Proc. IEEE CS 1998 High-Assurance Systems Engineering (HASE)
Symp., Washington, D.C., Nov. 1998, pp.50-57.

ROAFTS: A Middleware Architecture for Real-time Object-oriented Adaptive Fault Tolerance Support

(Invited)

K. H. (Kane) Kim

Department of Electrical & Computer Engineering
University of California
Irvine, CA, 92697 U.S.A.
khkim@uci.edu

Abstract: A middleware architecture named ROAFTS (Real-time Object-oriented Adaptive Fault Tolerance Support) is presented. ROAFTS is designed to support adaptive fault-tolerant execution of not only conventional process-structured distributed RT application software but also new-style object-structured distributed RT application software. While ROAFTS contains fault tolerance schemes devised for quantitatively guaranteed real-time fault tolerance, it is also designed to relax that characteristics while the application is in a soft real-time phase in order to reduce resource use. Through three different prototype implementation experiences using both commercial OS kernels and home-grown real-time kernels, the middleware architecture has been refined and its basic capabilities and effectiveness have been validated. The fault tolerance schemes supported and the integrating architecture are discussed in this paper. Implementation experiences and some future tasks are also discussed.

Keywords: fault tolerance, adaptive, middleware, ROAFTS, object, real-time, adaptation, thread, time-triggtered, kernel, AFT, DRB, recovery, SNS, network surveillance.

1. Introduction

Many emerging large-scale safety-critical applications require distributed computing systems that can *survive*, i.e., meet all the critical computing requirements, through a specified extent of dynamic losses and acquisitions of computing resources. While technologies for reliably constructing such application systems have been pursued for about two decades, well integrated distributed operating system (OS) facilities supporting such *adaptive fault-tolerant (FT) execution* of real-time (RT) distributed application software are not fully available yet [Kim92, Law95]. However, in recent years, most of the needed cornerstones of desirable integrated *adaptive fault tolerance* (AFT) support facilties started appearing in concrete forms. Therefore, time seems ripe for establishing architectures for distributed OS facilities that can be composed of recently established cornerstones to support adaptive FT execution of real-time distributed application software.

The architectures possessing the following characteristics are particularly desirable.

(1) <u>Middleware implementation on COTS (Commercial Off-The-Shelf) OS's</u>: AFT in a distributed computer system is basically a distributed resource management function and the most cost-effective implementation of such services can nowadays be realized in the form of middleware running on a COTS OS.

(2) <u>Quantitatively guaranteed RT FT</u>: In a system designed to realize FT via allocating resources for redundant computation, effective, let alone optimal, resource allocation is not possible in the absence of quantitative characterizations of FT schemes. In such characterization, the most important metrics are (1) *fault types and rates covered*, and (2) *recovery time bound*.

(3) <u>Maximization of application survivability</u>: The ultimate goal of incorporating FT mechanisms in a RT computing application system is to maximize the *survivability* of the system *through its mission period*. In an environment where resources cannot be added during the mission period, the initially configured resources must be utilized *intelligently and adaptively* as some of them get lost and/or *quantum changes in the resource needs* occur. If the application allows adding new computing resource components during its mission period, the capability for *non-disruptive incorporation* of new components to enable "non-stop RT task execution" must be built into the system. Therefore, the distributed OS architecture that maximizes the application survivability via adaptive use of resources is the most desirable one.

ROAFTS (Real-time Object-oriented Adaptive Fault Tolerance Support) is a middlware architecture devised to possess the characteristics mentioned above. Chareristics (2) in the above makes the distributed OS facility to be somewhat expensive in terms of resource use. Therefore, ROAFTS is designed to relax that characteristics while the application is in a soft RT phase. An important additional characteristics of ROAFTS is that it is capable of supporting not only conventional process-structured distributed RT application software but also new-style object-structured distributed RT application software [Kim97a].

Although the major backbone of the ROAFTS architecture was established in 1996, some of its detailed parts have undergone some refinements as three different prototype implementations have been undertaken. The first prototype implementation was realized at the author's laboratory on top of our home-grown RT OS kernel, the DREAM kernel v4.0 [Kim97a], and the second implementation was realized by SoHaR, Inc., with the assistance of the author and his research assistants on SUN Solaris platforms [Sho97]. Using these prototypes, two major distributed RT application prototypes with AFT capabilities were constructed and their performance was evaluated: (1) A military command-control application prototype, and (2) a factory control application prototype. The third prototype implementation of ROAFTS, which is still under development, will run on Microsoft NT platforms.

In Section 2, major RT FT schemes which play the cornerstone roles in ROAFTS are briefly described. These techniques are effective in a broad range of application environments. The architecture of ROAFTS is discussed in Section 3. Our experiences in prototype implementation are discussed in Section 4 along with some meaningful directions to pursue in the future.

2. Major real-time fault tolerance schemes incorporated into ROAFTS

In devising ROAFTS, we considered only those FT schemes for which *recovery time bounds* can be easily established. It is our judgment that as the basic OS and communication subsystem technologies become mature, rapidly growing demands for quantitatively guaranteed RT FT will be generated from the customers of distributed applications, not just from the customers of single-node computing system applications.

A natural and modular way of constructing fault-tolerant distributed and/or parallel computer systems is to
(1) construct various subsystems in the rugged fault-tolerant forms,
(2) interconnect them by use of rugged fault-tolerant communication subsystems, and
(3) operating them under the supervision of a *network surveillance and reconfiguration* (NSR) manager which detects the needs for system reconfiguration, performs or coordinates reconfiguration, and invokes detailed system diagnosis tasks periodically or as needs arise.

The most basic type of subsystems are *computing stations* each of which consists of a processing node (hardware and software) dedicated to the execution of one or a few application processes [Kim94]. Every complex subsystem can be constructed with such computing stations.

A brief overview of basic FT schemes is taken first.

2.1. Fault-tolerant computing stations with backward recovery (running primarily soft RT tasks)

The computing stations here all have *rollback* capabilities.

Type 1: Simplex type (i.e., single-node stations) without any application-dependent fault detection software component

Figure 1 depicts this computing station.

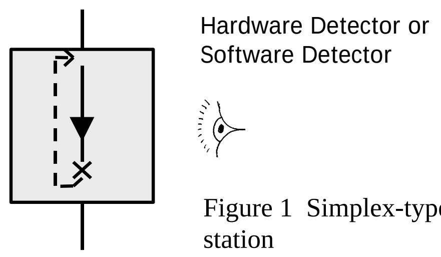


Figure 1 Simplex-type station

Systems using this approach must have either some fault-detection hardware or common (i.e., application-independent) fault-detection software components. Otherwise, systems cannot utilize their rollback capabilities and thus are not FT. A typical fault-detection software component considered here is one that does a consistency check on the data structures in OS. An extreme case in terms of the overhead requirement is to sequentially reexecute the same task and compare the results. In any case, the *fault coverage* achieved with common fault-detection software components in terms of the fault types and rates covered is somewhat limited.

Type 2: Duplex type (i.e., dual-node redundant operating stations) with result comparison

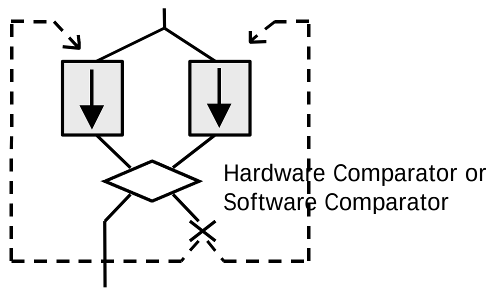


Figure 2. Duplex-type station

Figure 2 depicts this computing station.

The result comparison here can be performed by a special hardware comparator or a software-implemented comparator [Toy87]. The hardware comparator can be expensive in some applications and restricts reconfiguration scope. On the other hand, it incurs much less overhead in comparison to the software comparator.

Type 3: Recovery block station

Figure 3 depicts this computing station.

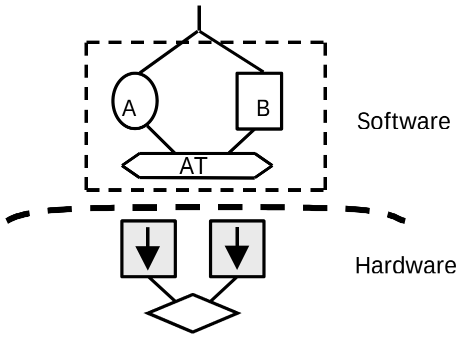


Figure 3. Recovery block station

The recovery block [Ran75, Ran95] station is essentially a Type-1 or Type-2 station plus at least one of the following two optional software components: (1) an *acceptance test* which is an application-dependent fault-detection software component and (2) alternate application algorithms. This computing station can handle both hardware and software faults to some extent but burdens the application task designer with the workload of providing acceptance tests and alternate algorithms.

2.2. Fault-tolerant computing stations with forward recovery (running primarily hard RT tasks)

Type 4: Pair of Self-checking Processing nodes (PSP)

Here two self-checking nodes execute the same application software.

Type 4a: Pair of Comparing Pairs (PCP) with result comparison

The result comparison can be performed by a

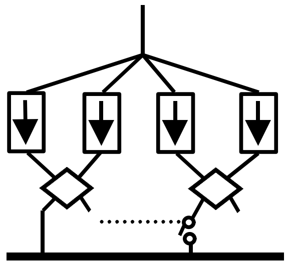


Figure 4. PCP station

special hardware comparator or a software-implemented comparator. Figure 4 depicts this computing station. The *primary-shadow cooperation* scheme is used in that the shadow pair delivers its output to the rest of the distributed computer system only when the primary pair cannot produce an output. If a hardware comparator is used, it becomes essentially the computing station architecture used in the Stratus machine [Wil85]. Since a computing station consists of 2 dual-processor nodes, each in turn consisting of 2 processors, the station requires four processors which could be viewed as expensive hardware requirements in some applications.

Type 4b: A pair of single-processor nodes without any application-dependent fault detection software component

This is essentially a pair of Type-1 stations using the primary-shadow cooperation scheme [Kim94, Kim95].

Type 5: Distributed recovery block (DRB) station

Figure 5 depicts this computing station.

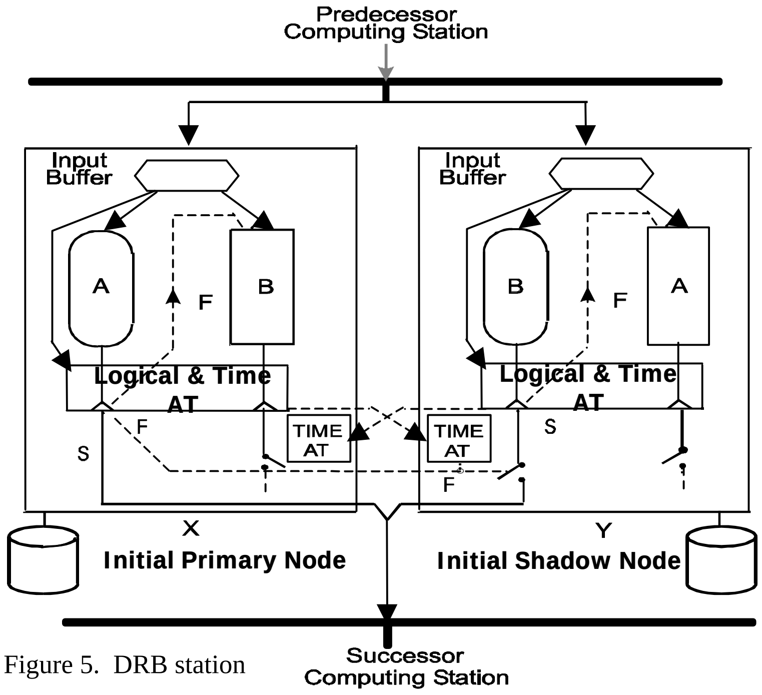


Figure 5. DRB station

The DRB station [Kim94, Kim95] is essentially a Type-4 (4a or 4b) station plus at least one of the following two optional software components: (1) an acceptance test which is an application-dependent fault-detection software component and (2) alternate application algorithms. As a part of the DRB development, a rigorous implementation model for the Type-4b PSP station has been formulated [Kim95]. When two different application algorithms are used, the two algorithms run on two different nodes. This computing station can handle both hardware and software faults to some extent but burdens the application task designer with the workload of providing acceptance tests and alternate algorithms.

Type 6: Voting triple modular redundant (TMR) station (more generally, Voting N-modular redundant station)

This computing station uses three copies of a computing component and take a vote with their execution results [Toy87]. When there is

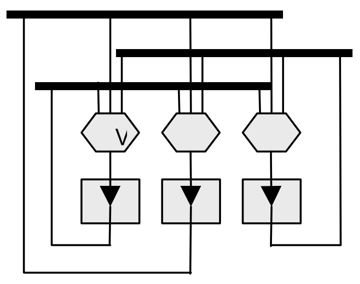


Figure 6. Voting TMR station

discrepancy, the result with a majority vote is used. The voter may be implemented in hardware or in software.

Type 7: Voting N-Version program (NVP) station

Figure 6 depicts this computing station.

This station is essentially a Type-6 station plus 3 or more different application algorithms for each application task [Avi95]. While the voting logic is application-independent, the scheme requires the design of multiple versions expected to generate *truly identical computation results*, which could be a restriction in cases where complexity of a program component is high.

2.3. Network surveillance and reconfiguration (NSR)

NSR, or network surveillance for short, is aimed for facilitating fast learning by each interested fault-free node in the distributed computer system of failure events and repair completion events occurring in other parts of the system. NSR schemes involve combinations of three basic types of actions.

Action 1: Heart beat generation and monitoring
Action 2: Broadcast/multicast of message receipt acknowledgments
Action 3: Extensive interrogation

Application of action 1 and action 2 to the nodes in execution of hard RT tasks is feasible but in general action 3 is not feasible.

In addition, NSR is aimed for facilitating *fast reconfiguration*, including
(2.1) functional amputation of faulty components, and
(2.2) redistribution of tasks, including assignment of tasks to newly incorporated or repaired nodes.

Basically, the following three types of approaches are conceivable:

(1) Centralized:
Supervisor node scheme [Hec91], and others [Gar82].

(2) Decentralized:
PRHB (periodic reception history broadcast) scheme [Kop89, Kim93], TTP (time-triggered protocol) [Kop93], and others [Cri88].

(3) Hybrid: SNS (supervisor based network surveillance) scheme [Kim97b], and others.

Research on NSR schemes is less mature than the research on RT FT computing stations. New schemes are expected to emerge continuously in the next few years. The important metrics in this area are again the *detection latency bound* [Kim93] and the recovery time bound which includes among other factors the reconfiguration latency. The SNS scheme is sketched below partly because the prototype implementations of ROAFTS built so far support the scheme.

2.3.1 The supervisor-based network surveillance (SNS) scheme

The SNS scheme [Kim97b] is effective in point-to-point networks with a variety of topologies. Figure 7 shows the basic operation of the SNS scheme. As shown in the figure, there are two types of nodes that execute the SNS scheme, the *worker* nodes and a *supervisor* node.

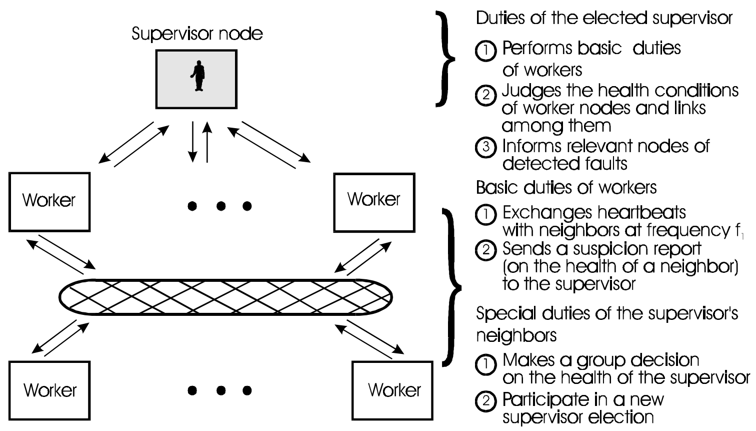

Supervisor node
Worker
Worker
Worker
Worker
Duties of the elected supervisor
① Performs basic duties of workers
② Judges the health conditions of worker nodes and links among them
③ Informs relevant nodes of detected faults
Basic duties of workers
① Exchanges heartbeats with neighbors at frequency f
② Sends a suspicion report (on the health of a neighbor) to the supervisor
Special duties of the supervisor's neighbors
① Makes a group decision on the health of the supervisor
② Participate in a new supervisor election

Figure 7. The SNS scheme (adapted from [Kim97c])

The worker nodes are mainly responsible for judging their own health status, the health status of their neighbor nodes, and the health status of the links attached to themselves. The supervisor node performs all the duties that a worker normally does. In addition, it is responsible for collecting *fault suspicion reports* from worker nodes, using the collected information to judge which area is the fault source, and then sending the *fault occurrence notice* to all the healthy worker nodes in the system. In case the current supervisor is judged to be faulty by the healthy neighbors, they participate in a new supervisor election and the newly elected supervisor informs all the healthy worker nodes about the fault in the old supervisor as well as the assumption of its new supervisor role.

2.4. The FT schemes supported by ROAFTS

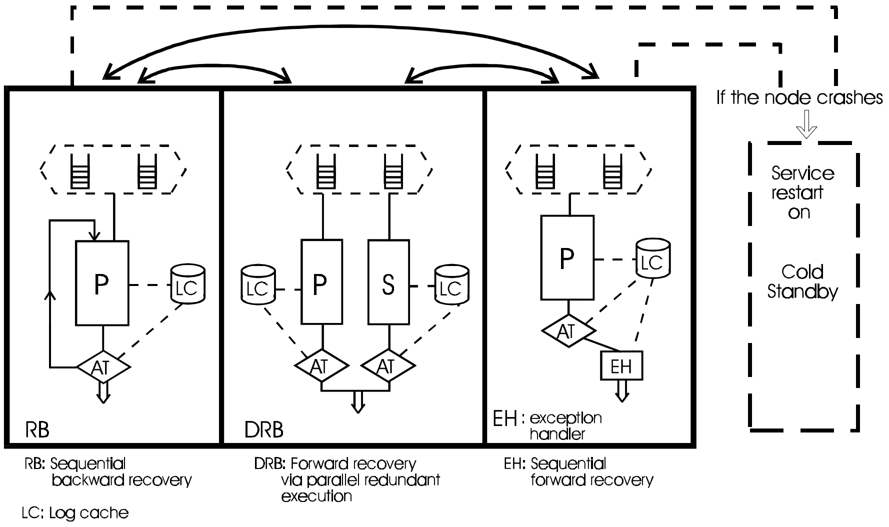
While about all the FT computing station structuring techniques reviewed in Sections 2.1 and 2.2 can be supported by ROAFTS, the following have been incorporated into the prototype implementations of ROAFTS.
(1) Self-checking node (Type-1 station) / Sequential recovery block (RB) scheme (Type-3 station)
(2) PSP (pair of self-checking processing nodes) (Type-4b station) / DRB (distributed RB) scheme (Type-5 station)

The reason why these techniques were incorporated into prototype implementions of ROAFTS first was because we judged that these had wider application ranges than other techniques. As for NSR schemes, the following were incorporated into the prototype implementations of ROAFTS.
(1) SNS (supervisor based network surveillance) scheme
(2) PRHB (periodic reception history broadcast) scheme

The schemes mentioned above were then integrated along with an adaptation manager into the *adaptable PSP / DRB (ADRB) scheme* [Kim97c] in ROAFTS. The ADRB scheme exploits several of the fundamental trade-offs that are found in computing systems in the dimensions of time, equipment, and service.

Under the ADRB scheme, a critical RT task can be executed not only (1) in the *parallel redundant mode*, which is the standard mode used in the basic DRB station (Type-5 station), but also (2) in the *sequential backward recovery mode*, which is the execution mode adopted in the original *recovery block scheme* (Type-3 station), and (3) in *the sequential forward recovery mode*, which has been considered in many previous research projects in the field of exception handling. Therefore, an *ADRB station* dynamically switches its operating mode in response to significant changes in the resource and application modes. Figure 8 depicts possible changes in the operating mode.

Also, the supervisor station under the DRB scheme is basically responsible for three functions with the support of a network surveillance scheme: (1) detection of various node failures and message communication failures in the system, (2) detection of misjudgments by the nodes in DRB stations about the status of their partner nodes, and (3) network reconfiguration including task redistribution. Under the ADRB scheme the supervisor station has three additional functions performed as quantum changes in the resource availability and application modes occur: (4) changing the set of real-time tasks to be executed, (5) deciding on the execution modes of real-time tasks, and (6) changing the parameters of the network surveillance scheme.



If the node crashes
Service restart on
Cold Standby
P
S
LC
AT
EH
RB
DRB
EH : exception handler
RB: Sequential backward recovery
DRB: Forward recovery via parallel redundant execution
EH: Sequential forward recovery
LC: Log cache

Figure 8. ADRB scheme

The ADRB scheme operates as follows. First, the given task set is mapped to the node set by the supervisor station. Each task may be assigned to one or more nodes to form an ADRB station. The execution mode of each ADRB station is chosen by the supervisor on the basis of the equipment availability and the criticality and recovery time requirement of the task assigned to the ADRB station. As the system resource condition changes and the application proceeds through different phases, the supervisor may order each ADRB station to change its execution mode. In addition, the supervisor may also decide to make changes to various parameters of the network surveillance scheme.

In addition, recently developed schemes for active replication of RT objects are also supported by ROAFTS. They were incorporated into prototype implementations of ROAFTS but they are not discussed in this paper due to space limit.

3. Overall Architecture of ROAFTS

Figure 9 depicts the protype implementation of ROAFTS on the DREAM kernel. The DREAM kernel itself can be viewed as an idealistic model of a RT OS kernel. Therefore, Figure 9 can be viewed as an abstract representation of the ROAFTS architecture and as we have experienced, this representation can be easily adapted to a representation of a reasonable prototype implementation on any COTS platform.

3.1. Basic kernel-threads

The basic active kernel components which ROAFTS depends on are two kinds besides the process/thread scheduler: (1) predictable communication support and (2) watchdog timer support. The communiciation support is provided by two kernel-threads, the *Incoming Communication Thread* (ICT) and the *Outgoing Communication Thread* (OCT), and the watchdog timer service is effected by the *Watchdog Timer Thread* (WTT). Therefore, these three kernel-threads or their equivalents which are required to support any serious distributed RT applications are also needed by ROAFTS.

Preferably all three kernel-threads should be structured as periodically executing *time-triggered* (TT-) threads. Their execution is driven by the real-time clock. This structuring eases the determination of the worst-case service times of these threads and hence eases the analysis of the performance of the ROAFTS with respect to the recovery time bound, the time bound for application task execution mode changes, etc.

The ICT is responsible for receiving messages from the network and distributing it to appropriate application processes or kernel-threads. The OCT is responsible for sending messages out to the network. The WTT functions as a watchdog timer which detects deadline violations of application processes and takes appropriate actions and also sends timer signals to application processes, if requested.

3.2. Components directly related to AFT

3.2.1 Network surveillance thread

SNS scheme was adopted as the basic NSR scheme. This TT-thread is responsible for almost everything in the SNS scheme described in Section 2.3.1 except the generation of the heartbeat signals. For example, some or all of the functions such as periodic analysis of the received heartbeat signals, reporting fault suspicions, election of a supervisor, and supervisory function, are handled by this kernel-thread, depending on the role of the host node (a worker node, supervisor node or a neighbor of the supervisor node). This thread also notifies the application task process (chosen by GFTST-core to be discussed below) of any detected faults in the host node. In addition, this thread honors any requests

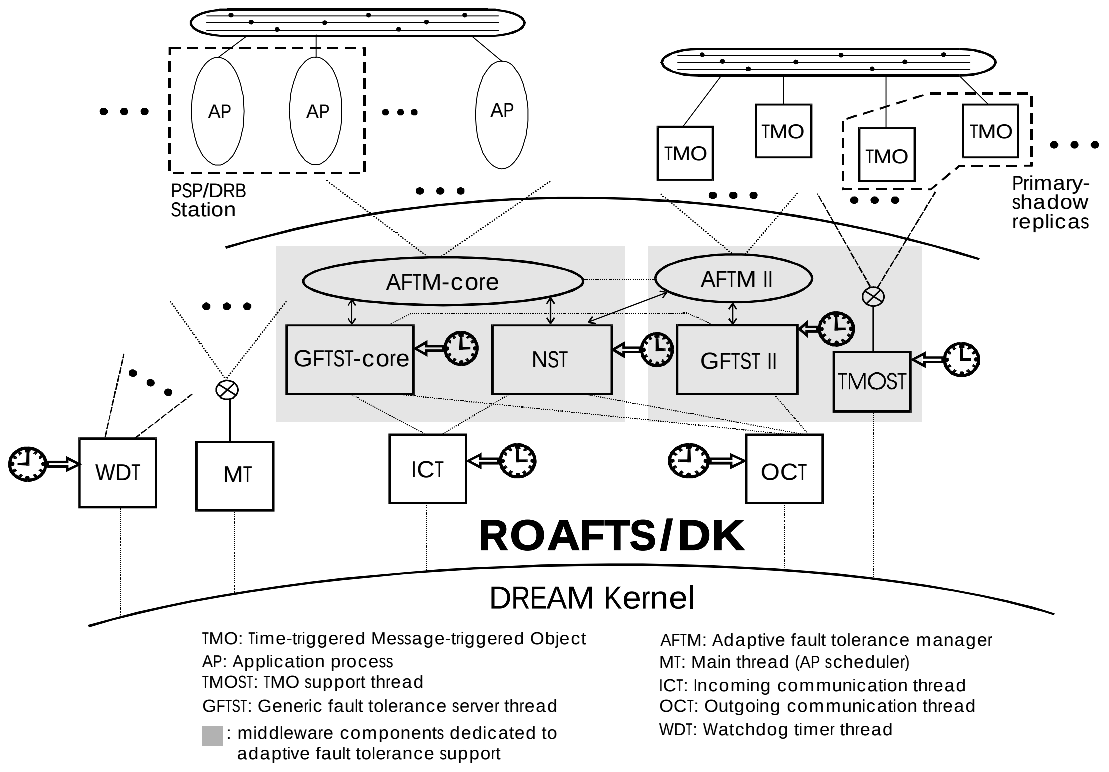

AP
AP
AP
PSP/DRB Station
TMO
TMO
TMO
TMO
Primary-shadow replicas
AFTM-core
AFTM II
GFTST-core
NST
GFTST II
TMOST
WDT
MT
ICT
OCT
ROAFTS/DK
DREAM Kernel
TMO: Time-triggered Message-triggered Object
AP: Application process
TMOST: TMO support thread
GFTST: Generic fault tolerance server thread
: middleware components dedicated to adaptive fault tolerance support
AFTM: Adaptive fault tolerance manager
MT: Main thread (AP scheduler)
ICT: Incoming communication thread
OCT: Outgoing communication thread
WDT: Watchdog timer thread

Figure 9. ROAFTS / DK

from the partner or supervisor. For instance, the partner may request for the most recent state information of the application task process saved in the log cache of the primary node or a particular missing input data item. The supervisor may also order the node to change its execution mode from the DRB mode to the RB mode. In another case the supervisor may order the heartbeat signal generation frequency to be changed.

3.2.2 Generic fault tolerance support thread - core (GFTST-core) and associated library

This thread is another TT-thread. It facilitates application-transparent fault tolerance and relieves the application software designer largely from the details of fault management. It handles most of the details of all the FT schemes supported. The supported FT operations include rejoin of a new shadow partner into a DRB station. The associated library includes application-independent fault management modules which application-specific modules can use.

3.2.3 Adaptive fault tolerance manager (AFTM - core)

AFTM-core which is a request-driven system process adjusts the system operating strategy, including the FT execution modes of computing stations, according to the state of fault tolerance requirement and resource availability. It may decide to reach a compromise among consistency, timeliness, functionality. It acts directly toward the goal of meeting the dynamically and widely changing fault tolerance requirement by efficiently and adaptively using a limited and dynamically changing available redundant processing resources.

3.3. Components directly related to AFT in object-structured applicatons

The components discussed in the preceding section (3.2) support adaptive FT execution of process-structured distributed application software. ROAFTS contains similar components designed to support adaptive FT execution of RT object-structured distributed application software. The RT object structuring scheme supported is called the *time-triggered message-triggered object (TMO)* scheme [Kim97a]. The TMO scheme and the ROAFTS components responsible for supporting FT execution of TMO's are not discussed in this paper due to space limit.

4. Experiences and future directions

Two prototype implementations of ROAFTS have been completed and two more are under way. As mentioned earlier, the first implementation runs on the DREAM kernel. Using this facility, we have constructed two major application prototypes and evaluated the AFT capabilities. One is a military command-control application prototype, and the other is a factory control application prototype. Various adaptation decisions were taken by AFTM-core in response to changes in the environment (such as those caused by the injection of faults) and it was observed that the worker application tasks could make the transitions among different execution modes within reasonably short delays. Rejoin of repaired nodes as shadow partners of active application processes was observed. The overhead bounds and recovery time bounds were also analyzed. Some of those analysis results have been reported and more reports will be made in the near future.

The second prototype implementation realized by collaborators in SoHaR, Inc., runs on Sun Solaris platforms. This implementation is a CORBA-compliant [OMG95] middleware. Both the military command-control application prototype and the factory control application prototype were again built using these facilities. The basic time grain supported by this middleware is larger than that supported by the DREAM kernel based ROAFTS. This is due to the overhead introduced by the interfact to the COTS OS, i.e., Solaris, and also the overhead introduced by the CORBA object request broker.

The third prototype implementation is under development and it will run on Microsoft NT platforms. CORBA facilities were not incorporated into this version in order to see how much performance could be obtained with middleware running on a COTS OS. The fourth prototype implementation is a joint work between the author's lab and SoHaR, Inc., and it is in an early phase.

During these experiences, the potential power of ROAFTS was amply demonstrated. Since quantitatively guaranteed RT FT was one of the achieved goals of ROAFTS, our confidence in the maximal survivability of the application prototypes developed to run on ROAFTS was high. The most important contributing factors are the effective use of periodic TT-threads and the employment of the FT schemes for which recovery time bounds were formally established.

One remaining technical challenge is to develop an efficient non-disruptive rejoin technique which accomplishes the job of copying the application state of an active RT process into a newly joining shadow partner process in multiple rounds. We have encountered situations where single-step copying introduced excessive disturbances into the active RT process which then ran into deadline violations.

Acknowledgments: The research work reported here was supported in part by the US Defense Advanced Research Project Agency under Contract N66001-97-C-8516 monitored by SPAWAR, in part by the University of California's MICRO Program under Grant 96-169, in part by Hitachi, and in part by USAF Rome Lab.

References

[Avi95] Avizienis, A., "The Methodology of N-Version

Programming", Ch. 2 in Michael R. Lyu, ed., '*Software Fault Tolerance*', 1995, pp. 22-46.

[Cri88] Cristian, F., "Agreeing on Who is Present and Who is Absent in a Synchronous Distributed System.", *Proc. IEEE CS 18th Int. Symp. on Fault-Tolerant Computing*, Tokyo, Japan, June 1988, pp. 206-211.

[Gar82] Garcia-Molina, H., "Elections in a distributed computing system.", *IEEE Trans. on Computers*, Jan. 1982, pp. 48-59.

[Hec91] Hecht, M., et al., "A Distributed Fault Tolerant Architecture for Nuclear Reactor and Other Critical Process Control Applications.", *Proc. IEEE CS 21st Int'l Symp. on Fault-Tolerant Computing*, June 1991, Montreal, pp. 462-469.

[Kim92] Kim, K.H., and Lawrence, T.F., "Adaptive Fault-tolerance in Complex Real-Time Distributed Computer System Applications", *Computer Communications*, May 1992, vol.15, (no.4) pp. 243-251.

[Kim93] Kim, K.H. and Shokri, E.H., "Minimal-Delay Decentralized Maintenance of Processor-Group Membership in TDMA-Bus LAN Systems", *Proc. IEEE CS 13th Int'l Conf. on Distributed Computing Systems*, Pittsburg, May 1993, pp. 410-419.

[Kim94] Kim, K.H., "Action-level Fault Tolerance", Ch. 17 in Sang H. Son ed., '*Advances in Real-Time Systems*', Prentice Hall, 1994, pp. 415-434.

[Kim95] Kim, K.H., "The Distributed Recovery Block Scheme", Ch. 8 in Michael R. Lyu, ed., '*Software Fault Tolerance*', 1995, pp. 189-209.

[Kim97a] Kim, K. H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, Vol. 30, No. 8, August 1997, pp. 62-70.

[Kim97b] Kim, K.H., and Subbaraman, C., "A Supervisor-Based Semi-Centralized Network Surveillance Scheme and the Fault Detection Latency Bound", *Proc. 16th Symp. on Reliable Dist. Systems*, Oct. 1997, pp. 146-155.

[Kim97c] Kim, K.H., Goldberg, J., Lawrence, T., and Subbaraman, C., "The Adaptable Distributed Recovery Block Scheme and a Modular Implementation Model", *Proc. Pacific Rim Int'l Symp. on Fault-Tolerant Systems*, Taiwan, Dec. 1997, pp. 131-138.

[Kop89] Kopetz, H., Grunsteidl, G, and Reisinger, J., "Fault-Tolerant Membership Service in a Synchronous Distributed Real-Time System.", *Proc. IFIP WG 10.4 Int'l Working Conf. on Dependable Computing for Critical Applications*, Santa Barbara, Aug. 1989, pp.167-174.

[Kop93] Kopetz, H. and Grunsteidl, G., "TTP-A: Time Triggered Protocol for Fault Tolerant Real-Time Systems", *Proc. IEEE CS FTCS-23*, Toulose, France, June 1993, pp. 524-533.

[Law95] Lawrence, T.F., "Anomaly Management in Complex Systems", *Proc. PRFTS '9*5, Newport Beach, Dec. '95, pp. 132-134.

[OMG95] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, 1995.

[Ran75] Randell, B., "System Structure for Software Fault Tolerance.", *IEEE Transactions on Software Engineering*, June 1975, pp.220-232.

[Ran95] Randell, B., and Xu, Jie, "The Evolution of the Recovery Block Concept", Chap. 2 in '*Software Fault Tolerance*', Michael R. Lyu, Ed., 1995, pp. 1-21.

[Sho97] Shokri, E., et al., "An Approach for Adaptive Fault Tolerance in Object-Oriented Open Distributed Systems", *Proc. 3rd Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 97)*, Newport Beach, CA, February 1997, pp. 89-98.

[Toy87] Toy, W.N., "Fault-Tolerant Computing.", A chapter in *Advances in Computers*, Vol. 26, Academic Press, 1987, pp.201-279.

[Wil85] Wilson, D., "The STRATUS computer system.", Chapter 12 in T. Anderson ed., '*Resilient Computing Systems - Volume I*', John Wiley & Sons Inc., 1985, pp. 45 - 67.

Proceedings

Third IEEE International High-Assurance Systems Engineering Symposium

November 13 – 14, 1998

Washington, DC

Sponsored by

IEEE Computer Society

IEEE COMPUTER SOCIETY

Los Alamitos, California

Washington • Brussels • Tokyo