	x
18	(0, 2, 0, 0, 1)	x	x	x
19	(0, 1, 2, 0, 0)	x	x	x
20	(0, 1, 1, 1, 0)	x	x	x
21	(0, 1, 1, 0, 1)	x	x	x
22	(0, 1, 0, 2, 0)	x	x	x
23	(0, 1, 0, 1, 1)	x	x	x
24	(0, 0, 3, 0, 0)	x	x	x
25	(0, 0, 2, 1, 0)	x	x	x
26	(0, 0, 2, 0, 1)	x	x	x
27	(0, 0, 1, 2, 0)	x	x	x
28	(0, 0, 1, 1, 1)	x	x	x
29	(0, 0, 0, 2, 1)	x	x	x

x = Does not apply for this case.

The probability there is an idle cluster is the sum of the time average steady-state probabilities for the Markov states where there are one or more clusters at Node 2, i.e., states 2, 6, 7, 8, 9, and 15–23. The analytic values determined for both workloads is compared to the experimental values in Table

TABLE X
COMPARISON OF EXPERIMENTAL AND ANALYTIC RESULTS

	Experimental	Analytical	Difference
Workload 1	0.95	0.82	13.7%
Workload 2	0.73	0.76	-4.1%

(a) Probability of an Idle Cluster

	Experimental	Analytical	Difference
Workload 1	0.51	0.46	9.8%
Workload 2	0.48	0.46	4.2%

(b) Probability of Bus Contention

	Experimental	Analytical	Difference
Workload 1	13.422 μsec	12.412 μsec	7.5%
Workload 2	12.532 μsec	10.423 μsec	16.8%

(c) Waiting Time for a Free Bus

X(a), and their percent difference is noted. The difference in Workload 1 is attributed to the assumption that the idle time was exponentially distributed. As demonstrated by Fig. 3(a), the idle time was usually one of three values. Since synthetic tasks have a constant execution time and a time frame is constant, the idle times become constant. When there was more variation in idle times, which is more realistic and was observed in Workload 2 [Fig. 3(b)], the analytic results are more accurate.

The probability of bus contention is the sum of the steady-state probabilities of states representing more than one cluster at Node 1, i.e., states 1–5. The analytic and experimental results are compared in Table X(b). The difference here is attributed to the adjustments necessary in deriving the parameter values and to the fact that bus transmission times were not data dependent.

Finally, the time for a cluster waiting for a free bus is calculated using the expression for W_q in Section IV. The comparison of results is shown in Table X(c). These calculations were heavily dependent on μ_S and the second moment of the service time. The differences are a result of the lack of variation in transmission durations. This is a problem of the SWG and not the model.

VIII. Conclusion

A closed queueing network model was presented to study the workload effects on performance for a highly reliable unibus multiprocessor used in critical real-time applications. Through extensive measurements on FTMP, the model was shown to be easily solved for a given set of parameters. We were able to experimentally justify the performance values demonstrated in the model. Despite the differences in the analytic and measured values, which have been accounted for, the queueing model produces acceptable results that justify its use as a tool for performance modeling.

The number of operating clusters remained constant throughout the analysis, because of the assumption that a reconfigured system will reach steady state before another cluster fails. The performance of a degraded system will be less than that of the previous system. To obtain the overall performance of the system operating over a given length of time, the performance contributions of each of the configurations can be combined, weighted by their relative time of operation. This analysis is similar to Meyer's performability [16].

The area that merits further research is determining the task partition equivalence relation for defining task classes. Being a more restricted problem than the workload characterization of a general-purpose computer motivates continued research in finding a solution. Once a characterization method is developed, the possibility of obtaining an optimal workload distribution to provide optimal performance can then be considered.

Ultimately, it would be desirable to be able to determine if a real-time multiprocessor system can handle a given workload and set of performance criteria. If it can, one would like to know how this might be accomplished in the sense of optimal performance, workload distribution, and scheduling. The model presented here is a device that can aid in solving some of these problems.

Acknowledgment

The authors are grateful to P. Padilla, C. Liceaga, and R. W. Butler at the NASA AIRLAB for their assistance in the FTMP experiments.

References

[1] M. Calzarossa and G. Serazzi, "A characterization of the variation in time of workload arrival patterns," *IEEE Trans. Comput.*, vol. C-34, pp. 156–162, Feb. 1985.
[2] M. A. Marsan, G. Balbo, and G. Conte, "Comparative performance analysis of single bus multiprocessor architectures," *IEEE Trans. Comput.*, vol. C-31, pp. 1179–1191, Dec. 1982.
[3] M. A. Marsan and M. Gerla, "Markov models for multiple bus multiprocessor systems," *IEEE Trans. Comput.*, vol. C-31, pp. 239–248, Mar. 1982.
[4] C. M. Krishna and K. G. Shin, "Performance measures for multiprocessor controllers," in *Performance '83*, A. K. Agrawala and S. K. Tripathi, Eds. New York: North-Holland, 1983, pp. 229–250.
[5] L. J. Miller, "A heterogeneous multiprocessor design and the distributed scheduling of its task group workload," in *Proc. 9th Symp. Comput. Architecture*, 1982, pp. 283–290.
[6] A. Singh and Z. Segall, "Synthetic workload generation for experimentation with multiprocessors," in *Proc. 3rd Int. Conf. Distributed Comput. Syst.*, Oct. 1982, pp. 778–785.
[7] M. H. MacDougall, "Instruction-level program and processor modeling," *Computer*, vol. 17, pp. 14–24, July 1984.
[8] D. Ferrari, G. Serazzi, and A. Zeigner, *Measurement and Tuning of Computer Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1983.
[9] D. Ferrari, "On the foundations of artificial workload design," in *Proc. 1984 ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, Aug. 1984, pp. 8–14.
[10] T. B. Smith and J. H. Lala, "Development and evaluation of a fault-tolerant multiprocessor (FTMP) computer: Volume I FTMP principles of operation," NASA Contractor Rep. 166071, May 1983.
[11] J. H. Lala and T. B. Smith, "Development and evaluation of a fault-tolerant multiprocessor (FTMP) computer: Volume II FTMP software," NASA Contractor Rep. 166072, May 1983.
[12] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[13] L. Kleinrock, *Queueing Systems, Vol. I: Theory*. New York: Wiley, 1975.

[14] D. R. Cox and W. L. Smith, *Queues*. London, England: Methuen, 1961.
[15] F. Feather, "Validation of a fault-tolerant multiprocessor: Baseline experiments and workload implementation," Master's Thesis, Dep. Elec. Eng. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, 1984.
[16] J. F. Meyer, "Closed-form solutions of performability," *IEEE Trans. Comput.*, vol. C-31, pp. 648–657, July 1982.

Michael H. Woodbury (S'83) received the B.S.E. and M.S.E. degrees in computer engineering, in 1983 and 1984, from the University of Michigan, Ann Arbor, where he is currently pursuing the Ph.D. degree.

His areas of technical interest include computer workload characterization, real-time systems and control, and fault-tolerant computing.

Mr. Woodbury is a member of Eta Kappa Nu, Tau Beta Pi, the IEEE Computer Society, and the Association for Computing Machinery.

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

He is a Professor in the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan, which he joined in 1982. He has been very active and authored/coauthored over 100 technical papers in the areas of fault-tolerant real-time computing, computer architecture, and robotics and automation. In 1986, he founded the Real-Time Computing Laboratory, where he and his students are currently building a 19-node hexagonal mesh multiprocessor to validate various architectures and analytic results in the area of distributed real-time computing. From 1970 to 1972 he served in the Korean Army as an ROTC officer and from 1972 to 1974 he was on the Research Staff of the Korea Institute of Science and Technology, Seoul, Korea, working on the design of VHF/UHF communication systems. From 1978 to 1982 he was an Assistant Professor at Rensselaer Polytechnic Institute, Troy, NY. He was also a Visiting Scientist at the U.S. Airforce Flight Dynamics Laboratory in Summer 1979 and at Bell Laboratories, Holmdel, NJ, in Summer 1980.

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