ELSEVIER

Journal of Systems Architecture 45 (1998) 1–13

JOURNAL OF SYSTEMS ARCHITECTURE

A new study for fault-tolerant real-time dynamic scheduling algorithms [1]

G. Manimaran [2], C. Siva Ram Murthy *

Department of Computer Science and Engineering, Indian Institute of Technology, Madras 600 036, India

Received 8 April 1996; received in revised form 7 February 1997; accepted 25 July 1997

Abstract

Many time-critical applications require predictable performance. Tasks corresponding to these applications have deadlines to be met despite the presence of faults. Failures can happen either due to processor faults or due to task errors. To tolerate both processor and task failures, the copies of every task have to be mutually excluded in space and also in time in the schedule. We assume that each task has two versions, namely, *primary copy* and *backup copy*. We believe that the position of the backup copy in the task queue with respect to the position of the primary copy (*distance*) is a crucial parameter which affects the performance of any fault-tolerant dynamic scheduling algorithm. To study the effect of distance parameter, we make fault-tolerant extensions to the well-known *myopic* scheduling algorithm [Ramamritham et al. IEEE Trans. Parallel Distr. sys. 1 (2) (1990) 184] which is a dynamic scheduling algorithm capable of handling resource constraints among tasks. We have conducted an extensive simulation to study the effect of distance parameter on the schedulability of the fault-tolerant myopic scheduling algorithm.

Keywords: Multiprocessor system; Real-time tasks; Dynamic scheduling; Fault-tolerance; Resource reclaiming

1. Introduction

Multiprocessors have emerged as a powerful computing means for real-time applications such as avionic control and nuclear plant control, because of their capability for high performance and reliability [1]. The problem of multiprocessor scheduling [2–6] is to determine when and where a given task executes. This can be done either statically or dynamically. In static algorithms, the assignment of tasks to processors and the time at which the tasks start execution are determined a priori. Static algorithms are often used to schedule periodic tasks with hard deadlines. The main

* Corresponding author. E-mail: murthy@iitm.ernet.in.

[1] This work was supported by the Department of Science and Technology, New Delhi.

[2] E-mail: gmani@bronto.iitm.ernet.in.

1383-7621/98/$ – see front matter
PII: S1383-7621(97)00072-6

advantage is that, if a solution is found, then one can be sure that all deadlines will be guaranteed. However, this approach is not applicable to aperiodic tasks whose arrival times and deadlines are not known a priori. Scheduling such tasks in a multiprocessor real-time system requires dynamic scheduling algorithms. In dynamic scheduling, when new tasks arrive, the scheduler dynamically determines the feasibility of scheduling these new tasks without jeopardizing the guarantees that have been provided for the previously scheduled tasks. Thus for predictable executions, schedulability analysis must be done before a task's execution is begun. A feasible schedule is generated if the timing, precedence, and resource constraints of all the tasks can be satisfied, i.e., if the schedulability analysis is successful. Tasks are dispatched according to this feasible schedule.

The general problem of optimal fault-tolerant scheduling of tasks in a multiprocessor system is NP-complete [13,15]. In a real-time multiprocessor system, fault-tolerance can be provided by scheduling multiple copies of tasks on different processors. *Primary/backup* (PB) and *triple modular redundancy* (TMR) are two basic approaches that allow multiple copies of a task to be scheduled on different processors. In the PB approach, if incorrect results are generated from the primary task, the backup task is activated. In the TMR approach, multiple copies are executed concurrently and their results are compared. In [8], a PB scheme has been proposed for preemptively scheduling periodic tasks in a uniprocessor system. In [9], another PB based algorithm for scheduling periodic tasks in a multiprocessor system has been proposed. In this strategy, a backup schedule is created for each set of tasks in the primary schedule. The tasks are then rotated such that primary and backup schedules are on different processors and do not overlap.

The PB strategy with *backup overloading* and *backup deallocation* has been proposed recently [10,11] for fault-tolerant dynamic scheduling of tasks in real-time multiprocessor systems. This scheme [10,11] allocates more than a single backup in a time interval (where time interval of a task is the interval between scheduled start time and scheduled end time of the task) and deallocates the resources unused by the backup copies in case of fault-free operation. This work [10,11] does not address the effect of relative positions of primary and backup copies in the task queue which is a crucial parameter that affects the schedulability of any dynamic scheduling algorithm. It does not also address resource constraints among tasks which is a practical requirement in any complex real-time system. The objective of our work is twofold: (i) to make fault-tolerant extensions to the well-known myopic scheduling algorithm [3], and (ii) to study the impact of distance parameter on schedulability of the fault-tolerant myopic scheduling algorithm.

The rest of the paper is structured as follows: System model and definitions are given in Section 2. Section 3 discusses the myopic scheduling algorithm and our proposed fault-tolerant extensions to it. Our simulation study is presented in Section 4. Finally, some concluding remarks are made in Section 5.

2. System model

In this section, we present the task model and the scheduler model used in this work, followed by some definitions.

2.1. Task model

1. Tasks are aperiodic, i.e., the task arrivals are not known a priori. Every task T_i has the attributes: *arrival time* (a_i), *ready time* (r_i), *worst case computation time* (c_i), and *deadline* (d_i).
2. The deadline d_i of every task T_i satisfies: $d_i \geqslant r_i + 2c_i$.

3. *Resource constraints*: A task might need some resources such as data structures, variables, and communication buffers for its execution. Every task can have two types of accesses to a resource: (a) exclusive access, in which case, no other task can use the resource with it or (b) shared access, in which case, it can share the resource with another task (the other task also should be willing to share the resource). We say that a resource conflict exists between two tasks T_i and T_j if one of these tasks cannot share the resources it requires, with the other.
4. Each task T_i has two versions, namely, *primary copy* and *backup copy*. The worst case computation time of the primary copy may be greater than that of the backup copy.
5. Tasks are non-preemptable. At any instant, at most one task can be executed on a given processor.
6. All the processors are identical and are prone to failures.
7. Every task can encounter at most one failure either due to processor fault or task error, i.e., if the primary copy fails, the backup copy always completes successfully.
8. There exists a fault-detection mechanism that detects processor faults and task errors.

2.2. Scheduler model

Dynamic scheduling algorithms can be either distributed or centralized. In a distributed dynamic scheduling scheme, tasks arrive independently at each processor. When a task arrives at a processor, the local scheduler at the processor determines whether or not it can satisfy the constraints of the incoming task. The task is accepted if they can be satisfied, otherwise the local scheduler tries to find another processor which can accept the task. In a centralized scheme, all the tasks arrive at a central processor called the *scheduler*, from where they are distributed to other processors in the system for execution. In this paper, we will assume a centralized scheduling scheme. The communication between the scheduler and the processors is through *dispatch queues*. Each processor has its own dispatch queue. This organization, shown in Fig.1, ensures that the processors will always find some tasks in the dispatch queues when they finish the execution of their current tasks. The scheduler will be running in parallel with the processors, scheduling the newly arriving tasks, and periodically updating the dispatch queues. The scheduler has to ensure that the dispatch queues are always filled to their minimum capacity (if there are tasks left with it) for this parallel operation. This minimum capacity depends on the average time required by the scheduler to reschedule its tasks upon the arrival of a new task [12].

Task deletion takes place when extra tasks are initially scheduled to account for fault tolerance, i.e., when the primary copy of the task completes execution successfully. When no faults occur, there is no necessity for these temporally redundant tasks to be executed and hence they can be deleted. Also, the actual time taken by a task during execution can be smaller than its worst case computation time. Hence, a lot of resources remain unused if we dispatch the tasks strictly based on their starting times of the feasible schedule. Each processor invokes the *resource reclaiming* algorithm [12] at the completion of its currently executing task, to utilize the resources left unused by a task when it executes less than its worst case computation time, or when a task is deleted from the current schedule. The scheduler is informed about the time reclaimed by the reclaiming algorithm so that it can schedule the new tasks correctly and effectively. A protocol for achieving this is suggested in [12].

Resource reclaiming in multiprocessor systems with independent tasks is straightforward. The

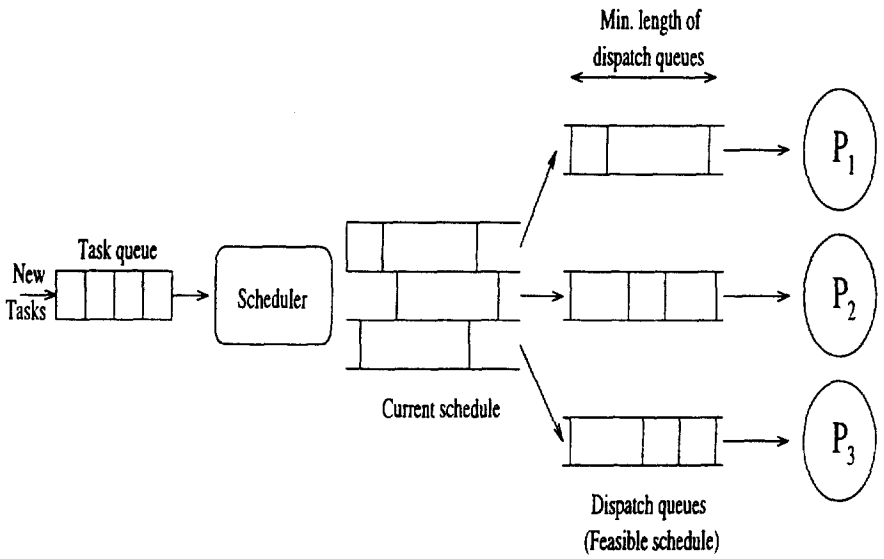


Fig. 1. Parallel execution of scheduler and processors.

resource reclaiming in such systems is *work-conserving* which means that the reclaiming never leaves a processor idle if there is a dispatchable task and the resource reclaiming in such systems is an integral part of the scheduler. But resource reclaiming on multiprocessor systems with resource constraints is more complicated. This is due to the potential parallelism provided by a multiprocessor, and potential resource conflicts among tasks. When early completion of tasks or deletion of tasks takes place in a non-preemptive multiprocessor schedule with resource constraints, run-time anomalies [16] may occur. These anomalies may cause some of the already guaranteed tasks to miss their deadlines. In particular, one cannot simply use a work-conserving scheme, like the one used in [10], without verifying that the task deadlines will not be missed. This justifies the need for separating the resource reclaiming from the scheduling algorithm and the same is adopted in our task model.

2.3. Terminology

Definition 1. The scheduler fixes a *feasible schedule* S taking into account the resource constraints and fault-tolerant requirements of all the tasks. The feasible schedule uses the worst case computation time of a task for scheduling it and ensures that the deadlines of primary and backup copies of all the tasks in S are met. A partial schedule is one which does not contain all the tasks.

Definition 2. A partial schedule is said to be *strongly feasible* if *all* the schedules obtained extending the current schedule by any one of the remaining tasks are also feasible [3].

Definition 3. $Proc(T_i)$ is the processor to which task T_i is scheduled. The processor to which task T_i should not get scheduled is denoted as *exclude* $proc(T_i)$. The time at which processor P_j is available for executing a new task is denoted as *avail* $time(j)$.

Definition 4. *Start time*(T_i) is the scheduled start time of task T_i which satisfies $r_i \leqslant starttime(T_i) \leqslant d_i - c_i$. *Finish time*($T_i$) is the scheduled finish time of task T_i which satisfies $r_i + c_i \leqslant finishtime(T_i) \leqslant d_i$.

Definition 5. The primary (Pr_i) and backup (Bk_i) copies of task T_i are said to be mutually exclusive in time, denoted as *time exclusion*(T_i), if $starttime(Bk_i) \geqslant finishtime(Pr_i)$.

Definition 6. The primary (Pr_i) and backup (Bk_i) copies of task T_i are said to be mutually exclusive in space, denoted as *space exclusion*(T_i), if $proc(Pr_i) \neq proc(Bk_i)$.

Definition 7. Distance between the primary and backup copies of a task T_i, denoted as *distance* (Pr_i, Bk_i), denotes the relative difference between their positions in the task queue. The distance parameter can have value between 1 and n, where n is the number of tasks.

Definition 8. EAT_k^s (EAT_k^e) is the earliest time when resource R_k becomes available for shared (exclusive) usage [3].

Definition 9. Let P be the set of processors and Q be the set of resources requested by task T_i. *Earliest start time* of a task T_i, denoted as $EST(T_i)$, is the earliest time when its execution can be started. $EST(T_i) = MAX(r_i, MIN_{j \in P}(availtime(j)), MAX_{k \in Q}(EAT_k^u))$, where $u = s$ for shared mode and $u = e$ for exclusive mode.

3. The Fault-tolerant scheduling algorithm

In this section, for the sake of completeness, we first present the myopic scheduling algorithm [3] and then the proposed fault-tolerant extensions to it.

3.1. The myopic algorithm

The myopic algorithm is a heuristic search algorithm that works as follows.

1. Tasks (in the task queue) are ordered in non-decreasing order of deadline.
2. The algorithm starts with an empty partial schedule.
3. Determines whether the current schedule is strongly feasible (strong feasibility is determined with respect to the first K (we call this, *feasibility check window*) tasks in the task queue).
4. If found to be feasible:
   (a) the heuristic function (H) is computed for the first K tasks;
   (b) the task with the best (smallest) H value is chosen to extend the schedule.
5. Else:
   (a) it backtracks to the previous search level;
   (b) it extends the schedule with the task having the next best H value.
6. The algorithm repeats steps 3–5 until a termination condition is met.

The termination conditions are either (a) a complete feasible schedule has been found, or (b) the maximum number of backtracks or H function evaluations has been reached, or (c) no more backtracking is possible. The authors of [3] have shown that the integrated heuristic function $d_i + EST(T_i)$ which captures the deadline and resource constraints of task T_i performs better than simple heuristics such as earliest deadline first and minimum processing first.

3.2. Fault-tolerant extensions

To tolerate processor and task failures, the copies of every task needs to be mutually excluded in space and also in time. Since in our model, every task, T_i, has two copies, we place both of them in

the task queue with relative difference of *distance*(Pr_i, Bk_i) in their positions. The primary copy of any task always precedes its backup copy in the task queue. The *distance* is an input parameter and in our study we assume that

$$\forall T_i, \mathit{distance}(\mathrm{Pr}_i, \mathrm{Bk}_i) = \begin{cases} \mathit{distance} & \text{for the first } (n-(n \bmod \mathit{distance})) \text{ } \mathit{tasks}, \\ n \bmod \mathit{distance} & \text{for the last } (n \bmod \mathit{distance}) \text{ } \mathit{tasks}, \end{cases}$$

where n is the number of tasks.

The *distance*(Pr_i, Bk_i) may change for some tasks due to backtracks during the process of scheduler building a feasible schedule. The following is an example task queue with $n = 5$ and *distance* = 3 assuming that the deadlines of tasks $T_1, T_2, \ldots, T_5$ are in the increasing order.

Pr_1	Pr_2	Pr_3	Bk_1	Bk_2	Bk_3	Pr_4	Pr_5	Bk_4	Bk_5

The skeleton of our fault-tolerant myopic algorithm is the same as the myopic algorithm. In the fault-tolerant myopic algorithm, we convert the *time exclusion* into precedence constraints and the *space exclusion* into non-access to the corresponding processor. The modifications are given below.

1. Establish precedence relation between primary and backup copies of every task T_i such that Pr_i precedes Bk_i. This is done when the backup tasks are inserted into the task queue.
2. When a task copy is considered for feasibility checking:
   If it is a backup copy (Bk_i) and its primary (i.e., Pr_i) has not been scheduled, then
   - assume feasibility of Bk_i is success, i.e., no need to check for feasibility since its primary (Pr_i) is known to ensure feasibility.
3. When a task copy is considered for H function evaluation:
   If it is a backup copy (Bk_i) and its predecessor (i.e., Pr_i) has not been scheduled, then
   - $\mathrm{EST}(\mathrm{Bk}_i) = \infty$.
4. When a task copy is considered for extending the schedule:
   If it is a primary copy (Pr_i), then
   - Set *ready time*(Bk_i) = *finish time*(Pr_i). This is to achieve *time exclusion*(T_i).
   - Set *exclude proc*(Bk_i) = *proc*(Pr_i). This is to achieve *space exclusion*(T_i).
5. When backtracking takes place, we do not extend the schedule from the previous level with a task having the second best H value, instead we perform a feasibility check for one more task (other K-1 tasks are known to ensure feasibility) and consider its H value also for finding the best H value, i.e., we always consider tasks of the entire feasibility check window.

4. Simulation studies

To study the effect of distance parameter on the schedulability of the fault-tolerant myopic algorithm, we have conducted extensive simulation studies. The parameters used in the simulation studies are given in Fig. 2. During the parallel operation of the scheduler and processor, the scheduler has a set of tasks to schedule. In our study, we do not concentrate on the functions of the processor. As mentioned in Section 2.2, any real-time dynamic scheduling approach has scheduling with associated resource reclaiming. Our simulation studies are carried out in two ways: (i) considering scheduling alone and (ii) considering scheduling with associated resource reclaiming.

4.1. Studies related to scheduling

In this section, we present the studies which are related to the fault-tolerant scheduling algorithm alone without considering the resource reclaiming.

parameter	explanation
MIN_C	minimum computation time of tasks, taken as 8.
MAX_C	maximum computation time of tasks, taken as 16.
R	laxity parameter denotes the tightness of the deadline, varies from 0.3 to 0.7.
UseP	probability that a task uses a resource, varies from 0.1 to 0.6.
ShareP	probability that a task uses a resource in shared mode, taken as 0.4.
K	size of feasibility check window, taken as {3,6,9,12,15,18}.
distance	relative difference in positions of primary and backup copies in the task queue, taken as {6,12,18,24}.
num_btrk	number of backtracks permitted in the search.
num_proc	number of processors considered for simulation.
num_res	number of resource types considered for simulation.

Fig. 2. Simulation parameters.

The task set for scheduling is generated in the following way.

1. A task set of only primary copies are generated till *schedule length*, which is an input parameter, with no idle time in the processors, as described in [3]. The computation times of primary copies are chosen randomly between MIN_C and MAX_C.
2. The deadline of a task T_i (primary copy) is randomly chosen in the range $r_i + 2c_i$ and $(1 + R) \times$ SC, where SC is the *shortest completion time* of the task set generated in the previous step.
3. The backup copies are assumed to have the same attributes of the primary copies including the resource requirements.

The performance metric is the schedulability of task sets, called *success ratio*, which is defined as the ratio of the number of task sets found feasible by the fault-tolerant myopic algorithm to the number of task sets considered for scheduling.

Each point in the performance curves (Figs. 3–6) is the average of five simulation runs, each with 100 task sets. Each task set contains approximately 100 primary copies by fixing the *schedule length* to 200 during the task set generation. In our study, the mutual exclusion of time and space is applicable only to the processors, but not to the resources. In each of Figs. 3–6, the first plot is for four processors with three resources and the second plot is for six processors with four resources. In all cases, the number of instances of every resource is taken as two. For Figs. 4 and 5, the cost of scheduling algorithm is fixed by fixing the number of backtracks.

In all the plots, the success ratio increases with increasing values of *distance* initially, and then starts decreasing at higher values of *distance*. The peaks are at different *distance* values for different number of processors. This behaviour depends on the value of K and the reason for this is as follows.

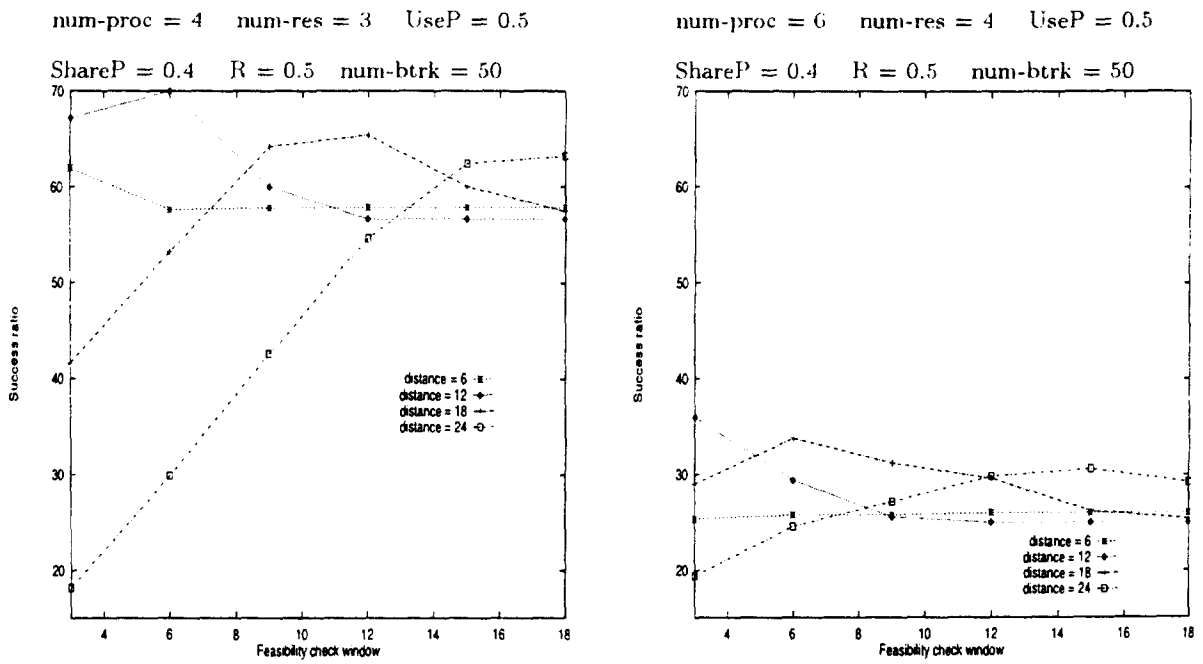


Fig. 3. (a) Effect of feasibility window with four processors; (b) Effect of feasibility window with six processors.

1. When *distance* is low, the positions of backup copies in the task queue is close to their respective primary copies and hence the possibility of scheduling these backup copies may get postponed (we call this, *backup postponement*) due to time and space exclusions. This makes more and more unscheduled backup copies getting accumulated. When this number exceeds K, the scheduler is forced to choose the best task (say T_b) among these backup copies, which results in creation of a hole (i.e., unusable time interval for scheduling) in the schedule since $\mathrm{EST}(T_b)$ is greater than the *availtime* of idle processors. This hole creation can be avoided by moving the feasibility window till a primary task falls into it. However, we do not consider this approach since it increases the scheduling cost.

2. When *distance* is high, the position of the backup copies in the task queue is far apart from their respective primary copies, i.e., tasks (backup copies) having lower deadlines may be placed after some tasks (primary copies) having higher deadlines. This may lead to backtracks when the feasibility check window reaches these backup copies (we call this, *forced backtrack*).

4.1.1. Effect of size of the feasibility check window

Fig. 3 shows the effect of varying feasibility check window (K) on success ratio when *distance* is equal to 6, 12, 18, and 24. Note that for larger values of K, the number of H function evaluations in a feasibility window is also larger, which gives rise to increase in scheduling cost, i.e., fixing the number of backtracks does not fix the scheduling cost for different values of K.

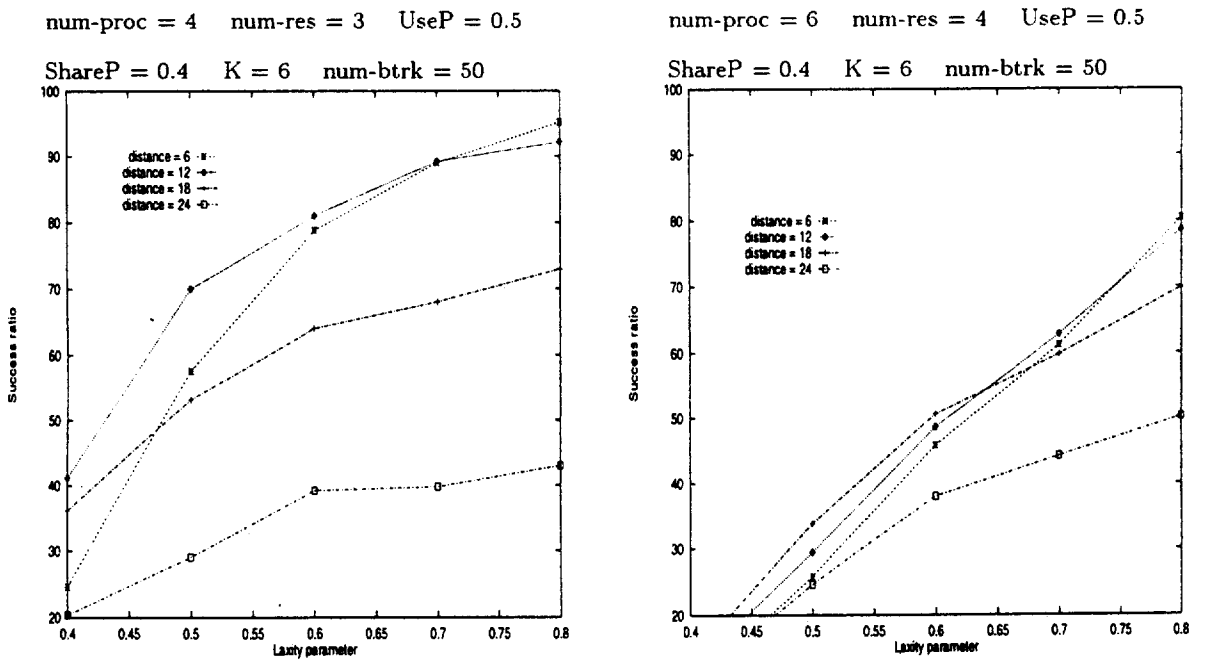


Fig. 4. (a) Effect of task laxity with four processors; (b) Effect of task laxity with six processors.

The success ratio increases with increasing K values for some time (*growing phase*) and then starts decreasing for higher values of K (*shrinking phase*). The shrinking phase starts at lower K values for lower values of *distance*, and at higher K values for higher values of *distance*. The trend is the same in both the plots (more clear in Fig. 3(a)). For example in Fig. 3a, 3, 6, and 12 are the values of K at which the shrinking phase starts when *distance* is equal to 6, 12, and 18, respectively. The reason for this is that the backup postponement is very high at the beginning of the growing phase, decreases along with it and reaches the lowest value at the end of it (equivalently, beginning of the shrinking phase), and the number of forced backtracks is very low at the beginning of the shrinking phase and increases along with it. This reveals three facts: (a) increased value of K does not necessarily increase the success ratio, (b) the optimal K for each *distance* is different, and (c) the global optimal *distance* is different for different numbers of processors. The right combination of K and *distance* offers the best success ratio.

4.1.2. *Effect of laxity parameter*

The effect on success ratio for various *distance* values by the laxity parameter (R) is shown in Fig. 4. For this, *UseP*, K, and *num_btrk* have been fixed at 0.5, 6, and 50, respectively. For lower values of laxity parameter (R), the impact on success ratio by distance is less significant compared with the one at higher values of R. This is due to the fact that for lower values of R, the deadline is very tight and hence there is less flexibility in making a feasible schedule (i.e., the number of feasible schedules is less). This reason is applicable for all

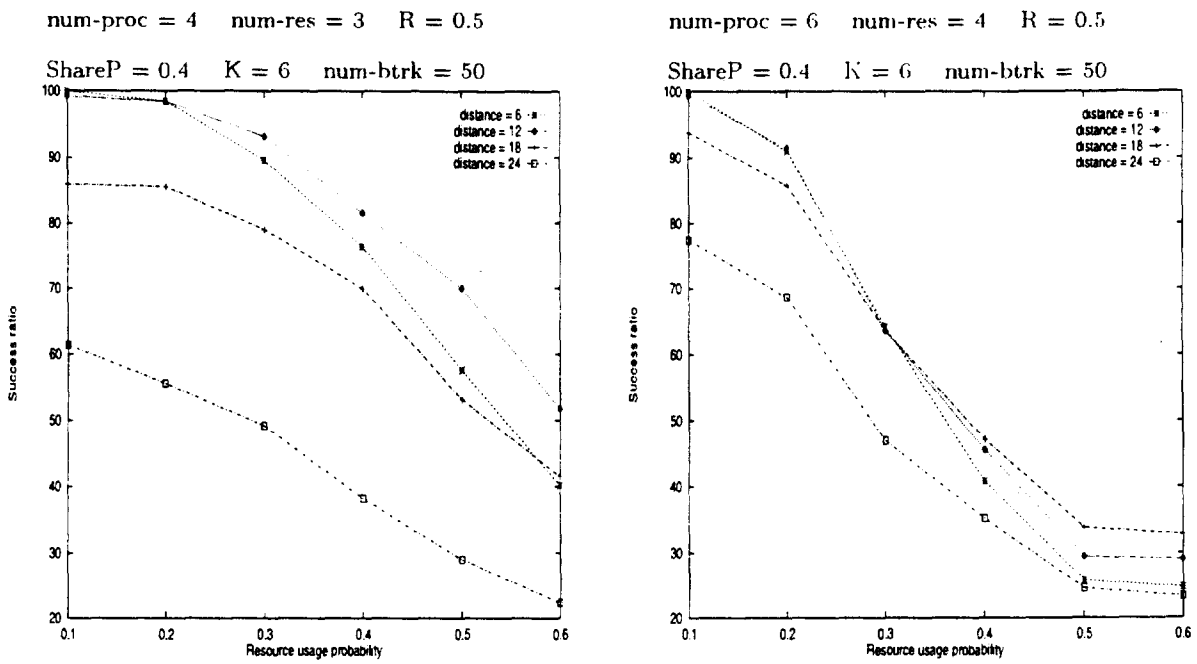


Fig. 5. (a) Effect of resource usage with four processor; (b) Effect of resource usage with six processors.

the *distance* values, but for higher R, the impact of the *distance* parameter is very effective since there is more flexibility in obtaining a feasible schedule.

4.1.3 Effect of resource constraints

Fig. 5 shows the effect on success ratio by the resource usage probability (*UseP*) for various values of *distance*. For all values of *UseP*, the trend of the success ratio remains the same (when *UseP* increases, success ratio decreases linearly) for all values of *distance*. This is because of resources not being considered for space and time exclusions. If *UseP* is fixed and *ShareP* is varied, one can expect a reverse of the above trend (when *ShareP* increases, success ratio increases linearly). If the resources are also considered for space and time exclusions, the trends might be different for different *distance* values.

4.1.4. Effect of number of backtracks

The effect of number of backtracks (*num_btrk*) on success ratio for varying *distance* has been plotted in Fig. 6 by fixing *UseP*, K, and R at 0.5, 6, and 0.6, respectively. The impact of *num_btrk* on success ratio for all values of *distance* is less significant compared to other parameters such as K, R, and *UseP*. This clearly shows that minor increments in number of backtracks without other interventions does not really improve the schedulability.

4.2. Studies related to scheduling and resource reclaiming

In this section, we present our simulation results considering scheduling with associated resource reclaiming. In this study, there is no concept of task set, instead, tasks arrive and get scheduled

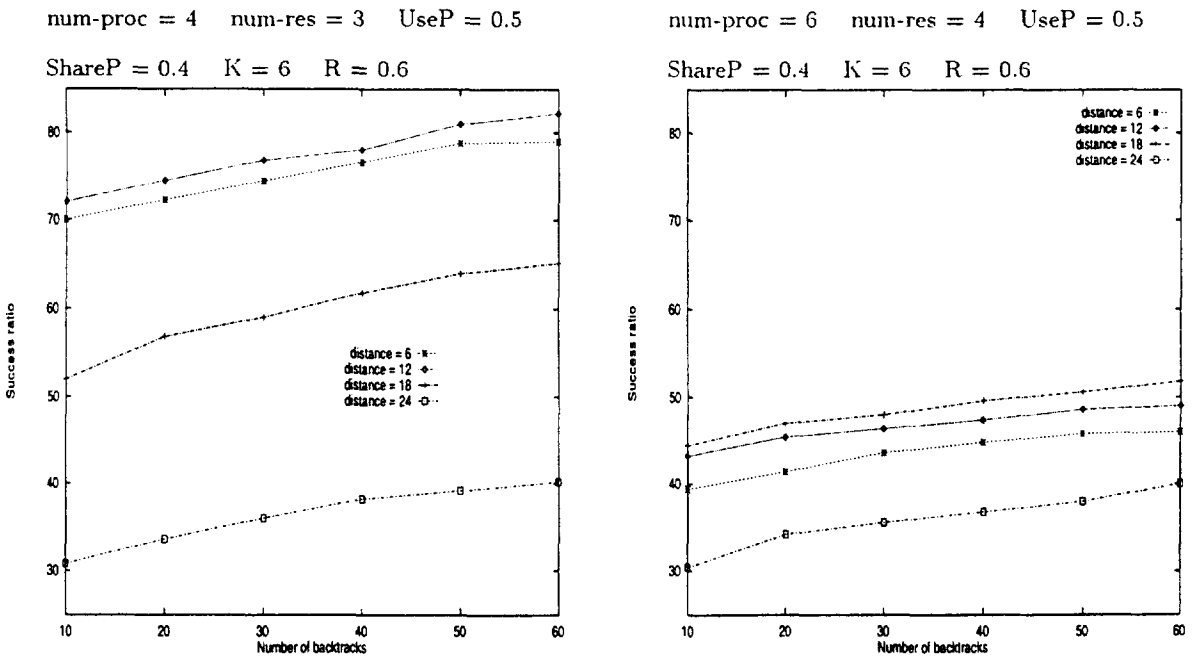


Fig. 6. (a) Effect of backtracks with four processor; (b) Effect of backtracks with six processors.

dynamically. Therefore, the metric is the *guarantee ratio* which is defined as the number of tasks whose deadlines are met by a scheduling algorithm to the number of tasks that have arrived in the system.

We have used the Restriction Vector (RV) based algorithm [14] for resource reclaiming. Each task T_i has an associated m-component vector, $RV_i[1..m]$, called Restriction Vector, where m is the number of processors. $RV_i[j]$ for a task T_i contains the *last task* in the set of tasks which have been scheduled on processor P_j to finish before task T_i begins and have resource constraints with T_i. For computing RV of a task T_i, the scheduler has to check at most k tasks in each of the other processors' dispatch queues. If T_i does not have any resource conflict with all the k tasks of a dispatch queue, the kth task becomes the restriction. The RV algorithm [14] says: *start executing a task only if all the tasks in its restriction vector have finished their execution.*

The parameter, in this study, *fault* – *p*, refers to the probability that a primary task fails the acceptance test and the parameter *aw* – *ratio* refers to the ratio of actual computation time to worst case computation time of a task. The inter-arrival time of tasks is exponentially distributed with mean $1/(\lambda \times m) \times$ (MIN_C + MAX_C)/2. The simulation results (Figs. 7 and 8) show the similar trend as observed when considering only the scheduling without resource reclaiming.

5. Conclusions

In this paper, we have brought out a new parameter called *distance* which affects the

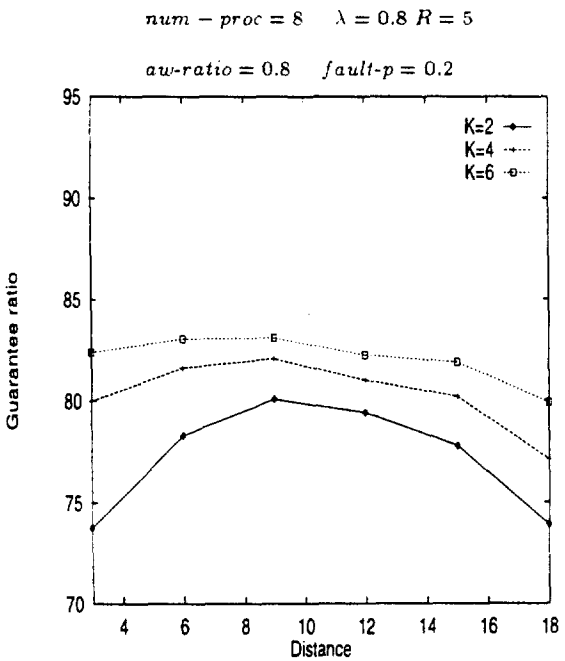


Fig. 7. Fault-tolerant myopic with varying distance.

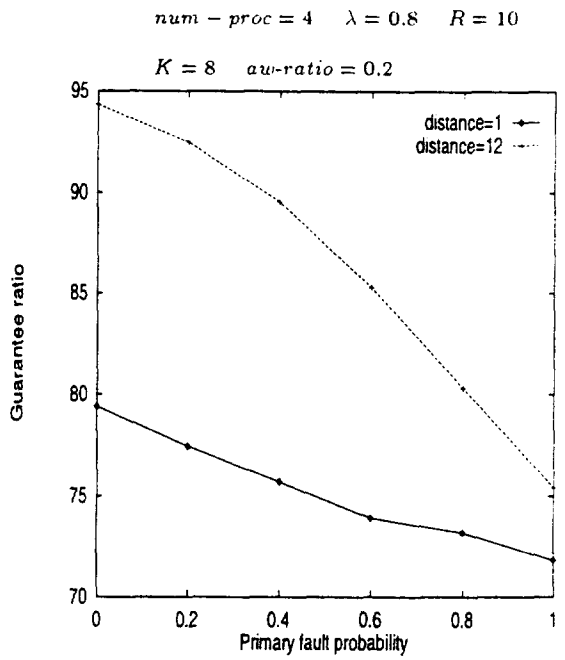


Fig. 8. Effect of primary fault probability.

schedulability of the fault-tolerant dynamic scheduling algorithms. To investigate its importance, we have proposed extensions to the well-known myopic scheduling algorithm and studied, through simulation, the impact of it on the schedulability of fault-tolerant myopic algorithm. We have evaluated our fault-tolerant scheduling algorithm considering (i) scheduling alone and (ii) scheduling and resource reclaiming. From our studies, the following inferences are made.

- For different number of processors the optimal distance parameter is different. From our simulation study, for four and six processors, the *distance* parameter 12 and 18, respectively, offer better success ratio than the other values of *distance* in most of the cases.
- Increasing the size of the feasibility check window (K) does not necessarily increase the success ratio. The right combination of K and *distance* decides the effectiveness of the scheduling algorithm.
- For lower values of laxity parameter (R), the impact on success ratio by *distance* is less significant compared to at higher values of R.
- For all values of resource usage probability, the effect on success ratio by various values of *distance* offer similar trend. This is because of non-consideration of resources for space and time exclusions.
- The impact of number of backtracks is less significant compared to the other parameters such as K, R, and *UseP* for many values of *distance*.
- In general, the effectiveness of fault-tolerant scheduling algorithms is heavily influenced by the choice of *distance* parameter along with other parameters.

References

[1] K.G. Shin, P. Ramanathan, Real-time computing: A new discipline of computer science and engineering, Proc. IEEE, vol. 82. no. 1, pp. 6–24 Jan. 1994.

[2] K. Ramamritham, J.A. Stankovic, Scheduling algorithms and operating systems support for real-time systems, Proc. IEEE, vol. 82, no. 1, pp. 55–67, Jan.1994.

[3] K. Ramamritham, J.A. Stankovic, Perng-Fei Shiah, Efficient scheduling algorithms for real-time multiprocessor systems, IEEE Trans. Parallel Distr. Systems. vol. 1. no. 2, pp. 184–194, Apr. 1990.

[4] W. Zhao, K. Ramamritham, J.A. Stankovic, Scheduling tasks with resource requirements in hard real-time systems, IEEE Trans. Software Eng., vol. 13, no. 5, pp. 564–577, May 1987.

[5] Ben A. Blake and Karsten Schwan, Experimental evaluation of a real-time scheduler for a multiprocessor system, IEEE Trans. Software Eng., vol. 17, no. 1, pp. 34–44, Jan. 1991.

[6] Jia Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence and exclusion constraints, IEEE Trans. Software Eng., vol. 19, no. 2, pp. 139–154, Feb. 1993.

[7] A.A. Bertossi, L.V. Mancini, Scheduling algorithms for fault-tolerance in hard-real-time systems, Real-Time Systems, vol. 7, no. 3, pp. 229–245, 1994.

[8] A.L. Liestman, R.H. Campbell, A fault tolerant scheduling problem, IEEE Trans. Software Eng., vol. 12, no. 11, pp. 1089–1095, Nov. 1986.

[9] Yingfeng Oh and Sang Son, Multiprocessor support for real-time fault-tolerant scheduling, In IEEE Workshop on Architectural Aspects of Real-time Systems, pp. 77–80, Dec. 1991.

[10] S. Ghosh, R. Melhem, D. Mosse, Fault-tolerant scheduling on a hard real-time multiprocessor system, In International Parallel Processing Symposium, Apr. 1994.

[11] D. Mosse, R. Melhem, S. Ghosh, Analysis of a fault-tolerant scheduling algorithm, In IEEE Fault Tolerant Computing Symposium, pp. 16–25, 1994.

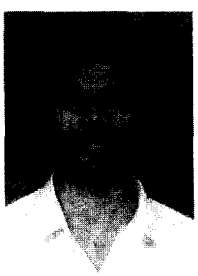
[12] Chia Shen, K. Ramamritham, J.A. Stankovic, Resource reclaiming in multiprocessor real-time systems, IEEE Trans. Parallel Distributed Systems, vol. 4, no. 4, pp. 382–397, Apr. 1993.

[13] M.R. Garey, D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, W.H. Freeman Company, San Francisco, 1979.

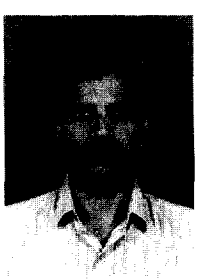
[14] G. Manimaran, C. Siva Ram Murthy, Machiraju Vijay, K. Ramamritham, New algorithms for resource reclaiming from precedence constrained tasks in multiprocessor real-time systems, to appear in Journal of Parallel and Distributed Computing.

[15] C.M. Krishna, K.G. Shin, On scheduling tasks with a quick recovery from failure, IEEE Trans. Computers, vol. 35, no. 5, pp. 448–455, May 1986.

[16] R.L. Graham, Bounds on multiprocessing timing anomalies, SIAM J. Appl. Math., vol. 17, no. 2, pp. 416–429 Mar. 1969.



G. Manimaran obtained a B.E. degree in Computer Science and Engineering from Bharathidasan University, Trichirapalli, in 1989, and M. Tech., in Computer Technology from Indian Institute of Technology, Delhi, in 1993. He is currently a doctoral student at the Indian Insitute of Technology, Madras. His research interest include scheduling in parallel and distributed real-time systems, real-time networks, and fault-tolerant computing.



C. Siva Ram Murthy obtained a B.Tech. degree in electronics and communications engineering from REC, Warangal, in 1982, and M.Tech., in computer engineering from the Indian Institute of Technology (IIT), Kharagpur, in 1984, and a Ph.D. in computer science from the Indian Institute of science (IISc), Bangalore, in 1988. From March 1988 to September 1988 he worked as a Scientific Officer in the Supercomputer Education and Research Certre at IISc. He subsequently joined IIT, Madras, as a Lecturer of Computer Science and Engineering, and became an Assistant Professor in August 1989. He is currently an Associate Professor at the same place. From October 1992 to December 1992 he was a Visiting Scientist at the German National Research Centre for Computer Science (GMD), Sankt Augustin, and from July 1995 to December 1995, a Visiting scholar at the University of Washington, Seattle, USA. He is a recipient of the Seshagiri Kaikini Medal for the best Ph.D. thesis and also the Indian National Science Academy Medal for Young Scientists.