IEEE TRANSACTIONS ON COMPUTERS, VOL. 37, NO. 4, APRIL 1988

The MAFT Architecture for Distributed Fault Tolerance

ROGER M. KIECKHAFER, MEMBER, IEEE, CHRIS J. WALTER, MEMBER, IEEE, ALAN M. FINN, MEMBER, IEEE, AND PHILIP M. THAMBIDURAI, MEMBER, IEEE

Abstract—**This paper describes the Multicomputer Architecture for Fault-Tolerance (MAFT), a distributed system designed to provide extremely reliable computation in real-time control systems. MAFT is based on the physical and functional partitioning of executive functions from application functions. The implementation of the executive functions in a special-purpose hardware processor allows the fault-tolerance functions to be transparent to the application programs and minimizes overhead. Byzantine Agreement and Approximate Agreement algorithms are employed for critical system parameters. MAFT supports the use of multiversion hardware and software to tolerate built-in or generic faults. Graceful degradation and restoration of the application workload is permitted in response to the exclusion and readmission of nodes, respectively.**

Index Terms—**Approximate agreement, distributed systems, fault tolerance, flight control, interactive consistency, real-time systems.**

I. Introduction

THE computerization of life-critical control systems has placed extreme safety requirements on real-time computing systems. Perhaps the most stringent requirement to date is that proposed for flight-critical control functions in advanced commercial transport aircraft. The failure probability of such systems is required to be about 10^{-10}/h; this is approximately three orders of magnitude more stringent than the corresponding requirement for military aircraft [1]. In those few systems designed to specifically address this extreme level of dependability, fault-tolerance is achieved through modular redundancy and the voting of data from replicated tasks [2]–[4].

Many problems unique to extremely reliable real-time systems were first addressed by the Software Implemented Fault-Tolerance (SIFT) distributed computer system [2], [9], leading to some fundamental results in fault-tolerance theory [8]. However, performance measurements have shown that SIFT executive functions can consume up to 80 percent of the system throughput [5]. This result demonstrates that the computational overhead of maintaining fault tolerance can seriously detract from the system resources available for application functions. A contemporary alternative to the software intensive approach of SIFT is the Fault Tolerant Multiprocessor (FTMP) [3]. FTMP provides hardware assistance for several system executive functions, such as voting and synchronization. Still, studies have shown that up to 60 percent of the system throughput can be consumed by FTMP executive functions [10]. A descendent of FTMP is the Fault Tolerant Processor (FTP) of the Advanced Information Processing System (AIPS) program [11]. FTP has shown much higher efficiencies than either SIFT or FTMP [4]. Whereas SIFT and FTMP are multiprocessor systems, FTP functions as a uniprocessor, employing redundant processing channels solely to provide fault tolerance. The effective processing power of FTP is thus limited to that of a single processor.

This paper describes a system called the Multicomputer Architecture for Fault-Tolerance (MAFT) [7], a distributed computer system designed to combine extreme reliability with high performance in a real-time environment, which is the result of an extended research effort in ultrareliable fault-tolerant systems [6].

Johnston *et al.* [1] have tabulated the system requirements for a variety of aerospace-related control applications. These requirements can be partitioned into two categories: performance and dependability. For a commercial flight-control system, the performance requirements are characterized by control loop frequencies (iteration rates) of up to 200 Hz, instruction throughputs of up to 5.5 million instructions per second (MIPS), I/O rates of up to one million bits per second (MBPS), and a transport lag (input to output delay) as short as 5 ms. These numbers are representative of the applications that MAFT was meant to support, and are taken as minimum performance objectives for the system. The major dependability objective of MAFT is the mission safety specification that the probability of system failure be approximately 10^{-10}/h over a 10 h mission, without repairs. To meet this objective, MAFT must tolerate subtle and highly improbable faults, such as multiple coincident faults, malicious or "Byzantine" faults, and built-in or "generic" faults.

Section II presents an overview of the MAFT architecture. Section III, comprising the bulk of the text, describes how the executive functions are implemented in MAFT. It also

Manuscript received June 8, 1987; revised November 23, 1987.

R. M. Kieckhafer was with the Bendix Aerospace Technology Center of the Allied Signal Corporation, Columbia, MD, 21045. He is now with the Department of Computer Science, University of Nebraska, Lincoln, NE 68588-0115.

C. J. Walter is with the Bendix Aerospace Technology Center of the Allied Signal Corporation, Columbia, MD 21045.

A. M. Finn was with the Bendix Aerospace Technology Center of the Allied Signal Corporation, Columbia, MD 21045. He is now with the United Technologies Research Center, East Hartford, CT 06108.

P. M. Thambidurai is with the Bendix Aerospace Technology Center of the Allied Signal Corporation and the Department of Electrical Engineering, Duke University, Durham, NC 27705.

IEEE Log Number 8719330.

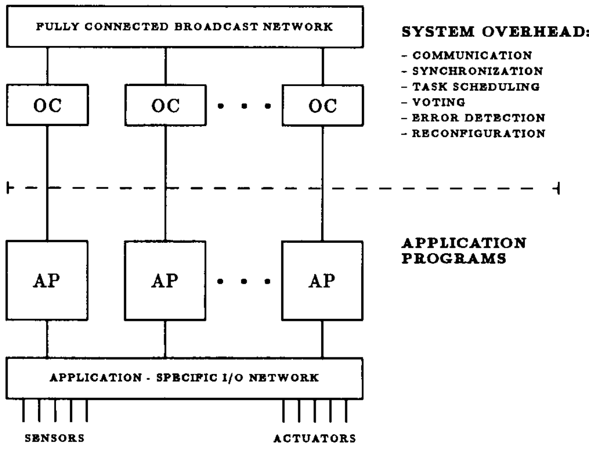


Fig. 1. MAFT system architecture.

discusses the capabilities and limitations of the current implementation of the architecture.

II. MAFT Architecture

A MAFT system consists of several semi-autonomous computers (nodes) connected by a broadcast bus network, as shown in Fig. 1. Each node is partitioned into two separate processors called the operations controller (OC) and the application processor (AP), respectively. The OC is a hardware-intensive data-driven processor designed to handle the vast majority of the system's executive functions. These functions include internode communication and synchronization, data voting, error detection, task scheduling, and system reconfiguration.

The OC/AP partitioning leaves the AP free to execute the application programs. Typical application functions performed by the AP include reading sensors, performing control law computations, and sending commands to actuators. While the OC is a specific device common to all MAFT systems, the AP may be any processor appropriate to a given application. The AP operating system may be extremely simple, since overall management of the distributed system is performed by the OC.

A problem facing any distributed system is maintaining agreement between nonfaulty nodes in the presence of faults. Various parameters may require exact or Byzantine Agreement [8], or Approximate Agreement [13]. A variety of solutions to the distributed agreement problem exist. The earliest solution employs hardware interstages in the communication paths [14], as used in FTP. Others make constraining assumptions about the behavior of the communication system [20], [21]. In MAFT, it is assumed that any error is possible, no matter how malicious. Therefore, interactive consistency [15] and convergent voting [13] algorithms are applied where required to maintain Byzantine Agreement and Approximate Agreement, respectively.

It is difficult to quantify the probability that generic faults exist in any given system. However, software fault-tolerance studies have shown that the existence of generic faults cannot be ignored [12]. To deal with generic faults, MAFT permits the use of design diversity in both software and hardware. This feature complicates the problem of detecting faults and managing the system. For example, truly dissimilar copies of a given task will most likely have slightly different execution times. Similarly, differences in such factors as numerical stability, arithmetic precision, and timing can cause slight differences in the various copies of a given data value. Unlike SIFT, FTMP, and FTP, equality among nonfaulty copies of a particular data value is not required in MAFT. Also, unlike FTMP and FTP, copies of a task are not required to run in exact synchrony.

Two MAFT prototypes are currently being implemented. The first is a demonstrator in which the OC's are implemented in standard 7400 series TTL technology. To date, four of six planned nodes have been assembled and operated as a system. In the second prototype, the OC's are implemented in a set of seven 2 μm CMOS chips. The current realization of the architecture will support up to eight nodes, providing sufficient redundancy to tolerate multiple coincident faults.

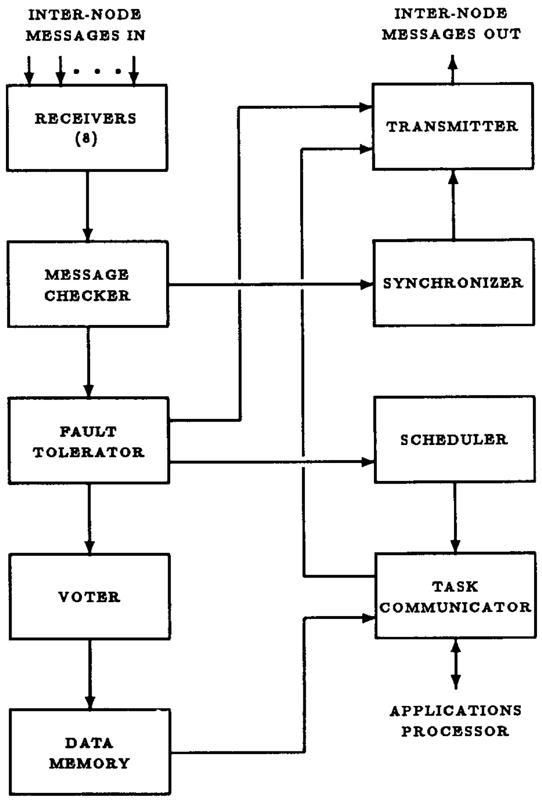


Fig. 2. Operations Controller block diagram.

III. OC Functions

The OC is a special purpose data-driven processor composed of several semi-autonomous subsystems. These subsystems, illustrated in Fig. 2, perform the executive functions listed in Fig. 1.

Communication

Each OC has two communication functions. It must communicate with all other OC's in the system, and with its own AP. An OC communicates with the other OC's through formatted messages transmitted over its dedicated serial broadcast link. The effective bandwidth in the demonstrator implementation is about one MBPS per node. Dedicated receivers in the receiver subsystem monitor each link and receive all messages broadcast by the OC associated with that link. The Message Checker subsystem subjects each received message to a variety of physical and logical checks. Messages deemed to be correct are passed to the other subsystems, as appropriate, for further processing.

Messages are classified as data, scheduling, synchronization, or error-management messages. A data message is broadcast by an OC whenever it receives a computed data value from its own AP. There are two types of scheduling messages: the task completed/started (CS) message, transmitted by an OC whenever its own AP completes an application task, and the task interactive consistency (TIC) message, transmitted by all nodes at predefined intervals. The synchronization or system state (SS) message is also transmitted by all nodes at predefined intervals. There are two types of error-management messages: the error report (ERR) message transmitted by a node when it detects an error committed by any node, and the base penalty count (BPC) message, transmitted by all nodes at predetermined intervals. The BPC message informs the recently powered-up node of the error status of all nodes in the system.

The contents of all internode messages are protected by a strict link protocol and an error control code (ECC). However, the fault tolerance of MAFT is not predicated upon the reliable delivery of all messages. MAFT relies on modular redundancy and distributed agreement algorithms to mask the effects of corrupted messages. While the ECC and link protocols cannot guarantee the detection of individual corrupted messages, they do probabilistically preclude the persistent undetectable corruption of multiple messages.

An OC communicates with its AP through an asynchronous parallel interface located in the Task Communicator subsystem. The interface appears as a simple input/output device to the AP. This approach eliminates the need for special purpose interfacing hardware or software in the AP. Upon completion of a task, the AP informs the OC. The OC responds with the next task to be started. The AP immediately starts the new task while the OC broadcasts a CS message to all OC's. Voted data required by the task are provided by the OC from the data memory as requested by the AP. Similarly, the OC receives from the AP any output data which require voting. Each data value is broadcast to all OC's in a data message as soon as it is received from the AP.

Synchronization

There are two major synchronization functions in MAFT; the first pertains to steady-state operation, the second to startup. The steady-state algorithm is responsible for maintaining synchronization of the nodes in the "operating set," which is defined as the subset of the system nodes that are synchronized to each other and considered to be nonfaulty. The startup algorithm has two modes: "cold start" mode for the initial synchronization of the system, and "warm start" mode for the synchronization of a node to an existing "operating set."

Steady State: The steady-state algorithm is similar to that used by SIFT [9]. Loose, frame-based synchronization is achieved through the exchange of system state (SS) messages whose transmission implicitly denotes the local clock time. One iteration of the steady-state algorithm entails the broadcast of two versions of the SS message. For the first message, each node counts a nominal number of local clock ticks and then broadcasts a "presync" SS message. Each node timestamps these presync messages on receipt. Since all nodes use the same nominal count parameter, each local timestamp is a function of the sending node's initial synchronization skew, the sending node's local clock rate, and the message transmission delays across the link. Each node computes an error estimate as the difference between the timestamp for its own presync message and its voted value for all the presync timestamps. The error estimate is used to adjust the number of clock ticks counted before sending the second message. For the second message, each node counts the locally adjusted nominal number of clock ticks and then broadcasts a "sync" SS message. Voted timestamp values are produced by the fault-tolerant voting algorithm described in the data management and voting section. The correctness and accuracy of this algorithm may be derived by regarding it as a hardware implementation of Srikanth and Toueg's optimal clock synchronization algorithm [17].

The accuracy of steady-state synchronization depends on several parameters: the length of the synchronization interval, clock drift, and message delivery delay. In a worst-case flight-control configuration, the nodes are physically distributed throughout a large aircraft. Given this configuration, the skew between any two nonfaulty nodes will not exceed 18 μs in the current OC implementation, even in the presence of a malicious fault.

Startup: In previous systems, guaranteed automatic startup was not considered necessary [16]. However, reliable "cold start" may be required to recover from catastrophic midmission events, such as a severe lightning strike or multiple power supply interruptions. Reliable "warm start" is required for graceful readmission of nodes previously excluded for transient faults.

For cold start, all nonfaulty nodes converge to and agree upon a single operating set. Each node synchronizes temporally by iteratively timestamping, in overlapping buffers, the receipt of SS messages for a given period of time. The period is chosen such that one iteration of the synchronization algorithm by any operational, nonfaulty node will be heard. Each node's synchronization interval is determined from these data and voting produces a target synchronization time. Only bounded corrections which shorten the second synchronization message interval are allowed in converging to the target time. This is an implementation of a nonterminating Approximate Agreement algorithm [13].

Termination of the cold start algorithm is achieved by Byzantine Agreement. In the interval preceeding the presync SS message, each node records a bit vector denoting which other nodes are "in sync with" (ISW) it. The ISW vectors are broadcast in the sync SS message. If all the nodes were started at approximately the same time, with no faults, it would be sufficient to determine the largest clique from the received ISW vectors. An operating set would be formed when enough nodes were in the clique. However, variations in message delivery time or the presence of faults may produce asymmetric ISW vectors (node i in sync with node j, but not vice versa). Therefore, the received ISW vectors are rebroadcast in the presync SS message. An interactive consistency algorithm is used to compute consistent ISW vectors, and reduce them to a potential operating set vector. When there are max $[(3f+1), (\lfloor N/2 \rfloor + 1)]$ nodes in the potential operating set vector (where f is the maximum number of maliciously faulty nodes, and N is the number of nodes in the system), reconfiguration to that operating set may be initiated.

During warm start, an operating set already exists. A starting node announces its intention to join the operating nodes by beginning to broadcast SS messages. If the starting node detects at least max $[(3f + 1), (\lfloor N/2 \rfloor + 1)]$ operating nodes, it converges to its voted value of the operating nodes' timestamps. When the starting node has synchronized itself with the operating nodes, the operating nodes may reconfigure to include it in the operating set.

Data Management and Voting

Each OC stores a copy of all shared application data values in its own data memory. This scheme provides N-way redundant storage of all data, and makes it immediately available to any node. The OC handles the management and voting of application data in a manner transparent to the AP.

Each data message is tagged with a data identification descriptor (DID) which uniquely labels the data contained in the message. Upon receipt of a data message from any node, each OC performs reasonableness checks to filter out data which are outside of predefined maximum and minimum limits. Data which pass the reasonableness checks are forwarded to the voter subsystem and trigger a new vote on the values with that DID. The voter does not wait for the arrival of all expected copies of a DID. Rather, it performs an "on-the-fly" vote using the new copy and any previously received copies. If one copy of the task generating the data value should fail or "hang," the voted value of the remaining copies is still available.

A dynamic deviance check verifies that all inputs used in the voting process are within a specified window of the voted value. The size of the deviance window is individually defined for each DID. The voter performs the deviance check every voting cycle and identifies the source node for each copy which fails the check. Deviance checking and on-the-fly voting permit truly dissimilar versions to be employed in both software and hardware.

Two algorithms are available for voting on application data. The first is the familiar "median select" (MS) algorithm which selects the center value for any odd number of inputs and averages the two central values for an even number of inputs. The second algorithm is the "mean of the medial extremes" (MME). These algorithms are just two variations in the more general Fault-Tolerant Midpoint voting strategy [13]. This strategy discards the μ most extreme values from either end of a sorted set of N inputs, and computes the mean of the two remaining extremal values. Computationally, the MS and MME algorithms vary only in their choice of μ. MS uses $\mu = \lfloor (N - 1)/2 \rfloor$, whereas MME used $\mu = \lfloor (N - 1)/3 \rfloor$.

The two algorithms offer different fault-tolerance properties. MS is the more robust algorithm since it discards more extremal values than MME. However, MME is convergent in the presence of up to μ erroneous copies regardless of the nature of those errors. Dolev [13] has shown that MME has a guaranteed convergence rate of 1/2. Conversely, a malicious error can prevent MS from converging. The application designer selects which of the two algorithms is to be used for each DID.

Application Task Scheduling

In MAFT, the application software is broken into tasks—indivisible blocks of code which must be executed without interruption on a single AP. As viewed by the OC, each task has several properties: iteration frequency, relative priority, desired redundancy, and intertask dependencies.

Dependencies between tasks may include concurrent forks and joins (AND-FORKS and AND-JOINS) or conditional branches (OR-FORKS and OR-JOINS). The MAFT scheduler subsystem treats all tasks as periodic. Nonperiodic behavior is obtained through conditional branching. MAFT supports tasks of varying frequencies within the constraints of a binary frequency distribution, i.e., all frequencies are 2^{-j} times the highest frequency, where j is a nonnegative integer.

The iteration period of a task is the reciprocal of its frequency. The shortest iteration period in the MAFT hierarchy is the "atomic period," so called because it is indivisible with respect to iteration periods. The boundaries between atomic periods coincide with the transmission of an SS message. While no single task may be repeated more than once per atomic period, several shorter tasks may be scheduled and executed on each node during the same atomic period. Conversely, a low-frequency task may run for several atomic periods.

The longest permissible iteration period is referred to as the "master period," and corresponds to the period of the lowest frequency task in the workload. The current OC implementation allows up to 1024 atomic periods per master period, permitting tasks of up to ten distinct frequencies to execute concurrently.

Scheduler Operation: The scheduling strategy selected for MAFT is a fault-tolerant variation of a deterministic priority-list algorithm. In this approach, each task is assigned a unique priority number. When a node becomes available, the list of tasks is scanned in order of decreasing priority. The first task encountered which is ready for execution is selected. In MAFT, the assignment of tasks to nodes is determined by the task reconfiguration process and is static for any given operating set. A separate priority list is thus maintained for each node.

The scheduling function is fully replicated so that the scheduler of each node selects tasks for every node in the system. The selection for its own node is used to dispatch tasks for its own application processor. The selections for all other nodes are used to monitor the actions of the other schedulers.

When an AP completes its currently assigned task, it passes a branch condition (BC) value to its OC, to be used by the schedulers in resolving OR-FORK dependencies. The OC immediately broadcasts a task completed/started (CS) message, containing the received BC.

To maintain Byzantine Agreeement on scheduling data, the scheduler employs an Interactive Consistency algorithm. This algorithm requires synchronized transmission rounds in which each node rebroadcasts the data received in the previous round. The number of rebroadcasts required to guarantee resolution of the disagreement is equal to the maximum number of simultaneous malicious faults to be tolerated [15]. MAFT incorporates one round of rebroadcast to deal with a single malicious fault.

Synchronization of the rebroadcasts is accomplished by defining a "subatomic" period whose boundaries are marked by the simultaneous transmission of task interactive consistency (TIC) messages by all nodes. An integer number of subatomic periods constitutes one atomic period. The TIC message contains two bytes, reflecting the scheduling activity of all nodes during the previous subatomic period. The first byte is a task completed (TC) byte indicating those nodes from which a CS message was received. The second byte is a BC byte indicating the branch condition contained in each received CS message. After the TIC messages have been received, each node executes an Interactive Consistency Algorithm on the contents of the TIC messages. Thus, all nodes reach Byzantine Agreement on the existence and content of each CS message transmitted.

Task execution timers synchronized to the subatomic period boundaries monitor the execution time of each task. If the execution time of a task is shorter than a predetermined minimum or longer than a predetermined maximum, then an execution timer error is reported. In addition, a task sequence error is reported whenever a node fails to execute tasks in the sequence expected by the schedulers.

Scheduler Performance: To ensure consistency among schedulers, all data structure modifications are based on the agreed upon contents of TIC messages, rather than CS messages. The "confirmation delay" created by waiting for TIC messages imposes a potential performance penalty in the release of successor tasks. If the task precedence graph defined by the intertask dependencies consists of a single path of dependent tasks, then the AP's will be idle while completions are confirmed. However, if the graph contains parallel paths, then independent tasks may be executed while dependencies are resolved. In the type of real-time workloads expected for MAFT applications, there is a significant amount of available parallelism, indicating that high-AP utilization can be maintained.

Fig. 3 shows the precedence graph for part of an actual workload currently being developed for a particular MAFT-based flight control system [22]. This graph concurrently processes the control laws for each of the three axes of rotation, along with a system monitoring function. All four paths converge by an AND-JOIN to a validity checking function,

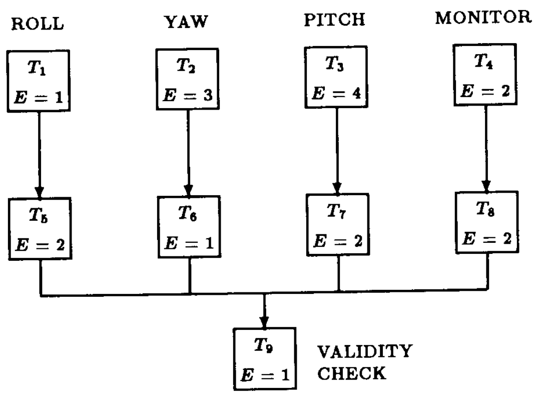


Fig. 3. Sample workload fragment.

PERIOD	1	2	3	4	5	6	7	8	9	10	11	12
GROUP 1	1	3				5		7		ϕ		9
GROUP 2	2			4		6	ϕ	8		ϕ		9

Fig. 4. Full six-node system standard Gantt chart.

task 9. The maximum execution time of each task, E, is listed in units of subatomic periods.

Assume that this workload requires triple redundancy in all tasks except task 9. Task 9 must be executed by all nodes in the operating set, and therefore has a variable redundancy of N, rather than a fixed redundancy of three. Fig. 4 is a standard Gantt chart [19] of the workload on a six-node system, with tasks 1, 3, 5, 7, and 9 assigned to nodes 1, 2, and 3 (Group 1), and tasks 2, 4, 6, 8, and 9 assigned to nodes 4, 5, and 6 (Group 2). The null task ϕ indicates processor time unavailable to this precedence graph due to confirmation delay.

The Gantt chart illustrates the effect of confirmation delay on processor utilization. During subatomic period 7, the AP's of nodes 4, 5, and 6 are idle because tasks 7 and 8 cannot be released until the completion confirmation of tasks 3 and 4, respectively. Similarly, all processors are idle during subatomic periods 10 and 11 because task 9 cannot be released until completion of tasks 7 and 8 is confirmed. This schedule utilizes 19 of the 24 available processor periods, for an overall processor utilization of 79 percent.

The minimum length of the subatomic period in the current OC implementation is 400 μs. For a full eight-node system, a minimum of seven subatomic periods is required per atomic period, yielding a maximum control loop frequency of 357 Hz. The flight-control workload above is designed for a 500 μs subatomic period. With this workload, the six-node system can support an atomic period as short as 6 ms for a frequency of 167 Hz.

Task Reconfiguration

In MAFT, task reconfiguration refers to the process of redistributing the application workload to account for changes in the system operating set. The overall objectives of task reconfiguration are to provide graceful degradation of application functions as resources are lost, and graceful restoration of those functions as resources are restored. The task reconfiguration process produces an eligibility table, indicating those nodes to which each task may be assigned. The actual selection and execution of tasks from the eligibility table are controlled by the normal scheduling process at run time. A new value for the system operating set is periodically computed by the fault tolerator subsystem and compared to the previous operating set. If the two sets differ, then task reconfiguration is initiated. Byzantine Agreement is assured by an Interactive Consistency algorithm performed on the operating set value before it is used.

A detailed description of the task reconfiguration algorithm is presented in [18]. Briefly, the algorithm employs three independent processes, combining their results to determine the eligibility of each task. Each process is path independent, and therefore reversible. The *Global Task Activation Process* activates or deactivates individual tasks to account for changes in the overall system capability. This process is realized by defining a set of relevant nodes and an initial activation status (either active or inactive) for each task. When the number of relevant nodes remaining in the operating set reaches a predetermined threshold, the activation status of the task is complemented. The *Task Reallocation Process* reallocates tasks among the operating nodes of the system to maintain the desired redundancy of each task. This process is realized by assigning tasks to nodes in a predetermined order of preference. If the redundancy of a task is R, then the task will be executed by the R most preferred nodes for that task in the current operating set. The *Task-Node Status Matching Process* allows tasks to be prohibited from executing on individual nodes based upon each node's operational status (i.e., included or excluded). For example, it may be desirable to prohibit output tasks from executing on an excluded node, assigning instead a comprehensive set of diagnostic tasks.

The ability to maintain functionality while the system degrades depends on the assignment of the node preferences for each task. A simple algorithm for defining preferences has been applied to the example workload of Fig. 3. This algorithm starts with the full six-node system, then attempts to equalize the workload as N, the number of nodes in the operating set, is reduced. For each value of N, preferences are assigned in order of decreasing task execution time. The preference orderings produced by this algorithm are shown in Table I. For each task i, the table lists its execution time E_i in subatomic periods, its required redundancy $R_i(N)$, and the system nodes in order of decreasing preference.

The effectiveness of task reconfiguration for the example workload of Fig. 3 is shown in Table II for all values of N from the minimal three-node system through the full six-node system. $W(N)$ is the total processing time required by the workload, computed as the summation of $E_iR_i(N)$ over all i. The value $t_0(N) = W(N)/N$ is the ideal (shortest) execution time for the workload. The value t is the actual execution time observed for each operating set. The AP utilization is then $t_0(N)/t$ for each case.

Table II shows that AP utilization is never less than 70 percent for any operating set. In the minimal three-node system, the workload requires 20 subatomic periods. Given

TABLE I
TASK-TO-NODE PREFERENCE ORDERING

i	E_i	$R_i(N)$	Node Preference Order
1	1	3	1 2 3 5 4 6
2	3	3	4 5 6 1 3 2
3	4	3	1 2 3 4 6 5
4	2	3	4 5 6 2 1 3
5	2	3	1 2 3 5 4 6
6	1	3	4 5 6 2 3 1
7	2	3	1 2 3 6 5 4
8	2	3	4 5 6 3 2 1
9	1	N	1 2 3 4 5 6

TABLE II
MULTIPROCESSING EFFICIENCY

N	$W(N)$	$t_o(N)$	t	% AP Utilization
3	54	18.00	20	90
4	55	13.75	17-18	76-81
5	56	11.20	15-16	70-75
6	57	9.50	12	79

the 500 μs subatomic period length, the minimum atomic period is 10 ms for a frequency of 100 Hz. Deactivation or replacement of less critical tasks can help to maintain loop frequency in severely degraded systems. For example, if the maximum permissible atomic period for this application is 18 subatomic periods (9 ms), then the three-node system exceeds its deadline by two subatomic periods. Deactivation of the four-period long monitoring function (tasks 7 and 8) when $N = 3$, and activation of a simpler two-period long monitoring function would permit the control functions to meet their deadlines.

Error Handling

An OC contains error detection mechanisms which continuously monitor the behavior of all nodes as revealed by their message traffic. These errors are reported in one or more of 31 error flags contained in an ERR message. The OC also uses a penalty counting mechanism to communicate the overall health of the node. A base penalty count (BPC) is maintained which indicates the current value of the accrued penalties for every node. An incremental penalty count (IPC) is also maintained for each node, containing a proposed penalty assessment for the node based on error detections during the current atomic period. Whenever an error is detected, a penalty weight, unique to the triggered detection mechanism, is added to the IPC. The value of the penalty weight for each detection mechanism is set by the application designer to reflect the relative severity of the type of error detected.

At the beginning of every atomic period, each node broadcasts an error report (ERR) message regarding each node in the system. The ERR message contains the accused node's identity, error flags, BPC, and IPC. Upon receipt of a round of ERR messages, each node votes on the contents of the error flag, BPC, and IPC fields. An updated BPC value is calculated for each node by adding its voted IPC to its voted BPC value. In this way, Byzantine Agreement on both the IPC and BPC counts is guaranteed. (The message which contained the original error is the first round broadcast of an interactive consistency algorithm; the ERR message constitutes the rebroadcast.)

Once Byzantine Agreement is reached, the BPC value of an accused node is updated and compared to a predefined exclusion threshold. If the value of the BPC exceeds the threshold, then each OC will recommend exclusion of the accused node from the operating set. The recommendation is made through a "next operating set" field contained in the next SS message. Upon receipt of the SS messages, each node votes on the contents of the next operating set fields in accordance with the Interactive Consistency algorithm, and takes the voted value as the new operating set of the system. Thus, Byzantine Agreement is also guaranteed on the inclusion or exclusion of each node in the operating set.

Reconfiguration of the application tasks may be initiated immediately or may be delayed until the beginning of the next master period, at the application designer's discretion. The excluded node is immediately prohibited from participation in any voting or decision making processes, masking its errors until reconfiguration is completed.

A faulty node will continue to accrue penalties, even after it has been excluded. If the node was excluded for a transient fault, it will subsequently exhibit correct behavior, and will *not* accrue penalties. At the beginning of each master period, the BPC of each node is decremented by a specified amount; this amount may be zero for permanent exclusion. When the BPC of an excluded node falls below a predetermined readmission threshold, each OC recommends readmission via the same sequence used to decide on exclusion.

Under normal circumstances, error detection mechanisms will not be exercised for extended periods of time, resulting in extremely long latency times for faults within the detection mechanisms themselves. Therefore, a system level self-test mechanism has been implemented in the OC. At a predetermined time, one node, selected on a rotating basis, broadcasts a stream of erroneous bits designed to exercise specific error detection mechanisms. All nonfaulty nodes respond by broadcasting error reports on the received errors. Addition of the IPC to the BPC is suspended to prevent exclusion of the originating node. Any node containing a faulty error detection mechanism is identified by the fact that its error report differs from the consensus. Using the consensus as the basis for correctness decouples the system self-test mechanism from the actual contents of the erroneous message stream. Thus, an error in the test messages generated by a specific source node does not affect the accuracy of the diagnosis. Rotation of source node duties ensures full test converage as long as at least one node is capable of correctly generating the self-test message stream.

IV. Summary

The MAFT system has been designed to provide high performance and extreme reliability across a broad spectrum of real-time applications. To satisfy these goals, MAFT relies on the functional and physical partitioning of system executive functions from application functions. The executive functions are performed by hardware-intensive data-driven operations controllers (OC) while application functions are assigned to general-purpose application processors (AP).

MAFT meets or exceeds its stated performance objectives by providing control loop frequencies in excess of 350 Hz, and bus bandwidths of about one MBPS. Since MAFT supports multiprocessing and permits a wide choice of AP's, the throughput of 5.5 MIPS is not difficult to attain. MAFT employs Interactive Consistency and Convergent Voting algorithms to maintain Byzantine Agreement and Approximate Agreement, respectively, on critical system and application parameters. Specific MAFT features such as on-the-fly voting and deviance checking support the use of dissimilar hardware and software in the system. Two different prototypes have been implemented and are currently undergoing testing.

Acknowledgment

The MAFT architecture is the result of long-term research and development efforts involving several divisions of the Allied Signal Corporation. The authors wish to thank all those who have contributed to MAFT and to the production of this paper, particularly M. C. McElvany. The example workload, due to F. Herman, is also acknowledged.

References

[1] M. W. Johnston *et al.*, "AIPS system requirements (revision 1)," CSDL-C-5738, Charles Stark Draper Lab., Inc., Cambridge, MA, Aug. 1983.

[2] J. H. Wensley *et al.*, "SIFT: Design and analysis of a fault-tolerant computer for aircraft control," *Proc. IEEE*, vol. 66, Oct. 1978.

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

[4] T. B. Smith, "Fault-tolerant processor concepts and operation," in *Proc. Fourteenth IEEE Fault-Tolerant Comput. Symp.*, June 1984.

[5] D. L. Palumbo and R. W. Butler, "Measurement of SIFT operating system overhead," NASA Tech. Memo. 86322, 1985.

[6] A. Whiteside *et al.*, "Fault-tolerant multicomputer system for control applications," in *Proc. Eleventh IEEE Fault-Tolerant Comput. Symp.*, June 1981.

[7] C. J. Walter *et al.*, "MAFT: A multicomputer architecture for fault-tolerance in real-time control systems," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 1985.

[8] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM TOPLAS*, vol. 4, pp. 382–401, July 1982.

[9] J. Goldberg *et al.*, "Development and analysis of the software implemented fault-tolerance (SIFT) computer," Final Rep. NASA Contract NASA-CR-172146, Feb. 1984.

[10] E. W. Czeck, D. P. Siewiorek, and Z. Segall, "Fault free performance validation of a fault-tolerant multiprocessor: Baseline and synthetic workload measurements," Carnegie Mellon Univ., Dep. Comput. Sci., CMU-CS-85-117.

[11] ——, "Advanced information processing system (AIPS) system requirements (revision 1)," Rep. CSDL-C-5709, Charles Stark Draper Lab., Inc., Cambridge, MA, Oct. 1984.

[12] J. C. Knight *et al.*, "A large scale experiment in N-version programming," in *Proc. Fifteenth IEEE Fault-Tolerant Comput. Symp.*, June 1985, pp. 135–139.

[13] D. Dolev *et al.*, "Reaching approximate agreement in the presence of faults," in *Proc. Third Symp. Reliability Distributed Software Database Syst.*, Oct. 1983.

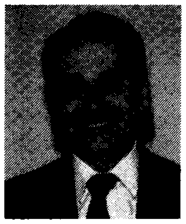
[14] D. Davies and J. Wakerly, "Synchronization and matching in redundant systems," *IEEE Trans. Comput.*, vol. C-27, pp. 531–539, June 1978.

[15] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *J. Ass. Comput. Mach.*, vol. 27, pp. 228–234, Apr. 1980.

[16] R. W. Butler, "An assessment of the real-time application capabilities of the SIFT computer system," NASA Tech. Memo. 84432, Apr. 1982.

[17] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *J. Ass. Comput. Mach.*, vol. 34, July 1987.

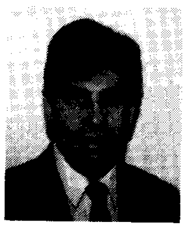
[18] R. M. Kieckhafer, "Task reconfiguration in a distributed real-time system," in *Proc. Eighth IEEE Real-Time Syst. Symp.*, Dec. 1987.
[19] G. K. Manacher, "Production and stabilization of real-time task schedules," *J. Ass. Comput. Mach.*, vol. 14, July 1967.
[20] O. Babaoglu and R. Drummond, "Streets of Byzantium: Network architectures for fast reliable broadcasts," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 546-554, June 1985.
[21] K. Perry, "Randomized Byzantine agreement," in *Proc. Fourth Symp. Rel. Distributed Software Database Syst.*, Silver Springs, MD, Oct. 1984, pp. 107-118.
[22] D. P. Gluch and M. J. Paul, "Fault-tolerance in distributed digital fly-by-wire flight control systems," in *Proc. AIAA/IEEE Seventh Digital Avion. Syst. Conf.*, Oct. 13-16, 1986.



Roger M. Kieckhafer (S'81-M'83) received the B.S. degree in nuclear engineering from the University of Wisconsin, Madison, in 1974 and the M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, NY, in 1982 and 1983, respectively.

From 1983 to 1987 he was with the Bendix Aerospace Technology Center of the Allied Signal Corporation, Columbia, MD, as a member of the Technical Staff. While there, he was engaged in research on fault-tolerant distributed computer systems, and was one of the designers of the Multicomputer Architecture for Fault-Tolerance (MAFT). He is currently with the University of Nebraska, Lincoln, as an Assistant Professor in the Department of Computer Science. His research interests include computer architecture, distributed systems, and fault- tolerance.

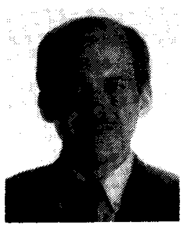
Dr. Kieckhafer is a member of the Association for Computing Machinery and the IEEE Computer Society.



Chris J. Walter (M'75) received the B.S. degree in electrical engineering from the University of Notre Dame, Notre Dame, IN, in 1975 and the M.S. degree in computer science and electrical engineering from the University of Michigan, Dearborn, in 1978 and is completing the D.Sc. degree in computer science at the George Washington University, Washington, DC.

Since 1980 he has been a member of the Technical Staff at the Bendix Aerospace Technology Center of the Allied Signal Corporation working in the design of fault-tolerant computer architectures and leads the System Integration and Reliability Group. His research interests include fault tolerance, computer architecture, and distributed systems. He is currently involved in the development of a highly architecture for artificial intelligence applications.

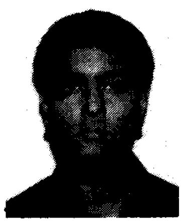
Mr. Walter is a member of the IEEE Technical Committee on Fault Tolerance.



Alan M. Finn (S'80-M'83) received B.S. degrees in mathematics and electrical engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1977. In 1979 he matriculated at Cornell University, Ithaca, NY. Between 1977 and 1979 he was employed by IBM, Poughkeepsie, NY, where he worked in Advanced Processor Development for the 308x series of machines and received the Ph.D. degree in 1983.

From 1983 to 1986, he was employed by the Bendix Aerospace Technology Center of the Allied Signal Corporation where he worked on the MAFT fault-tolerant real time control system for commerical aircraft. He is currently employed by the United Technologies Research Center and is working on fault-tolerance and signal processing. His research interests include computer architecture, fault tolerance, signal processing, and finite field mathematics.

Dr. Finn is a member of the IEEE Computer Society and the ACM.



Philip M. Thambidurai (M'87) received the B.S. degree in physics from Geneva College, Beaver Falls, PA, in 1978, and the M.S. degree in nuclear physics from North Carolina State University, Raleigh, in 1981.

Since 1985 he has been a member of the Technical Staff at the Bendix Aerospace Technology Center of the Allied Signal Corporation. He is currently involved in research and development related to future versions of the MAFT system. His research interests include fault tolerance, computer architecture, real-time systems, and modeling. He is also working towards the Ph.D. in electrical engineering at Duke University, Durham, NC.

Mr. Thambidurai is a member of the IEEE Computer Society.