ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.10, October 2009, pp.2628−2636
doi: 10.3724/SP.J.1001.2009.00577

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

容错多处理机中一种高效的实时调度算法*

王 健, 孙建伶+, 王新宇, 杨小虎, 王申康, 陈俊波

(浙江大学 计算机科学与技术学院,浙江 杭州 310027)

Efficient Scheduling Algorithm for Hard Real-Time Tasks in Primary-Backup Based Multiprocessor Systems

WANG Jian, SUN Jian-Ling+, WANG Xin-Yu, YANG Xiao-Hu, WANG Shen-Kang, CHEN Jun-Bo

(College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China)

+ Corresponding author: E-mail: sunjl@zju.edu.cn, http://www.cs.zju.edu.cn

Wang J, Sun JL, Wang XY, Yang XH, Wang SK, Chen JB. Efficient scheduling algorithm for hard real-time tasks in primary-backup based multiprocessor systems. *Journal of Software*, 2009,20(10):2628−2636. http://www.jos.org.cn/1000-9825/577.htm

Abstract: This paper has considered the problem of preemptively scheduling a set of independent periodic hard real-time tasks in primary-backup based multiprocessor systems. An efficient scheduling algorithm—Task Partition based Fault Tolerant Rate-Monotonic (TPFTRM) is proposed which extends RM algorithm to primary-backup based multiprocessor to provide fault tolerance. Compared with previous scheduling algorithms in this area, TPFTRM abandons active backup copies and only uses passive and overlapping backup copies to maximize the backup over-booking and deallocation, thus improves the scheduling performance. Moreover, TPFTRM proposes the task partitioning and processors grouping technique, which reduce the scheduling computation time and also make an easy way to understand and implement it. Extensive simulations experiments are also carried out based on task sets with different parameters. And the simulation result shows a remarkable saving of processors as well as scheduling computation time compared with previous algorithms, which proves the feasibility and effectiveness of the proposed TPFTRM algorithm.

Key words: hard real-time; primary-backup; fault-tolerant; scheduling algorithm; multiprocessor; periodic task

摘 要: 针对基于主副版本容错的多处理机中独立的、抢占性的硬实时任务,提出了一种高效的调度算法——TPFTRM(task partition based fault tolerant rate-monotonic)算法.该算法将单机实时 RM 算法扩展到容错多处理机上,并且调度过程中从不使用主动执行的任务副版本,而仅使用被动执行和主副重叠方式执行的任务副版本,从而最大限度地利用副版本重叠和分离技术提高了算法调度性能.此外,TPFTRM 根据任务负载不同将任务集合划分成两个不相交的子集进行分配;还根据处理机调度的任务版本不同,将处理机集合划分成 3 个不相交的子集进行调度,从而使 TPFTRM 调度算法便于理解、实现以及减少了调度所需要的运行时间.模拟实验对各种具有不同周期和任务负载的任务集合进行了调度测试.实验结果表明,TPFTRM 与目前所知同类算法相比,在调度相同参数的任务集合时不仅明显减少了调度所需要的处理机数目,还减少了调度所需要的运行时间,从而证实了 TPFTRM 算法的高效性.

关键词: 硬实时;主副版本;容错;调度算法;多处理机;周期任务

* Received 2008-07-11; Accepted 2009-01-14

中图法分类号: TP316 文献标识码: A

1 Introduction

In recent years, hard real-time systems have been used in many applications which have stringent timing constraints, such as defense systems, air-traffic control systems, telecommunications and launch vehicle control. The classic RM algorithm[1] is becoming an industry standard because of its simplicity and flexibility for preemptive periodic hard real-time scheduling in uniprocessor[2]. However, scheduling algorithms in current hard real-time systems require critical tasks be executed correctly and timely even in the presence of processor failures, thus have led to the choice of multiprocessors systems as a natural candidate for them[3]. Some algorithms such as RMFF and RMST[4,5] generalize RM to multiprocessor systems but none of them provides fault-tolerance. Therefore, many fault-tolerant scheduling algorithms[2,6−11] have been proposed to extend RM to multiprocessor and meanwhile ensure that hard real-time tasks meet their deadlines before as well as after the occurrence of a processor fault.

Most of these fault-tolerant scheduling algorithms are based on the Primary-Backup (PB) approach, which is one of the most common fault-tolerant approaches in multiprocessors systems. In PB approach, each task has primary and backup copies and the copies are scheduled on two different processors[12−14]. In general, PB approach provides a variety of schemes and they can be partitioned into three broad classes. In the first class, backup copies are called *passive* backup copies and when a primary copy fails, the passive backup copies are restarted on the backup processor. In the second class, backup copies are called *active* backup copies which run concurrently with the primary copies. In the third class, backup copies are called *overlapping* backup copies and their execution are started concurrently with the primary copies but terminated when corresponding primary copies are successfully completed.

The most important metric for measuring the performance of a fault-tolerant scheduling algorithm in multiprocessor is the used processor number. For a given task set, the less processor a fault-tolerant scheduling algorithm requires, the better scheduling performance it has. Another metric is the computation time required by allocating tasks to processors, although the task allocation happens only once in the scheduling (after tasks are assigned to processors, RM algorithm is used to schedule them), the less task allocation computation time a scheduling algorithm requires, the better scalability it has. However, according to these two metrics, none of the previous proposed algorithms[2,6−11] have good scheduling performance for all the cases when the task set has different upper bound for the task load. Therefore, in this paper, we present an efficient scheduling algorithm named Task Partition based Fault-Tolerant Rate-Monotonic (TPFTRM). TPFTRM abandons active backup copies and only uses the passive and overlapping backup copies to maximize the backup over-booking and deallocation, thus reduces the required processor number. Moreover, TPFTRM proposes the task partitioning and processors grouping technique, which reduce the scheduling computation time and also make an easy way to understand and implement it.

2 Related Work

Scheduling periodic hard real-time tasks on multiprocessor even without fault-tolerance consideration has been found to be NP-hard, hence several heuristic algorithms have been proposed. Dhall and Liu firstly proposed Rate-Monotonic First-Fit (RMFF) heuristic[4]. Burchard, *et al*. gave more refined Rate-Monotonic Small Tasks (RMST) and Rate-Monotonic General Tasks (RMGT) heuristics[5]. Since RMFF, RMST and RMGT algorithms do not provide fault-tolerance, Bertossi, *et al*. proposed Fault-Tolerant Rate-Monotonic First-Fit (FTRMFF) heuristic, which firstly extends RMFF algorithm by combining in the same schedule both active and passive backup copies, thus exploiting the advantages of PB fault-tolerant approach[2]. Later, the Efficient Fault-tolerant Rate-Monotonic Best-fit (ERMBF)

algorithm is proposed in Ref.[6] which uses the *Best-Fit* heuristic to improve scheduling performance. S-Priority Passive algorithm (S-PR-Pass), Active Resource Reclaiming (ARR) algorithm and Deferred Active Backup-Copy based Best-Fit (DABCBF) algorithms were also proposed in Refs.[7,8] which have better scheduling performance than FTRMFF and ERMBF. Finally, they were extended to heterogeneous distributed environments and with resource constrains in Refs.[9−11]. However, none of these algorithms[2,6−11] has good scheduling performance in all the cases when the task set has different upper bound for the task load. S-PR-Pass and DABCBF algorithms have better scheduling performance than ARR2 when tasks have light load while ARR2 algorithm has better scheduling performance than S-PR-Pass and DABCBF when tasks have heavy load. Moreover, ARR2 algorithm is based on the phasing delay technique which has a time offset restriction for the backup copies[7]. Therefore, in this paper, we propose TPFTRM algorithm, which does not have any time offset restrictions and also requires little computation time; and meanwhile TPFTRM has good scheduling performance as the same as S-PR-Pass when tasks have light load and also has the similar scheduling performance as ARR2 when tasks have heavy load.

3 Problem Formulation

As for the fault-tolerance model, failure characteristics of the hardware are the following: (1) processors fail in a fail-stop manner, which means a processor is either operational or cease functioning; (2) hardware provides fault isolation mechanism, that is a faulty processor can not cause incorrect behaviors in a non-faulty processor; (3) the failure of a processor is detected by the remaining ones within the closest completion time of a task scheduled on the faulty processor[2]; (4) a second processor does not fail before the system recovers from the first failure.

A periodic task t_i is characterized by a pair (C_i,T_i), where C_i is the *computation time* and T_i is the *request* (or *arrival*) *period*. Each request of any task must be completely executed before the next request of the same task and the first request of t_i occurs at time 0. W_i denotes the *worst-case response time* of t_i. Periodic tasks $t_i,\ldots,t_n$ are *independent* and *preemptive*. The ratio U_i is the *load* (or *utilization*) of t_i, which equals C_i/T_i and U is *the load of the task set* $t_1,\ldots,t_n$ hence $U=\sum_{1\le i\le n}U_i$.

In order to achieve fault tolerance, each task has two copies p_i and b_i, called primary and backup copies which are located on different processors. For the sake of simplicity, it is assumed that each backup copy has the same computation time and period parameters as its primary copy. The backup copy has three statues: active, passive and overlapping. O_i denotes the length of the computation overlap between p_i and b_i. Therefore, the status of backup copy is determined by the following:

$$status(b_i)=\begin{cases} active, & O_i=C_i \\ passive, & O_i=0 \\ overlapping, & O_i>0 \text{ and } O_i<C_i \end{cases} \quad (1)$$

All the task copies assigned to the same processor are scheduled by the RM algorithm. Given n periodic independent tasks $\{t_i,\ldots,t_n\}$, the fault-tolerant scheduling problem considered in this paper is to find a minimum number of processors to ensure each task request can be executed by the end of its period before as well as after the occurrence of a fault.

4 TPFTRM Scheduling Algorithm

When we allocate task primary and backup copies to processors, there are three questions we have to consider. First, how should the algorithm decide the status for the backup copy? Second, what order should task copies follow to be assigned to processors? Third, what criteria should be taken for checking the schedulability of task copies?

The following sections will address these questions one by one.

4.1 Task partitioning and status of backup copy

In TPFTRM, primary copies are firstly partitioned into two groups according to U_i: Big task group B and small task group S. Let $g(p_i)$ denote the group which p_i belongs to, thus

$$g(p_i)=\begin{cases}S, & U_i \le 0.5\\ B, & U_i > 0.5\end{cases} \tag{2}$$

The reason to partition primary copies according to U_i is that passive backup copies can be used for all the tasks whose U_i≤0.5 while they are not able to be used for the tasks whose U_i>0.5. As shown in Fig.1, there is not any length of the computation overlap between p_i and b_i when U_i≤0.5 while in Fig.2, the length of the computation overlap is O_i=2·C_i−T_i when U_i>0.5.

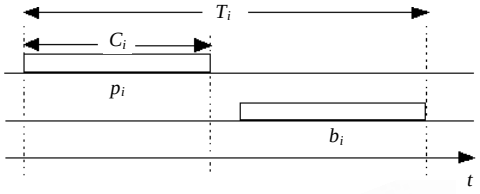

Fig.1 Passive backup is used

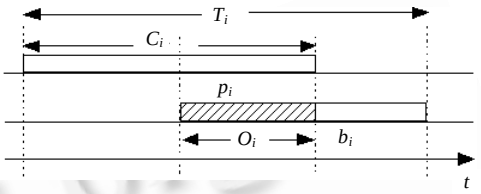

Fig.2 Overlapping backup is used

Therefore, tasks in different groups have different status of the backup copy. In TPFTRM, the status of backup copy is determined by the following:

$$status(b_i)=\begin{cases}passive, & g(p_i)=S\\ overlapping, & g(p_i)=B\end{cases} \tag{3}$$

Unlike previous algorithms[2,6–11], TPFTRM algorithm does not use active backup copies. The reason is that active backup copy introduces more redundant computation time which has a big impact on the scheduling performance. In contrast, passive and overlapping backup copies do not need execute if their primaries execute successfully hence reduce the redundant computation time. Moreover, passive or overlapping copies whose primary copies are assigned to different processors can be scheduled on the same processor so as to share the same time interval, which called the backup overbooking technique[15], has great advantages of saving processors.

4.2 Processors grouping and assignment of tasks to processors

Unlike previous algorithms[2,6–11], TPFTRM divides processors into three groups to avoid mixing backup copies and primary copies coming from different task groups on the same processor. Therefore, in TPFTRM, processors in the first group G_1 are only for primary copies coming from S; processors in the second group G_2 are only for primary copies coming from B; and processors in the third group G_3 are only for passive or overlapping backup copies coming from S and B.

The next problem we consider is what order the algorithm should follow to allocate primary and backup copies to processors. One approach is called RMFF, which assigns task copies by decreasing RM priorities[2,8–10]. This approach simplifies the algorithm since only lower priority tasks are assigned later to the same processor hence time intervals for already assigned tasks will remain unchanged. Another approach is called RMST, which assigns task copies by the increasing value s_i, where $s_i=\log_2 T_i-\lfloor \log_2 T_i \rfloor$[5,7]. This approach allocates task copies whose periods are equal to or multiple on the same processor, thus producing a more compact schedule and requiring fewer processors when tasks have light load.

TPFTRM uses RMST approach to allocate task copies. The first reason is that it has better performance than

RMFF in many cases. The second reason is that in TPFTRM, the backup status has already been decided beforehand and processors are also partitioned; hence using RMST won't require much computation time than using RMFF. For each task group, when we assign a task, we assign its primary copy before assigning its backup copy.

Theorem 1. In TPFTRM, if the overlapping part of an overlapping backup copy b_i is able to be finished before W_i on the processor P_j in the absence of failure, then it is also schedulable on P_j in the presence of a failure occurs on P_f ($P_f \neq P_j$).

Proof: Assume b_i's corresponding primary copy p_i is assigned to the processor P_k. It is worth noting that p_i is the only one task copy located on P_k. (to prove this, assuming there is another task copy p_m on P_k besides p_i, then p_i and p_m are both from task group B and not able to schedule on P_k because of $U_i+U_m>1$). Firstly, assuming a fault occurs on P_k, then P_j can reclaim all the time slots reserved for other passive or overlapping backup copies to recovery p_i. So if the overlapping part of b_i is able to be finished before W_i, then the left unfinished part of b_i is also able to be finished before T_i. Secondly, assuming a fault occurs on P_f ($P_f \neq P_j$ and $P_f \neq P_k$), then time slots reserved for b_i can be reclaimed by P_j to execute backup copies which have faulty primary copies.

In previous algorithms[2,6–11] except S-PR-Pass, to find a processor a task copy can be assigned to, both the situations in which no processor fails and any processor fails have to be considered when checking the schedulability of primary copies and overlapping backup copies. However, in TPFTRM, both the primary copies and overlapping backup copies have only one condition to be checked, therefore, there are only three assignments and schedulability testing cases in TPFTRM hence greatly reduces the computation time:

1) To assign a *primary* copy p_i to a processor P_j, only one condition has to be checked.
   - p_i must be schedulable together with all the primary copies already assigned to P_j.
2) To assign a *passive backup* copy b_i to a processor P_j, assuming its corresponding primary copy p_i is already assigned to processor P_f ($P_f \neq P_j$), then only one condition has to be checked.
   - p_i must be schedulable together with all the passive backup copies assigned to P_j such that their corresponding primary copies are all assigned to the same processor P_f.
3) According to theorem 1, to assign an *overlapping backup* copy b_i to a processor P_j, only one condition has to be checked.
   - The overlapping part of b_i must be schedulable together with the overlapping part of the other overlapping backup copies already assigned to P_j.

If a task copy is not able to be assigned to any of existing processors, then the task copy is assigned to a new processor.

4.3 Schedulability condition in TPFTRM

The classical *Completion Time Test* (CTT)[16] gives the following necessary and sufficient schedulability criterion for independent periodic tasks:

Theorem 2. Let periodic tasks $\tau_1, \tau_2, \ldots, \tau_n$ be given in priority order and scheduled by a fixed-priority algorithm. All periodic requests of τ_i will meet deadlines if and only if

$$\min_{0<t\le T_i}\left\{\sum_{1\le k\le i} C_k \cdot \lceil t/T_k \rceil / t\right\} \le 1.$$

The entire set of tasks $\tau_1, \tau_2, \ldots, \tau_n$ is schedulable if and only if

$$\max_{1\le i\le n} \min_{0<t\le T_i}\left\{\sum_{1\le k\le i} C_k \cdot \lceil t/T_k \rceil / t\right\} \le 1.$$

In this section, we extend the CTT for checking the schedulability of task copies in TPFTRM algorithm. Let τ_i be a task copy (either the primary copy p_i or the backup copy b_i) to be assigned to a processor P_j. *primary*(P_j),

passive(P_j) and *overlapping*(P_j) represent primary copies, passive backup copies and overlapping backup copies assigned to processor P_j. *recover*(P_j,P_f) represents all the passive backup copies assigned to processor P_j with their corresponding primary copies assigned to P_f.

Theorem 3. (1) Let primary copies $p_1,p_2,\ldots,p_n$ be given in priority order and scheduled by TPFTRM. The entire set of primary copies $p_1,p_2,\ldots,p_n$ is schedulable on P_j ($P_j\in G_1$) if and only if:

$$\max_{p_i\in primary(P_j)}\min_{0<t\le T_i-C_i}\left\{\sum_{1\le k\le i}C_k\cdot\lceil t/T_k\rceil/t\right\}\le 1.$$

(2) A primary copy p_i is schedulable on P_j ($P_j\in G_2$) if and only if P_j is the new processor.

Proof: In TPFTRM, processors in G_1 or G_2 only have primary copies allocated on them. Firstly, all the primary copies in G_1 must be executed completely no later than T_i–C_i to ensure that their passive backup copies have enough time to recovery in the case a fault occurs. Secondly, each processor in G_2 only has one primary copy allocated on it. If two primary copies were assigned to P_j ($P_j\in G_2$), then the requests of one of them would not meet deadlines because no scheduling algorithm exits to schedule the task set with U>1.

Theorem 4. Let passive backup copies $b_1,b_2,\ldots,b_n$ be given in priority order and scheduled by TPFTRM. Assuming b_i's corresponding primary copy is already assigned to processor P_f ($P_f\neq P_j$), then the entire set of passive backup copies $b_1,b_2,\ldots,b_n$ is schedulable on P_j ($P_j\in G_3$) if and only if

$$\max_{b_i\in recover(P_j,P_f)}\min_{0<t\le T_i-W_i}\left\{\sum_{1\le k\le i}C_k\cdot\lceil t/T_k\rceil/t\right\}\le 1.$$

Proof: Unlike primary copy and active backup copy, the *critical instant* for a passive copy occurs not only when b_i and all the task copies with higher priority than b_i are released simultaneously, but also when it is the first request of b_i which must be executed in the limited recovery period, which is shorter than the task period T_i and is at least T_i–W_i time units long. Therefore, if b_i is schedulable on P_j starting from the *critical instant*, which represents the worst case, then it is also schedulable at any moment when its corresponding primary copy p_i fails.

Theorem 5. Let overlapping backup copies $b_1,b_2,\ldots,b_n$ be given in priority order and scheduled by TPFTRM. Assuming b_i's corresponding primary copy is already assigned to processor P_f ($P_f\neq P_j$), then the entire set of overlapping backup copies $b_1,b_2,\ldots,b_n$ is schedulable on P_j ($P_j\in G_3$) if and only if

$$\max_{b_i\in overlapping(P_j)}\min_{0<t\le C_i}\left\{\sum_{1\le k\le i}O_k\cdot\lceil t/T_k\rceil/t\right\}\le 1.$$

Proof: According to theorem 1, we only need to test the schedulability of b_i in absence of the failure. P_j ($P_j\in G_3$) has mixed passive and overlapping backup copies. However, the workload of passive backup copies can be ignored in absence of the failure because passive backup copies are only executed when a fault occurs. For b_i, its overlapping part O_i equals $2\cdot C_i$–T_i and will execute T_i–C_i unfinished part when p_i fails, therefore, O_i must be finished before C_i, otherwise, when a fault occurs, b_i will not meet its deadline T_i.

4.4 TPFTRM algorithm

Let $P(\tau_i)$ be the processor to which the task copy τ_i is assigned. The algorithm is described below:

1. Partitioning task copies into big task group B and small task group S according to Formula (2).
2. Set the status for backup copies according to Formula (3).
3. m represents the number of used processors in G_1, v represents the number of used processors in G_2, h represents the number of used processors in G_3. Initialize m, v and h to 0.
4. If S is empty, go to step 6. Otherwise, let primary copies $p_1,p_2,\ldots,p_n$ ($p_i\in S$) be indexed by the increasing value s_i, where $s_i=\log_2 T_i-\lfloor\log_2 T_i\rfloor$; set m to 1 and allocate P_m to G_1; set h to 1 and allocate p_h to G_3.
5. Repeat the following steps for i=1,2,…,n:

1) Assign the primary copy p_i to the first processor P_j ($P_j \in G_1$) on which the task set $p_i \cup primary(P_j)$ is schedulable by means of the formula given in Theorem 3; if no such a processor exists, set m to m+1 and assign p_i to P_m, setting $P(p_i)=P_m$ and allocate P_m to G_1;

2) Assign passive backup copy b_i to the first processor $P_j \neq P(p_i)$ ($P_j \in G_3$) on which the task set $b_i \cup recover(P_j, P(p_i))$ is schedulable by means of the formula given in Theorem 4; if no such processor exists, set h to h+1 and assign b_i to p_h, setting $P(b_i)=p_h$ and allocate p_h to G_3;

6. If B is not empty, let primary copies $p_1,p_2,\ldots,p_k$ ($p_i \in B$) be indexed by the increasing value s_i, where $s_i=\log_2 T_i - \lfloor \log_2 T_i \rfloor$; repeat the following steps for i=1,2,…,k:

1) Set v to v+1 and assign p_i to P_v, setting $P(p_i)=P_v$ and allocate P_v to G_2;

2) Assign overlapping backup copy b_i to the first processor $P_j \neq P(p_i)$ ($P_j \in G_3$) on which the task set $b_i \cup overlapping(P_j)$ is schedulable by means of the formula given in Theorem 5. If no such processor exists, set h to h+1 and assign b_i to p_h, setting $P(b_i)=p_h$ and allocate p_h to G_3;

7. Return the number $m+h+v$ of processors used and the task assignment found.

5 Performance Evaluation

This section presents simulation results for the comparison of our algorithm and previous research. We use the simulated method and task set as the same as that in Refs.[2,6−8]. Task sets with $100 \le n \le 1000$ tasks are generated. The parameters of each task τ_i are chosen as follows. The period T_i is an integer uniformly chosen from [1,500], while the computation time C_i is an integer uniformly distributed in $[1, \alpha T_i]$, where $\alpha=\max(U_i)$ is the upper bound for the task load. In this simulation, we use three values for α($0<\alpha<1$), which is 0.2, 0.5 and 0.8. For the chosen n and α, the experiment is repeated 10 times and the average result is computed.

Let M be the number of processors used for scheduling both primary and backup copies. M/U($M/U>1$) is the ratio of processor number to task set load. Figure 3 shows the ratio M/U for the experiments executed by FTRMFF, ARR2, S-PR-Pass and TPFTRM when α=0.2 while Fig.4 shows the same experiments when α=0.5. As shown in Figs.3 and 4, TPFTRM and S-PR-Pass have the smallest M/U hence have the best performance, and as task number increases, their advantages of saving processors are more obvious. It is worth noting that when $\alpha \le 0.5$, both TPFTRM and S-PR-Pass only use passive backup copies hence have the same performance.

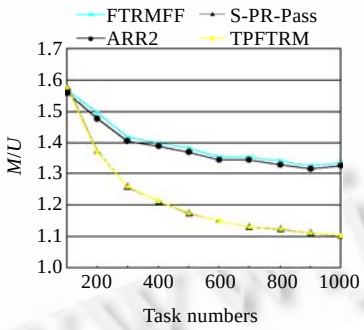

Fig.3 M/U ratios comparison when α=0.2

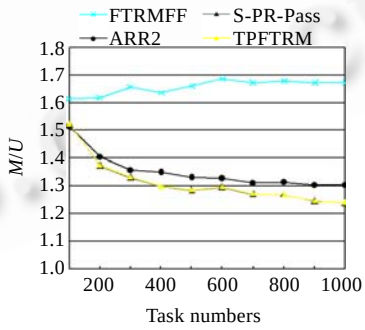

Fig.4 M/U ratios comparison when α=0.5

Figure 5 reports the comparison results when α=0.8. As shown in Fig.5, ARR2 has the smallest M/U hence have the best performance. However, compared with ARR2, TPFTRM have the similar small M/U value and in the meantime, TPFTRM does not have any time offset restrictions for backup copies. When α>0.5, TPFTRM uses overlapping backup copies as well as passive backup copies hence have much better performance than S-PR-Pass which only uses passive backup copies. It is observed that TPFTRM always has good scheduling performance in all the cases

when the task set has different upper bound for the task load.

All the algorithms were written in Java and ran on a Windows 2000 PC with 1GB memory and Intel Pentium 4 2.26GHz. Figure 6 shows the total running time required by each algorithm in this simulation. As shown in Fig.6, TPFTRM and SP-PR-Pass only take 5% of the running time required by FTRMFF and ARR2. We also get the similar simulation results on other different PCs hence the conclusions are not dependent on a particular PC platform.

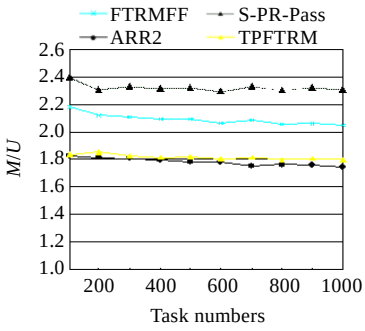

Fig.5 M/U ratios comparison when α=0.8

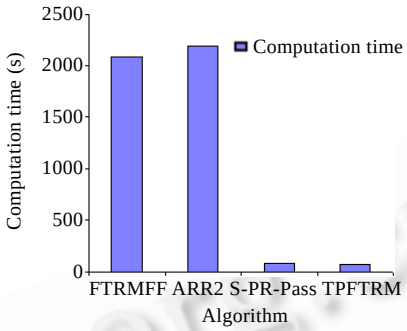

Fig.6 Computation time comparison

6 Conclusion

In this paper, we propose TPFTRM algorithm which extends Rate-Monotonic algorithm to tolerate failures based on the primary-backup approach. TPFTRM algorithm abandons active backup copies and only uses passive and overlapping backup copies to maximize the backup over-booking and deallocation, thus improves the scheduling performance. Moreover, another contribution of TPFTRM algorithm is that it partitions the task set into two groups according to their load and partitions processors into three groups to avoid mixing backup copies and primary copies coming from different task groups, which reduce the scheduling computation time and make an easy way to understand and implement it. Simulation results also show a remarkable saving of processors as well as scheduling computation time compared with previous algorithms. However, this paper has assumed that tasks are all independent. Further research includes scheduling a set of tasks subject to precedence constraints or resource requirements[11]. Also, further research includes looking for some new heuristics strategies for the task assignment and extending to heterogeneous distributed systems.

References:

[1] Liu CL, Layland JW. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM, 1973,20(1):46–61.

[2] Bertossi AA, Mancini LV, Rossini F. Fault-Tolerant rate-monotonic first-fit scheduling in hard-real-time systems. IEEE Trans. on Parallel and Distributed Systems, 1999,10(9):934–945.

[3] Manimaran G, Murthy CSR. An efficient dynamic scheduling algorithm for multiprocessor real-time systems. IEEE Trans. on Parallel and Distributed Systems, 1998,9(3):312–319.

[4] Dhall SK, Liu CL. On a real-time scheduling problem. Operations Research, 1978,26(1):127–140.

[5] Burchard A, Liebeherr J, Oh YF, Son SH. New strategies for assigning real-time tasks to multiprocessor systems. IEEE Trans. on Computers, 1995,44(12):1429–1442.

[6] Yang FM, Luo W, Pang LP. An efficient real-time fault-tolerant scheduling algorithm based on multiprocessor systems. Wuhan University Journal of Natural Sciences, 2007,12(1):113–116.

[7] Bertossi AA, Mancini LV, Menapace A. Scheduling hard-real-time tasks with backup phasing delay. In: Alba E, Turner SJ, Roberts D, Taylor SJE, eds. Proc. of the 10th IEEE Int'l Symp. on Distributed Simulation and Real-Time Applications. Los Alamitos: IEEE Computer Society, 2006. 107−116.

[8] Luo W, Yang FM, Pang LP, Li J. A real-time fault-tolerant scheduling algorithm for distributed systems based on deferred active backup-copy. Journal of Computer Research and Development, 2007,44(3):521−528 (in Chinese with English abstract).

[9] Luo W, Yang FM, Pang LP, Tu G. A real-time fault-tolerant scheduling algorithm of periodic tasks in heterogeneous distributed systems. Chinese Journal of Computers, 2007,20(10):1740−1749 (in Chinese with English abstract).

[10] Qin X, Jiang H, Swanson DR. An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems. In: Abdelrahman TS, ed. Proc. of the Int'l Conf. on Parallel Processing. Los Alamitos: IEEE Computer Society, 2002. 360−368.

[11] Yang CH, Deconink G, Gui WH. Fault-Tolerant scheduling for real-time embedded control systems. Journal of Computer Science & Technology, 2004,19(2):191−202.

[12] Qin X, Han ZF, Pang LP, Li SL. Design and performance analysis of a hybrid real-time scheduling algorithm with fault-tolerance. Journal of Software, 2000,11(5):686−693 (in Chinese with English abstract). http://www.jos.org.cn/ch/reader/view_abstract.aspx?flag=1&file_no=20000516&journal_id=jos

[13] Qin X, Pang LP, Han ZF, Li SL. Algorithms of fault-tolerant scheduling in distributed real-time systems. Chinese Journal of Computers, 2000,23(10):1056−1063 (in Chinese with English abstract).

[14] Yang CH, Gui WH, Ji L. A fault-tolerant scheduling algorithm of hybrid real-time tasks based on multiprocessors. Chinese Journal of Computers, 2003,26(11):1479−1486 (in Chinese with English abstract).

[15] Ghosh S, Melhem R, Mosse D. Fault-Tolerance through scheduling of aperiodic tasks in hard-real-time systems. IEEE Trans. on Parallel and Distributed Systems, 1997,8(3):272−284.

[16] Joseph M, Pandya P. Finding response times in a real-time system. The Computer Journal, 1986,29(5):390−395.

附中文参考文献:

[8] 罗威,阳富民,庞丽萍,李俊.基于延迟主动副版本的分布式实时容错调度算法.计算机研究与发展,2007,44(3):521−528.

[9] 罗威,阳富民,庞丽萍,涂刚.异构分布式系统中实时周期任务的容错调度算法.计算机学报,2007,20(10):1740−1749.

[12] 秦啸,韩宗芬,庞丽萍,李胜利.混合型实时容错调度算法的设计和性能分析.软件学报,2000,11(5):686−693. http://www.jos.org.cn/ch/reader/view_abstract.aspx?flag=1&file_no=20000516&journal_id=jos

[13] 秦啸,庞丽萍,韩宗芬,李胜利.分布式实时系统的容错调度算法.计算机学报,2000,23(10):1056−1063.

[14] 阳春华,桂卫华,计莉.基于多处理机的混合实时任务容错调度.计算机学报,2003,26(11):1479−1486.

WANG Jian was born in 1982. He is a Ph.D. candidate at the Zhejiang University and a CCF student member. His current research areas are real-time scheduling and fault-tolerant computing.

YANG Xiao-Hu was born in 1966. He is a professor at the Zhejiang University. His current research areas are software engineering and software technology financial services.

SUN Jian-Ling was born in 1964. He is a professor at the Zhejiang University and a CCF senior member. His current research areas are real-time systems and database systems.

WANG Shen-Kang was born in 1945. He is a professor at the Zhejiang University. His current research areas are distributed computing.

WANG Xin-Yu was born in 1979. He is a lecturer at the Zhejiang University. His current research areas are software engineering.

CHEN Jun-Bo was born in 1979. He is a Ph.D. candidate at the Zhejiang University. His current research areas are distributed computing and data mining.