An Integrated Scheduling Mechanism for Fault-Tolerant Modular Avionics Systems

Yann-Hang Lee

CISE Department
University of Florida
Gainesville, FL 32611
yhlee@cise.ufl.edu

Mohamed Younis Jeff Zhou

Advanced System Technology Group
AlliedSignal Aerospace
Columbia, MD 21045
younis, zhou@batc.allied.com

Abstract— In this paper, we present an effective scheduling approach for a fault-tolerant IMA (Integrated Modular Avionics)-based system. The system architecture consists of connected cabinets that are made of multiple line replaceable modules, such as core processor and I/O modules. To provide fault tolerance, the system is incorporated with fault resilient capability and executes replicated tasks on different cabinets. Thus application output will be ready after a task processing stage and a consistency checking stage. To schedule the two-stage operations at task processing nodes and at the voter, we adopt fixed priority executives and investigate two priority assignment algorithms. Several experiments have been conducted to measure the success ratios of finding feasible schedules under various conditions. The evaluation results reveal a proper design space in which feasible schedules can be found easily.

TABLE OF CONTENTS

1. INTRODUCTION

Computer controllers are the core units used in real-time embedded systems. Such controllers or embedded processors may deviate from general-purpose computer processors since they are designed for some special applications and have substantially different performance and implementation constraints. Various high-end 32-bit processors have found a fast expansion in avionics, telecommunication, military, aerospace, manufacture plants, and medical monitoring applications where computation power and safe operation are rigidly required. These systems not only must be fault tolerant, but also must meet task execution deadlines as their applications are often mission- and safety-critical.

Consider an example of the autolanding systems of wide-bodied jumbo passage jets [10] which control critical functions of aircraft's motion in each axis, i.e., roll, pitch, and yaw. They are required to operate in all visibility condition, including zero horizontal visibility and zero ceiling, and become truly critical in a 15 second interval just before touchdown. To certify the systems, a design must demonstrate a probability failure less than 10^{-9} during this critical interval. This high reliability requirement implies that the systems must tolerate most cases of component failures and must guarantee 100% timing correctness. A cost-effective design to reach this strict requirement becomes extremely important considering that a failure can be catastrophic and that the avionics systems account for some 30% of the total cost of a new airplane [5].

One approach to reduce the design and maintenance cost of avionics system is to take a modular approach. Traditional avionics systems are implemented with autonomous and federated architectures [1]. They consist of a number of interconnected but functionally independent (or loosely coupled) subsystems. In order to improve performance, flexibility, and availability of avionics systems, it is necessary to avoid possible duplication of functions and to allow resource sharing of system components and standard modules. One significant approach for cost-effective avionics systems is based on the integrated modular avionics (IMA) approach in which hardware and software systems are decomposed into modules and then integrated for various avionics applications [1]. The IMA approach suggests an architecture that consists of a set of interconnected cabinets. Each cabinet, containing a standard backplane interconnection and multiple line replaceable modules (LRM), forms a common platform to house the execution of software modules. With standard interfaces, hardware and software modules become interchangeable and can be reused. They can also be upgraded using new technology to add new functions. Thus, it is expected that the life-cycle cost of avionics systems can be reduced, and the processes of system development, and maintenance can be simplified.

As we gain the advantages of IMA approaches, it is expected that the whole system as well as the interactions between modules must be considered in the design and integration process. Modules must be put together to collectively perform application functions with a high reliability. In addition, the performance requirements must include a guarantee of responsiveness. A cabinet that cannot schedule all critical tasks to meet their deadlines may cause a timing error that can be catastrophic for time-critical tasks. For instance, a miss of deadline in the control loop of the autolanding system may cause a crash. Thus in order to implement the IMA approach for avionics applications, we require the hardware and software platforms to be able to:

1. allocate task modules of different applications into several candidate processors in which tasks can be scheduled according to their individual timing constraints.
2. provide fault tolerance mechanisms such that replicated task executions and checking operations can be managed coherently.

The design guidance report for IMA [1] lists several example architectures that can utilize modular components and install fault tolerance mechanisms at various levels. As the application modules are integrated in one or more cabinets, we may assume that operation executives can dispatch ready tasks based on either cyclic or priority-driven schedules [14]. On the other hand, the fault tolerance capability at system and cabinet levels can be established by incorporating redundant task execution at remote cabinets or at redundant processors. For either of these fault tolerant arrangements, the executions should be done before a checking process can verify the correctness of the results, and the accumulated response times must be bounded within the task's deadlines. This response time requirement clearly indicates that the scheduling of task replications and the result checking process must be addressed altogether.

The issues of scheduling tasks in fault tolerant systems have been investigated in previous research work. For instance, Krishna and Shin devised a ghost allocation mechanism in order to generate backup tasks [9]. The algorithm assumes that there exists a scheduling algorithm that checks the schedulability. The optimal allocation of replicated tasks under rate monotonic scheduling was studied in [16]. In addition, a dynamic scheduling and redundancy management approach was proposed in the Spring system in which replications can be created during system operation stages [6]. On the other hand, there are approaches to replicate identical subsystems. Then, similar to the cyclic scheduling executives, task executions and checking process are scheduled at specific instances [8, 10]. In these studies, the main focus is the scheduling of primary and backup tasks.
In this paper, we look into two design issues: how to embed a scheduling mechanism into a fault tolerant IMA system, and how to implement fixed-priority scheduling algorithms for task execution and result checking. Building fault resilience at system level can be an adequate approach for avionics systems. This is due to the fault containment introduced by the physically distributed cabinets that are often equipped with independent power and clock sources. The task execution in each cabinet should meet a target time which is shorter than the task deadline. Thus, the computation results can be released to the following checking process in order to reach interactive consistency. In fact, as the task processing is modeled by a two-stage pipeline, we need to find feasible priority assignments such that the sum of processing delays at the stages is bounded to the given timing constraint. Through an extendible timing analysis, the system behavior under the fixed priority scheduling algorithm can be predicted [18]. Most importantly, the approach does not have to examine every execution instance, thus makes it easy to accommodate any changes of system load.

In the following chapters, we first present the system architecture and describe the fault tolerance implementation with a Redundancy Management System (RMS) unit. Also, we show the scheduling model for such a system. In Section 3, we focus on the proposed algorithms to determine suitable priority assignments for a two-stage fixed priority schedule. The performance of the priority assignment algorithms is evaluated in several experiments. The success ratios of the algorithms are reported in Section 4. Finally, a short conclusion follows in Section 5.

2. System Architecture and Scheduling Model

Avionics systems typically consist of a number of cabinets that contains various modules to perform application processing, data communication, and local I/O operations to sensors and actuators. Each cabinet is made of multiple line replaceable modules (LRMs) of different types, such as CPU module, standard I/O and communication module, special I/O module, power supply module, etc. With the consideration of I/O requirement, wire length, maintainability, and payload areas, cabinets are physically distributed throughout the airplane. For avionics applications, this set of cabinets can be viewed as a distributed multi-computer system where application tasks and their redundant copies can be initiated in multiple cabinets and/or multiple modules. Thus, a failure of a cabinet or a LRM can be tolerated and the system functions continuously with a proper fault management scheme.

In the following, we will present a fault tolerance design based on AlliedSignal's MAFT architecture to provide system level fault resilience in a cabinet-based avionics system. Then, we show a scheduling model suitable for this architecture. We assume that the cabinets are organized according to the example architecture "C" of the IMA report [1]. Under the architecture "C", signal I/O is handled by remote data concentrators, thus the processing resources are physically independent of their I/O data. At each CPU module, the software structure includes a single executive and multiple application tasks. Application tasks are replicated across the redundant cabinets and a consistency

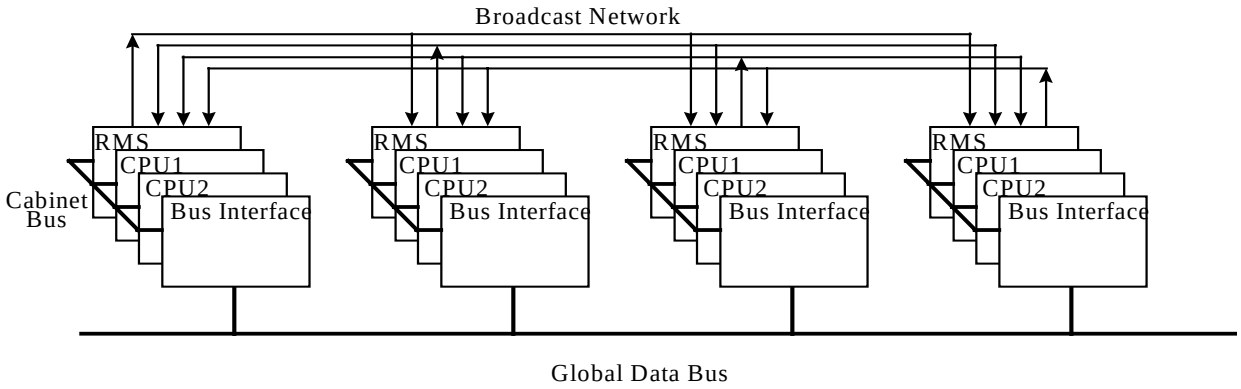

Figure 1. The fault-tolerant cabinets for IMA systems

checking operation ensures that correct results are always available given a limited degree of failure.

Architecture Model

The architecture model of our fault tolerant IMA system is shown in Figure 1. In addition to typical LRMs, each cabinet is equipped with a Redundancy Management System (RMS) module which provides system executive functions such as synchronization and data voting. With this quad-redundant architecture, the system can tolerate a single Byzantine-type failure. The approach can let a system developer concentrate on system application design and rely on the RMS module to achieve fault tolerance and redundancy management at the system level. The design of RMS is originated from the MAFT (Multi-computer Architecture for Fault Tolerance) system [8, 19] and can be implemented by a mix of hardware and software components.

The primary function of the RMS modules is to provide a consistency checking and voting mechanism. With a fully connected broadcast network, RMS performs voting on data values collected from replicated applications that are allocated throughout the redundant cabinets. Such data voting maintains consistency between the cabinets. In addition, it assists in recovering from transient and intermittent faults by replacing any corrupted application data with the voted values. Moreover, RMS votes on its internal state and error reports to maintain a global consistent system view of the system health status.

In order to perform checking and voting operations, the RMS modules of multiple cabinets must be synchronized and a global clock must be maintained in this loosely coupled distributed system. Each RMS has its own clock and the system synchronization is achieved by exchanging the local time among all RMS modules and correcting the local clock according to the cardinality of clocks from all healthy RMS units. A distributed agreement mechanism is used to prevent any single point of failure and a fault-tolerant voting algorithm is used to protect the global system clock from failure by any type of faults including Byzantine faults. This synchronization will be invoked periodically so the system can limit the clock skew and detect a failing cabinet immediately.

The ultimate goal of RMS is to prevent a system failure during the duration of a critical mission as a result of some error manifested by a fault on one node. After voting, RMS can detect, contain and recover from errors. By comparing the voted data values with the data submitted by the cabinet, RMS detects errors and penalizes the faulty one. Since all modules will be using the voted data values, errors can be tolerated and the faulty module will get a chance to recover by using the voted data. In addition, RMS supports dynamic system configuration by excluding faulty modules and readmission of recovered modules.

From an application's point of view, a task is replicated and statically allocated to CPU modules of different cabinets. The replicated instances of a task are executed synchronously, i.e. they must execute with the similar input data and produce results before a scheduled voting instance. When a result is generated by the task, it is passed from a host CPU module to the RMS module. When the RMS modules agree on the voting process on and completes the voting operation, the result is verified and become available for further computations or I/O operations. Thus, a set of CPU modules of different cabinets can be regarded as a logical processing node if they are running the same set of tasks. The RMS modules maintain the consistency between the replicated executions automatically as long as enough execution and voting times are scheduled.

Scheduling Model

To build the scheduling algorithms in the proposed system, we may investigate the task processing model depicted in the following figure. The system consists of m processing nodes and each node has a proper degree of redundancy. Also included in the system is a voter (implemented by the RMS units) which verifies computation results before they are put out. We assume that there are n_i tasks allocated to

processing node i, where $1 \le i \le m$. A task TS_j^i will be invoked periodically with a period T_j^i. For each invocation, it takes a worst-case execution time (WCET) C_j^i at processing node i and produces at most k_j^i data items in the voting queue. The voting process of these items must be completed before the task's deadline D_j^i, where $C_j^i \le D_j^i \le T_j^i$.

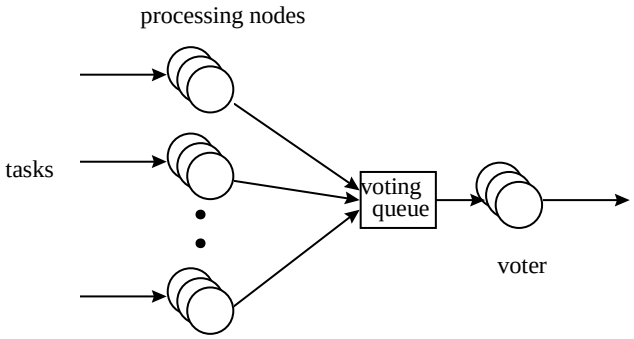

Figure 2 Scheduling model for task processing and voting

To model the operations in the RMS, we assume that voting cycles are initiated every VT seconds as shown in Figure 3. During each voting cycle, the voter takes the ready items from the voting queue and makes the verified data available at the end of cycle. Given b as the base overhead to perform the synchronization and to initiate voting actions, and τ as the voting time for each data item, the voter can verify at most $\lfloor (VT-b)/\tau \rfloor$ data items in one cycle. The remaining items in the queue and the newly arrived items will be voted in the subsequent cycles. Note that the voting queue is divided into two parts: an on-time arrival queue and an early arrival queue, where the data items in the on-time arrival queue has a higher priority to join the voting process than the data items in the early arrival queue. For each task TS_j^i, let VRT_j^i be the target voting ready time that its output data items enter the voting queue. This target voting ready time can be viewed as the execution deadline at the processing node and D_j^i - VRT_j^i becomes the deadline of the voting processing for task TS_j. If the task is completed earlier than VRT_j^i, the data items can wait in the early arrival queue. On the other hand, if its data items have not been voted before the target voting ready time, the items are waiting in the on-time arrival queue. With this two-queue scheme, we can avoid the impact of arrival jitters[1] to the scheduling process at the voter [13]. The worst-case arrival sequence to the voter occurs when all tasks release their data items at their target voting ready times, thus forming periodic requests to the voter.

In this paper, we explore the feasibility of using fixed priority preemptive scheduling [13] at the processing nodes and the voter. With static priority assignments at the processing nodes and at the voter, the task dispatchers can be easily implemented and the choice of data items to be voted in each cycle can be readily determined. Also, we can

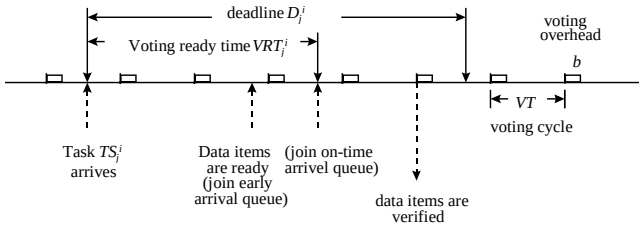

Figure 3. The timing diagram of voting operations

compute the worst-case response times for each task at the processing nodes and at the voter in order to check the schedulability. Given the execution priority $p_i(j)$ for each task TS_j^i, the worst-case response time RS_j^i at processing node i can be computed by the following recursive equation [18]:

$$RS_j^i = C_j^i + \sum_{l \in hp^i(j)} \left\lceil \frac{RS_j^i}{T_l^i} \right\rceil C_l^i$$

where $hp^i(j)$ is the set of tasks of higher priority than task TS_j^i at processing node i. Similarly, with the voting priority $p_v(i,j)$ for task TS_j^i, the maximum voting delay VD_j^i can be calculated as the sum of two parts: the initial waiting for the beginning of a voting cycle and the voting response time, VRS_j^i:

$$VD_j^i = VT + \left\lceil \frac{VRS_j^i}{VT} \right\rceil VT$$

$$VRS_j^i = \tau k_j^i + \left\lceil \frac{VRS_j^i}{VT} \right\rceil b + \tau \sum_{l \in hp^v(i,j)} \left\lceil \frac{\left\lceil VRS_j^i / VT \right\rceil VT}{T_l^i} \right\rceil k_l^i$$

where $hp^v(i,j)$ is the set of tasks of higher voting priority than task TS_j^i. The equation for VRS_j^i assumes that the data items from each task TS_j^i arrive periodically with a period T_j^i. This case occurs when all task computations are done at the target voting ready times.

3. SCHEDULING ALGORITHMS

As the task processing is decomposed into two stages at the processing nodes and the voter, a scheduling algorithm must provide the priority assignments $p_i(j)$ and $p_v(i,j)$. In addition, it must supply the target voting ready time VRT_j^i to begin the voting process. A feasible schedule is a set of $(p_i(j), p_v(i,j), VRT_j^i)$ for all tasks TS_j^i under which $RS_j^i + VD_j^i \le D_j^i$ [2]. Among all static priority schemes, we adopt a deadline-monotonic approach at each stage. Note that under a deadline-monotonic priority assignment, task priorities are assigned inversely proportional to the length of the deadlines

[1] Note that an arrival jitter caused by the variation of execution time can make the voting arrival process irregular.

[2] An unnecessary stringent definition of schedulability is to have $RS_j^i \le RL_j^i$ and $VD_j^i \le D_j^i - RL_j^i$.

and such an assignment is an optimal station priority scheme [3,12]. In fact, with deadline-monotonic approach, we only need to determine one of the three parameters, $p_i(j)$, $p_v(i,j)$, and RL_j^i, and solve the other two based on their dependency. Thus, if VRT_j^i is known, we can use VRT_j^i and $D_j^i - VRT_j^i$ as the deadlines at the processing node and the voter to assign $p_i(j)$ and $p_v(i,j)$ according to a deadline monotonic approach. Similarly, if $p_i(j)$ is known, we can compute the response time RS_j^i and assign it to VRT_j^i. Then, $p_v(i,j)$ can be determined.

The scheduling approach of determining VRT_j^i is similar to solving a 2-stage deadline distribution problem in which D_j^i is partitioned into two parts in order to meet the end-to-end deadline [7]. For a general deadline distribution problem, several heuristic approaches have been proposed for distributed real-time systems and dependent task sets. For instance, the deadline of a task can be evenly partitioned and used as the deadlines of its subtasks [4]. Search algorithms have also been applied for the problems, e.g., an iterative deadline assignment approach to improve the schedulability [17]. Recently, Di Natale and Stankovic suggested a slicing technique, which allocates slack time[3] to the subtasks in the critical path of a task graph according to normalized laxity and pure laxity metrics [15]. Under the normalized laxity metric, slack time is distributed in proportional to subtask execution times, whereas, under the pure laxity metric, each subtask receives an equal share of slack time.

Consider the computation and voting pipeline in our system. The voting processing time is expected to be much smaller than the task execution time at a processing node when the system has only one voter shared by all tasks. The pure laxity metric cannot be effective since, with an equal share of slack time, the computation deadlines become tight and the scheduling at each processing node can be feasible only if the processor utilization is low. On the other hand, the approach with normalized laxity metric assigns a bigger slice of slack time to the computation stage than to the voting stage. This can be a reasonable approach since the scheduling of short voting processes is easier than the scheduling of long task executions. The approach uses

$$VRT_j^i = C_j^i \left(1 + \frac{D_j^i - C_j^i - VT - \lceil \tau k_j^i / (VT - b) \rceil VT}{C_j^i}\right)$$

to set the deadline of task computation. Thus the priorities at the processing nodes and the voter, $p_i(j)$ and $p_v(i,j)$, can be assigned based on deadline-monotonic assignment. The schedulability can then be determined once we compute RS_j^i and VD_j^i, and check that $RS_j^i + VD_j^i \leq D_j^i$.

Instead of partitioning the deadlines, an alternate scheduling approach is to determine $p_i(j)$ and $p_v(i,j)$ directly. Let Φ and Θ be optimal priority assignments at the processing nodes and the voter. Denote $RS_j^i(\Phi)$ and $VD_j^i(\Theta)$ as the task response times at processing nodes and the voting delays at the voter for task TS_j^i under the priority assignments Φ and Θ, respectively. Given that deadline-monotonic is an optimal static schedule [3,12], we can easily observe that Φ and Θ must be deadline-monotonic priority assignments based on the deadlines $\{D_j^i - VD_j^i(\Theta)\}$ and $\{D_j^i - RS_j^i(\Phi)\}$, respectively. This dependence suggests an iterative approach which begins with a priority assignment at the processing nodes Φ_1. After $RS_j^i(\Phi_1)$ is computed, a deadline-monotonic priority assignment for the voter Θ_1 can be obtained. Then, a deadline-monotonic priority assignment for the processing nodes Φ_2 can be defined based on $VD_j^i(\Theta_1)$. The iteration can continue until either the priority assignments are feasible or there is no more improvement in schedulability. This scheduling algorithm, called DMA2, is given in the Figure 4.

Algorithm DMA2:

```
VD_j^i = ⌈τk_j^i / (VT - b)⌉VT;
old_tradiness=0; no_tries=0;
repeat
    for each processing node i {
        for each task TS_j^i  (1≤j≤n_i) {
            assign p_i(j) inversely proportional to
                D_j^i - VD_j^i;
        }
        compute RS_j^i for each task TS_j^i (1≤j≤n_i);
    }
    for each task TS_j^i (1≤j≤n_i and 1≤i≤m) {
        assign p_v(i,j) inversely proportional to
            D_j^i - RS_j^i ;
    }
    compute VD_j^i for each task TS_j^i
                (1≤j≤n_i and 1≤i≤m);
    tardiness = max { RS_j^i + VD_j^i - D_j^i };
    if (tardiness >= old_tardiness)
        no_tries++; old_tardines s= tardiness;

until { tardiness < 0,                    # schedulable
    or no_trial = a maximal threshold }
                    # too many unsuccessful trials
```

Figure 4. The DMA2 algorithm for priority assignments

Note that, in DMA2 algorithm, the computation starts with a zero voting delay initially. This initial setting leads to a deadline-monotonic assignment of $p_i(j)$ almost similar to the one based on the deadline D_j^i. In fact, as the voting delay is much smaller than the task computation time, this initialization selects an assignment that focuses on the task delay caused in the processing nodes. Also, DMA2 algorithm search for the priority assignments that are mutually deadline-monotonic. It may not be able to find such a pair of assignment or fail to find a feasible one. In these cases, the algorithm will terminate after the number of trials reaches a threshold.

[3] A task's slack time is defined as the difference between its deadline period and its total execution time.

4. SCHEDULING EVALUATION AND EXPERIMENTS

We performed several experiments to determine the performance of the slicing approach based on the normalized laxity metric and DMA2 priority assignment algorithm. Since the performance of the scheduling algorithms is likely affected by various parameters, we intend to set the values of the parameters in various ranges. The experimental environment has 4 processing nodes and each node has 6 tasks. Thus, data items generated by a total of 24 tasks will be voted. For each experiment, we collect the success ratios of the scheduling algorithms among 5000 random cases.

To define various parameters in the experiments, we first assume the voting overhead per each voting cycle, i.e., *b*, is a constant and is equal to one unit of time. The voting cycle is set to a period of 20, except the experiment that the period is a control parameter. The task periods are distributed uniformly in the range of [50, 500]. Then, for a given utilization at each processing node, we assign the task execution times such that the fraction of processor time spent in executing each task is randomly distributed in the range of [5%, 25%]. To determine voting processing times, we first assign the voting processing time for task TS_j^i to be uniformly distributed between [15%, 45%] of the task's execution time at the processing node. Then, all voting processing times are adjusted proportionally such that the overall utilization of the voter (including the voting initialization overhead) is in the range of 0.45 to 0.90.

Our first experiment is to examine the performance of slicing technique and DMA2 algorithm under different utilization. As shown in Figure 5, two sets of curves are plotted in which the utilization of each processing node is set to 0.45 and 0.65, respectively. The cases with a utilization of 0.65 probably have the highest utilization we need to consider since it is slightly less than the theoretical utilization bound (i.e. $\cong ln\ 2$) of rate monotonic algorithm in a single server [13]. The experiments assume that the task deadlines are equal to their periods, and the voting period is 20. The utilization at the voter varies from 0.45 to 0.90. The figure reveals several interesting properties of the two scheduling algorithms. When the utilization at the voter is less than 50%, both algorithms can reach a success ratio of 100% even if the utilization of each processing node reaches 0.65. Caused by the small voting processing times, the voter is likely to complete the voting process for any output data in one cycle and impose at most two cycle delay in verifying the correctness of output data. This delay doesn't make many disturbances in selecting a good scheduling at the processing nodes. However, when the utilization of the voter increases, the success ratios of the algorithms begin to diverge. A significant difference can be observed when the utilization of the voter is 0.75 and the utilization of the processing nodes is 0.65. The DMA2 algorithm has a success ratio of 0.98 whereas the slicing approach can make only 2% of the cases feasible

Figure 5 also shows substantial differences in the change of success ratios for each algorithm when the utilization of the

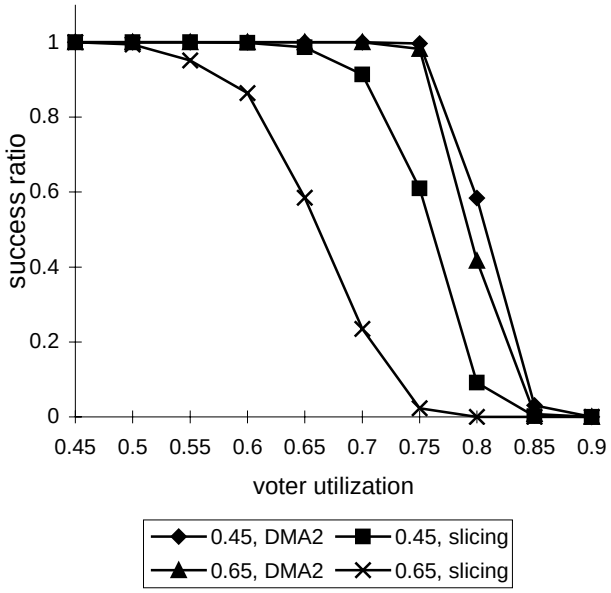

Figure 5. The success ratios of slicing and DMA2 algorithms measured in Experiment 1

processing node varies from 0.45 to 0.65. With a voter utilization larger than 0.55, the success ratio under the slicing approach drops drastically with an increase of processor utilization. It suggests the setting of voting ready times does not weigh the increased delays at both the processing node and the voter. Conversely, DMA2 algorithm can adjust the priorities properly, thus avoid the adverse effort caused by the increase of utilization at the processing nodes. This scheduling implication can be explained in a simple example. Consider two tasks with slightly different deadlines. If their ratios of the execution time at the processing node to that at the voter are similar, the deadlines are distributed to each stage with the same proportion. Then, the task with a shorter deadline will be assigned a higher priority at both the processing node and the voter. It can be completed much earlier in the expense of the task with a longer deadline that may miss its deadline. On the other hand, under DMA2 algorithm, the task receiving a higher priority at the processing node is likely assigned with a lower priority at the voter. Thus, the total delays of these two tasks are limited due to the shuffle of priorities, and can meet the deadlines as long as the utilization is not high.

To reassess the priority shuffling under DMA2 algorithm, we conduct the second experiment that compares the success ratios of two cases. The utilization at the processing nodes and the voting cycle are fixed at 0.45 and 20, respectively. In the first case, we set task deadlines randomly in the range of 60% to 100% of the task periods, whereas the second case assumes task deadlines are identical to the task periods. As the deadline shrinks, we expect that the tasks with long response time and voting delay will miss their deadlines. Thus, by examining the success ratios, we can detect the existence of long task delays. The results of the experiment

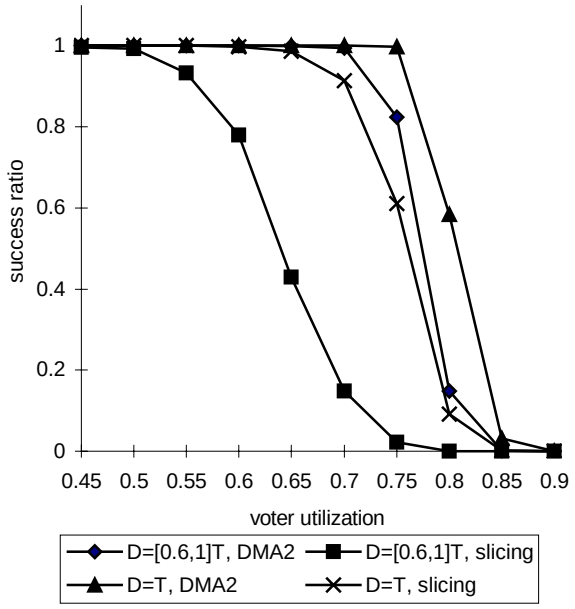

Figure 6. The success ratios of slicing and DMA2 algorithms measured in Experiment 2

are shown in Figure 6. The reduction of deadlines leads to a bigger decrease of success ratio for the slicing approach than for DMA2 algorithm. This difference implies that the slicing approach may result in a delay distribution that has a long percentile. Also, the effect of priority shuffling under DMA2 algorithm can eliminate the long percentile and allow individual tasks to meet their deadlines.

As the first two experiments assume that all processing nodes have the similar utilization, we look into the cases that the loads in processing nodes are not balanced in our third experiment. We set the average utilization of processing node to 0.55. Under the balanced case, each node has an equal utilization, whereas, in the unbalanced case, the utilization is set to 0.45 and 0.65 for two pairs of nodes. The success ratios with the slicing approach and DMA2 algorithm are shown in Figure 7. The success ratios under DMA2 algorithm are almost identical in both balanced and unbalanced cases. Apparently, the priority assignment at the voter adapts to the utilization of each processing node by comparing D_j^i - RS_j^i. Thus, the task which experiences a long computation delay in a heavily loaded processing node can still meet its deadline by holding a high priority at the voter. On the contrary, the slicing approach fails to make any compensation to the variation of node utilization as it does not check the loading situation at all in the determination of the target voting ready times.

Our next experiment is to investigate the effect of various periods of voting cycles. To make the synchronization between RMS units easy, the voting is initiated periodically by a hardware clock. This results in a periodic voting server and an initiation overhead per each cycle. Consider the impact of voting cycle times to the schedulability. We can conjecture that a short period may increase the utilization of the voter, and a long period may bring up additional waiting time to the voting requests. These suppositions are illustrated in Figure 8 that plots the success ratios with the periods vary from 5 to 50. Note that, in all cases, the utilization due to voting process (excluding the synchronization overhead) is set to 0.55. By adding the synchronization overhead per cycle, the net utilization at the voter varies from 0.75 to 0.57. In addition, the figure also confirms the stability of the DMA2 algorithm against any change of processing load and voting cycle.

Comparing the cases of experiments 1 and 4, we may observe that feasible schedules can be found easily when the voting period is short. In experiment 1, we have a case that the utilization of the voter is 0.75 and the voting period equals to 20. With the same utilization, we have a case in experiment 4 in which the voting period equals to 5. The resulting success ratios are quite different, i.e., much higher for the cases in the experiment 4 than that in the experiment 1. As our voter begins a voting process periodically, the voting delay increases linearly as we prolong the voting period. A good design of the voter should maintain a utilization less that 65% while choosing a small voting period.

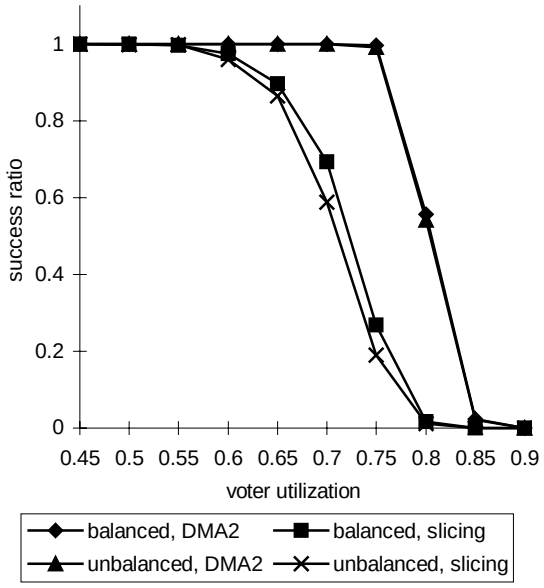

Figure 7. The success ratios of slicing and DMA2 algorithms measured in Experiment 3

Our last experiment is to investigate the effect of task periods to the schedulability of these two algorithms. In the previous experiments, the task periods are uniformly distributed between 50 to 500. Thus, the maximal period is likely several times more than the minimal period. If we change the range in which the task periods are distributed, it may get difficult to find a feasible schedule as that the most difficult scheduling condition occurs when all periods are less than the twice of the minimal period [13]. We assume that the mean of task periods is 275. The range is set to 275-*task_period_range* to 275+*task_period_range*, where

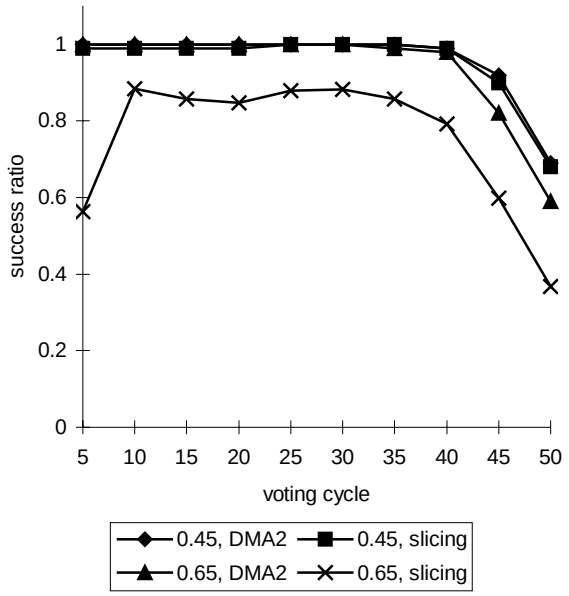

Figure 8. The success ratios of slicing and DMA2 algorithms measured in Experiment 4

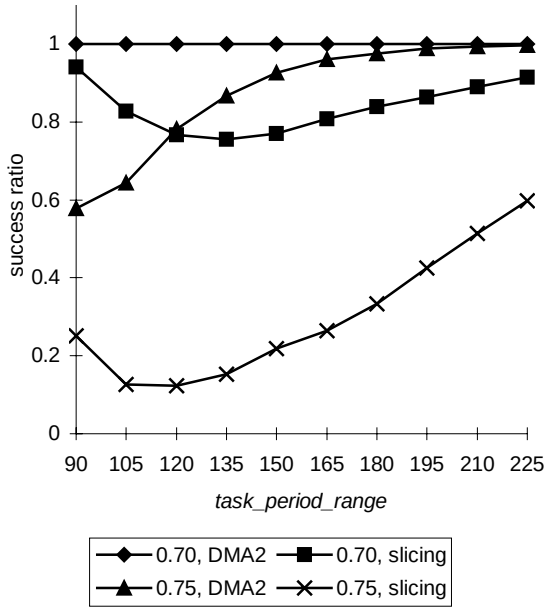

Figure 9. The success ratios of slicing and DMA2 algorithms measured in Experiment 5

task_period_range varies from 90 to 225. The experiment also assumes that the utilization of the processing node is equal to 0.45. Two sets of curves are plotted in Figure 9 to represent the cases that the voter utilization is equal to 0.70 and 0.75, respectively. They illustrate the property that the scheduling algorithms result in different success ratios as the range of task periods changes. For instance, with a voter utilization of 0.75, the success ratios of various period distributions may drop more than 50% and 40% under the slicing approach and DMA2 algorithm, respectively.

The results from the 5 experiments clearly indicate that DMA2 algorithm outperforms the slicing approach in determining feasible schedules, and is robust under various conditions. In fact, the experiments show that feasible schedules can be found even if the utilization at the processing node and the voter closes to the theoretical bound of rate-monotonic algorithm. This finding is interesting since, in order to make a schedule feasible, we expect there is a need to limit the utilization at the two processing stages such that the sum of the experienced delays at both processing stages is less than the task deadline. However, when DMA2 algorithm is able to shuffle the priorities at the two stages, no task needs to undergo long delays at both stages and feasible schedules can be obtained when the utilization is not high.

5. CONCLUSION

In this paper, we have proposed an effective scheduling mechanism that can be incorporated in fault-tolerant IMA systems. The emphases are placed in the issues of designing fault resilience at system level and scheduling consistency checking operations along with task execution. To find feasible solutions for fixed priority scheduling, we have examines two priority assignment schemes and evaluated their corresponding performance through a set of experiments. In addition, the measured data indicates a suitable design space in terms of the utilization at each processing node and the RMS unit. This result will be extremely useful in preparing system requirements in the early stage of the design process.

To continue the research work in the integrated mechanisms for scheduling and fault tolerance, we plan to investigate the scheduling algorithms in the APEX (application/executive interface) environment. The APEX environment calls for a partitioning approach to set up fault containment [2]. The temporal partition limits the execution of a task set to specific partition windows of a major time frame. Apparently, this deterministic approach favors a cyclic task scheduling which in turn results in a deterministic checking process. In this context, we need to look into efficient approaches of sharing the checking mechanisms among partitions, and to schedule partitions, the task set within each partition, and consistency checking operations.

REFERENCES

[1] "Design guidance for integrated modular avionics," ARINC Report 651, Aeronautical Radio Inc., Annapolis, Maryland, Oct. 1991.

[2] "Avionics application software standard interface," ARINC Specification 653, Aeronautical Radio Inc., Annapolis, Maryland, Jan. 1997.

[3] N. Audsley, A. Burns, M. Richardson, and A. Wellings, "Hard real-time scheduling: the deadline-monotonic approach," *Eighth IEEE Workshop on Real-time Operating Systems and Software*, 1991, pp. 133-137.

[4] R. Bettati and J. W.-S. Liu, "End-to-end scheduling to meet deadlines in distributed systems," *Proc. Of the IEEE Int'l Conf. on Distributed Computing Systems*, 1992, pp. 452-459.

[5] R. P. G. Collinson, *Introduction to Avionics*, Chapman & Hall Publisher, 1996.

[6] O. Gonzalez, H. Shrikumar, K. Ramamritham, and J. A. Stankovic, "Adaptive fault tolerance and graceful degradation under dynamic hard real-time scheduling," To appear in *Proc. IEEE Real-Time Systems Symposium*, Dec. 1997.

[7] J. Jonsson and K. Shin, "Deadline assignment in distributed hard real-time systems with relaxed locality constraints," *Proc. Of the IEEE Int'l Conf. on Distributed Computing Systems*, 1997, pp. 432-440.

[8] R. Kieckhafer, C. Walter, A. Finn, and P. Thambidurai, "The MAFT architecture for distributed fault tolerance," *IEEE Trans. on Computers*, Vol. 37, No. 4, 1988, pp.398-405.

[9] C. M. Krishna and G. Shin, "On scheduling tasks with a quick recovery from failure," *IEEE Trans. on Computers*, Vol. 35, No. 5, 1986, pp.448-455.

[10] J. H. Lala and R. E. Harper, "Architectural principles for safety-critical real-time applications," *Proceedings of the IEEE*, Vol. 82, No. 1, Jan. 1994, pp. 25-40.

[11] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Algorithm," *ACM Trans. `Programming Languages and Systems*, No. 4, 1982, pp. 382-401.

[12] J. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks," Performance Evaluation, Vol. 2, No. 4, 1982, pp. 237-250.

[13] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in hard real time environment," *J. Assoc. Comput. Mach.*, Vol. 20, No. 1, 1973, pp.46-61.

[14] C. D. Locke, "Software architecture for hard real-time applications: Cyclic executives vs. Fixed priority executives," *The Journal of Real-Time Systems*, Vol. 4, No. 1, 1992, pp. 37-53.

[15] M. Di Natale and J. Stankovic, "Dynamic end-to-end guarantees in distributed real-time systems," *Proc. IEEE Real-Time Systems Symposium*, Dec. 1994, pp. 216-227.

[16] Y. Oh and S. Son, "Enhancing fault-tolerance in rate-monotonic scheduling," *Real-time Systems*, Vol. 7, 1994, pp. 315-329.

[17] M Saksena and S. Hong, "An engineering approach to decomposing end-to-end delays on a distributed real-time system," *Proc. of the IEEE Workshop on Parallel and Distributed Real-time Systems*, 1996, pp. 244-251.

[18] K. Tindell, A. Burns, and A. Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks," *Real-time Systems*, Vol. 6, 1994, pp. 133-151.

[19] C. J. Walter, "Evaluation and design of an ultra-reliable distributed architecture for fault tolerance," *IEEE Trans. on Reliability*, Vol. 39, No. 4, 1990, pp. 492-499.

Yann-Hang Lee *is an associate professor with the Computer and Information Science and Engineering Department, University of Florida. His research interests include real-time systems, fault-tolerant computing, communication networks, computer architecture, and performance evaluation. He co-edited two special issues in Real-Time Systems in IEEE Computer (May 1992) and IEEE Proceedings (Jan. 1994), and co-chaired the Real-Time system Symposium, 1996. He received his Ph.D. degree in Computer, Information, and Control Engineering from the University of Michigan, Ann Arbor, MI, in 1984.*

Mohamed F. Younis *received the B.S. degree in computer science and the M.S. in engineering mathematics from Alexandria University in Egypt in 1987 and 1992, respectively. In 1996, he received his Ph.D. in computer science from New Jersey Institute of Technology. Dr. Younis is currently a research scientist with the AlliedSingal Advanced Systems Technology Group, Columbia MD, where he is leading multiple projects for building integrated fault tolerant avionics. His technical interests are in fault tolerant computing, system integration, real-time distributed systems and compile-time analysis.*

Jeffrey Zhou, *Sr. Manager of AlliedSignal Advanced Systems Technology Group, received his B.S. degree in 1980 from Shanghai Science and Technology University, M.E. degree in 1983 from Shanghai Jiao-Tong University, and Ph.D degree in 1989 from the University of Florida. His major is computer engineering specialized in real-time and fault-tolerant computing systems, complex computer-based systems and avionics systems. He joined AlliedSignal Aerospace in 1990 where he was involved in the development of the real-time engine simulator FAST. He also participated in the Shuttle Main Engine Control and Health Monitoring System (EC&HMS) program funded by NASA and was a key contributor for the development of the Real-Time Executive Module (RTEM), a real-time and fault-tolerant operating system for the EC&HMS program. Since 1996, he has been leading an AlliedSignal team in developing the Redundancy Management System which is a key component for the Vehicle and Mission Computer currently under the development for NASA's X-33 space launch vehicle. Dr. Zhou has served different committees and has numerous publications in his research field.*