A Hybrid Reconfiguration Scheme for Real-Time Fault-Tolerant Systems

Jong Kim, Heejo Lee

Dept. of CSE
POSTECH
Pohang 790-784, KOREA

Kang G. Shin

Dept. of EECS
The University of Michigan
Ann Arbor, MI 48109–2122

Abstract

Non-stop operation, despite the occurrence of faults, is a key requirement for highly-reliable, real-time systems, such as aircraft and nuclear reactor controller computers. Reconfiguration is commonly used to mean the process of transforming the execution structure of a system from one state to another when a fault occurs. We identified two types of reconfiguration: fault-tolerating reconfiguration (upon occurrence of each fault) and performability-enhancing reconfiguration (independently of fault occurrences).

This paper proposes a hybrid reconfiguration scheme for real-time fault-tolerant systems. The proposed scheme not only meets the timing constraints or deadlines of hard real-time jobs, but also maintains high performability. In this scheme, a system reconfigures itself either when a fault occurs or when a job arrives or completes its execution.

1 Introduction

The development of high-performance microprocessors & memory chips and new parallel processing techniques has increased significantly the computation speed of today's computer systems. In addition to this performance enhancement, there is a strong need to devise techniques for tolerating component failures in a computer system, especially for such safety-critical applications as defense systems, avionics, and process controls. One must consider two important requirements when designing a reliable and stable computer system for these applications. First, the system must satisfy the timing requirements of the underlying applications. Second, the system must operate continuously even in the case of random fault occurrences.

There are several methods proposed for finding an"optimal" configuration upon the occurrence of a fault [1-3]. These methods focused on "optimal" reconfiguration in the sense of maximizing reliability upon detection of a fault, but did not address such issues as how many jobs are currently executing in the system and how soon the reconfiguration must be completed to meet the timing constraints of currently-executing jobs.

Melhem [4] modeled state-dependent reconfiguration. He identified the system to be in strict- or relaxed- mode, depending on the environment of currently-running jobs. The strict-mode represents both a heavy computational load and a strict demand for response within a tight timing bound. The relaxed-mode, on the other hand, is characterized by a light computational load with relatively relaxed constraints on the response time. For example, a radar system would be in strict-mode when it is tracking a target, and is in relaxed-mode when there is no object to be tracked.

When a system is in strict-mode and needs to be reconfigured, for example, due to the occurrence of a component failure, the system is required to do it fast using only local information of the failed component. On the other hand, when the system is in relaxed-mode and needs to be reconfigured, the system may gather information on all live components and find an optimal system configuration by maximizing its stability and reliability. Reconfiguration based on local information as shown in [4] is not the only fast method. The method proposed in [3] is also fast using precomputed reconfiguration tables. In this method, possible system configurations are determined *a priori* and stored in a reconfiguration table, and when a fault is detected, a possible system configuration is selected

The work reported in this paper was supported in part by the Office of Naval Research under Grant N00014-91-J-1115, by Texas Instrument, and by POSTECH internal fund. Any opinions, findings, and recommendations in this paper are those of the authors and do not reflect the views of the funding agencies.

from the table. This kind of reconfiguration is said to be *static*. On the other hand, the reconfiguration that gathers information, whenever needed, from the entire system and finds an optimal configuration, is said to be *dynamic*. The weakness of the method in [4] is that there are no key parameters that can be used to clearly identify the system to be in strict- or relaxed-mode.

System reconfiguration may be invoked only upon the occurrence of a fault, and, in such a case, may find the "best" (in some sense) configuration, which may still contain the faulty component. The reconfigured structure will no longer be the best when a new job arrives or the currently running job finishes its execution. Upon completion of one of the currently-running job or upon start of a new job, the system must be reconfigured in order to enhance the system's performance and reliability (as opposed to a single task performance). The objective of this reconfiguration is to maximize the system's performance *and* reliability — commonly known as *performability*. The reconfiguration to handle the occurrence of a fault is said to be *passive* and the reconfiguration to maximize performability is said to be *active*. We will henceforth call the former "fault-tolerating" reconfiguration and the latter "performability-enhancing" reconfiguration.

In this paper, we propose a hybrid reconfiguration scheme for real-time fault-tolerant systems. The proposed scheme classifies the system to be in one of three possible states: C (constrained), U (unconstrained), F (fault). For each of these system states, the proposed scheme prescribes an optimal configuration by maximizing system performability. In other words, the scheme accounts for not only fault-tolerating reconfiguration, but also performability-enhancing reconfiguration. As a result, the proposed hybrid reconfiguration satisfies the real-time constraints and maximizes the system performability.

This paper is organized as follows. The proposed hybrid reconfiguration scheme is introduced in Section 2. State identification and system reconfiguration are treated in Section 3. In Section 4, the performance of the proposed hybrid reconfiguration scheme is compared against a simple reconfiguration scheme. The paper concludes with Section 5.

2 Reconfiguration Scheme

2.1 Background

In a real-time system, high performance and continuous operation are two key attributes to the correct, timely completion of critical jobs. High performance can be achieved by employing a well-structured system and utilizing system resources efficiently. Continuous operation can be achieved by adopting and maintaining fault-tolerant structures. For performance-enhancement and fault-tolerance, the system has to be reconfigured dynamically so as to choose and maintain an optimal system configuration. That is, the system needs to be reconfigured either upon the occurrence of a fault or at anytime if it can enhance system performability. In [3], the reconfiguration needed to enhance system performability was called *active reconfiguration*, while the one needed upon occurrence of a fault was called *passive reconfiguration*.

Depending on the strictness or importance of meeting task deadlines, a real-time system is said to be *hard* or *soft*. In a hard real-time system, all critical tasks must be completed before their deadline to avoid *dynamic failure* [5]. A real-time system has two operating modes: *constrained mode* and *unconstrained mode*. When it is in the constrained mode, the system is heavily-loaded with time-critical jobs and busy executing them to meet their deadlines, thus needing to minimize any disruption (caused by system reconfiguration) in executing the tasks. By contrast, when it is in the unconstrained mode, the system has only a small number of critical jobs to execute and has time to do other things, such as thorough health checking and maintenance.

The reconfiguration algorithm for the constrained mode should differ from that for the unconstrained mode, since there isn't enough time to find an efficient or optimal configuration. It should be fast enough not to miss the deadline of any of the currently-executing jobs. More details about reconfiguration are discussed in next subsections.

2.2 Hybrid Reconfiguration

We propose here a hybrid reconfiguration scheme for a real-time fault-tolerant system which adopts different reconfiguration strategies depending on its operating status. Although the system status can be classified in many ways, we classify it in the following two ways relevant to system reconfiguration. First,

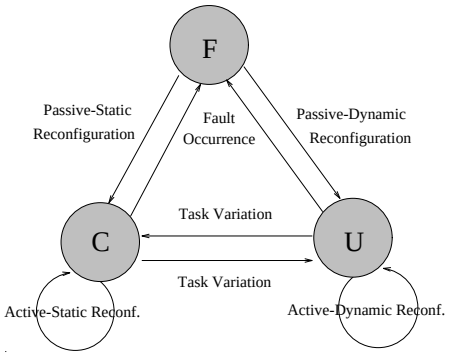

Figure 1: System state transition diagram.

the system status is divided into *fault-free* mode and *fault-occurred* mode, depending on whether a fault has already occurred or not. The system should reconfigure itself not only to tolerate or handle a fault upon its occurrence, but also to enhance the performability of the system, even when no fault has occurred. We will call the "fault-tolerating" reconfiguration *passive* and the "performance-enhancing" reconfiguration *active*.

Second, as discussed above, we distinguish the operating status/condition of a real-time system with two modes, *constrained* and *unconstrained*, depending on the tightness of timing constraints. Conceptually, the system is considered to be in the constrained mode if the total remaining time before task deadlines expire is small. Otherwise, the system is in the unconstrained mode. When a system is in the constrained mode, all but urgent operations should be avoided or shortened. For example, if a fault occurs in a constrained system, the system should perform "minimal" or fast reconfiguration so as to minimize the disruption in the execution of those tasks whose deadlines are tight. *Static* or pre-determined reconfiguration is a typical example of this type. This kind of system reconfiguration is suitable for hard real-time systems. When the system is in the unconstrained mode, since there are no tight deadlines to be met, the system derives an optimal reconfiguration structure dynamically as needed. This kind of system reconfiguration is said to be *dynamic*.

The proposed hybrid reconfiguration scheme adopts one of four reconfiguration strategies, depending on the system status, to enhance the system performance and dependability. The four reconfiguration strategies are passive–static, passive–dynamic, active–static, and active–dynamic. Applying these four strategies, we can find three distinct system states. These states and transitions among them are shown in Fig. 1.

The system is classified into one of three possible states: F (faulty), C (constrained), and U (unconstrained). The state F represents that the system has a fault. When a faulty system reconfigures and it is in the constrained mode, passive–static reconfiguration is necessary to meet the tightness of timing constraints. The state after this reconfiguration is thus represented as C. Passive–dynamic reconfiguration is performed in the unconstrained mode and the state after this reconfiguration is represented as U. The state transition from C to U or from U to C occurs when one of the currently-running job finishes or a new job is scheduled to execute by timing constraint. Active–dynamic reconfiguration is performed in the state U to enhance the performability of the unconstrained system. Similarly, active-static reconfiguration is performed in the state C to enhance the performability of the constrained system.

The proposed hybrid reconfiguration scheme is attractive and of practical significance since it can minimize the possibility of missing task deadlines *and* maximize the performability of a real-time fault-tolerant system. However, one obvious problem of the proposed scheme is state identification. Although it is easy to say that the system is in the constrained or unconstrained mode, it is difficult to differentiate between the two modes precisely. In the next section, we will discuss how to identify the systems state.

3 System State Identification

A system enters the reconfiguration phase upon occurrence of a fault or upon scheduling of a new job or upon completion of a currently-running job. The reconfiguration method could be static or dynamic. Determining which reconfiguration method to use depends on the system state at the time of reconfiguration. If the system is determined to be in the constrained mode, then a static reconfiguration method will be used. Otherwise a dynamic reconfiguration method will be used. Although there are many parameters describing the system constraints, such as response time, number of jobs, job arrival rate, job service rate, and deadline tightness, most of them give only a partial view of the system state. Among these parameters, deadline tightness is the most important to real-time applications. We will therefore focus on the system mode determined by the deadline tightness of real-time jobs.

Each real-time job can be described with three parameters: its start time, execution time, and

deadline [6]. Suppose there are m real-time jobs, $T_1, T_2, ..., T_m$, in the system. The start time, execution time, and deadline of the k-th job or T_k are denoted as $s(T_k), e(T_k), d(T_k)$ $(1 \leq k \leq m)$. When these m jobs are sorted in ascending order of their deadlines, job 1 through job k in the sorted list have to be completed before the deadline of job k. Hence, the sum of execution times of the first k jobs should be smaller than the deadline of job k; that is,

$$\sum_{i=1}^{k} e(T_i) \leq d(T_k), \ (1 \leq k \leq m)$$

The fault-tolerating reconfiguration is triggered by the occurrence of a fault. The reconfiguration process requires a certain amount of time, called the *reconfiguration delay*, to find a suitable reconfigured structure. In order to meet the deadlines of real-time jobs, the addition of reconfiguration delay to the execution time of each job should be within the job's deadline. Let $R(t)$ denote the reconfiguration delay, then

$$\sum_{i=1}^{k} e(T_i) + R(t) \leq d(T_k), \ (1 \leq k \leq m)$$

This equation restricts the reconfiguration delay, $R(t)$, to be less than $d(T_k) - \sum_{i=1}^{k} e(T_k)$ $(1 \leq k \leq m)$.

In the hybrid reconfiguration scheme, there exist two kinds of reconfiguration delay functions for passive and active reconfigurations: in constrained and unconstrained modes. Let $R_c(t)$ and $R_u(t)$ denote the reconfiguration delay in constrained mode and unconstrained mode, respectively. By determining the system's operating mode upon occurrence of a fault, we estimate the reconfiguration delay. For example, if the allowable reconfiguration delay $R(t)$ is larger than both $R_c(t)$ and $R_u(t)$, then system is in unconstrained mode. In unconstrained mode, the system has time to do optimal reconfiguration. If $R(t)$ is smaller than $R_u(t)$ and larger than $R_c(t)$, then the system is in constrained mode and should be reconfigured as fast as possible to minimize the percentage of jobs missing deadlines, or the *deadline miss ratio* for short. But if the allowable reconfiguration delay $R(t)$ is smaller than $R_c(t)$, then there is a job that cannot afford any reconfiguration delay. In such a case, the job will miss its deadline irrespective of the reconfiguration method used. However, to reduce the number of jobs missing deadlines, it would be better for the system to use static reconfiguration. Thus, hybrid reconfiguration minimizes the deadline miss ratio. Also, reconfiguration in unconstrained mode makes it possible to enhance the performability of the system. The hybrid reconfiguration scheme maximizes the system performance by minimizing the deadline miss ratio for real-time jobs.

4 Performance Evaluation

The advantages of hybrid reconfiguration are demonstrated using simulations. The selected measure for our evaluation is the probability of meeting the deadlines of real-time jobs, called the *deadline hit ratio*. The deadline hit ratio of hybrid reconfiguration is compared against a simple reconfiguration scheme that uses the same parameters. We assume that job inter-arrival times, job execution time, and fault inter-arrival times are exponentially distributed with parameters λ, μ, and γ, respectively. Since the system becomes stable after a long period of time since the occurrence of a fault, we observed the system behavior only for 1000 units of time measured from the point of failure with hybrid and simple reconfiguration schemes. The time to do static reconfiguration is assumed to be α % of the time to do dynamic reconfiguration. In the simulation, we generated jobs without considering whether the generated jobs would finish within their deadline or not, thus increasing the deadline miss ratio. The deadline of each task is computed as the summation of its execution time and a randomly-generated delay between 1 and 5.

Fig. 2 shows the deadline hit ratio as a function of failure rate when $\lambda = 0.5$, $\mu = 0.7$, and $\alpha = 50\%$. The difference between the hybrid and simple schemes becomes more pronounced as the failure rate increases. When the failure rate γ is 0.01, only 10 % of the submitted jobs are completed in time with the simple reconfiguration scheme, while 50 % of the submitted jobs are finished within their deadlines with the hybrid reconfiguration scheme.

Fig. 3 shows the effect of varying the ratio of the time to do static reconfiguration to the time to do dynamic reconfiguration. In this simulation, the failure rate is given as 0.001. When the time to do static reconfiguration is 10% of the time to do dynamic reconfiguration, the deadline hit ratio is greater than 98 %. The reason why the deadline hit ratio is less than 100 % is that jobs are not checked for the feasibility of their timely completion.

In Fig. 4, the range of arrival rates that hybrid reconfiguration can reduce the deadline miss ratio is

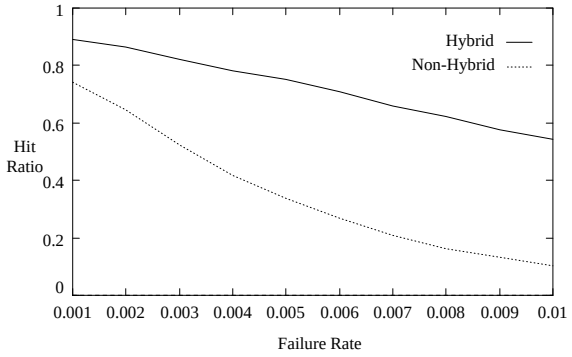

Figure 2: Deadline hit ratio versus failure rate.

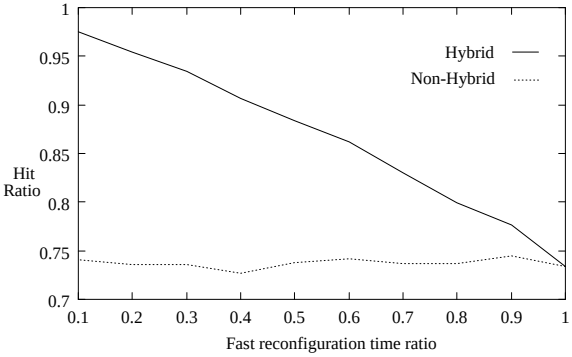

Figure 3: Deadline hit ratio versus the ratio of reconfiguration times.

evaluated in the simulation. In this simulation, μ, γ, α are set to 0.7, 0.001, 50%, respectively. Utilization factor ($\rho = \lambda/\mu$) must be less than 1 for the system to be stable. Hybrid reconfiguration is shown to improve the deadline hit ratio for all stable conditions. Especially, the larger the utilization factor the better hybrid reconfiguration becomes.

5 Conclusion

Though many researchers studied the problem of system reconfiguration, none of their results are suitable for real-time applications. To remedy this deficiency, we proposed a hybrid reconfiguration scheme for fault-tolerant real-time systems. The proposed scheme classified the operating status of a fault-tolerant real-time system into two types. First, the operating status is divided into constrained and unconstrained modes, depending on the tightness of deadlines of real-time jobs. Second, the operating status is divided into fault-tolerating and performability-enhancing modes, depending on whether a fault has

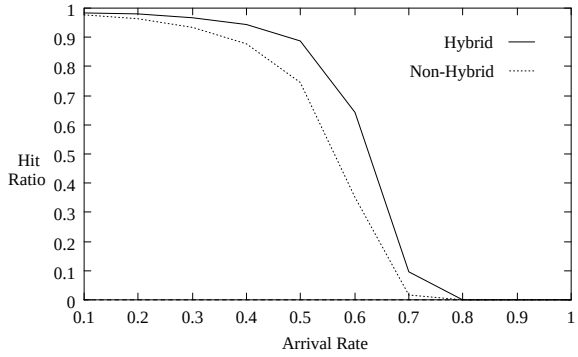

Figure 4: Deadline hit ratio versus job arrival rate.

occurred or not. From these two different classifications, we identified four distinct system's operating modes, each of which requires a different reconfiguration scheme to have better system performance and dependability. The hybrid reconfiguration scheme adopts a different reconfiguration strategy for each operating mode. Our simulation results have shown that the proposed hybrid reconfiguration scheme outperforms a simple reconfiguration scheme.

References

[1] T. K. Chang Chen, An Feng and K. Torii, "Reconfiguration algorithm for fault-tolerant arrays with minimum number of dangerous processors," in *Proc. of FTCS-21*, pp. 452–459, 1991.

[2] P. Banerjee and M. Peercy, "Design and evaluation of hardware strategies for reconfiguring hypercubes and meshes under faults," *IEEE Trans. on Computers*, vol. 43, pp. 841–848, July 1994.

[3] Y.-H. Lee and K. G. Shin, "Optimal reconfiguration strategy for a degradable multimodule computing system," *J. of ACM*, vol. 34, pp. 326–348, Apr. 1987.

[4] R. G. Melhem, "Bi-level reconfigurations of fault tolerant arrays in bi-modal computational environments," in *Proc. of FTCS-19*, pp. 488–495, 1989.

[5] M. H. Woodbury and K. G. Shin, "Evaluation of the probability of dynamic failure and processor utilization for real-time systems," in *Proc. of 9th Real-Time System Symposium*, pp. 222–231, Dec. 1988.

[6] J. Y.-T. Leung, "Research in real-time scheduling," in *Foundations of Real-Time Computing: Scheduling and Resource Management* (A. M. van Tilborg and G. M. Koob, eds.), ch. 2, pp. 31–62, Kluwer Academic Publishers, 1991.