A Fault-tolerant Dynamic Task Scheduling Algorithm for Real-time Systems on Heterogeneous Multiprocessor

Ming-Dien Chang, Yi-Hsuan Lee, Wen-Pin Liu, and Cheng Chen

Department of Computer Science and Information Engineering
1001 Ta Hsueh Road, Hsinchu, Taiwan, 30050, Republic of China
Tel: (8863) 5712121 EXT: 54734, Fax: (8863) 5724176
E-mail: {mdchang, yslee, wpliu, cchen}@csie.nctu.edu.tw

Abstract

Real-time Systems are being increasingly used in several applications which are time critical. Tasks corresponding to these applications have deadlines to be met. Fault-tolerance is an important requirement of such systems, due to the catastrophic consequences of not tolerating faults. In this paper, we propose density first with minimum non-overlap scheduling algorithm (*DNA*)*, which schedule real-time tasks dynamically with primary/backup* (*PB*) *scheme on heterogeneous multiprocessor. In DNA we define a heuristic function density to prioritize tasks. Besides, we also propose minimum non-overlap* (*MNO*) *mechanism for backup scheduling. According to our simulations, DNA can achieve higher guarantee ratio in any environment compared with related methods.*

1 Introduction

Real-time systems are defined as those systems in which its correctness depends not only on the logical result of computation, but also on the time at which the results are produced [1-3]. It can be broadly classified into three categories [4]. Among them the hard real-time system is strictest, and in this paper we will focus on it.

Due to the critical nature of tasks in a hard real-time system, fault-tolerance becomes an important issue [1-2]. In multiprocessor systems, fault-tolerance can be provided by scheduling multiple versions of tasks on different processors [5-6]. Among different schemes for fault-tolerant scheduling, we choose the primary/backup (PB) scheme which is the most popular one.

Some effective scheduling algorithms used for real-time multiprocessor system have been proposed [1-2]. Most of them are designed for homogeneous system, and we have enhanced them to heterogeneous multiprocessor [8]. In these algorithms, tasks with earlier deadlines will be given higher scheduling priorities. However, a task with smaller *schedulable interval* should be schedule early to decrease the reject ratio. Thus, in our *density first with minimum non-overlap scheduling algorithm* (*DNA*), tasks are prioritized with a new heuristic function *density*. Furthermore, we also propose *minimum non-overlap* (*MNO*) mechanism for backup scheduling, which can minimize the reserved time intervals for backups. Based on simulation results, DNA can achieve higher guarantee ratio in any environment compare with other related methods.

The remainder of this paper is organized as follows. Section 2 describes the system model and related work. Design issues and principles of DNA are introduced in Section 3. In Section 4, some experimental results are given. Finally, we give some conclusions in Section 5.

2 Fundamental Background

2.1 System, Task, and Fault Models [1-2]

The *heterogeneous multiprocessor* consists of *m* processors $P_1 \ldots P_m$ connected by a network. Every processor may fail due to hardware or software failure.

The faults are independent, and only occur in one processor at a time.

Real-rime task scheduling algorithms usually assume that tasks are independent, because [8] have proven that precedence constraints can be actually removed. Thus, we simply assume tasks are aperiodic, independent, non-preemptive, and not parallelizable. Every task T_i has following attributes: *ready time* (r_i), *computation time* on processor P_j (c_{ij}), and *deadline* (d_i). Each task T_i has *primary* (Pr_i) and *backup* (Bk_i) copies with identical attributes. Since tasks are not parallelizable, $d_i - r_i$ should be long enough to schedule both primary and backup copies of T_i.

In dynamic multiprocessor scheduling, all tasks arrive at a central scheduler and execute on other processors. The scheduler runs in parallel with other processors, and periodically schedules newly arriving tasks with a small time quantum or as a task arrive. Simply, we assume the scheduler is fault free.

2.2 Related Work

Because PB scheme schedules two copies of a task on different processors, the entire schedulability is obviously decreased. Therefore, *BB-overloading* technique, proposed in [1], describes that Bk_i and Bk_j scheduled on the same processor can be overlapped if Pr_i and Pr_j are scheduled on different processors. *Backup deallocation* reclaims resources reserved for backups when their corresponding primaries complete successfully, which is another technique to increase the entire schedulability [1].

Distance myopic algorithm (*DMA*) is a heuristic search algorithm that schedules real-time tasks on homogeneous multiprocessor with fault-tolerant [2]. It treats Pr_i and Bk_i of task T_i as separate tasks and constructs a task queue according to deadlines and variable *distance*. DMA contains two features doing scheduling. The first one is using a *feasibility check window* (with size *K*) to achieve look-ahead nature. The second one is to use an integrated heuristic function to select task. It will rearrange the sequence of tasks being scheduled, which can lead to select the most appropriate task. Moreover, DMA has the capability of *backtracking*. If the current schedule cannot be extended any more, it will deallocate the last scheduled task and try to schedule another one.

In [7], we extend DMA to heterogeneous multiprocessor and propose *heterogeneous distance myopic algorithm* (*HDMA*). Besides, we also design *fault-tolerant myopic algorithm* (*FTMA*), which is modified from HDMA [7]. In FTMA, primaries and backups are ordered in separate task queues. In each scheduling step, the first primary or backup with smaller heuristic value will be moved into the feasibility check window. Compared with HDMA, FTMA can select more appropriate tasks to schedule every step. Thus, from simulation results, FTMA exactly outperforms HDMA in any environment.

3 Density First with Minimum Non-overlap Scheduling Algorithm (DNA)

In the first two subsections, we will introduce a new heuristic function *density* and the *minimum non-overlap* (*MNO*) mechanism respectively. The overall scheduling steps of our *density first with minimum non-overlap scheduling algorithm* (*DNA*) will be listed in Section 3.3.

3.1 Density Heuristic Function

As mentioned above, almost all previous methods use deadline to prioritize tasks. However, in order to decrease the reject ratio, a task with smaller *schedulable interval* seems much urgent to be schedule first. We define a *density* heuristic function, which gives higher priorities for urgent tasks.

Definition 3.1 For a task T_i, the *latest finish time of primary* (*LFP*) is defined as

$$LFP(T_i) = d_i - \mathbf{min}\{c_{ij}\} \quad (1)$$

Definition 3.2 For a task T_i, $EST_j(Pr_i)$ indicates the *earliest start time* of its primary on processor P_j.

$EST_j(Pr_i) = \mathbf{max}\{r_i, avail(j)\}$, where $avail(j)$ is the time that P_j is available to execute Pr_i (2)

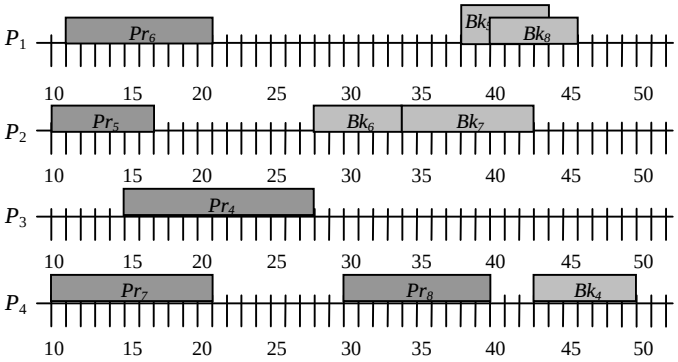

Figure 1. The partial schedule.

Table 1. Attributes of T_9 to T_{11}.

	T_9	T_{10}	T_{11}
r_i	10	10	10
d_i	45	50	37
c_{i1}	5	15	5
c_{i2}	11	14	7
c_{i3}	15	11	5
c_{i4}	10	10	8

Table 2. Density calculation for T_9 to T_{11} in Table 1.

	LFP	*ESB*	*availP* (Pr_i)	*availP* (Bk_i)	*Pr_m*	*Bk_m*	∑ *prslot*	∑ *bkslot*	***density***
T_9	40	26	P_1, P_2	P_1, P_2, P_3	8	10.33	28	55	**0.22**
T_{10}	40	36	P_1, P_3	P_2, P_3, P_4	13	11.67	29	38	**0.37**
T_{11}	32	15	P_1, P_2, P_3, P_4	P_1, P_2, P_3, P_4	6.25	6.25	36	54	**0.14**

Definition 3.3 For a task T_i, the *earliest finish time of primary* (*EFP*) is defined as

$$EFT(T_i) = \mathbf{min}\{EST_j(Pr_i) + c_{ij}\} \quad (3)$$

We assume that Bk_i should be started after Pr_i finishes, hence, the *earliest start time of backup* (*ESB*) is defined as $ESB(T_i) = EFP(T_i)$ (4)

Definition 3.4 For a task T_i, *availP* (Pr_i) contains all processors that can execute Pr_i between r_i and *LFP* (T_i) without overlapping with scheduled primaries and backups.

Definition 3.5 For a task T_i, *availP* (Bk_i) contains all processors that can execute Bk_i between *ESB* (T_i) and d_i without overlapping with scheduled backups.

Definition 3.6 For a processor P_j, $prslot_{ij}$ and $bkslot_{ij}$ are lengths of time interval on it which can successfully execute Pr_i and Bk_i respectively.

Definition 3.7 For a task T_i, *Pr_m* (T_i) and *Bk_m* (T_i) indicate the average computation time of Pr_i and Bk_i on *availP* (Pr_i) and *availP* (Bk_i) respectively.

$$Pr\_m(T_i) = \sum_{j \in availP(Pr_i)} c_{ij} \Big/ \left|availP(Pr_i)\right| \quad (5)$$

$$Bk\_m(T_i) = \sum_{j \in availP(Bk_i)} c_{ij} \Big/ \left|availP(Bk_i)\right| \quad (6)$$

Definition 3.8 For a task T_i, we define its *density* heuristic function as follows:

$$density(T_i) = \frac{Pr\_m(Ti) + Bk\_m(T_i)}{\sum_{j \in availP(Pr_i)} prslot_{ij} + \sum_{j \in availP(Bk_i)} Bkslot_{ij}} \quad (7)$$

For example, Figure 1 and Table l contains a partial schedule and attributes of task T_9 to T_{11}. Table 2 lists all variables defined above of T_9 to T_{11}.

3.2 Minimum Non-overlap (MNO) Mechanism

In PB scheme, both primary and backup must be scheduled before deadline to tolerant processor failure. However, the time intervals reserved for backups are redundant if their primaries finish successfully. Thus, minimizing processor time occupied by backups is an issue for increasing schedulability.

In other algorithms, backup usually be scheduled ALAP or overlapped with other scheduled backups (primaries) as much as possible. Because computation times of a task on heterogeneous processors are different, maximizing the overlapped time intervals may still occupy longer non-overlapped intervals. Therefore, we propose *minimum non-overlap* (*MNO*) mechanism for backup scheduling, which minimizes non-overlap time intervals to reserve much time for unscheduled tasks. For example, from Table 2, we select T_{10} to schedule because it has the maximum density. After scheduling Pr_{10} to P_1 in time interval

1. Calculate the density heuristic values for all tasks initially
2. (a) Select T_i with the maximum density heuristic value
(b) Schedule Pr_i by EFT
(c) **if** (Pr_i is scheduled successfully)
Schedule Bk_i by MNO mechanism
if (Bk_i is scheduled successfully)
Modify heuristic values of tasks
else Deallocate Pr_i and reject T_i
else Reject T_i
3. Repeat Step 2 until all tasks are scheduled

Figure 2. The scheduling steps of DNA.

(21, 36), Bk_{10} can be scheduled to P_2 in (36, 50), or to P_3 in (36, 47), or to P_4 in (40, 50). Lengths of non-overlap time intervals on P_2 to P_4 are 7, 11, and 3 respectively. Hence, Bk_{10} will be scheduled to P_4.

3.3 DNA Algorithm

After introducing heuristic function density and MNO mechanism, Figure 2 contains the scheduling steps of our DNA algorithm. Different from three methods mentioned in Section 2.2, DNA schedules both copies of a task simultaneously. This feature makes DNA more efficient, because it avoids the reconstructing of task queues, the checking of strong feasibility, and backtracking. Notices that either primary or backup of a task cannot be successfully scheduled, the task should be rejected.

The complexity of DNA is $O(n^2)$, where n is the number of tasks in the task queue. Since our central scheduler schedules newly arriving tasks periodically with a small time quantum or as a task arrive, we believe that n shall not be very large.

4 Performance Studies

4.1 Simulation Environment

We construct a dynamic simulation environment to evaluate DNA. It contains two parts named task generator and scheduler, and we describe them in detail in the following.

The task generator generates a set of real-time tasks in the non-decreasing order of ready times. Table 3 lists parameters used in the task generator, which can generate task set with any characteristic. The computation times of a task are chosen uniformly between (MIN_C, $MIN\_C + (MAX\_C - MIN\_C) \times h$), where h is the heterogeneity of computation times of a task. The inter-arrival times between tasks is exponentially distributed with mean ($MIN\_C + MAX\_C$) / $2lP$ [2]. We also simulate bursts of tasks, which the mean of inter-arrival times becomes MIN_C / $10lP$. In order to make sure that both copies can be successfully scheduled, the deadline of task T_i is chosen uniformly between (r_i + **max** c_{ij} + **second max** c_{ij}, $r_i + R \times$ **max** c_{ij}).

We construct a discrete-event dynamic scheduler, which simulates events including task arrivals, task starts and completions, start of scheduling, backup deallocations, and occurrences of fault. The failure may be due to hardware or software. A software fault terminates the task that causes the fault immediately, and the hardware fault caused by processor failure may be permanent or transient. The failed processor with permanent fault will never be available, and will be available after *MAX_Recovery* if the fault is transient. Related parameters and probabilities for failure events are listed in Table 4.

4.2 Experimental Results

In this subsection, we evaluate performances of DNA, HDMA, and FTMA. The total number of tasks is 20,000, and 20 task sets are generated for each set of parameters. Because HDMA and FTMA require additional variables K and *distance*, we evaluate them with various K and *distance* combinations and select the best result. As for the objective, we use the *guarantee ratio* (GR) defined below that means the percentage of tasks whose deadlines are met [9].

$$GR = \frac{number\ of\ tasks\ whose\ deadlines\ are\ met}{total\ number\ of\ tasks\ arrived\ in\ the\ system} \times 100\% \quad (8)$$

With fixed probability of primary fails (*FaultP* =0.2), Figures 3~5 show results of various task arrival rate, laxity, and number of processors, respectively.

Table 3. Parameters for task generator.

parameter	explain	range
MIN_C	min. computation time	10
MAX_C	max. computation time	80
l	task arrival rate	(0.3, 0.9)
R	laxity	(2, 7)
P	number of processors	(3, 10)
BurstP	probability of a burst	*l* / 100
MIN_Burst	min. number of tasks for a burst	10
MAX_Burst	max. number of tasks for a burst	30

Table 4. Parameters for fault probability.

parameter	explain	range
FaultP	probability that a primary fails	(0, 0.5)
Soft_FP	probability that a primary fails due to software fault	0.2
Hard_FP	probability that a primary fails due to hardware fault	0.8
PermHard_FP	probability that a hardware fault is permanent	10^{-6}
MAX_Recovery	max. recovery time after a transient hardware fault	50

These results clearly indicate that DNA outperforms HDMA and FTMA, which means using the *density* heuristic function can select more appropriate tasks to schedule at each scheduling step. Meanwhile, the GR is higher with lower *l*, larger *R*, and larger *P*. The reason of lower *l* and larger *R* seems trivially. As for larger *P*, using MNO mechanism will schedule backups as tight as possible, so more time intervals can be reserved for primaries when *P* is larger.

In Figures 6~8, the probability of primary fails is varied with parameters *l*, *R*, and *P*. While the fault probability increases, more backups are active to be executed. So, less time intervals reserved for backups can be reutilized, which decreases the entire GR. In these results, DNA still has higher GR than that of FTMA in any simulation environment. That is to say, our DNA is not only an efficient algorithm, but also quite effective compared with related methods.

5 Concluding Remarks

In this paper, we propose a fault-tolerant dynamic task scheduling algorithm DNA for real-time systems on heterogeneous multiprocessor. In DNA we design *density* heuristic function and MNO mechanism, which are used for task prioritizing and backup scheduling. According to our simulation results, DNA achieves higher *guarantee ratio* than other related methods in any simulation environment. Besides, it is quite efficient, which is appropriate to be used in real-time systems.

Reference

[1] S. Ghosh, R. Melhem, and D. Mosse, "Fault-tolerance Through Scheduling of Aperiodic Tasks in Hard Real-time Multiprocessor Systems", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 8, No. 3, pp. 272-284, March 1997.

[2] G. Manimaran and C. S. M. Murthy, "A Fault-tolerant Dynamic Scheduling Algorithm for Multiprocessor Real-time Systems and Its Analysis", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 9, No. 11, pp. 1137-1152, Nov. 1998.

[3] K. Ramamritham and J. A. Stankovic, "Scheduling Algorithms and Operating Systems Support for Real-time Systems", *Proc. of IEEE*, Vol. 82, No. 1, pp. 55-67, Jan. 1994.

[4] K. G. Shin and P. Ramanathan, "Real-time Computing: A New Discipline of Computer Science and Engineering", *Proc. of IEEE*, Vol. 82, No. 1, pp. 6-24, Jan. 1994.

[5] A. L. Liestman and R. H. Campbell, "A Fault-tolerant Scheduling Problem", *IEEE Trans. on Software Engineering*, Vol. 12, No. 11, pp. 1089-1095, Nov. 1988.

[6] Y. Oh and S. Son, "Multiprocessor Support for Real-time Fault-tolerant Scheduling", *Proc. of*

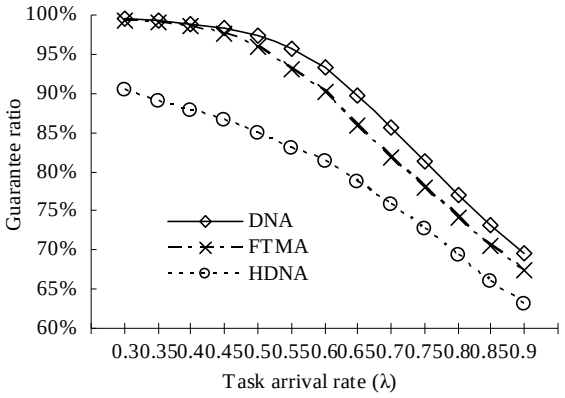

Figure 3. Effect of task load (R = 3, P = 8).

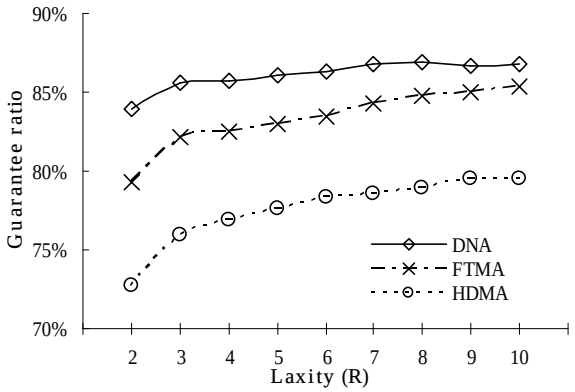

Figure 4. Effect of laxity (l = 0.7, P = 8).

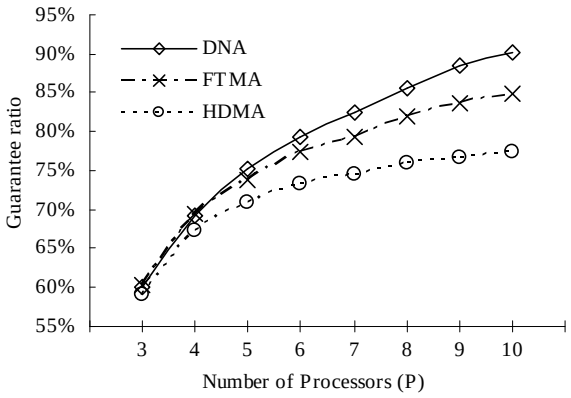

Figure 5. Effect of number of processor (l = 0.7, R = 8).

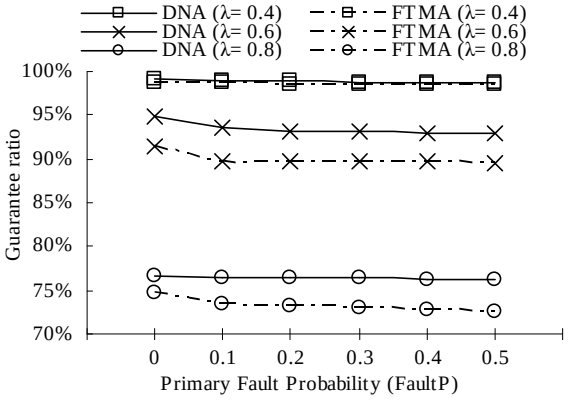

Figure 6. Effect of FaultP with various l (R = 3, P = 8).

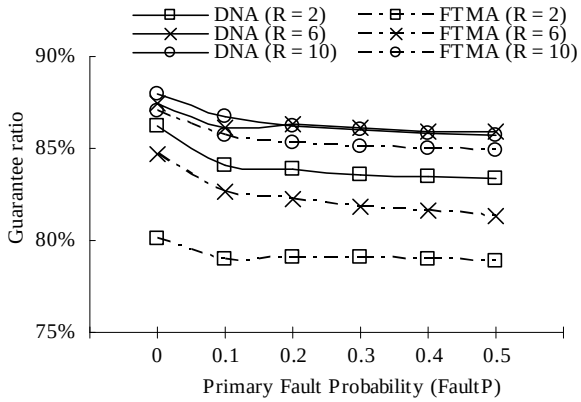

Figure 7. Effect of FaultP with various R (l = 0.7, P = 8).

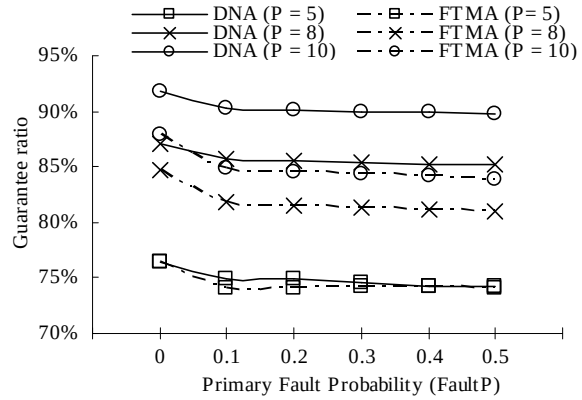

Figure 8. Effect of FaultP with various P (l = 0.7, R = 3).

IEEE Workshop Architectural Aspects of Real-time Systems, pp. 76-80, Dec. 1991.

[7] Y. H. Lee and C. Chen, "Effective Fault-tolerant Scheduling Algorithm for Real-time Tasks on Heterogeneous Systems", *Proc. of National Computer Symposium*, Dec. 2003.

[8] J. W. S. Liu, W. K. Shih, K. J. Lin, R. Bettati, and J. Y. Chung, "Imprecise Computations", *Proc. of IEEE*, Vol. 82, No. 1, pp. 83-94, Jan. 1994.

[9] K. Ramamritham, J. A. Stankovic, and P. -F. Shiah, "Efficient Scheduling Algorithms for Real-time Multiprocessor Systems", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 1, No. 2, pp. 184-194, Apr. 1990.