Adaptive Fault Tolerance and Graceful Degradation Under Dynamic Hard Real-time Scheduling

Oscar González, H. Shrikumar, John A. Stankovic†, Krithi Ramamritham

Dept. of Computer Science, University of Massachusetts, Amherst, MA 01003
† Dept. of Computer Science, University of Virginia, Charlottesville, VA 22903
phone: (413) 545-4753 Fax: (413) 545-1249
e-mail: {ogonzale, shri, krithi}@cs.umass.edu, stankovic@cs.virginia.edu

Abstract

Static redundancy allocation is inappropriate in hard real-time systems that operate in variable and dynamic environments, (e.g., radar tracking, avionics). Adaptive Fault Tolerance (AFT) can assure adequate reliability of critical modules, under temporal and resources constraints, by allocating just as much redundancy to less critical modules as can be afforded, thus gracefully reducing their resource requirement.

In this paper, we propose a mechanism for supporting adaptive fault tolerance in a real-time system. Adaptation is achieved by choosing a suitable redundancy strategy for a dynamically arriving computation to assure required reliability and to maximize the potential for fault tolerance while ensuring that deadlines are met. The proposed approach is evaluated using a real-life workload simulating radar tracking software in AWACS early warning aircraft. The results demonstrate that our technique outperforms static fault tolerance strategies in terms of tasks meeting their timing constraints. Further, we show that the gain in this timing-centric performance metric does not reduce the fault tolerance of the executing tasks below a predefined minimum level. Overall, the evaluation indicates that the proposed ideas result in a system that dynamically provides QOS guarantees along the fault-tolerance dimension.

1 Introduction

1.1 Complex Systems need AFT

Mission-critical real-time applications require a reliable environment to guarantee that their deadlines are met despite the occurrence of faults either in hardware, software, workload, or the environment. Many current fault tolerance systems developed for real-time distributed applications are dedicated to specific applications, suffer substantial performance overheads, or require special hardware [15]. Static fault tolerance techniques allocate an amount of redundancy that is fixed at design time, and the system is sized appropriately. Under high loads with dynamic arrival, this capacity can be found to be inadequate leading to some critical tasks getting rejected while earlier less critical tasks have been guaranteed at high redundancy. Increasing system capacity to avoid this is often not meaningful due to system weight and power constraints inherent in embedded systems.

Flexibility in the management of redundancy is desirable in applications where drastic changes in environmental conditions and/or workload take place rapidly. Examples of such applications include air traffic control [5], digital avionics [3, 18], satellite systems and Radar Tracking (AWACS) and Digital Flight Control Systems.

If a unified framework can be designed to combine the different types of fault tolerance approaches and in addition, explicitly address real-time scheduling to meet timing and fault tolerance requirements, then some of the performance overheads and costs associated with static fault tolerance techniques can be reduced. Such a framework is essential for applications that operate in dynamic environments, and has not been adequately studied thus far.

What is desired in such environments is an assurance of high reliability for critical processes, and an attempt at the best possible allocation of the remaining resources to the less critical processes depending on dynamic arrival patterns. An efficient integration of fault tolerance and real-time scheduling is therefore called for. A promising approach for dynamic environments is the use of *adaptive fault tolerance* techniques in conjunction with a reflective real-time OS. *Adaptive fault tolerance* is defined as "an approach to meeting the dynamically and widely changing fault tolerance requirements by efficiently and adaptively utilizing a limited and dynamically changing amount of available redundant processing resources" [7]. The main advantage of this approach is the addition of flexibility for managing redundancy while preserving timing-related predictability [2].

1.2 AFT in a Hard Real Time System

A key issue in developing such a framework is the efficient integration of on-line adaptive management of redundancy and the real-time scheduler in a multi-processor hard real-time environment. Since an adaptive technique must function under timing constraints and effect the reconfiguration in a stable and predictable manner, it is desirable to implement it within the scope of a dynamic scheduler that guarantees predictable task execution.

In this paper, we take the Spring [20] system as a concrete example of a dynamic real-time system. With appropriate adaptations, our work should be applicable to other similar dynamic real-time systems.

The Spring system is a distributed testbed composed of multiprocessor nodes with both local and global shared memory as well as a reflective memory synchronized between the nodes with a fiber optic ring, and is supported by a dynamic planning mode scheduler, and related compiler tools.

The AFT strategies of each type of task is specified using the **`FERT`** notation [2], which is translated into *process groups* in the System Description Language, SDL, and submitted to Spring by the adaptive redundancy allocator, along with reflective information (e.g., importance, deadline, precedence constraints, fault tolerance requirement, etc.). An alternative would be selected in such a way that the corresponding process group has the highest specified reliability that can be guaranteed by Spring to finish by its deadline.

As new tasks arrive, the kernel takes advantage of this reflective information and attempts to guarantee each of them at the highest requested redundancy level that is feasible. An advantage of a planning-mode scheduler is its ability to predict that a timing constraint will be violated, enabling early action to handle the fault.

1.3 AFT Techniques Studied, and Results

To evaluate the performance of the AFT mechanism, we compare an adaptive fault tolerance technique with Triple-Modular Redundancy (TMR), Primary/Backup (PB) and Primary/Exception (PE) techniques. The set of alternatives in the adaptive technique is constructed from the three individual static techniques.

We first study the characteristics of different adaptive fault tolerance mechanisms using a synthetic workload. Then the results are finally verified using the realistic **`FERTstones`** benchmark [19], which simulates the demanding workload generated by the radar tracking application in AWACS airborne radar systems. The synthetic workload is parameterized for different overload and failure situations, since timing constraints in dynamic systems are often violated under these conditions. Also, experiments are conducted to determine the influence of tightness of the deadline, task value and the scheduling costs on the effectiveness of the adaptive technique.

Results of the performance tests indicate that all the AFT techniques studied achieve higher or comparable task schedulability compared to using static fault tolerance strategies in a real-time multiprocessor system, and they do this at a reliability level that is no worse than the minimum specified for the respective modules. Further, the best adaptive technique significantly increases the task schedulability without lowering the reliability of any module below specification by a graceful degradation of redundancy allocated to non-critical tasks.

A second important result is the observation that when the AFT mechanism selects an alternative strategy under dynamic arrival of tasks, it must be capable of rescinding prior strategy selections if the load generated by future task arrivals demand it. The AFT algorithm needs to be able to assure a minimal acceptable redundancy early upon task arrival, perhaps only provisionally guaranteeing any redundancy higher than that minimum. It must be able to postpone the final commitment of increases of redundancy until it becomes clear that such an increase can be actually afforded with a low probability of affecting the minimal performance of future tasks. Naturally, it needs to be able to do this rescinding of provisional acceptance of a higher strategy with an assurance of never dropping the task redundancy below acceptable limits.

Other contributions include : the development, implementation and evaluation of an adaptive fault tolerance mechanism supported by extensions to a real-time scheduler operating in planning-mode, a detailed performance evaluation that compares the adaptive strategy with static fault tolerance techniques; and easy and cost effective extensions to the scheduler to accomplish this selection.

The remainder of this paper is organized as follows: Section 2 discusses related work. Section 3 contains a description of the fault tolerance techniques and how they are integrated in our adaptive fault tolerant strategy. This section also describes the extensions made to the Spring kernel in order to dynamically support adaptive fault tolerance, and includes a discussion of the important design and research issues that require attention. We then discuss the metrics and results of the experimental study, first with a synthetic workload in section 4 and with a real-world workload in section 5. Finally, section 6 summarizes and concludes the paper.

2 Related Work

Static approaches as represented by MARS [9], and those capable of tolerating limited faults, as in [10], prove to be overconstrained and inflexible in dealing with dynamic overloads. This motivates the need for an integrated approach to fault tolerance and real-time scheduling.

Different modifications to static real-time scheduling algorithms to provide for redundant tasks, as by Liestman [13], and Krishna *et al.* [10] have been proposed. More recently, Oh *et al.* have studied the dynamic scheduling of copies of tasks on redundant CPUs [17], and in combination with Rate-Monotonic Scheduling. As far as we know, none of these projects has provided support for adaptive fault tolerance to deal with overloads and specifications of graceful degradation.

In [11], Laprie discusses the need for an assurance of dependability in complex systems, the kind where adaptive fault tolerance would be required. In [7], Kim and Lawrence provide a concrete definition of adaptive fault tolerance and identify major research issues that need to be resolved. The authors proposed that each distinct mode of operation of a system should have a different and effective set of fault tolerance mechanisms. As part of our study, we address two of the research topics discussed in [7]: adaptive decision under time constraints and cost-effective integration of fault tolerance techniques.

Bondavalli, Stankovic and Strigini introduce a framework for software implemented, adaptive fault tolerance in a real-time context [2]. The authors present the design of FERT (Fault Tolerant Entities for Real-Time), a specification notation which allows the abstraction of the functionality, timing constraint and adaptive fault-tolerance requirements of a particular application module. By using the design notation, a designer of a FERT is able to specify alternative fault tolerance strategies for a module. The information can be accessed by both off-line and on-line schedulers to select strategies, and control their execution, adapting to the actual load and fault situations [2]. The authors describe the on-line scheduling support needed by the framework. In our study, we implement a mechanism for supporting the on-line scheduling of adaptive fault tolerance techniques. A comprehensive survey in software fault tolerance and the necessary support mechanisms needed in the operating system can be found in [21].

Tai [23] introduces performability concepts and modeling methods to adaptive fault tolerance. The purpose of this work is to demonstrate the feasibility of performability-driven realization of adaptive fault tolerance. The target system is a multiprocessor environment supporting adaptation between Recovery Blocks and Distributed Recovery Blocks. Our fault tolerance techniques are different than the ones studied in [23] and the focus of our work is on dynamically scheduled hard real-time systems.

Ghosh, Melhem and Mossé [6] present a fault-tolerant scheduling approach for real-time multiprocessors systems. A Primary/Backup technique is used to schedule tasks. To achieve high schedulability and still provide fault tolerance, backup overloading and dynamic deallocation of backup techniques are incorporated for aperiodic tasks in a dynamic real-time multiprocessor system. We are interested in a similar task and system model, but consider other fault tolerance techniques in addition to Primary/Backup. Additional work in fault tolerant scheduling can be found in [4, 13, 17]. Adaptive fault-tolerance has been studied in the context of routing in multiprocessor interconnects, for example in [22].

Lee and Shin introduced an active reconfiguration strategy for a degradable multi-module computing system with a static set of tasks [12]. They recognized that the system should reconfigure itself after a certain amount of mission time has passed, even without any failure. Their model is a state-based approach which is represented as a Markov reward process. In [16] Muppala, Woolet, and Trivedi have combined two approaches for modeling soft and hard real-time systems. Their approach is based on the addition of transitions to the Markov model of a system's behavior for modeling a system failure due to the missing of a hard deadline. The system's response time and throughput distributions are used to denote the reward rates.

The Simplex architecture, in [1], provides a different approach to software fault tolerance. A high performance software system is combined with a highly reliable one in order to exploit their differences. This approach anticipates when

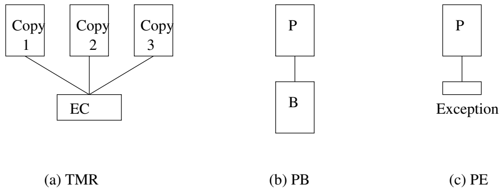

Figure 1. **Precedence Constraints**

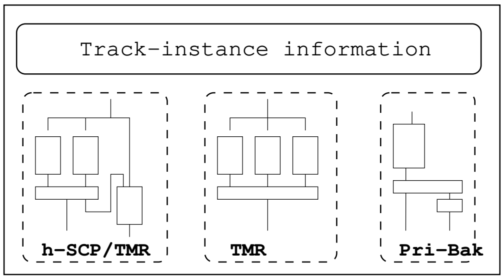

Figure 2. **Alternatives in `FERTstones`**

a failure is about to occur in order to activate an adaptation mechanism. When this happens, the high performance software is replaced by the reliable one. But this architecture does not deal explicitly with issues related to guarantee of schedulability.

3 Adaptive Fault Tolerance (AFT)

As mentioned in the introduction, adaptation is achieved by choosing a suitable fault tolerance technique from a set of alternatives for each instance of a dynamically arriving computation. The fault tolerance techniques considered as alternatives are *Triple Modular Redundancy* (TMR), *Primary/Backup* (PB) and *Primary/Exception* (PE). In all cases here, we have studied exact replicas, given identical inputs and producing identical outputs under fault-free conditions.

Triple modular redundancy (**TMR**) is a technique where three copies of a computation are scheduled on different processors, introducing location constraints. The copies are executed and error checking is done by comparing the results after completion [6]. The execution order between the copies and the error checking task (EC) is enforced by placing precedence constraints as shown in figure 1 (a).

In the Primary/Backup (**PB**) technique, two copies of a computation are scheduled on different processors (location constraints), but the backup task (B) is executed only in case the primary task (P) produces incorrect results. The precedence constraints between the tasks in PB are shown in figure 1 (b). Primary/Exception (**PE**) is similar to PB. The only difference is that in case of failure of the primary task, an exception handler is executed instead of the backup task, see figure 1 (c).

In all of the techniques the task graph as a whole has a single deadline.

In the case of the **`FERTstones`** benchmark, the **PB** strategy is used to implement a hybrid scheme, called hybrid-SCP-TMR. In this scheme, two copies are executed first, implementing a Self Checking Pair (SCP). If they terminate with unequal results, or a signaled fault, a third copy is executed as a backup and the three results are voted for a majority agreement, thus giving the same reliability level as (Triple Modular Redundancy) TMR, but with a lower CPU utilization under no-fault conditions. A suitable mechanism for ensuring exact consistency of inputs to the replicas is assumed, the costs of such a mechanism are subsumed within the execution time per task as seen by the real-time scheduler.

Fig. 2 shows the three different software redundancy schemes in a tracking FERT - hybrid-SCP/TMR scheme, TMR and Primary-Exception. Note that no one scheme is a winner in all circumstances. TMR is preferable when the slack is less than half the deadline, and a high reliability is desired; this preference is irrespective of global conditions. Under higher total system loads, the hybrid self-checking/TMR set becomes preferable, but is ruled out for tasks that have a slack less than half their deadline. Under heavier system loads, and lower reliability requirements, the primary-backup scheme is preferable, but again is applicable only to instances with large slack. Clearly this shows the need for adaptive fault-tolerance.

3.1 Characteristics of an Adaptation Mechanism

In order for a framework to effectively support adaptive fault tolerance within a real-time context, the adaptation mechanism needs to be *inexpensive, simple* to implement, and have a flexible and efficient *selection strategy*.

The adaptation needs to be as *inexpensive* as possible, in order to keep to a minimum the overhead costs introduced by the dynamic adaptation. Overheads must be small in planning-based scheduling algorithms operating in highly dynamic environments, so as to not impinge on the time already guaranteed to the application tasks or other system activities.

Simple and easy to understand schemes are not only more likely to be more reliable compared to a more complex one, but also enable us to obtain much needed data [7] to characterize and understand the behavior of adaptive fault tolerance techniques under various conditions.

A major problem in AFT is the implementation of an adequate selection strategy among the different fault tolerance techniques. We identify two types of selection strategies: (i) *sequential selection* and (ii) *dynamic selection*. In sequential selection, a list of different fault tolerance techniques created off-line *specifies the order* in which each of the techniques should be considered. In the dynamic strategy, an on-line evaluation module *decides the order* in which each of the fault tolerance mechanisms of the predefined set will be considered. The evaluation module takes into account the current system conditions (i.e., load, components failures, forecast of faults and their type, etc) to form the list of alternative fault tolerant techniques.

On the one hand, a clear advantage of the first form is its ease of implementation. There is no significant extra overhead cost added to the on-line scheduler. Ironically, its major disadvantage is its inability to adapt, at a finer level of granularity, to new environmental conditions. The dynamic strategy will not only add cost to the scheduling process, but requires the implementation of the evaluation module. We believe that finding an appropriate parameter or mechanism for the evaluation module is an open research question and a better understanding of different sequential strategies under different conditions is needed before an effective dynamic selection strategy can be designed. Therefore, in this paper we report on work done using a sequential alternative selection mechanism.

3.2 The Adaptive Mechanism

Our adaptive fault tolerance (AFT) mechanism is based on the combination of the three software fault tolerance techniques described in the previous section.

Software modules are programmed in the **`FERT`** notation, where a **`FERT`** encapsulates a set of alternative strategies for the same functional goal. The fault tolerance mechanism appropriate for a particular task is chosen when it arrives.

The temporal parameters include the task's deadline, type of deadline and laxity. The redundancy strategy indicates the precedence constraints between the various subtasks and constraints on their layout (such as, redundant copies must execute on different processing elements) that make up the computation and the redundancy of the computation. Location constraints are placed at run-time. Finally, the selection strategy is an ordered list of possible fault tolerance techniques that should be considered at run-time.

As tasks arrive in the system, the scheduler uses information about the task and attempts to guarantee the new task. If the new task has an alternative list, then the scheduler attempts to build a feasible schedule using the first fault tolerant alternative on the list. If no guarantee can be provided using the selected alternative, the next alternative is selected and the scheduler makes another attempt. The process continues until a feasible schedule is found or the alternative list is exhausted. In the latter case, the task is rejected. While building a feasible schedule, the planning-mode scheduler takes into account the location of task code and precedence constraints that each fault tolerance technique requires.

In this paper, we evaluate three adaptive fault-tolerance techniques.

1. **AFT-3**: Adaptive Fault Tolerance where the alternative list has the following three options: TMR, PB and PE.
2. **AFT-2**: Adaptive Fault Tolerance where the alternative list has the following two options: TMR and PE.
3. **AFT-quarantine**: Adaptive Fault Tolerance, with rescindable provisional acceptance, where the run-time provides an irreevocable guarantee of the primary exception for all non-critical tasks, holding up the allocation of greater redundancy in quarantaine till such a later point in time when it becomes clear that it is least likely to affect allocation of resources to critial tasks.

Whereas AFT-2 and AFT-3 are easy to understand, AFT-quarantine perhaps needs some explanation. It allows for the provisional acceptance of some tasks, with a possibility of rescinding the guarantee upon the arrival of overloads in the future. Specifically, critical tasks are processed in the same way, but non-critical tasks are handled a bit differently. They are first split into two component tasks upon arrival, in a manner that is specified off-line. The first component reserves resources for enough redundancy to satisfy the minimum acceptable redundancy specification of the task in question. The second component attempts to increase the redundancy level allocated to the first component of the the same task. The second component is provisionally accepted but held in quarantine for a fixed delay, which is an adjustable parameter. A committed guarantee is given only at the expiry of the quarantine interval.

The above mechanism takes full advantage of the reflective architecture of the Spring system and its planning-mode scheduler. These two key features provide a solid foundation that supports adaptive decisions, with respect to the management of redundancy, under time constrains in a flexible, predictable and cost effective way. In our approach, the adaptive selection between the different fault-tolerance alternatives is driven by the current system state and timing constraints of tasks, enabling fast response to rapid changes in the environment.

4 Performance Evaluation, with Synthetic Workload

4.1 Synthetic Workload Generator

A task generator creates a task set and the workload for each of the experiments. The task set is described in terms of information such as worst case execution time, deadline, precedence constraints, and fault tolerance requirements. The generator also creates the workload by creating a sequence of aperiodic arrivals of the tasks in the set. Each task arrival is described by its arrival time and the type of fault tolerance technique associated with the task. The following parameters are used to generate the task set and workload.

- **Min_WCET, Max_WCET**: the minimum and maximum worst case execution time of each task (WCET).
- **Mean interarrival rate, a**: the task interarrival time is assumed to be an exponential distribution with mean a.
- **RP**: the replication probability. RP is used to determine the percentage of arrivals requiring fault tolerance. For example, if $RP = 0.5$ then 50% of the arrivals will be replicated. A uniform distribution between 0 and 1 is used to determine whether a task is replicated.
- **DF**: the deadline factor, a laxity parameter that denotes the tightness of the deadline.
- **L**: the length of the simulation in number of arrivals.

The worst case execution time, $WCET$, of each of the tasks is randomly chosen using a uniform distribution between the minimum and maximum worst case execution times. The deadline of a task is set to $2 * WCET + (1 + DF) * T$, where T is obtained uniformly between 1 and 25 units. Hence a tighter deadline can be specified by reducing the range of T or by reducing DF. For all of the experiments the value of Min_WCET is equal to 20 and the value of Max_WCET is equal to 40 units.

3000 dynamic arrivals are considered for each run. Tasks in the set are assumed to be independent of each other. The interarrival time is used to control the load of the system. All of the fault tolerance techniques are evaluated over a wide range of load conditions.

Each point in the graphs (Figures 3 - 10) is the average of 5 simulations runs. The variance for these averages were computed and they were within 5% of the average in each case.

4.2 Performance Metrics

The experiments were conducted using Spring's scheduling simulator. The simulator accurately reproduces many aspects of the actual Spring system. Support mechanisms for adaptive fault tolerance (section 3.2) were added to the simulator.

All the AFT mechanisms studied make their adaptive selections for every task arrival, and the best alternative selected at each instance. While this increases the scheduling cost for each task arrival, we found the overheads to be not significant. Further, since the experiments were designed to compare the maximum short-term gains in performance and reliability that can be extracted without unduly compromising minimum requirements for both performance and reliability in the long-term, reevaluating the dynamic selection of alternatives for each task arrival is appropriate.

In all of the experiments, the static fault tolerance techniques described in section 3 are compared with the adaptive techniques, also described in section 3.2.

Assuming independent faults, all of the techniques can detect a single transient or permanent fault. TMR and PB are able to tolerate the fault since task replication is used to achieve a correct result despite the occurrence of the fault. By increasing the redundancy of tasks, tasks are able to complete under the presence of faults. However this reduces the number of tasks which the system is able to guarantee. The simulations enable us to quantify the tradeoffs between performance and redundancy levels of each of the fault tolerance techniques.

To estimate the performance overhead costs of providing fault tolerance, all of the techniques listed above are compared to a system where no redundancy is provided. Throughout the experiments this baseline is referred to as the *No fault tolerance* technique (**No FT**). A real-time system with no provisions for redundancy is not desirable, since the presence of a fault could have catastrophic results. Hence, the Primary/Exception technique serves as a baseline to estimate the performance overhead where a minimum degree of reliability is provided.

The performance metric used throughout the simulation is the *guarantee ratio*. The redundancy metric used is *the average replication factor*. The definition of these metrics are:

Guarantee Ratio (GR) : ratio of the number of tasks guaranteed by the scheduler to the total number of task arrivals, separately for critical tasks and non-critical tasks.

Average Replication Factor (ARF) : ratio of the sum of the number of completed task copies to the sum of the number of total task arrivals to the system, separately for critical tasks and non-critical tasks. Given that fault tolerance is achieved via redundancy the rationale behind the use of ARF should be clear: the higher the ARF value, the higher the expected fault tolerance.

An exception task in the PE technique does not count as one copy, instead the number of copies used for an exception task is obtained by dividing the worst case execution time of the exception by the worst case execution time of the primary. In all of the experiments with the synthetic workload, this ratio was set to 0.2 allowing the exception task enough time to deal with the fault. Thus, the number of copies for a task instance schedule using TMR, PB, PE and NoFT are 3, 2, 1.2 and 1 respectively.

4.3 Experiments

The simulation experiments are divided into two types. In the first type of experiments (experiment 1 through 4), different fault tolerance techniques are compared to the no fault tolerant baseline (NoFT) in an effort to estimate the loss of schedulability caused by the added redundancy and measure their redundancy levels using the ARF metric. For these types of experiments a *fault-free model* is assumed. in order to understand the effect of different adaptive fault tolerance techniques under several conditions (e.g., overload, tighter deadlines, percentage of replication of tasks, etc.).

The second type of experiments is designed to demonstrate the efficacy and power of the adaptive technique using a realistic AWACS workload. The **FERTstones** benchmark suite generates a workload that simulates the highly variable conditions in a radar tracking system. The different AFT mechanisms were tested against this workload, and the performance of adaptive fault tolerance was verified as indicated by the first set of experiments.

In all the experiments the number of number of processors per node is 3, and the scheduling is performed on a separate systems processor, as described in [20].

4.3.1 Expt. 1: Performance and Replication Factor

The performance results of all of the fault tolerance techniques as a function of load without considering scheduling costs, with $RP = 1$ and $DF = 1$ are shown in Figure 3. As expected, the guarantee ratio of all of the fault tolerance techniques increases with decreasing load. The results show that Primary/Exception (PE) performs much better than TMR technique, in particular under overload conditions. For example, when the mean interarrival rate is 20 units, the guarantee ratio for PE is near 100% and the guarantee ratio for TMR is down to 55%. The performance of both adaptive techniques (AFT-3 and AFT-2) and Primary/Backup is very similar throughout the range of interarrival rates. The guarantee ratio of these techniques is acceptable even under overload conditions, for example 80% at interarrival rates of 20 units. The results show that adaptive techniques are bounded by PB. This is because the adaptive techniques first try to schedule a task using TMR, leaving less cpu time available for future tasks compared to the same task scheduled using PB. From the results, we can conclude, as expected, that redundancy affects the guarantee ratio. PE is the technique with the highest performance and it achieves similar level of schedulability as the no fault tolerance (NoFT) baseline, except in overload conditions. On the other hand, the adaptive techniques provide performance similar to PB.

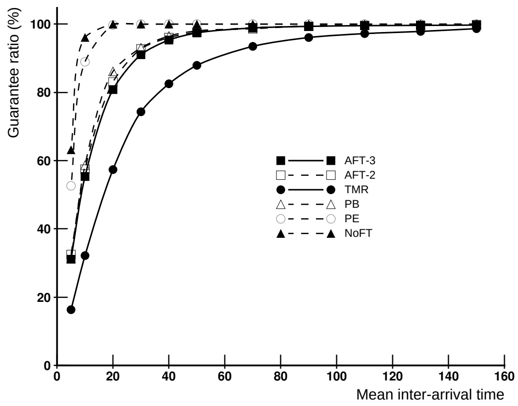

Figure 3. GR: no costs, RP=1.0, DF=1.0

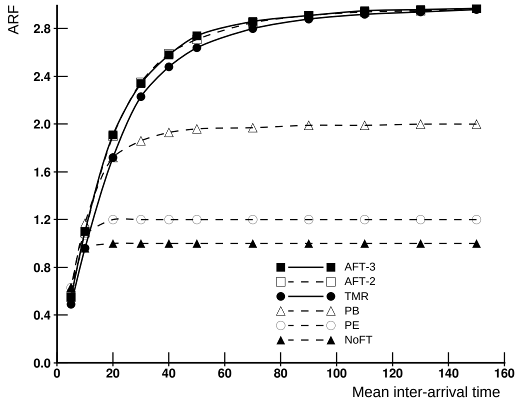

Figure 4. ARF: no costs, RP=1.0, DF=1.0

Figure 4 shows the average replication factor, of all of the fault tolerance techniques. The results illustrate the tradeoff between redundancy levels and performance for TMR, PB and PE. For mean inter-arrival values greater than 10, the ARF of the adaptive techniques is higher than that of PB and TMR, and the GR of the adaptive techniques is similar to PB and higher than TMR.

This clearly shows that adaptive techniques produce a higher average redundancy than any static technique, at a guarantee ratio that is comparable to PB. The guarantee ratio is poorer only to PE, which has no redundancy.

4.3.2 Experiment 2: Effects of Scheduling Costs

The effects of the Spring scheduling costs on the guarantee ratio and average replication factor are shown in Figures 5 and 6 respectively. The scheduling costs are calculated before each invocation as follows $SC = overhead\_cost + n * per\_task\_cost$, where n is the number of tasks to be scheduled

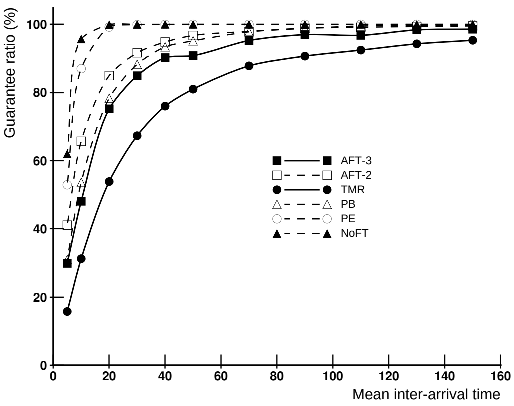

Figure 5. GR: with costs, RP=DF=1.0

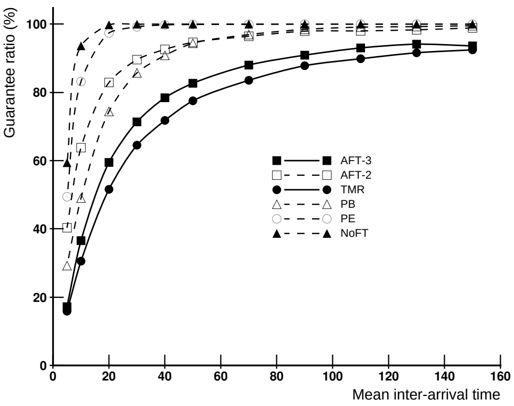

Figure 7. GR: w/costs, RP=1.0, DF=0.5

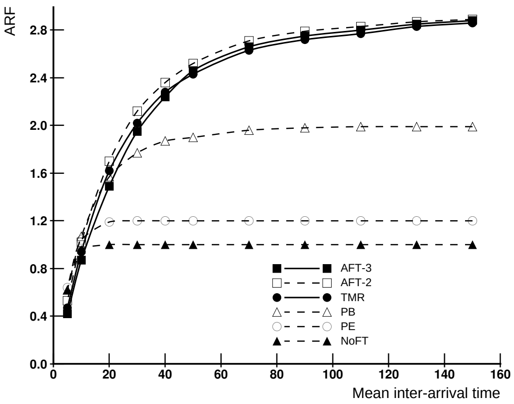

Figure 6. ARF: with costs, RP=DF=1.0

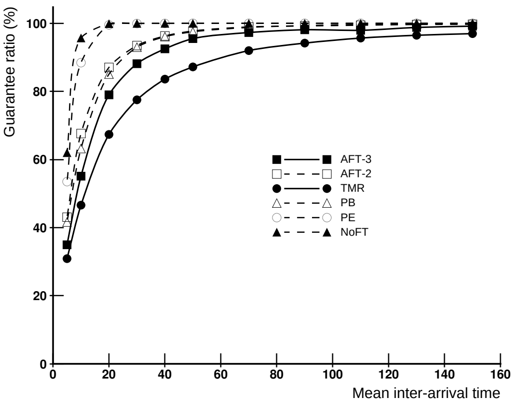

Figure 8. GR: w/costs, RP=0.75, DF=1.0

for the current invocation. The values used in the experiments for $overhead\_cost = 12.7$ units and $per\_task\_cost = 0.04$ units. These values produce scheduling costs that agree with measurements taken from the actual Spring system. It is important to notice that the scheduling costs are quite conservative, since these costs are on a 68020, which is quite slow compared to current processors.

Taking into account scheduling costs, we can see a decrease in the guarantee ratio for all of the techniques, with the exception of PE. Both TMR and AFT-3 are more sensitive to this effect under heavy loads. Under these conditions AFT-2 achieves a slightly better GR than PB. This is because the overhead of scheduling delays the start time of guaranteed tasks, which reduces the available interval for scheduling a task. A less drastic effect can be seen in the average replication factor. PB and TMR are affected more with respect to ARF. The ARF of both adaptive techniques (AFT-3 and AFT-2) is similar to the one obtained in experiment 1.

In conclusion, scheduling overheads do not significantly skew the results of Experiment 1.

4.3.3 Experiment 3: Effects of Deadline Factor

Figure 7 shows the guarantee ratio of all five techniques and the NoFT baseline, as a function of load with a smaller deadline factor (DF) and scheduling cost included.

It is clear that GR of all of the techniques is significantly lower under overload conditions. However, notice that the performance of AFT-3 drops for all interarrival rates. This is because AFT-3 is not able to schedule some tasks under TMR and PB due to the tigher deadline. AFT-2 is not affected as much as AFT-3 with tighter deadlines, since the scheduling of the feasible PE technique only inccurs one scheduling overhead cost. The important aspect is that under tight deadlines the performance of adaptive techniques with a significant number of *alternatives* may suffer. This result indicates that with tight deadlines *dynamic selection* strategy should be used.

The adaptive techniques continue to achieve high ARF compared to the other techniques, even as the deadlines of the tasks get tighter. In particular AFT-2 continues to perform well.

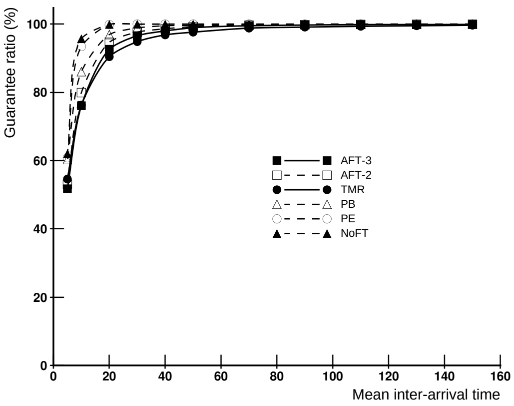

Figure 9. **GR: w/costs, RP=0.25, DF=1.0**

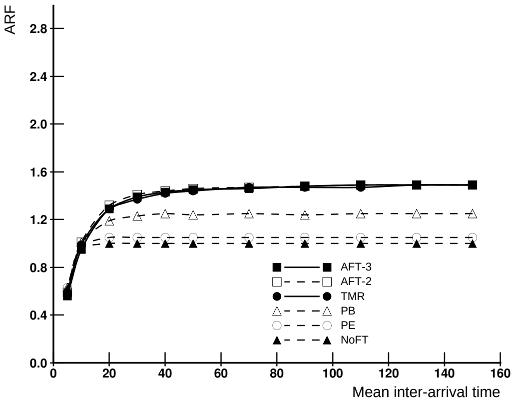

Figure 10. **ARF: w/costs, RP=0.25, DF=1.0**

4.3.4 Experiment 4: Effects of Probability of Replication

Figures 8 and 9 show the guarantee ratio of all the techniques as a function of load with a probability of replication of 0.75 and 0.25 respectively. As expected, the guarantee ratio for all of the techniques improves and the gap between them reduces (compare with Figure 5) as less tasks are replicated. TMR is the technique that benefits the most. Similarly, the average replication factor reduces for all of the techniques, see Figure 10 and 6 for probability of replication of 0.25 and 1.0 respectively. Therefore, to take advantage of the benefits of adaptive fault tolerance techniques, applications must need moderate to high level of replication of tasks.

Thus, static TMR is shown to be most sensitive to changes in probability of replication. Further, both adaptive techniques continue to perform adequately, in particular AFT-2 provides the best guarantee ratio at its average replication factor.

5 Performance Evaluation, with Real-World Workload

We wanted to evaluate our ideas in real-world environments in a way that different adaptive fault-tolerance approaches can be evaluated. To this end we studied several benchmarks in the literature, and finally developed the **FERTstones** workload generator. It is based on a radar tracking and control application. This was developed after a study of the **AOCP** program (Airborne Operations Control Program) used on the currently deployed **AWACS** aircraft (Airborne Warning And Control System), and certain anticipated requirements of future generations of similar systems. **FERTstones** furnishes a large variance of dynamically generated application load, and supports an adaptive fault-tolerance requirement specification.

Other benchmarks have been suggested in the literature [14, 24, 8]. However, these benchmarks do not make it easy to compare, qualitatively or quantitatively, results of different experiments of adaptive fault-tolerance mechanisms. Also, these do not lend themselves to experiments subjecting different implementations of adaptive fault-tolerance in comparitive tests under conditions that produces a workload similar to the high loads and high variability found in a real-world complex hard real-time application operating in a dynamic environment with adaptive fault-tolerance requirements.

5.1 FERTstones - Radar Tracking Benchmark

Radar tracking is a very good example of a complex software application operating under hard temporal deadlines that requires assurance of continued operation tolerant of hardware faults due to the hostile environment. Also, it operates in a dynamic environment subject to unpredictable overloads (due to targets arriving in bursts, and high-G maneuvers being performed by aircraft), so the worst case characteristics of the stimulus received by the system or the computational load generated by the software cannot be bounded. This makes the application interesting in that it captures some properties of a periodic real-time workload while at the same time modelling very bursty arrivals over larger periods of time. Such an application requires the engineering of a dynamic system capable of adaptive fault-tolerance, and hence is a good candidate for a benchmark.

The radar antenna carried in a radome above the aircraft has a range of 200 miles or more with a 360 degree view of the horizon and the ability to “look-down” on low-flying or surface targets. In a tactical role, such radars usually have a rate of scan of between 2 to 13 seconds, with targets capable of speeds greater that Mach 2.3 and maneuvers of 7-G or more. The **FERTstones** is modeled to simulate these conditions.

5.2 Characteristics of the Generated Workload

Radar returns are generated either by noise sources, or by reflection from actual target aircraft. A set of returns over multiple scans that corresponds to an identified target is classified as a confirmed track, each of which is tracked by one instance of a tracker task. Typically, about 1000 radar returns can be seen in every scan, each a potential target or threat. About a 100 of these prove to be confirmed targets that are tracked through multiple scans, but these number can show wide variations, and burstiness.

The benchmark simulates the CPU processing time demanded by the Multiple Hypothesis Testing (MHT) algorithm

Generator	Parameter
p_1 (target creation)	LogNormal(5.454,1.667) || Exp(0.007)
p_2 (lifetime)	Beta(3.000,1.455) * 720

Table 1. Standard Benchmark, `FERTstones`

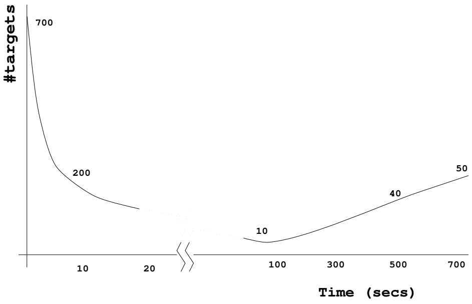

Figure 11. Target lifetime

[25], so named after the approach of hypothesising the maneuvers possibly being executed by a confirmed target from a previous scan, and extending its track with one of the returns from the current scan. The computation time required for each return depends on the number of such hypotheses to be tested per track, and therefore on dynamic conditions such as the number of radar returns that were observed with a small azimuth angle from a confirmed track.

Confirmed tracks require significant computation time, the requirement being proportional to the number of hypotheses being generated, ranging from 4.2 to 35.125 msec per track, and they are mandated to be run with the highest level of redundancy available. The remaining time is to be used to increase the redundancy level allocated to the unconfirmed tracks. Since this computation time for each unconfirmed track, 4.2msec, is too small to be handled efficiently by the the Spring dynamic scheduler, and since they occur in large numbers, they are bunched together to amortize the scheduling costs per invocation.

The **`FERTstones`** benchmark suite has composite probablisitic generators, listed in Table 1 and described below, that simulate the bursty arrival of true targets (aircraft), and noise echos in the radar. It also estimates the variability of the computation load under different target arrival patterns, and variation in target life-time on the radar.

5.2.1 Track creation

Tracks are generated by p_1 using two generators, one generating true targets (aircrafts) and the other simulating false reports (Table 1). Every track is specified as performing one input and one output interaction per iteration.

True targets are created with a log-normal interarrival time distribution, with the parameters $\mu = 5.545$ and $\sigma^2 = 1.667$. Noise reports are generated with an exponential distribution of interarrival times with a mean of 0.007 seconds.

The distinction between these two types of tracks is internal to the statistical generators, and not intended to be visible to the system under test. Both of these are combined together and presented as undistinguished new track creation events.

5.2.2 Track deletion

The library internally generates simulated life-times of each target, based on a simplified simulation of the expected lifetime of true and false tracks in a typical tracking radar system. It keeps track of event-lists of each track that it has issued an arrival event for, and it issues the deletion events in the appropriate chronological order.

The lifetime of targets is generated internally with p_2, using a beta distribution, with the parameters $\alpha_1 = 3.000$, and $\alpha_2 = 1.455$, with scale = 720 (Table 1). Noise returns only last for one scan, or 2 secs. This simulates the quick deletion of false tracks while the distribution of the lifetime of true tracks is consistent with the behavior of high speed aircraft overflying the range of the radar, and is shown in Figure 11.

5.2.3 Workload due to Multiple Track Instances

Each track-instance arrives with a requirement to be run at a certain specified redundancy level, which can then change dynamically. All confirmed tracks are run with the highest redundancy level that is feasible, with a preference given to hybrid-SCP-TMR and TMR over Primary-Exception. All unconfirmed tracks, typically tracks that have been in the system for less than two scans, are specifed to require a minimal redundacy level as provided by Primary-Exception, and any spare capacity available is allocated to them to enhance their redundancy level to hybrid-SCP-TMR or TMR, as feasible.

The application has statistical generators that trigger the birth and death of these tracks, as well as the promotion and demotion of their importance levels.

The scanning of the radar antenna every 2 secs is simulated, and independant sporadic events are generated every scan for every radar return from a target or noise source in the system.

5.2.4 Computational Load per Track Instance

The **`FERTstones`** libraries estimate the computational load associated with each iteration of each instance, simulating the MHT algorithm described in section 5.2. Once every scan cycle of the (simulated) radar (every 2 secs), the computational load presented by each track is updated. This number is an integral multiple ($n \geq 1$) of the nominal unit computational load per track.

The internal generator function, included with the benchmark, that simulates the estimation of computational load, assumes a target detection probability, P_D, of 0.9, and estimates the number of hypotheses or models applied to that track as a function of the total number of reports, both true and false, currently in the system. It generates this number by counting the number of radar reports that are in a specified neighbourhood of an existing track and therefore are candidates for track extension. This yields an execution time (WCET) for evaluation of each track between 4 msec and 36 msec.

5.3 Experiment 5: Testing with `FERTstones`

This final experiment was first performed with the **`FERTstones`** workload and the AFT-2 mechanism. For

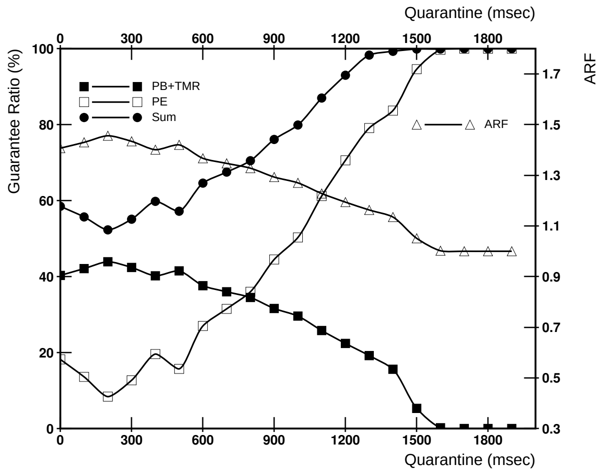

Figure 12. AFT-quarantine

the tasks representing confirmed tracks, we found AFT-2 to produce results consistent with the earlier experiments, as expected. However, interestingly we found that the larger deadlines in this workload led to a number of instances of tasks belonging to (less-critical) unconfirmed tracks being guaranteed with the alternatives of higher redundancy. The unrescindable commitment of the alternative selection of these tasks led to a significant rejection of similar tasks arriving later. Therefore, developing the AFT-2 and AFT-3 mechanisms further, we defined the AFT-quarantine mechanism, by guaranteeing only the minimal redundancy alternative upon arrival, provisionally accepting feasible alternatives requiring higher resources and holding the higher redundancy alternatives for a quarantine delay before committing on the allocation.

The amount of delay in the quarantine was varied as a tunable parameter, adjusted between 0 and $deadline$, and its effect on the guarantee ratio and replication factor were studied.

A delay of 0 is the case of the earliest possible commitment of resources to the alternative with the largest feasible redundancy, and is essentially the same as AFT-2. As expected, this strategy tends to generously overallocate resources to tasks that come earlier, at the expense of the minimal operating requirements of the tasks arriving later.

Figure 12 shows the variation of guarantee ratio with quarantine delay for the (less critical) tasks which represent unconfirmed tracks. As the delay is increased above zero, the guarantee ratio (sum = PE+PB+TMR, where PB implements h-TMR-SCP) goes up as more tasks are getting their minimal redundancy levels guaranteed. With a quarantine delay greater than 1300 msec, the guarantee ratio for the higher redundancy alternatives (PB+TMR) begins to drop quickly as there is not enough time left before the deadline to succesfully schedule them.

Figure 12 also shows the average redundancy levels (ARF) allocated to the tasks with different values of quarantine delay. It is useful to observe that with the introduction of quarantine delay, the average redundancy drops only very slowly initially. It begins to drop quickly to the redundancy level of the minimal alternative as the delay is increased beyond 1300 msec due to the delay being too much making the delayed copies infeasible to execute.

A good tradeoff is observed with a quarantine delay of between 800 to 1000 msec (approximately 40-50% of effective deadline), with which 69-80% of the nominal deadlines are met even under heavy overload. At the expense of a small lowering of the redundancy level of less critical tasks, (though never below the specified minimum and always without affecting the guarantee of the critical tasks), an increase in guarantee ratio of 10-20% over no quarantine is obtained.

Thus, this scheme shows a graceful degradation of the levels of redundancy provisionally allocated to tasks that have arrived earlier in order to favour transfer of resources to the minimal strategy for more arrivals.

6 Conclusion

The work presented in this article has integrated the framework for software implemented adaptive fault tolerance proposed in [2] with a dynamic real-time system in an effort to demonstrate the effectiveness of AFT. We have developed an efficient on-line mechanism that dynamically addresses the real-time constraints and fault tolerance properties of a software module. The idea of an adaptive strategy based on an ordered list of *alternative* fault tolerance techniques is presented.

A performance study shows the excellent performance of adaptive fault tolerance techniques in normal and overload conditions. The results were verified with a real-life workload, the **`FERTstones`** benchmark. This benchmark is based on a study of the computational needs of radar tracking software in AWACS early warning aircraft. We found that the adaptive approaches developed here are effective in that application.

Specifically, in experiments 1 through 4, we evaluated the performance of two adaptive fault tolerance strategies, **`AFT-2`** and **`AFT-3`**, with a very demanding synthetic workload. The performance of both was compared with static fault-tolerance strategies, and the impact of scheduling costs was also evaluated. It was concluded that **`AFT-2`** performed the best under most conditions of dynamic task arrival, and further outperformed all static fault-tolerance schemes.

We then tested the performance of the **`AFT-2`** mechanism with the realistic radar tracking workload generated by the **`FERTstones`** benchmark. While the results were found to be as expected, the longer deadlines in this workload suggested that further gains could be had by the introduction of provisional guarantees that are rescindable. This led to the development of of **`AFT-quarantine`** mechanism, which gave an additional improvement af 10% in the guarantee ratio.

We also examined various design issues related to the support mechanism of alternative selection in adaptive fault tolerance. Two types of strategies were introduced: (i) *sequential selection* and (ii) *dynamic selection*. In this paper, we evaluated sequential selection, which was shown to be simple, and yet quite effective. The dynamic selection strategy which interacts efficiently with an evaluation module might provide additional flexibility for managing redundancy under workloads that have tasks with very tight deadlines, but such tasks are not seen in the radar tracking benchmark and therefore this possibility is open for future work.

An important result is the observation that whenever an AFT mechanism selects an alternative strategy under dynamic arrival of tasks, significant gains in overall guarantee ratios were not obtained if the AFT mechanism was also incapable of rescinding prior strategy selection whenever future task arrivals proved to demand it. An effective policy was found to be one that (a) allocated resources for minimal redundancy levels soon after arrival to provide a high guarantee ratio but with minimum redundancy, (b) provisionallly accepted alternatives with greater redundancy and (c) committed the alternatives only later when it proves prudent.

Future work includes the development of algorithms that support adaptive fault tolerance in a distributed environment, performing additional experiments under different fault models and environment, evaluating other lists of alternatives, support for dynamic selection of AFT-strategy depending on some suitable system state-variables and comparing the performance of adaptive techniques with other models of scheduling which do not include planning. In addition, we need to address the deallocation of a backup (in the case of PB and PE) upon the successful completion of the primary component, since previous studies have demostrated [6] that resource reclamation increases the schedulability of real-time tasks on a multiprocessor system.

7 Acknowledgments

The authors wish to thank Gary Wallace for his suggestions in the implementation of the adaptive fault tolerance support mechanism. We acknowledge the support from Mitre Corporation and Mitsubishi Electric Research Laboratories.

References

[1] M. Bodson, J. Lehoczky, R. Rajkumar, L. Sha, and J. Stephan. *Fault-Tolerant Automatic Control, D. Fussell and M. Malek, eds.*, chapter 11. Kluwer Academic Publishers, 1995.

[2] A. Bondavalli, J. Stankovic, and L. Strigini. Adaptable Fault Tolerance for Real-Time Systems. *Proc. Third International Workshop on Responsive Computer Systems*, September 1993.

[3] D. Briere and P. Traverse. AIRBUS A320/A330/A340 Electrical Flight Controls A Family of Fault-Tolerant Systems. *Proc. of the 23th International Symposium on Fault Tolerant Computing, Toulose, France*, pages 616–623, June 1993.

[4] G. Buttazzo and J. Stankovic. *Adding Robustness in Dynamic Preemptive Scheduling, D. Fussell and M. Malek, eds.*, chapter 4. Kluwer Academic Publishers, 1995.

[5] F. Cristian, B. Dancey, and J. Dehn. Fault-Tolerance in the Advanced Automation System. *Proc. of the 20th International Symposium on Fault Tolerant Computing, Chapel Hill, North Carolina*, pages 6–17, June 1990.

[6] S. Ghosh, R. Melhem, and D. Mossé. Fault-Tolerant Scheduling on a Hard Real-Time Multiprocessor System. *Proc. IPPS-94*, pages 775–782, 1994.

[7] K. Kim and T. Lawrence. Adaptive Fault Tolerance in complex real-time distributed computer applications. *Computer Communications*, 15(4), May 1992.

[8] D. Kiskis and K. Shin. SWSL: A Synthetic Workload Specification Language for Real-Time Systems. *IEEE Transactions on Software Engineering*, 20(10), 1996.

[9] H. Kopetz et al. Distributed Fault-Tolerant Real-Time Systems: The MARS Approach. *IEEE Micro*, 9(1):25–40, February 1989.

[10] K. Krishna and K. Shin. On Scheduling Tasks with a Quick Recovery from Failure. *IEEE Transacations on Computers*, 35(5), May 1986.

[11] J. Laprie et al. Panel Session on Limits in Dependability. *Twenty-third IEEE Symposium on Fault-Tolerant Computing, Toulouse, France*, July 1993.

[12] Y. Lee and K. Shin. Optimal Reconfiguration Strategy for a Degradable Multimodule Computing System. *Journal of the ACM*, pages 326–348, April 1987.

[13] A. Liestman and R. Campbell. A Fault-tolerant Scheduling Problem. *Trans. Software Engineering*, 12(11), November 1988.

[14] J. Molini, S. Maimon, and P. Watson. Real-time System Scenarios. *Eleventh Real-time Systems Symposium*, December 1990.

[15] D. Mossé, R. Melhem, and S. Ghosh. Analysis of a Fault Tolerant Multiprocessor Scheduling Algorithm. *Proc. of the 24th International Symposium on Fault Tolerant Computing, Austin, Texas*, pages 16–25, June 1994.

[16] J. Muppala, S. Woolet, and K. Trivedi. Real-Time Systems Performance in the Presence of Failures. *IEEE Computer*, 24(5), May 1991.

[17] Y. Oh. *The Design and Analysis of Scheduling Algorithms for Real-Time and Fault-Tolerant Computer Systems*. PhD thesis, University of Virginia, 1994.

[18] R. Riter. Modeling and Testing a Critical Fault-Tolerant Multi-Process System. *Proc. of the 25th International Symposium on Fault Tolerant Computing, Pasadena, California*, pages 516–521, June 1995.

[19] H. Shrikumar and J. Stankovic. FERTstones A Synthetic Benchmark Suite for Complex and Adaptive Hard Real-Time Systems. Technical report, University of Massachusetts, 1997.

[20] J. A. Stankovic and K. Ramamritham. A New Paradigm for Hard Real-Time Operating Systems. *IEEE Software*, 8(3), May 1991.

[21] L. Strigini. Software Fault Tolerance. Technical Report PDCS 23, IEI-CNR, July 1990.

[22] C. Su and K. Shin. Adaptive Fault-Tolerant Deadlock-Free Routing in Meshes and Hypercubes. *IEEE Transactions on Computers*, 45(6), June 1996.

[23] A. Tai. Performability Driven Adaptive Fault Tolerance. *Proc. of the 24th International Symposium on Fault Tolerant Computing, Austin, Texas*, pages 176–185, June 1994.

[24] N. Weiderman and N. Kamenoff. Hartstone Uniprocessor Benchmark: Definitions and Experiments for Real-Time Systems. *Journal of Real-Time Systems*, 4, 1992.

[25] Y. Bar-Shalom. *Multitarget Multisensor Tracking, Advanced Applications*. Artech House, 1990.