A Fault–Tolerant Dynamic Scheduling Algorithm for Multiprocessor Real–Time Systems

R. Al-Omari romari@iastate.edu
G. Manimaran gmani@iastate.edu
A. Somani arun@iastate.edu
Department of Computer Engineering
Iowa State University, Ames, IA 50011

1 Introduction

Scheduling multiple versions of the tasks on different processors can provide fault tolerant. One of the models that is used for fault-tolerant scheduling of real–time tasks, namely, is the Primary backup (PB) model. In this approach, two versions are executed serially on two different processors and an acceptance test is used to check the result. The backup version is executed only if the output of the primary version fails the acceptance test, either due to the processor or the software fault.

In this work, we address the scheduling of dynamically arriving real-time tasks with PB fault-tolerant requirements onto a set of processors in such a way that the versions of the tasks are feasible in the schedule. The objective of any dynamic real-time scheduling algorithm is to improve the guarantee ratio which is defined as the percentage of tasks, that arrived into the system, whose deadlines are met.

2 Task model and assumptions

1. Tasks are aperiodic, i.e., the task arrivals are not known a priori. 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).
2. Each task T_i has two versions, namely primary copy (Pr_i) and backup copy (Bk_i).
3. Tasks are non-preemptable.
4. The system has multiple identical processors, which are connected through a shared medium.
5. Processor faults can be transient or permanent and are independent.The minimum interval time between two successive faults is greater than or equal to the maximum of ($d_i - r_i$) for all tasks.
6. There exists a fault–detection mechanism such as acceptance tests to detect processor faults. Also the scheduler will not assign tasks to a faulty processor.
7. All the tasks arrive at a central processor, called the scheduler, from where they are distributed to other processors in the system for execution.

3 Backup overloading and de–allocating

The backup overloading algorithm allocates more than a single backup in a time interval and de-allocates the resources unused by the backup copies in case of fault -free operation. This is done in order to make efficient utilization of available processors and resources time. This introduces a trade-off between the number of faults and utilization in the system. The conditions under which two backups can be overloaded are:

- If Pr_1 and Pr_2 are scheduled on two different processor.
- And if $||st(Pr_1) - ft(Pr_2)|| <$ minimum interval between successive faults. Where st(Pr_1) and ft(Pr_2) are the start time of Pr_1 and the finish time of Pr_2 respectively.

Then their backups Bk_1 and Bk_2 can overlap in execution on a third processor. This backup overloaded is valid under the assumption that there is at most one fault at any instant of time in these three processors until the two tasks are execute successfully.

4 Related work and motivations

A PB based algorithm with backup overloading and backup de–allocation has been proposed [2] for fault-tolerant dynamic scheduling of real–time tasks in multiprocessor systems, which we call as backup overloading algorithm. Though this algorithm is for dynamic scheduling, it does not consider resource constraints among tasks, which is a practical requirements in any complex real-time system. Additionally it assumes at most one failure at any instant of time in the whole system, which is pessimistic.

Another PB based algorithm with resource requirement has been proposed in [1] for fault–tolerant dynamically scheduling of real-time tasks in multiprocessor system. The algorithm can tolerate more than one fault at a time, and employs techniques such as flexible backup overloading and resource reclaiming to improve the guarantee ratio of the system. Though this algorithm can tolerate more than one fault at a time (number of faults equal number of groups),

they have an unpractical assumption which says "if the primary fails, it's backup always succeeds". Also to tolerate more than one fault and allow backup overloaded to happen, they divided the system statically to groups of three processors. This static division restricts the flexibility of overloaded thus reduces system utilization and reduces the guarantee ratio.

Our algorithm works within the Spring scheduling approach [3] and builds fault-tolerant solutions around it to support PB based fault tolerant with backup overloaded and de–allocated. To avoid the assumption of having one fault at a time per system and the disadvantages of static grouping, we divided our system to groups dynamically as the tasks arrived and finished. Each group can have at most one fault at any instant of time to allow backup over–loaded to occur within these groups. Also we gave a minimum limit for the time interval between two successive faults for each group which is the maximum of $(d_i - r_i)$ for all tasks. Since the two copies of the task are scheduled within this interval and only one fault can be introduced with this group in this interval, this will make sure that at most one copy of a task will be faulty.

5 Dynamic logical groups

Since in our task model, every task, T_i, has two copies (Pr_i and Bk_i), and we have a fault-tolerant scheduling algorithm, then these two copies will be scheduled in a way that they will have time and space exclusion, i.e., they will be scheduled onto two different processors and two different time interval. To make sure that at most one of these copies can fail, we have the two processors combined in a logic group that can have at most one processor fault at any instant of time. Since the minimum interval between two successive faults for this group will be greater than $(d_i - r_i)$, causing us to ensure that at most one of the copies will fail. This logic group stays either until the primary is successfully executed and the backup is de–allocate or the primary failed and the backup executed. Also if another primary from a third processor overloaded its backup with an existing backup, then we can form a dynamic logical group that contains three processor (Proc(Pr_i), Proc(Pr_j), Proc(Bk_i, Bk_j)). This logical group will allow to have only one fault at any instant of time, and it will stay until the two tasks are executed successfully.

These logical groups are dynamically formed when the copies of the tasks are scheduled onto different processors. A processor can be a member of more than one group, and these groups change with time. Figure 1.a shows a task T_1 that has its primary copy at processor 1 and its backup at processor 2. Then these two processors will be formed in a logical group that will stay until one of the copies executes successfully. Figure 1.b shows the same situation as Figure 1.a, but now the scheduler has decided to put the primary of T_2 in processor 3, and its backup overloaded with Bk_1 at processor 2. So the group will expand to have three processor. Figure 1.c shows the same situation as Figure 1.b but now the scheduler has decide to put the primary of T_3 in processor 4, and its backup on processor 3 then these two processors will form a logical group (group2). Figure 1.d shows the situation when Pr_1 has execute successfully so Bk_1 will be de–allocated, then group1 will reduce to two processors (processors 1 and 2).

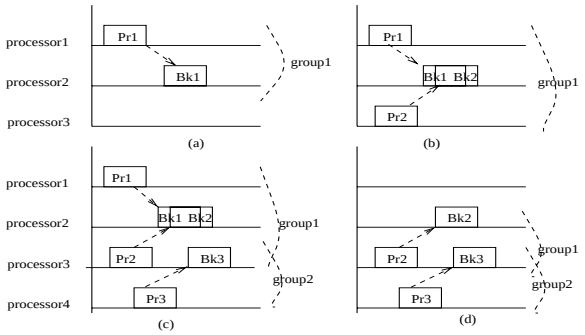

Figure 1: **Dynamic grouping**

6 Conclusion

In this paper, we made extensions to the original Myopic algorithm to support PB based fault–tolerants, with backup overloaded and de-allocated using dynamic logical grouping concept, for scheduling a set of real–time tasks.

We expected that this dynamic logical grouping will work better than static grouping, because it will increase system utilization by allowing more overloading with all the processors in the system. Also this concept will increase the fault–tolerant degree for the system which may change with time. All of these factors will increase the guarantee ratio for the system. Through simulation and analytic work, we will study the system performance.

References

[1] G. Manimaran and C. Siva Ram Murth, " 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.

[2] 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.

[3] K. Ramamritham, J.A. Stankovic, and P.-F. Shiah, "Efficient Scheduling Algorithms for Real-Time Multiprocessor systems," IEEE Transaction on parallel and distributed systems, VOL 1, NO.2, PP. 184–194, Apr. 90.

[4] K. Ramamritham and J-A. Stankovic, "Scheduling Algorithms and Operating System Support for Real-Time Systems", Proc. IEEE, VOL 82, NO 1, PP. 55–67, Jan. 94 .