

Real-time Fault-tolerant Scheduling in Heterogeneous Distributed Systems^{*}

Xiao Qin, Zongfen Han
Liping Pang, Shengli Li

Huazhong University of Science and Technology
Wuhan, China, 430074

Hai Jin

University of Southern California
Los Angeles, California, 90089

Abstract: *Some works have been done in addressing real-time fault-tolerant scheduling algorithms. However, they all based on homogeneous distributed systems or multiprocessor systems, which have identical processors. This paper presents two fault-tolerant scheduling algorithms, RTFTNO and RTFTRC, for periodic real-time tasks in heterogeneous distributed systems. Reliability cost, a main performance metric, is applied. RTFTRC algorithm tries to minimize the reliability cost, while RTFTNO does not consider such metric. The results of the performance evaluation for two algorithms are discussed. Simulation experiments show that RTFTRC has better performance than RTFTNO.*

Keywords: Fault-tolerant, Real-time, Scheduling, Heterogeneous distributed systems, Performance evaluation

1. Introduction

Distributed heterogeneous computing is being increasingly applied to a variety of large size computational problems. In general, such computations consist of multiple heterogeneous modules, which interact with each other to solve the problem [3]. Many real-time systems are also distributed [14]; they consist of a set of heterogeneous or homogeneous nodes interconnected by a real-time communication subsystem. One typical real-time system is mission-critical real-time system, which has two characteristics: guaranteeing real-time applications to meet timing constraints and continuing the specified operations in case of hardware or software failure. Therefore, fault-

tolerance and real-time techniques should be combined in such critical real-time systems [4].

Researchers have made a close study on fault-tolerant theory and practice. For example, [16][7] investigate voting scheme in distributed systems and optimal rollback recovery, respectively. Traditional fault-tolerant techniques also focus on roll-back, which is impossible in many real-time systems. Most traditional fault-tolerant techniques are not suitable in real-time environment, since these techniques assume that tasks in the system have no timing constraints. On the other hand, much work has been done in exploring real-time scheduling algorithms [1][2][9][8]. These real-time scheduling algorithms fall into two categories: static and dynamic scheduling. Most of these scheduling algorithms have an assumption that no error occurs.

Real-time scheduling algorithms with fault-tolerant must be studied in order to make critical real-time systems dependable [6][10]. Such algorithms can improve the reliability of the real-time distributed systems without extra hardware cost. Several special cases of the real-time fault-tolerant scheduling problem have been explored in [5][11]. We have investigated real-time scheduling algorithms that schedule dependable and non-dependable real-time tasks [12]. A hybrid scheduling algorithm [15], which is a combination of static and dynamic scheduling, greatly enhances the performance of static scheduling algorithms discussed in [12].

Most real-time fault-tolerant scheduling algorithms assume that processors in the system are identical. Our main motivation is to

^{*} This work was supported by National Defense Pre-research Foundation of China.

investigate real-time fault-tolerant scheduling algorithms in the heterogeneous distributed systems. We assume that each task has different computation time on different processors.

The paper is organized as follows. In Section 2, the scheduling model and assumptions are presented. Two algorithms of fault-tolerant and real-time scheduling are proposed in Section 3. The performance analysis is presented in Section 4. In Section 5, we summarize the contributions and the open problems of this work.

2. System Model and Assumptions

The real-time task set is represented as $T = \{t_1, t_2, \dots, t_n\}$. Assume that tasks in T are independent to each other and non-preemptive. Task t_i is defined as a tuple, $t_i = (p_i, d_i, t_i^p, t_i^b)$, where p_i and d_i are period and deadline, t_i^p and t_i^b are primary and backup copies. Software versions of t_i^p and t_i^b are identical; $t_i^p = (C_i, s_i^p, \rho_i^p)$, $t_i^b = (C_i, s_i^b, \rho_i^b)$, where s_i^p and s_i^b are start time, ρ_i^p and ρ_i^b indicate on which processor t_i^p and t_i^b are allocated. In heterogeneous distributed system, each task t_i has different computation time on different processors, so a vector C_i is introduced [15], $C_i = [c(i, 1), \dots, c(i, m)]$, $c(i, j)$ represents the computation time of t_i^p and t_i^b on processor p_j .

The heterogeneous distributed system is modeled as a set of processors, $\Omega = \{P_1, P_2, \dots, P_m\}$, $P_i = (\Delta_i, \xi_i, \lambda_i)$, where task t in task set Δ_i is a task allocated on P_i , ξ_i denotes schedule length and λ_i denotes failure rate. We use the same model of reliability as in [13][15]. In such model, processor failures are assumed to be independent, and follow a Poisson Process with the constant failure rate [13][15]. $RC_0(\Omega)$, which denotes the reliability cost with no failure in the system, is defined as below, similar as [13].

$$RC_0(\Omega) = \sum_{j=1}^m \sum_{t_i^p \in \Delta_j} \mathbf{I}_j c(i, j)$$

If processor P_i fails, we use $RC_1(\Omega, i)$ to represent the reliability cost of the system, $RC_1(\Omega, i)$ is given by the following equation,

$$RC_1(\Omega, i) = \sum_{j=1, j \neq i}^m \sum_{t_k^p \in \Delta_j} \mathbf{I}_j c(k, j) + \sum_{j=1}^m \sum_{t_k^p \in \Delta_i \wedge t_k^b \in \Delta_j} \mathbf{I}_j c(k, j)$$

$RC_1(\Omega)$ denotes the reliability cost with one processor failure, it is derived from $RC_1(\Omega, i)$,

$$RC_1(\Omega) = \sum_{i=1}^n (\mathbf{I}_i RC_1(\Omega, i)) / \sum_{j=1}^m \mathbf{I}_j$$

A real-time fault-tolerant task meets its deadline if either its primary copy or backup copy completes before specific deadline. 1-Timely-Fault-Tolerant schedule [11] is defined as a schedule in which no task deadlines are missed in spite of one processor failure. The scheduling goal is to decide a 1-Timely-Fault-Tolerant schedule, which leads to the minimum reliability cost of the system. Our model assumes that all the tasks have the same deadline and the deadline coincides with the period. It is also assumed that if t_i^b and t_j^b are allocated on the same processor, there must be no overlapping in time between them, formally speaking, $\forall i, j \in [1, n], i \neq j: \rho_i^b = \rho_j^b \rightarrow ((s_i^b + c(i, \rho_i^b) \leq s_j^b) \vee (s_j^b + c(j, \rho_j^b) \leq s_i^b))$.

Property 1: If one processor failure is tolerated, the sum of computation time for t_i^p and t_i^b must be less than or equal to the deadline, formally described as, (One processor failure is tolerated) $\rightarrow \forall i \in [1, n]: c(i, \rho_i^p) + c(i, \rho_i^b) \leq DL$.

Property 2: One processor failure is tolerant, for each task t_i , t_i^p and t_i^b must be scheduled on different processors with no overlapping in time between them, thus, (One processor failure is tolerated) $\rightarrow \forall i \in [1, n]: (\rho_i^p \neq \rho_i^b) \wedge (s_i^p + c(i, \rho_i^p) \leq s_i^b)$.

Property 3: For each processor P_j in the system, we have, $\forall j \in [1, m]$:

$$\sum_{t_i^p \in \Delta_j} c(i, j) + \sum_{t_i^b \in \Delta_j} c(i, j) \leq DL.$$

3. Real-time and Fault-tolerant Scheduling

Since scheduling problem is NP-Complete, two heuristic algorithms are proposed in this section. First, we present an algorithm RTFTNO that does not take the reliability cost into consideration. RTFTNO, the traditional static real-time fault-tolerant scheduling, tries to minimize the schedule length on each processor, meanwhile, all tasks can complete before their deadlines. It is presented as follows:

Algorithm RTFTNO (Input: T, Ω, DL ; **Output:** $\text{sch_success}, \Omega, RC_0(\Omega)$)

1. Initialize information in Ω, T and $RC_0(\Omega)$;
2. for each real-time task t_i^p {

```

3. /*  $t_i^p$  is allocated on processor  $P_j$  */
   Find  $j$ , which satisfies:  $\forall k \in [1, m]: \xi_j \leq \xi_k$ ;
4.  $\Delta_j = \{t_i^p\} + \Delta_j$ ;  $\rho_i^p = j$ ;  $s_i^p = \xi_j$ ;  $\xi_j = \xi_j + c(i, j)$ ;
    $RC_0(\Omega) = RC_0(\Omega) + \lambda_j \times c(i, j)$ ;
}
5. for each real-time task  $t_i^b$  {
6. /*  $t_i^b$  is allocated on processor  $P_j$  */
   Find  $j \neq \rho_i^p$ , which satisfies:  $\forall k \in [1, m]$ :
        $k \neq \rho_i^p \wedge \xi_j \leq \xi_k$ ;
7.  $\Delta_j = \{t_i^b\} + \Delta_j$ ;  $\rho_i^b = j$ ;  $s_i^b = \xi_j$ ;  $\xi_j = \xi_j + c(i, j)$ ;
    $RC_0(\Omega) = RC_0(\Omega) + \lambda_j \times c(i, j)$ ;
8. if ( $\xi_j > DL$ ) return(FAIL);
}
9. return(SUCCESS).

```

Compared with RTFTNO, algorithm RTFTRC enhanced the reliability of the system with no extra hardware cost. The objectives of algorithm RTFTRC are to minimize the schedule length and maximize the reliability. For each primary copy t_i^p , it is allocated on the processor with the minimum reliability cost. However, if the schedule length L_p for primary copies on a processor is too long, it may result in the possibility that some backup copies can not meet their deadlines.

For each backup copy t_i^b , it should satisfy three conditions. (1) Allocated on the processor, which is different from that of its primary copy. (2) Allocated on the processor that leads to the increase of reliability cost as minimum as possible. (3) t_i^b can finish before deadline.

Before presenting RTFTRC, we introduce a vector of reliability cost for each task t_i , $RCV_i = [rcv_{i1}, \dots, rcv_{im}]$, where $rcv_{ij} = (rc_{ij}, \rho_{ij})$. rcv_{ij} is derived from vector C_i and Ω , $rc_{ij} = c(i, \rho_{ij}) \times \lambda_{\rho_{ij}}$. Elements in RCV_i is sorted in order of non decreasing reliability cost, in other words, $\forall i \in [1, n]: \forall j, k \in [1, m] (j < k \rightarrow rcv_{ij} \leq rcv_{ik})$. The algorithm RTFTRC is described as follow.

Algorithm RTFTRC(Input: T, Ω, L_p, DL ; Output: $sch_success, \Omega, RC_0(\Omega)$)

```

1. Initialize information in  $\Omega, T$  and  $RC_0(\Omega)$ ;
2. Generate each vector of reliability cost  $RCV_i$  from  $C_i$  and  $\Omega$ ;
3. Sort all vectors of reliability cost in order of non-decreasing reliability cost;
4. for each real-time task  $t_i$  {
5. for ( $j = 1$  to  $m$ ) and (Not find a proper one)
   {
6.  $k = \rho_{ij}$ ;
7. if ( $\xi_k + c(i, k) < L_p$ )
   { /*  $t_i^p$  is allocated on processor  $P_k$  */
8.  $\Delta_k = \{t_i^p\} + \Delta_k$ ;  $\rho_i^p = k$ ;  $s_i^p = \xi_k$ ;
    $\xi_k = \xi_k + c(i, k)$ ;
    $RC_0(\Omega) = RC_0(\Omega) + rc_{ij}$ ;

```

```

}
}
9. if (Can not find a proper processor)
   return (FAIL);
} /* End of Schedule primary copies */
10. /* Schedule backup copies */
for (each real-time task  $t_i$ ) {
11. for (( $j = 1$  to  $m$ ) and (Not find a proper processor)) {
12.  $k = \rho_{ij}$ ;
13. /*  $t_i^b$  is allocated on processor  $P_k$  */
   if (( $MAX(\xi_k, s_i^p + c(i, \rho_i^p)) + c(i, k) < DL \wedge (k \neq \rho_i^p)$ )) {
13.1  $\Delta_k = \{t_i^b\} + \Delta_k$ ;  $\rho_i^b = k$ ;
13.2  $s_i^b = MAX(\xi_k, s_i^p + c(i, \rho_i^p))$ ;
13.3  $\xi_k = s_i^b + c(i, k)$ ;
13.4  $RC_0(\Omega) = RC_0(\Omega) + rc_{ij}$ ;
}
}
14. if (Can not find a proper processor)
   return (FAIL);
} /* End of schedule backup copies */
16. return (SUCCESS).

```

Theorem 1: The time complexity of RTFTRC is $O(nm(3 + \log m))$, where n and m are the numbers of real-time tasks and processors respectively.

Proof: Generation of reliability cost vectors takes $O(nm)$ time. Sorting all RCVs in order of non-decreasing reliability cost takes $O(nm \log m)$ time. Scheduling primary copies and scheduling backup copies are both bounded by $O(nm)$. RTFTRC takes $O(3nm + nm \log m)$, time complexity is therefore $O(nm(3 + \log m))$. ■

Theorem 2: If algorithm RTFTRC is run for a task set that is schedulable, RTFTRC can generate a 1-Timely-Fault-Tolerant schedule for this task set.

Proof: According to property 1-2, we need to prove that for each task t_i : (a) its primary copy t_i^p and its backup copy t_i^b are scheduled on two different processors. (b) no overlapping in time between t_i^p and t_i^b . (c) t_i^p and t_i^b can finish before deadline DL . Thus, we need to prove $\forall i \in [1, n]: (\rho_i^p \neq \rho_i^b) \wedge (s_i^p + c(i, \rho_i^p) \leq s_i^b) \wedge (s_i^b + c(i, \rho_i^b) \leq DL)$.

(1) Since $\forall i \in [1, n]: \rho_i^b = k \wedge k \neq \rho_i^p$ (see step 12. and 13 in algorithm RTFTRC.), we have $\forall i \in [1, n]: (\rho_i^p \neq \rho_i^b)$;

(2) $\forall i \in [1, n]: s_i^b = MAX(\xi_k, s_i^p + c(i, \rho_i^p))$, (see step 13.2 in algorithm RTFTRC), so, $\forall i \in [1, n]: s_i^p + c(i, \rho_i^p) \leq s_i^b$;

(3) $\xi_k = MAX(\xi_k, s_i^p + c(i, \rho_i^p)) + c(i, k) = s_i^b + c(i, k)$ and $MAX(\xi_k, s_i^p + c(i, \rho_i^p)) + c(i, k) < DL$, (see step 13 in algorithm RTFTRC), we can

guarantee that $\forall i \in [1, n]: s_i^b + c(i, k) \leq DL$, since $k = \rho_i^b$, we have $\forall i \in [1, n]: s_i^b + c(i, \rho_i^b) \leq DL$.

According to (1) - (3), if output `sch_success` is SUCCESS, RTFTRC generates a 1-Timely-Fault-Tolerant schedule. The theorem holds. ■

4. Performance Evaluation

The simulation experiments are carried out in following steps, which are similar as [15]. (1) Deadline DL and threshold for schedule length (L_p) are selected. (2) The number of processors and their failure rates are determined. (3) When number of real-time tasks is chosen, the vector C_i for each task t_i is generated, each element c_{ij} in C_i is chosen uniformly between $[5, 100]$. (4) For each task t_i , its reliability cost vector RCV_i is calculated. (5) RTFTNO and RTFTRC are invoked respectively. If the task set is schedulable, $RC_0(\Omega)$ and $RC_1(\Omega)$ can be calculated after the scheduling.

Under the same workload, 10000 task sets are generated independently to run the algorithms. We focus on the average values of $RC_0(\Omega)$ and $RC_1(\Omega)$ for two algorithms. We use *percentage of missed deadlines* (PMD) as metric to determine the schedulability of the algorithm in our performance evaluation. PMD is defined as follow:

$$PMD = \frac{\text{Number of schedules which is not schedulable} \times 100}{\text{Total schedules}}$$

where total schedules is set to 10000 in our experiments.

4.1 Reliability cost

The parameters in the simulation are $DL=1400$, $L_p = 700$ and $m = 5$. Failure rates of processors are 0.9, 0.95, 1, 1.05 and 1.10 respectively, the unit of failure rate is 10^{-6} per hour. Since the metric measured is reliability cost, each task set randomly generated is schedulable for two algorithms. In other words, PMD for each algorithm is always zero. We will discuss PMD in next subsection. As far as RTFTNO algorithm is concerned, there is no difference between RC_0 and RC_1 , so data of RC_1 for RTFTNO is omitted.

From Fig. 1, we see that, as the number of tasks goes up, reliability costs of both algorithms increase. With the same input, reliability cost of RTFTRC is less than that of RTFTNO. This is due to the fact that RTFTRC schedules tasks on the processors which leads to a reliability cost as minimum as possible, while RTFTNO does not

concern about reliability cost. For RTFTRC, data of RC_0 and RC_1 are close. However, with the same number of tasks, RC_1 is a little bit larger than RC_0 . In terms of reliability cost, performance of RTFTRC is better than that of RTFTNO.

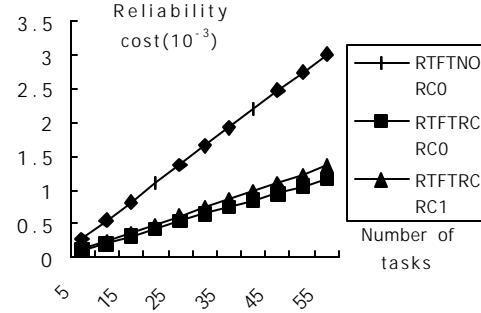


Figure 1 Comparison of RTFTNO and RTFTRC ($m = 5$, $DL = 1400$, $L_p = 700$)

4.2 Relation between computation time and reliability cost

This experiment concerns about the relationship between computation time and the reliability cost. We only exam algorithm RTFTRC, since algorithm RTFTNO has the same feature. Simulation parameters in this experiment are also the same as the first experiments, except that each element c_{ij} in C_i is chosen uniformly between $[5, 200]$. Fig. 2 demonstrates the simulation results. We observe that, when c_{ij} in C_i is chosen uniformly between $[5, 200]$ instead of $[5, 100]$, the reliability cost becomes higher. For example, the number of tasks is set to 45, if computation time in each vector C_i are generated uniformly between $[5, 100]$, the values of RC_1 and RC_0 will be 1.10×10^{-3} and 0.95×10^{-3} respectively.

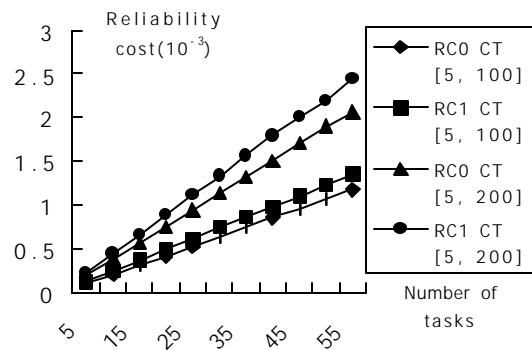


Figure 2 Relationship between computation time and reliability cost ($m = 5$, $DL = 1400$, $L_p = 700$)

If computation time in each vector C_i are chosen according to uniform distribution between $[5, 200]$, RC_1 and RC_0 will be 1.10×10^{-3} and 0.95×10^{-3} . This is due to the fact that, when computation time in each C_i raise, the rc_{ij} in the vector of reliability cost RCV_i also increase. We can conclude that, as the computation time in each vector C_i go up, the reliability cost of the system also increases. The result of this experiment also proves the conclusion drawn from 4.1 that RC_1 is a little bit larger than RC_0 under the same set of parameters.

4.3 Percentage of missed deadlines

The basic simulation parameters are the same as those in experiment 4.1. The metric PMD measures the efficiency of the algorithm. The algorithm with lower PMD has higher efficiency. Table 1 compares PMD between algorithms RTFTNO and RTFTRC. For RTFTNO, PMD is zero when n is less than 56, and if n is greater than 70, PMD will always be 100%. Again, we observe that, PMD in RTFTRC only changes when n is in the interval $[97, 103]$; PMD is equal to zero when n is less than 97 and PMD is 100% when n is greater than 103. It illustrates that, in terms of PMD, performance of RTFTRC is better than that of RTFTNO. Because in the case that n is less than 104, PMD of RTFTRC is always less than that of RTFTNO under the same number of tasks.

For example, we intentionally let PMD be 1%, RTFTRC can schedule 97 tasks, while RTFTNO can only schedule 56 tasks. As shown in table 1, as the number of tasks goes up, PMD of both algorithms increase sharply. The result also proves that a larger task set would mean a less possibility that the task set is schedulable. Next experiment will do a further study on schedulability.

4.4 Comparison of minimum number of processors

Minimum number of processors (MNP) is a metric that measures the schedulability of the scheduling algorithm. In other words, the

algorithm with better schedulability needs fewer processors to run a set of tasks that is given. The algorithm, which is used to calculate the minimum number of processors-FMNP, is devised in [12]. Failure rates of all processors are set to be 10^{-6} per hour, and other basic parameters are the same as those in section 4.2.

Fig. 3 shows the impact of the number of tasks on the minimum number of processors. As the number of tasks increases, the minimum numbers of processors for two algorithms both increase. It is because the simple fact that, the more tasks in the system, the more processors are need in order to guarantee all the tasks to complete before their deadlines.

We compare the minimum number of processors produced by RTFTRC algorithm and RTFTNO algorithm. We found that, when the number of tasks is small ($N < 30$), there is no significant difference of MNP between RTFTRC and RTFTNO. However, such difference increases if the number of tasks raises, and for the same number of tasks, MNP of RTFTNO is greater than that of RTFTRC.

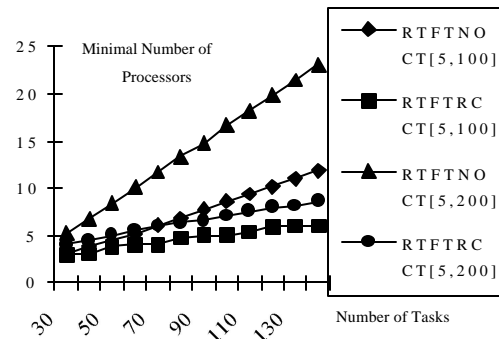


Figure 3 Comparison of minimal number of processors (DL = 1400, $L_p = 600$)

Hence, it is clear that the schedulability of RTFTRC algorithm is better than that of RTFTNO algorithm, especially when the number of tasks is larger. For instance, the number of tasks is set to be 100, if computation time in each vector C_i is taken uniformly between $[5, 100]$, MNP of RTFTNO algorithm

Table 1 Percentage of Missed Deadlines for RTFTNO and RTFTRC
(DL = 1400, $L_p = 700$, $m = 5$)

N	56	58	60	62	64	66	68	70
RTFTNO	1%	8%	24%	50%	74%	89%	97%	99%
N	97	98	99	100	101	102	103	
RTFTRC	1%	2%	3%	5%	35%	97%	99%	

is 5.36. If computation time in each vector C_i is chosen uniformly between [5, 200], MNP of RTFTNO and RTFTRC are 16.64 and 7.09, respectively.

For the same number of tasks, the longer the computation time of each task, the larger the MNP for two algorithms. This is because that, with the same number of tasks, if computation time of each task increases, the workload for the system is also raises. Thus, more processors need to complete the tasks before deadline.

4.5 Relationship between threshold of schedule length and minimum number of processors

The threshold of schedule length (L_p) has no impact on the overall performance of RTFTNO algorithm, though L_p do affect performance of RTFTRC algorithm. It is due to the fact that RTFTNO balance the scheduling length on each processor, while RTFTRC does not do so. Therefore, figure 4 plots the MNP of RTFTRC as a function of the threshold for schedule length. The computation time in each vector C_i is generated uniformly from 5 to 200, and other basic parameters are the same as those in experiment 4.1.

Fig. 4 illustrates MNP with three different number of tasks of 40, 60 and 100, respectively. The result is that the minimum number of processors decreases as L_p increases. However, if the threshold of the schedule length has insignificant influence on MNP as L_p exceeds 700. For example, as the number of tasks is 40, MNP is in the range of 4.48 and 4.50; as the number of tasks is 100, MNP only decreases from 7.07 to 7.08. Thus, for RTFTRC algorithm, L_p has little impact on performance of the algorithm if L_p is larger than $DL/2$. As an interesting result, threshold of schedule length, which is an important parameter of load balancing in traditional distributed system, has non-sense in RTFTRC algorithm.

5. Conclusions

Some works have been done in past in real-time fault-tolerant scheduling in homogeneous distributed systems. This paper presents two real-

time scheduling algorithms with fault-tolerance on heterogeneous distributed systems. All processors in heterogeneous distributed system have different failure rates, and each real-time task has different computation time when it is allocated on different processors. Reliability cost is a main performance metric in our scheduling algorithms.

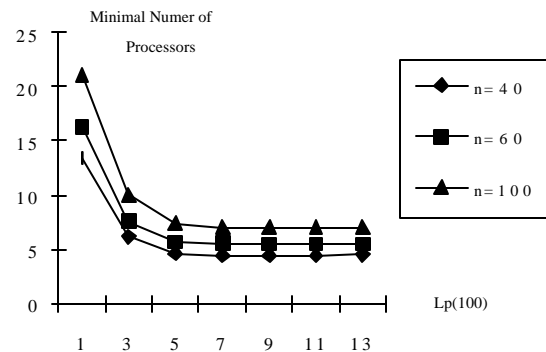


Figure 4 Relationship between minimal number of processors and schedule length (DL = 1400, CT [5, 200])

RTFTNO and RTFTRC are two static algorithms that schedule periodic real-time tasks with a common deadline. RTFTRC algorithm regards the reliability cost as the most important issue in scheduling, while RTFTNO does not. The overall performance of RTFTRC, in terms of reliability cost and PMD, is better than that of RTFTNO. Real-time fault-tolerant scheduling algorithm, RTFTRC, will apply to many real world systems including real-time database applications, real-time manufacturing, defense applications, and distributed multimedia.

Our future studies in this area include (1) Present static fault-tolerant algorithms that schedule real-time tasks with different deadlines. (2) Study dynamic scheduling algorithms with fault-tolerant for aperiodic real-time tasks. (3) Based on RTFTRC, study an efficient algorithm, which releases assumption 2 in this paper.

References

- [1] P. Berman and B. DasGupta. Improvements in Throughput Maximization for Real-Time Scheduling. *DIMACS, Technical Report, TR-99-52*, 1999.

- [2] G. L. Chang, H. Joosun, M. S. Yang et. al. Analysis of Cache-Related Preemption Delay in Fixed-Priority Preemptive Scheduling. *IEEE Trans. on Computers*, 47(6): 700-713, 1998.
- [3] Z. Chen, K. Maly, P. Mehrotra, V. K. Praveen and M. Zubair. An Architecture to Support Collaborative Distributed Heterogeneous Computing Applications. *Technical Report, TR97-26*, Department of Computer Science, University of Bristol, UK, 1997.
- [4] A. Egan, D. Kutz, D. Mikulin, R. Melhem and D. Mosse. Fault-Tolerant RT-Mach (FT-RT-Mach) and its Application to Real-Time Train Control. *Software Practice and Experience*, 29(3):1-17, 1999.
- [5] S. Lauzac, R. Melhem, and D. Mosse. Adding Fault-Tolerance to P-Fair Real-Time Scheduling. *Proceedings of Workshop on Embedded Fault-Tolerant Systems*, pp.34-37, 1998.
- [6] F. Liberato, S. Lauzac, R. Melhem and D. Mosse. Fault Tolerant Real-Time Global Scheduling on Multiprocessors. *Proceedings of Euromicro Workshop in Real-Time Systems*, 1999.
- [7] T.H. Lin and Shin K.G. Damage Assessment for Optimal Rollback Recovery. *IEEE Trans. on Computers*, 47(5): 603-613, 1998
- [8] H. Liu and M.E. Zarki. Adaptive Source Rate Control for Real-Time Wireless Video Transmission. *Mobile Networks and Application*, 3:49-60, 1998.
- [9] G. Manimaran and C. Siva Ram Murthy. An Efficient Dynamic Scheduling Algorithms for Multiprocessor Real-Time Systems. *IEEE Trans. On Parallel and Distributed Systems*, 9(3): 312-319, 1998.
- [10] P. Mejia Alvarez and D. Mosse. A Responsiveness Approach for Scheduling Fault Recovery in Real-Time Systems. *Proceedings of Fifth IEEE Real-Time Technology and Applications Symposium*, Canada, pp.1-10, June 1999.
- [11] Y. Oh and S. H. Son. Scheduling Real-Time Tasks for Dependability. *Journal of Operational Research Society*, 48(6):629-639, June 1997.
- [12] X. Qin, L. P. Pang, S. L. Li, and Z. F Han. Efficient Scheduling Algorithm with Fault-tolerance for Real-time Tasks in Distributed Systems. *Proc. 5th Int'l conf. for Young Computer Scientists*, Vol.2: 721-725, Aug. 1999.
- [13] S. M. Shatz, J. P. Wang, and M. Goto. Task Allocation for Maximizing Reliability of Distributed Computer Systems. *IEEE Trans. On Computers*, 1992, 41(9):1156-1168.
- [14] S. H. Son and C. Chaney. Supporting the Requirements for Multilevel Secure and Real-time Databases in Distributed Environments. *Database Security: Status and Prospects*, T. Y. Lin and S. Qian (eds.), Chapman and Hall Publishing, pp.73-91, 1998.
- [15] S. Srinivasan, and N.K. Jha. Safety and Reliability Driven Task Allocation in Distributed Systems. *IEEE Trans. On Parallel and Distributed Systems*, 10(3): 238-251, 1999.
- [16] L. Xu and J. Bruck. Deterministic Voting in distributed Systems Using Error-Correcting Codes. *IEEE Trans. on Parallel and Distributed Systems*, 9(8): 813-824, 1998.