An Experiment with Adaptive Fault Tolerance in Highly-Constraint Systems

Eltefaat Shokri
Sun MicroSystems
Eltefaat.shokri@eng.sun.com

Periklis Beltas
Nuera Communications
pbeltas@nuera.com

Abstract

Highly resource constrained dependable real-time systems have a very limited number of resources, which have to be managed dynamically through their lives. These systems also require self-contained fault-tolerance capabilities while operating under dynamic environments. Varying environmental conditions, dynamic application profile, and severe constraints in resource availability necessitates a flexible and adjustable resource allocation and fault-management strategy. In other words, the fault tolerance mechanism must be able to adapt itself to changes in both available resources and environmental conditions. A fault-tolerance mechanism that possesses such a characteristics is known as adaptive fault tolerance mechanism. This paper reports an experimental investigation effort for application of adaptive fault tolerance for autonomous spacecraft.

1. Introduction

In advanced space technology there is a need for autonomous spacecraft that can operate for a prolonged period of time, without intervention from a ground-based control center [Alk93, Joh93, And83]. On-board functions such as navigation and spacecraft subsystem management should be autonomous since:

- Communication with ground station may be possible for a short period of time, even during fault-free system operations.
- There is an increased need to minimize the cost of ground-based support during the long duration of a space mission.
- For missions in outer space the capabilities of ground control to respond in emergency situations are limited, mainly because of long round trip communication delays.
- In hostile environments the ground control could be interrupted for a long time, while continued spacecraft operation would be necessary more than ever.

To achieve the goal of autonomous operation the spacecraft should be designed such that the critical functions are executed autonomously, and without any severe performance penalties and/or safety loss, even in the presence of undesired events such as hardware problems, malfunctioning software, and excessive environmental hostility.

In addition autonomous spacecraft systems have a very limited number of resources, due to constrains in power consumption and spacecraft weight and size. Spacecrafts should also support self-contained fault tolerance capabilities [Alk93] while operation in potentially adverse and dynamic environments.

Autonomy, long life span, varying environmental conditions and severe constrains in resource availability necessitate a flexible and adjustable resource allocation and fault tolerance strategy. Thus the fault-tolerance mechanism to be applicable in such systems must be able to adapt itself to the changes in (i) available resources, (ii) environmental demands and conditions, and (iii) mission phases. A fault tolerance mechanism with such desirable characteristics is known as *adaptive fault-tolerance* mechanism (AFT) [Kim92, Sho97a].

This paper reports on a research and development effort to create a middleware that provides adaptive fault-tolerance mechanisms for spacecraft applications. The middleware was developed on top of a COTS operating system (VxWorks) and provides multiple forms of fault tolerance execution to satisfy different requirements for various phases of space missions as well as operation environments. The middleware also provides adaptation mechanisms to transition among these fault-tolerance execution modes in response to changing environmental and computational conditions.

2. Adaptive Fault-Tolerance Concepts and Architecture

2.1. Adaptive Middleware Concept

This work was done when the authors were with SoHaR Inc.

The main objective of AFT is to enhance system reliability, performance, and survivability through the following [Sho97b, Sho99]:

- *Enhance Resource Utilization*: The goal of the adaptation mechanism is minimizing resource utilization, while retaining required reliability requirements. AFT should enable the utilization of resources, or alternate the use of redundant channels.
- *Generality*: The AFT mechanisms must be sufficiently general to handle a variety of credible fault classes. More specific AFT should enable system designers to handle sensor, processor hardware, system software, and application software failures. The mechanism should be able to handle both transient and permanent failures either on the hardware, or the software.
- *Appropriate response*: The AFT mechanisms must respond to changes in the environment, mission phases, system state, and user profiles, within an acceptable time-period.
- *Configurability*: The user and/or developer of an application may provide application specific parameters that affect the adaptation decisions. An effective AFT manager should be able to accept these parameters during the execution of the application, and considering them during the decision-making process.
- *Versatility*: A suitable adaptation policy should consider various modes of adapting the system in response to anomalous behaviors. The adaptation can be executed for example in one of the following ways: (i) switching from one fault-handling mode to another, (ii) modifying parameters of the currently used fault handling mode, or (iii) modifying service attributes.

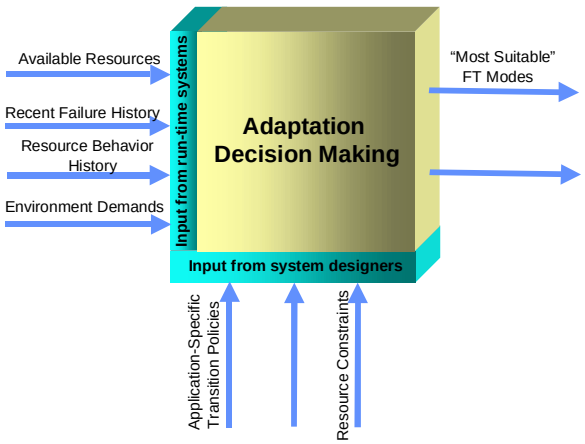

Figure 1: Adaptation Decision Making

- *Simple interfacing*: The AFT mechanism should be implemented as a separate layer and must have a simple and clear interface with the application, to maximize its openness and reusability. The application should interact with the AFT layer only through this interface.

In order to effectively respond to changes in mission requirements, resource availability, and user input the AFT should be initiated by the following events (Figure 1)

- *Changes in available resources*: When a resource becomes unavailable (as a result of a permanent failure or volunteer power off), AFT should intelligently respond to the change.
- *Recent Failure History*: The basic idea is that if the recent failure history indicates that the probability of more transient failures in the near future is very high, it is desirable for AFT to switch from an optimist fault-handling mode to a more pessimistic fault-handling mode. In the opposite case (low probability of transient failures) AFT should switch to a more optimistic (less resource demanding) fault-handling mode
- *Resource Behavior History*: A major goal of the adaptation process is to achieve an efficient resource allocation. A resource, which seems more reliable, should be utilized to perform high critical functions.
- *Environmental demands*: During the long life of a spacecraft, there are several phases with changing performance and reliability requirements. For instance, during the launch phase engineering data tasks should be executed more frequently, while other phases, such as in-orbit phase, the frequency of these tasks can be kept very small. An effective AFT should be able to recognize such changes in the emission requirements and adjust the fault-handling mode toward optimizing the resource utilization.
- *Application-Specific Transition Policies*: The system designer can supply the application specific inputs, such as application performance/reliability attributes and preferred transition policies.

2.2. Adaptive Middleware Architecture

The overall architecture of AFT is shown in Figure 2. The adaptive fault tolerance subsystem was implemented as a layer denoted as Adaptive Fault Tolerance Middleware (AFTM) such that will be reusable and easy to use, since the application designer can use the provided services through a well define API.

The middleware is built on top of a Generic Communication Layer, which encapsulates platform/protocol dependent aspects of the underlying communication subsystem and thereby provides the middleware with a generic set of inter-node

communication services. The middleware depends on time services of the operating system. Modern

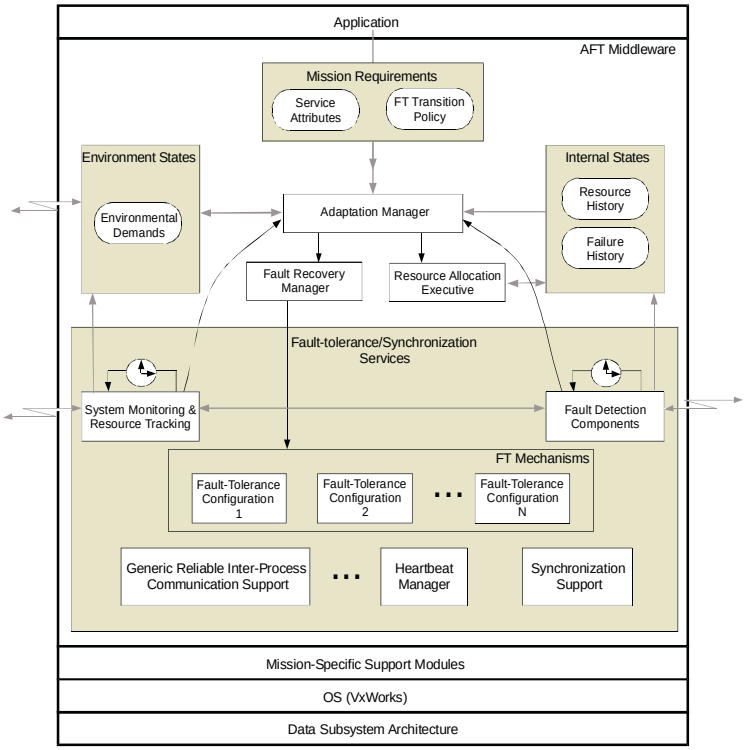

Figure 2. AFTM Architecture

commercial operating systems, such as Solaris and Windows NT, may provide the minimum requirements providing that the system operates in an isolated inter-network environment. The adaptation mechanism employed by the AFTM utilizes the following three databases [Sho99]:

Environmental Database: This database maintains a representation of the environment, in which the system operates, including any recent changes in environmental conditions or demands that should be taken into consideration by the AFTM. Updates to the database are made by sensors interfacing with the environment and by AFTM components responsible for monitoring and diagnosis (such as the Application Execution Monitoring component) based on the recent diagnosis knowledge.

Internal State Database: This database maintains the recent failure history as well as the history of recent behavior of system resources. The Network Reconfiguration Manager (NRM) and the Resource Allocation Executive (RAE) update this database when an anomalous behavior of a system component is diagnosed.

User Requirement Database: The user informs the AFTM about the specific characteristics and requirements of the application by providing (1) application-specific adaptation policies (if they exist), and (2) application attributes such as acceptable reliability and performance characteristics of each application task.

The Adaptation Manager (AM) component selects the most suitable fault-handling and resource-allocation modes of the system based on the current contents of these three databases. The adaptation decision made by AM is forwarded to the RAE component to enforce the changes by reallocating available resources.

Any change in the health status of the software and hardware components of the cooperating distributed AFTM, as well as the status of the application components, must be made known by the Adaptation Manager within an acceptable time period (i.e., a minimal delay). The Network Reconfiguration Manager (NRM) component is responsible for the fast detection of any anomalous behavior of software and/or hardware components of the application, or AFTM. NRM also reconfigures the network when a permanent malfunction of a resource is diagnosed.

The Application Execution Monitoring (AEM) component provides the user with the current status of application components. It is mainly responsible for monitoring the health status of active objects running in the nodes in the distributed system. This system monitoring facility enables the user to selectively monitor the health-status of various objects and instruct AFTM to take an appropriate action when the system enters an unexpected abnormal state.

3. The AFT Middleware: Applying the AFT

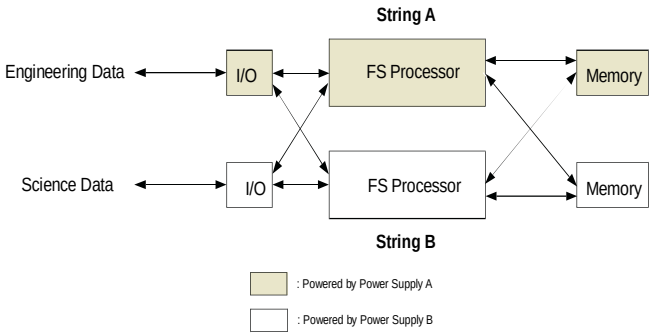

Figure 3. Simplified an autonomous spacecraft

concept to Spacecraft Application

3.1 System Architecture

This Adaptive Fault Tolerance (AFT) was based on the fault tolerance architecture proposed for the Pluto Express data system [Lab95, Cha95, Cha96.

As shown in Figure 3, the architecture is consisted of two identical processors (with local memories), which are powered by separate power sources. The speed of each processor can by adjusted dynamically by software. Under normal conditions one string is executing the engineer data tasks, while the other is executing the science data tasks. If a permanent failure of one processor occurs, then over the peer's responsibilities, doubling its speed if necessary. The AFTM extends this mechanism to respond not only to permanent failures in hardware components, but also to the frequency of transient failures in each processor as well as to changes in mission phases.

3.2. Fault Tolerance execution modes in AFT

Conventionally, application software consists of (i) the engineering subsystem for performing spacecraft functions, such as navigation and control, and (ii) the science subsystem for spacecraft specific missions such as gathering scientific data. Application software is usually composed of both periodic and non-periodic tasks. The majority of essential software such as guidance, navigation and control (GNC) are indeed periodic in nature. However there are application functions, which are supposed to be performed irregularly when needed. Depending on t emission objectives and requirements, application tasks can be specified as critical and non-critical tasks. Critical tasks should be executed as specified, otherwise the spacecraft survivability and stability cannot be guaranteed. On the other hand, if the system fails to execute some non-critical tasks for a short period of time, there will be no catastrophic consequences. Generally, engineering tasks are mapped into critical tasks. However there can be some science tasks, which are also considered critical.

In order to apply the AFT concept to the Pluto Express adaptive architecture an attempt was made to identify possible states for the spacecraft, based on three of the criteria from Figure 2, Recent Failure History, Available Resources, and Environmental Demands [Sho97b]. A simplified classification is shown in Figure 4.

The Recent Failure History is indicated either as "low rate of recent failures", or "high rate of recent failures". Although more detailed and complete classifications are possible, this classification enables us to make the adaptation process simpler and more responsive. Environmental demand is usually manifested by (i) changes in failure rate, or (ii) changes in the application attributes such as periods and execution deadlines. Since the first factor was considered earlier in Recent Failure History, changes in executions of application tasks was considered as the main factor for the environmental demands. As shown in Figure 5, eight different states were identifies, each of which needs a different task execution policy in order to maintain required reliability and survivability.

	Leader String			Follower String		
Config.	Speed	Execution Mode for Critical Tasks	Execution Mode for Non-critical Tasks	Speed	Execution Mode for Critical Tasks	Execution Mode for Non-critical Tasks
Mode 1	Single	Primary	---	Single	---	Primary
Mode 2	Double	Primary	---	Single	---	Primary
Mode 3	Single	Primary	Primary	---	---	---
Mode 4	Double	Primary	Primary	---	---	---
Mode 5	Single	Primary	---	Double	Backup	Primary
Mode 6	Double	Primary	---	Double	Backup	Primary
Mode 7	Double	Primary	Primary	---	---	---
Mode 8	Double	Primary	---	---	---	---

Figure 5: Characteristics of fault-tolerant execution modes

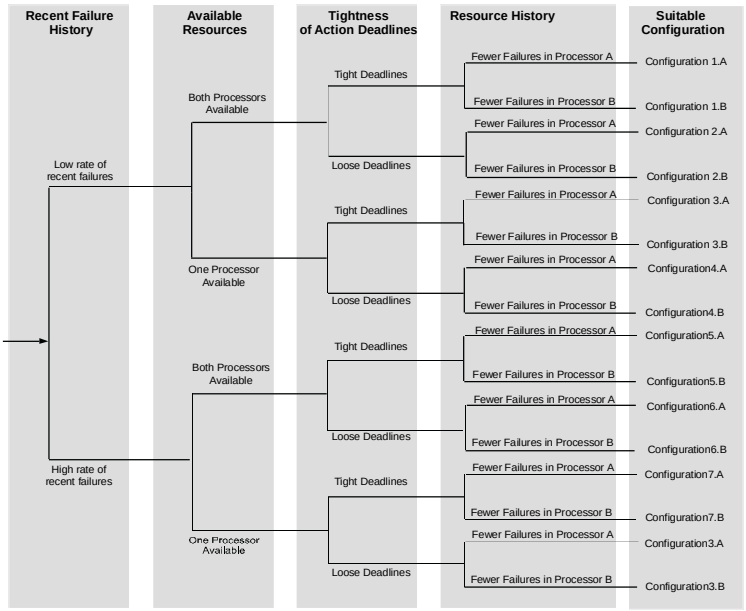

Figure 4: Classification of adaptation states

The following guidelines were used to decide on the execution modes for both critical and non-critical tasks [Sho97b].

If the recent failure is high, critical tasks should be executed concurrently in both processors (if possible) in order to reduce fault-recovery tolerance.

The load on the processor executing the primary copy of the critical tasks should be kept low to decrease the possibility of missing action deadlines.

If the deadlines are tight, such that the probability of late actions taken by critical tasks cannot be ignored, the processing speed of the

corresponding processors should be increased. However maximum care should be taken to minimize use of this option to optimize power usage.
In very rare cases, in which there is not enough processing power to reliability execute both critical and non-critical tasks, some of the non-critical tasks should be temporarily abandoned. However the duration of such periods should be kept minimal.

3.3 Fault Recovery Mechanisms Supported by AFT

The most essential factor in devising an adaptive fault-tolerance and resource allocation provision is to keep its development cost and execution overhead in acceptable ranges while maximizing the responsiveness of the system. Towards that goal a careful analysis was conducted in order to: (i) specify a realistic fault pattern, (ii) specify possible environmental conditions, (iii) designing an effective switching mechanism for tuning the operational fault tolerant execution mode, and (iv) specify

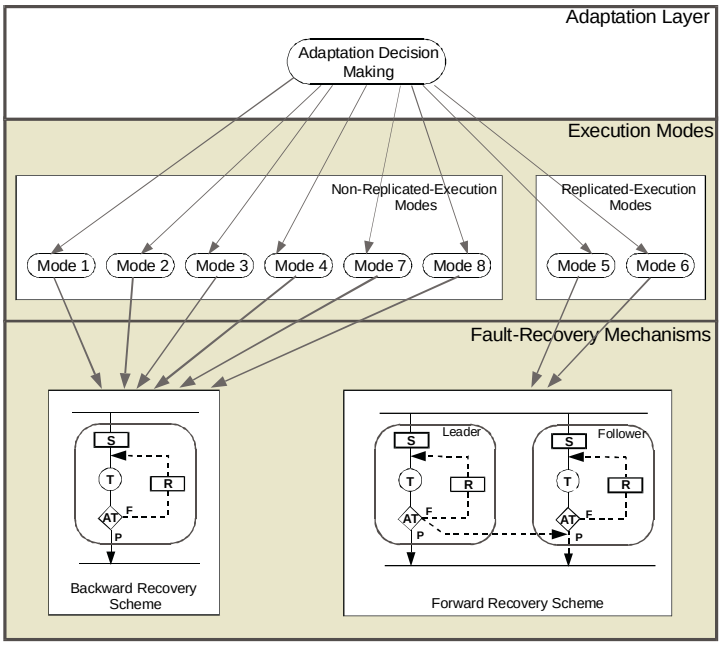

Figure 6: Fault recovery for the execution modes

a minimal set of fault-tolerant modes.

A minimal set of fault-tolerant execution modes was selected, because a careless selection of fault-tolerant execution modes may result in a large and very slow software, which is against the initial designing goals [Sho97b].

Finally the following fault tolerant mechanisms were supported: (Figure 6)

- *Non-Replicated Execution Mode*: Each task is executed in at most one processor.
- *Replicated Execution Modes*: In this case each critical task is executed in parallel in both processors. This precautionary action is taken to avoid long fault-recovery latency.

Non-replicated execution modes employ Rollback and Retry scheme [Ran75] to tolerate transient hardware and/or software faults. As shown in Figure 6, the state of the task is saved before the execution. Then the task is executed and in the case of a failure the acceptance test will detect the failure. Then the state of the task will be restored and the task (or an alternative version of it) will be executed. Under the replicated execution modes, a task (or an alternative version of it) will be executed using a variation of the PSP scheme introduced in [Kim95]. If the leader processor succeeds the correct execution of the task, it will take appropriate actions. If the leader fails then it will notify the follower processor, which will take the leadership and produce the results.

3.4 Generic Middleware Services

In addition the fault recovery mechanisms, which were described in 3.3, the middleware provides a number of additional services [Sho99]:
Vehicle Time Service: The purpose of it is to maintain a consistent time base on applications that can perform real time calculations and in which the middleware can be used as a basis for timeout detection.
Reliable Messaging: Using lower level communication primitives a high level abstraction for reliable communication among tasks with the following characteristics: (i) location transparent inter-task communications, (ii) Logical channel based communications, (iii) multicast communication and (iv) flexibility.
Replicated data management: The purpose of replicated data management is the maintenance of current consistent state data among replicated objects
Resource monitoring: It maintains the system status information needed by the middleware for reconfiguration and adaptation decisions. Such system status information is also needed by the telemetry function to provide status reports to the ground operation center.
Data System management: It provides the interface with the vehicle autonomy and other high-level decision makers to provide input to the adaptation manager.

4. Concept Demonstration

To illustrate the feasibility and potential benefits of the

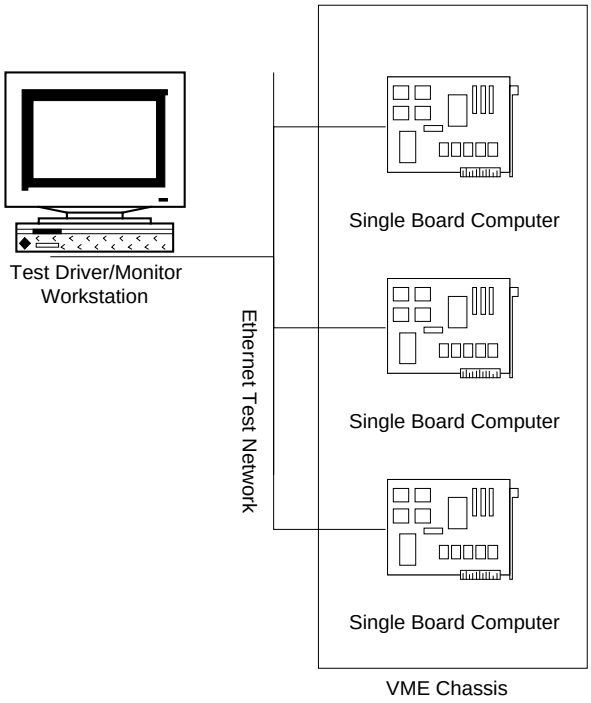

Figure 7 : Testbed environment

AFT design a demonstration version was implemented. Figure 7 shows the platform that was used for the demonstration [Sho99]. Three MC68040 single board computers are located in a MVME167 chassis and can communicate with each other, as well as with the SPARC/ Solaris computer using the Reliable Messaging facility. In order to isolate fault domains of the processors, the shared memory provided by the MVME was not used.

8. Conclusion

This paper described the results of a research and development effort on creating an adaptive fault-tolerance for spacecraft applications. The research has demonstrated the feasibility of applying fault-tolerance concepts in space applications.

9. Reference

[Alk93] L. Alkalaj, "Advanced flight computing program" in proceedings of 1st Workshop on Advanced Flight Computing and Industry, (Pasadena CA), pp. 104-166, 1993.

[And83] J. L. Anderson, "Space station autonomy requirements" in proceedings of AIAA Computers in Aerospace Conference, (Hartford, CN), pp. 164-170, 1983

[Cha95] S. Chau, "Pluto Express Data System:FY95 Final Report" technical report J.P.L. (Pasadena CA) 1995

[Cha96] S.Chau, "Pluto Express data system fault protection", technical report, J.P.L., (Pasadena Ca) 1996

[Joh93] G. I. Johnson, "Commercialization and technology transfer: A NASA view" in proceedings of 1st Workshop on Advanced Flight Computing and Industry", (Pasadena CA), pp. 62-74.

[Kim92] K. Kim and T. Lawrence, "Adaptive fault tolerance in complex real-time distributed applications" Computer Communication, vol 15, no 4, pp 243-251

[Kim95] K Kim, "The distributed recovery block scheme" in Software Fault Tolerance (M. R. Luy, ed.) pp. 189-207, John Wiley, 1995.

[Sho97a] E. Shokri et al, "An approach for adaptive fault-tolerance in object-oriented open distributed systems" in Proceeding of Third Workshop on Object-Oriented Real-time Dependable Systems, 1997.

[Sho97b] E. Shokri, "Adaptive Fault-tolerance for Autonomous Spacecraft", Phase 1 Final Report, 1997.

[Sho99] E. Shokri, "Adaptive Fault-tolerance for Autonomous Spacecraft", Phase 2 Final Report, 1999.