An Efficient Backup-Overloading for Fault-Tolerant Scheduling of Real-Time Tasks

R. Al-Omari, G. Manimaran, and Arun K. Somani

Dept. of Electrical and Computer Engineering
Iowa State University, USA
{romari,gmani,arun}@iastate.edu

Abstract. Many time-critical applications require dynamic scheduling with predictable performance. Tasks corresponding to these applications have deadlines to be met despite the presence of faults. In this paper, we propose a technique called *dynamic grouping*, to be used with backup overloading in a primary-backup based fault-tolerant dynamic scheduling algorithm in multiprocessor real-time systems. In dynamic grouping, the processors are dynamically grouped into logical groups in order to achieve efficient overloading of backups, thereby improving the schedulability. We compare the performance of dynamic grouping with that of static grouping and no-grouping schemes through extensive simulation studies and show the effectiveness of dynamic grouping.

1 Introduction

Due to the critical nature of tasks in a hard real-time system, it is essential that every task admitted in the system completes its execution even in the presence of faults. Therefore, fault tolerance is an important requirement in such systems. Scheduling multiple versions of tasks on different processors can provide fault tolerance. One of the models that is used for fault-tolerant scheduling of real-time tasks is the Primary-Backup (PB) model, in which two versions of a task are scheduled on two different processors and an acceptance test is used to check the correctness of the execution result [1, 2]. The backup version is executed only if the output of the primary version fails the acceptance test, otherwise it is deallocated from the schedule. A concept, called *backup overloading*, was introduced in [1] and has been extended in [2] to capture the trade-off between the number of faults in the system and system utilization.

In this paper, we address the problem of dynamically scheduling real-time tasks with PB fault-tolerance onto a set of processors in such a way that the versions of the tasks are feasible in the schedule. In order to improve the performance of backup overloading, we propose a technique called dynamic grouping, which dynamically divides the processors of the system into logical groups as tasks arrive into the system and finish executing.

Task Model: (i) Tasks are aperiodic. Every task Ti has the attributes arrival time (a_i), ready time (r_i), worst case computation time (c_i) and a deadline (d_i). **(ii)** Each task T_i has two versions, namely primary copy (Pr_i) and backup copy

(Bk_i). They have identical attributes. **(iii)** Tasks are non-preemptable. **(iv)** All tasks arrive to a centralized scheduler.

Fault Model: *Assumption 1:* Processor faults can be transient or permanent and are independent. *Assumption 2:* The maximum number of faulty processor at any instant of time in a "group" is limited to one (group is defined later). *Assumption 3:* The minimum interval between two successive faults within a group of processors $>$ Max $\{d_i - r_i\}$ $\forall T_i \in T$, where T is the set of tasks. *Assumption 4:* There exists a fault-detection mechanism to detect processor faults. The scheduler does not assign tasks to a faulty processor.

Related Work and Motivation for Our Work

Backup overloading concept was proposed in [1] to allow backups of different tasks to overlap in time on the same processor. This is valid only if the following conditions are satisfied: *Condition 1:* The primaries should be scheduled on different processors. *Condition 2:* At most one of the primaries can encounter a fault. *Condition 3:* At most one version of a task can encounter a fault.

Condition 1 is needed to handle permanent faults. Condition 2 is needed to ensure that at most one backup can be executed among the overloaded backups. Condition 3 is needed to ensure that at least one version of each task will execute without any fault.

The algorithm proposed in [1] assumes at most one fault at any instant of time in the entire system in order to satisfy condition 2, which is optimistic. Another PB based algorithm for tasks with resource requirements was proposed in [2]. This algorithm can tolerate more than one fault at a time by employing a technique called flexible backup overloading. Though this algorithm can tolerate more than one fault at a time, it statically divides the processors into groups of three or more. This static division restricts the flexibility of backup overloading, thus reducing the guarantee ratio (% of tasks accepted).

2 Dynamic Logical Groups

In this section, we propose a technique called dynamic grouping of processors which overcomes the limitations of static grouping [2] and no-grouping [1]. Dynamic logical grouping is defined as the process of dynamically dividing the processors of the system into logical groups as tasks arrive into the system and finish executing. The logical groups are determined when the scheduler decides where to schedule the two versions of a task. The number of groups and the size of the groups will vary with time, in contrast to static grouping where the number and size of the groups are fixed and are known a priori. Moreover, in static grouping a processor can be a member of only one group. Whereas, in dynamic grouping a processor can be a member of more than one group which allow efficient use of backup overloading.

We will show that dynamic grouping offers better schedulability than static grouping due to its flexible nature of overloading backups. We will also show that dynamic grouping offer higher fault-tolerance degree than the static grouping due to its dynamic nature of forming the groups. However, dynamic grouping

involves more scheduling overhead compared to static grouping. The dynamic grouping concept works under the following propositions:

Proposition 1: At most one version of a task will encounter a fault.

Proof: Since the two versions (Pr_i and Bk_i) of a task T_i are scheduled on two different processors in the same group in the interval $[r_i, d_i]$. Assumptions 1 and 2 of the fault model will ensure that at most one version of the task will encounter a fault, which satisfy condition 3.

Proposition 2:. Two primaries (Pr_1 and Pr_2) of tasks T_1 and T_2 that are scheduled on two different processors ($Proc(Pr_1)$ and $Proc(Pr_2)$) in the same group can overload their backups (Bk_1 and Bk_2) on a third processor ($Proc(Bk_1, Bk_2)$) in the same group if $|st(Pr_1) - ft(Pr_2)| <$ Max $\{d_i - r_i\}$ $\forall T_i \in T$, where T is the set of tasks and $|X|$ is the absolute value of X.

Proof: Since these three processors ($Proc(Pr_1)$, $Proc(Pr_2)$), and $Proc(Bk_1, Bk_2)$) are within the same group, this group can have only one fault at a time (Assumption 1), and since $|st(Pr_1) - ft(Pr_2)| <$ minimum interval between successive faults (assumption 2), at most the fault can be in one of these primaries which satisfy condition 2.

Group Dynamics: The creation, deletion, expansion, and shrinking of logical groups is controlled by the following rules:

Rule 1: A logical group G_k is dynamically formed from two processors (Proc(Pr_i), Proc(Bk_i)) when the two versions of a task T_i are scheduled to these two processors. This logic group stays either until Pr_i is successfully executed and Bk_i is de-allocated or the Pr_i fails and the Bk_i is executed.

Rule 2: If another primary (Pr_j) scheduled on a third processor Proc(Pr_j) whose backup (Bk_j) overloads with Bk_i, then the logical group (G_k) will expand to have three processors (Proc(Pr_i), Proc(Pr_j), Proc(Bk_i)=Proc(Bk_j)). This logical group will stay until the two tasks (T_i and T_j) are successfully executed.

Rule 3: If primary (Pr_i) is successfully executed then its backup (Bk_i) will be de-allocated from its processor Proc(Bk_i), and the logical group (G_k) will shrink to have two processors (Proc(Pr_j), Proc(Bk_j)). The same is true for Pr_j.

Rule 4: If primary (Pr_i) is failed then the group will stay until primary (Pr_j) and (Bk_i) are successfully executed. The same is true for Pr_j.

Figure 1 shows an example that illustrates how the groups are formed and removed dynamically as tasks arrive into the system and finish executing. Figure 1a shows that primary and backup copies of task T_1 are scheduled on processors 1 and 2, respectively. Then, these two processors will form a logical group (group 1) that will stay until one of these copies executes successfully. Figure 1b shows the same situation as in Figure 1a, but this time the primary of T_2 is scheduled on processor 3, and its backup is overloaded with Bk_1 on processor 2. This results in expanding the group to have three processors. Figure 1c shows the same situation as in Figure 1b, but now the scheduler has decided to schedule the primary of T_3 on processor 4, and its backup on processor 3, then processors 3 and 4 will form a logical group (group 2). Figure 1d shows the situation when Pr_1 has executed successfully. Therefore Bk_1 is de-allocated, which results in shrinking group 1 to have two processors 1 and 2.

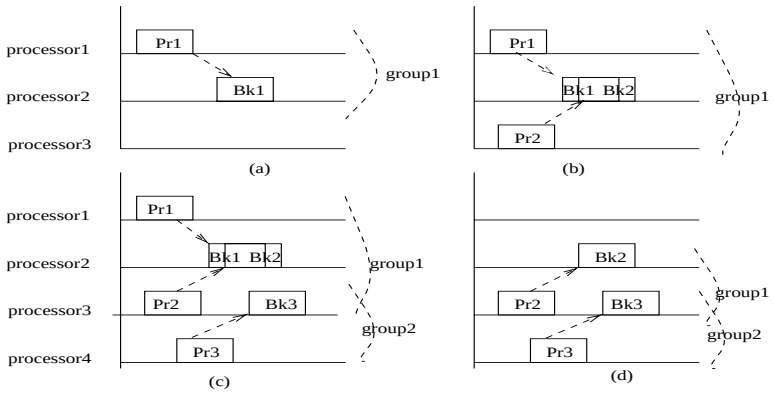

Fig. 1. Dynamic grouping

On the other hand, if static grouping is employed for the same example, processors 1, 2, and 3 will form group 1 and processor 4 will be in group 2. In that case, the situations shown in Figures 1c and 1d cannot occur because the groups are disjoint and their sizes are fixed. It can be seen that dynamic grouping results in higher utilization for processor 3 (in Figure 1c), because in static grouping Bk_3 can not be scheduled to processor 3 since Pr_3 is in group 2. Also, dynamic grouping increases the number of faults tolerable in these processors as tasks T_2 and T_3 can be executed successfully even in the presence of faults in processors 2 and 4 (in Figure 1d). Similarly, if no-grouping was employed for the same example, all the processors would be in one group. In this case the scheduling will be the same except that only one fault can be tolerated in the entire system.

3 Performance Study

We compare the performance of the proposed dynamic grouping based fault-tolerant scheduling algorithm with that of the static grouping based algorithm [2] and also with the no-grouping algorithm [1] using guarantee ratio as the performance metric. In simulation, the number of processors is taken to be 12.

Effect of Task Load: The task load (L) has been varied in Figure 2. The figure shows that increasing L decreases the guarantee ratio for all the algorithms. From the figure, the difference in the performance for the dynamic and static grouping algorithms is maximum for medium task load. This is explained as follows: For higher L, the guarantee ratio is lower for all the algorithms because the task load is much more than the capacity of the system. On the other hand, for lower L, the guarantee ratio is higher for all the algorithms because the task load is less than the system capacity, which means that most of the tasks are schedulable by all the algorithms. Also, note that the difference in performance between no-grouping and dynamic grouping is small which means that the dynamic grouping algorithm increases the utilization of the system to a point equal

to the no-grouping algorithm and the difference in performance is partly due to the overhead cost associated with dynamic grouping.

Effect of Number of Faults: Figure 3 shows the effect of varying number of fault occurrences in the system for task loads of 0.25, 0.5 and 1. The figure shows that the difference in the guarantee ratio between the algorithms is significant at low load ($L = 0.25$) and medium load ($L = 0.5$) compared to at high load ($L = 1$). This is explained as follows. For light load (L=0.25), the dynamic grouping can compensate the degradation in guarantee ratio due to faults by rearranging the groups which is not possible in the static grouping. For full load (L=1), the dynamic grouping tends to behave similar to static grouping as all the processors are heavily loaded. The figure also shows that the guarantee ratio offered by the static grouping algorithm decreases more rapidly than the dynamic grouping as the number of faults increases due to the same reason.

4 Conclusions

In this paper, we have proposed a concept called dynamic logical grouping of processors for overloading of backups in a PB-based fault-tolerant dynamic scheduling algorithm. Our simulation studies show that the dynamic grouping offers significantly better guarantee ratio than the static grouping under all the interesting conditions that we have simulated in the system.

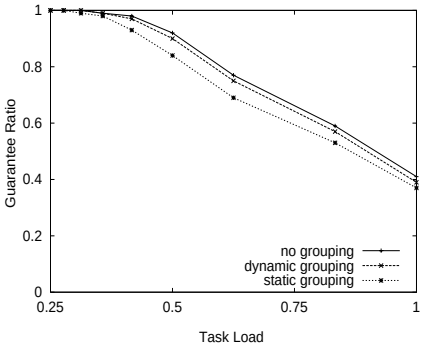

Fig. 2. Effect of task load

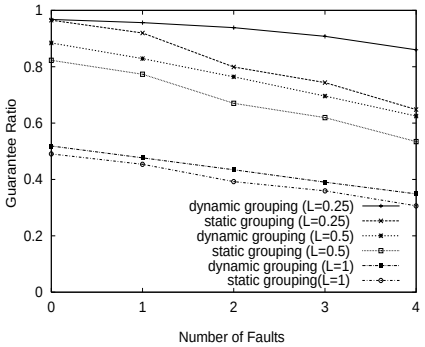

Fig. 3. Effect of number of faults

References

1. S. Ghosh, R. Melhem, and D. Mosse, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems," *IEEE Transaction on Parallel and Distributed Systems,* vol. 8, no. 3, pp. 272-284, Mar. 97.
2. G. Manimaran and C. Siva Ram Murthy, " A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis," *IEEE Transaction on Parallel and Distributed Systems,* vol. 9, no. 11, pp. 1137-1152, Nov. 98.
3. R. Al-Omari, G. Manimaran, and A. K. Somani, " A Fault- tolerant dynamic scheduling algorithm for multiprocessor real- time systems," *Fault-tolerant Computing Symp. (FTCS), FAST ABSTRACTS,* pp. 63-64, Jun. 99.