FAULT-TOLERANCE IN REAL-TIME SYSTEMS

K. G. Shin (Chairman), *Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122, USA*

G. Koob, *Office of Naval Research, Code 1133, 800 N. Quincy Street, Arlington, VA 22217-5000, USA*

F. Jahanian, *IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA*

Abstract. With ever-increasing reliance on digital computers in embedded systems such as in space, avionics, manufacturing, and life-support monitoring/control applications, the need for dependable systems that deliver services in a timely manner has become crucial. Embedded systems often interact with the external environment and operate under strict timeliness and reliability requirements. Fault-tolerance and real-time requirements on a system often influence one another in subtle ways. For example, the requirements on a highly-available system, such as an air traffic control system, may derive the timing constraints imposed on certain critical tasks. In cases where a set of interacting tasks operate under strict timing constraints, missing a deadline may result in a catastrophic system failure, which was termed a *dynamic failure* in (Shin, Krishna, and Lee, 1985). This paper argues that fault-tolerance and real-time requirements are not orthogonal and it addresses some of the challenges that confront the designers of fault-tolerant real-time systems. These challenges include formal specification of reliability and timing requirements, appropriate language and operating system support for providing fault-tolerance in a time-critical system, new scheduling theories which consider multiple resources and fault-tolerance, tradeoff between time and space redundancy, and predictable redundancy management in the presence of faults.

Keywords. Fault-tolerance; real-time systems; reliability; time and space redundancy; run-time monitoring; specification.

ARE FAULT-TOLERANCE AND REAL-TIME ORTHOGONAL ?

Fault-tolerance is defined informally as the ability of a system to deliver the expected service even in the presence of faults. A common misconception about real-time computing is that fault-tolerance is orthogonal to real-time requirements. It is often assumed that the availability and reliability requirements of a system can be addressed independent of its timing constraints. This assumption, however, does not consider the distinguishing characteristic of real-time computing: *the correctness of a system is dependent not only on the correctness of its result, but also on meeting stringent timing requirements.* In other words, a real-time system may fail to function correctly either because of errors in its hardware and/or software or because of not responding in time to meet the timing requirements that are usually imposed by its "environment." Hence, a real-time system can be viewed as one that must deliver the expected service in a timely manner even in the presence of faults. A missed deadline can be potentially as disastrous as a system crash or an incorrect behavior of a critical task, e.g., a digital control system may lose stability.

In fact, if the logical correctness of a system may be dependent on the timing correctness of certain components, separating the functional specification from the timing specification is a very difficult task. Moreover, timeliness and fault-tolerance could sometimes pull each other into opposite directions. For example, frequent extra checks and exotic error recovery routines will enhance fault-tolerance but may increase the chance of missing the deadlines of application tasks.

When a system specification requires certain service in a timely manner, then the inability of the system to meet the specified timing constraint can be viewed as a failure. However, a simple approach of applying existing fault-tolerant system design methods by treating a missed deadline as a timing fault does not address fully the needs of real-time applications. The fundamental difference is that real-time systems must be predictable, even in the presence of faults. Hence, fault-tolerance and real-time requirements must be considered jointly and simultaneously when designing such systems. The challenge is to include the timing and the fault-tolerance requirements in the specification of the system at every level of abstraction and to adopt a design methodology that considers system predictability even during fault detection, isolation, system reconfiguration and recovery phases. Formal specification of the reliability requirements and their impact on meeting timing constraints is an area which requires further exploration. Determining the

This is a summary of the panel on real-time, fault-tolerant computing at the 9-th IEEE Workshop on Real-Time Operating Systems and Software, May 15-17, 1991, Atlanta, GA. The order of authors names appeared in this paper represents the order of presentation at the RTOSS.

timing constraints on a system from its availability requirements is a very difficult problem.

SPACE AND TIME TRADEOFF

The design methodologies for fault-tolerant systems have often been characterized by the tradeoff between time and space redundancy. In non-real-time systems, time is treated as a cheap resource and most methods concentrate on space optimization. In a real-time environment, the tendency would be to trade space for time since meeting the stringent timing constraints is essential in ensuring correct system behavior. Although time-space tradeoff forms the basis for most fault-tolerant system design methodologies, it is unclear whether it is an appropriate paradigm for characterizing fault-tolerance in a real-time environment. In particular, redundancy must be considered in the context of achieving both predictability and dependability in a system. For example, tolerating transient faults by retrying a computation is an acceptable technique if the timing constraints can be met. The same assertion holds for techniques based on the notion of recovery blocks where a different version of the software module is used in the retry. However, alternative approaches must be considered when time is a scarce resource. In particular, the quality of the computation can become a third dimension in the design space. That is, in a new twist on the principle of graceful degradation, a fault in a real-time system could result in a (temporary) reduction in the quality of the services provided in order to allow the system to continue to meet critical-task deadlines. Although general methods that delete or reduce the number of less critical tasks would certainly fall into this category, "quality" must ultimately be defined in terms of the detailed semantics of the application. Real-time control systems, for example, are characterized by continuous variables whose values can be approximated or estimated if time does not permit precise computation. The imprecise computation approach (Liu, Lin, Shih, and others, 1991) is one technique that sacrifices accuracy for time in iteratively-improving calculations. As discussed in the next section, trading space for time also has potential limits since spatial redundancy introduces additional overhead (in time) for managing the redundancy (see (Krishna, Shin, and Butler, 1984) for an example).

PREDICTABLE REDUNDANCY MANAGEMENT

Although advances in distributed and parallel systems provide the opportunities for achieving real-time performance while satisfying fault-tolerance requirements, using the inherent redundancy provided by these systems is not free. For example, the overhead associated with synchronization and the nondeterminism due to communication delay contribute to the complexity of building systems with predictable timing behavior. Predictable redundancy management remains an open research problem. For example, a set of identical servers on multiple processors provide fault-tolerance in a system with crash or performance failures. However, predictable redundancy requires addressing issues such as synchronization of servers, agreement on the order of request messages, and the cost of failure detection and recovery (Krishna, Shin, and Butler, 1984). In this case, redundancy in space incurs additional cost in time. The overhead associated with managing redundancy must be quantified precisely so that certain guarantees about the real-time behavior of the system can be made.

Managing redundancy in a predictable fashion is related closely to the demands on real-time scheduling theory. Satisfying the timing requirements of a real-time system demands the scheduling of system resources such that the timing behavior of the system is "understandable, predictable, and maintainable." Most existing scheduling algorithms consider one resource at a time and ignore fault-tolerance. The requirement for meeting timing constraints in the presence of faults imposes additional demands on the scheduling algorithms. New resource allocation techniques are necessary to address the predictability and reliability requirements of complex real-time systems. A simple case of this was treated in (Krishna and Shin, 1986).

RUN-TIME MONITORING

In designing real-time systems, we often make assumptions about the behavior of the system and its environment. These assumptions take many forms: upper bounds on interprocess communication delay and task execution time, deadlines on the execution of tasks, or minimum separations between occurrences of two events. They are often made to deal with the unpredictability of the external environment or to simplify a problem that is otherwise intractable or very hard to solve. Such assumptions may be expressed as part of the formal specification of the system or as scheduling requirements on the real-time tasks. Despite the contribution of formal verification methods and recent advances in real-time scheduling, the need to perform run-time monitoring of these systems is not diminished for several reasons: the execution environment of most systems is imperfect and the interaction with the external world introduces additional unpredictability; design assumptions can be violated at run-time due to unexpected conditions such as transient overload; application of formal techniques or scheduling algorithms in turn requires assumptions about the underlying system; and it may be infeasible (or impossible) to verify formally some properties at design time, thus further necessitating run-time checks (Jahanian and Goyal, 1990, Chodrow, Jahanian, and Donner, 1991).

Run-time monitoring of a system requires timestamping and recording of the relevant event occurrences, analyzing the past history as other events are recorded and providing feedback to the rest of the system. The non-intrusiveness requirement of real-time monitoring often leads to use of special monitoring hardware (Haban and Shin, 1990, Tsai, Fang, and Chen, 1990). Many important issues must be addressed before run-time monitoring is fully utilized in real-time systems. Some of these issues are:

- *Language support:* What is an appropriate set of language constructs for the specification of run-time constraints? Should the specification language be tied closely to the underlying implementation language?
- *Run-time system support:* What level of support should be provided by the operating system? What internal operating system events, such as task preemption, should be made visible to the monitoring facility?
- *Scheduling support:* How intrusive is run-time system monitoring on the critical tasks within a system? Is it possible to make the intrusiveness of run-time monitoring predictable? See (Tokuda, Koreta, and Mercer, 1989) for more on this.

In addition to detecting violation of design assumptions, run-time monitoring can be used to detect application specific exception conditions. One can envision a system in

which specification-based fault-detection is done by a monitoring facility (Jahanian and Goyal, 1990). Furthermore, beside detecting exception conditions, a run-time monitoring facility can provide feedback to the rest of the system. The information collected by the monitoring facility can be used to provide feedback to the system operator, the application tasks, or the scheduler (Haban and Shin, 1990). For example, exceeding the maximum computation time estimated for a task can be reported by the monitoring facility. The feedback to the scheduler can be used to build a robust system that is capable of adapting to the changes in the environment and the system load. Investigating the utilities of monitoring application tasks and operating system events as a way of providing feedback to the rest of the system is an area of ongoing research. A related topic is scheduling run-time monitoring tasks with the real-time application tasks in a system.

REFERENCES

Chodrow, S., F. Jahanian, and M. Donner (1991). Run-time monitoring of real-time systems. *Proc. of 12th Real-Time Systems Symposium.*

Haban, D., and K. G. Shin (1990). Application of real-time monitoring for scheduling tasks with random execution times. *IEEE Trans. on Software Engineering*, Vol. 16, No. 12, pp. 1374-1389.

Jahanian, F., and A. Goyal (1990). A formalism for monitoring real-time constraints at run-time. *Proc. of Fault-Tolerant Computing Symposium (FTCS-20).*

Krishna, C. M., and K. G. Shin (1986). On scheduling tasks with a quick recovery from failure. *IEEE Trans. on Comput.* Vol. C-35, No. 5, pp. 448-455.

Krishna, C. M., K. G. Shin, and R. W. Butler (1984). Synchronization and fault-masking in redundant real-time systems. Digest of Papers, FTCS-14. pp. 152-157.

Liu, J., K.-J. Lin, W.-K. Shih , A. Yu, A.-Y. Chung, and W. Zhao (1991). Algorithms for scheduling imprecise computations. *Computer.* Vol. 24, No. 5, pp. 58-69.

Shin, K. G., C. M. Krishna, and Y.-H. Lee (1985). A unified method for evaluating real-time computer controllers and its application. *IEEE Trans. on Automatic Control.* Vol. AC-30, No. 4, pp. 357-366.

Tokuda, H., M. Koreta, and C. Mercer (1989). A real-time monitor for a distributed real-time o.s. *ACM SIGPlan Notices.* Vol. 24, No. 1, pp. 68-77.

Tsai, J. P., K.-Y. Fang, and H.-Y. Chen (1990). A noninvasive architecture to monitor real-time distributed systems. *Computer.* Vol. 23, No. 3, pp. 11-23.