Fault Tolerant Global Scheduling for Multiprocessor Hard Real Time Systems

Hao Peng, Fan Yang
School of Computer and Information, Hefei University of Technology, Hefei, 230009, China

Keywords: multiprocessor; hard real-time systems; primary-backup; fault tolerant; global scheduling

Abstract. In hard real-time systems, faults happen every now and then. With respect to the criticality of this kind of system, the capability of fault tolerance is necessary. A fault tolerant scheduling algorithm is capable of guaranteeing timing requirements of tasks even when faults occur. A primary-backup based fault-tolerant global scheduling algorithm RRFTGS (resource reclaim fault tolerant global scheduling) is proposed in this paper. RRFTGS pushes the execution of backup as late as possible and reclaims the resource distributed to backups when it is no longer needed. By this method the additional cost for achieving fault tolerance is significantly reduced. Simulation result shows that, comparing to the scheduling algorithm which only use passive backups, RRFTGS needs much less processors to tolerate a single fault. RRFTGS also could be adopted when high utilization task exists.

Introduction

Due to the rapid increase of performance demanding, multiprocessor chips have been wildly adopted in hard real time systems, which brings about new challenges to system design. In hard real time systems, each task has strict timing constrains. A single violation to these timing constrains would induce system failure. The timing constrains can be guaranteed by one of the hard real time scheduling algorithms, for instance, rate monotonic (RM), earliest deadline first (EDF), etc. However in the real world these systems encounter faults occasionally because of environmental interferences or design flaws. Therefore fault tolerance is crucial in hard real time scheduling.

Redundancy is the major approach of fault tolerance. There are two different kinds: hardware redundancy and time redundancy[1]. The former, for instance, includes spare hardware, triple modular redundancy and hybrid dual duplex redundancy[2], etc. comes with considerably cost. While in time redundancy software is executed several times to tolerant faults. The cost of time redundancy is relatively much less.

In hard real time systems, faults can be classified as permanent or transient. Permanent faults are usually hardware failure, which do not disappear with time. Redundant hardware is the only way to tolerant permanent faults. Transient faults which are highly related with the environment only exist for a very short period of time[3]. In this paper, we focus on tolerating transient faults by time redundancy.

Related works

Most fault-tolerant scheduling algorithms are based on primary-backup approach. Each task has a primary copy and one or several backup copies. It is considered as one successful execution of a task when any one of its primary and backup copies responses before its deadline[4]. In this paper, we adopt this approach for fault tolerance.

Hard real time scheduling algorithms can be classified into partitioned fault-tolerant scheduling or global fault-tolerant scheduling. In the past decade, partitioned fault-tolerant scheduling is wildly studied[5-8]. Comparing with partitioned fault-tolerant scheduling, there are much less studies for the global approach. Berten et al.[9] built a schedulability test for global fault-tolerant scheduling with a probabilistic approach. This utilization-based test has low complexity but low performance. Pathan and Goossens[10] proposed a global fault-tolerant scheduling algorithm called FTGS. FTGS only

considered passive backup copy so that it is infeasible when high utilization task exists. Samala et al.[11] adopted hybrid genetic algorithm to online schedule tasks dynamically. This algorithm has massive online computation so it is not practical.

In this paper we propose a global fault-tolerant scheduling algorithm which is called resource-reclaim fault-tolerant global scheduling (RRFTGS). RRFTGS uses both passive and active backups and determines which one should be used by itself. By pushing backups as late as possible and reclaiming the resources not used by backups RRFTGS reduces the cost for fault tolerance. A response time analysis[12] based schedulability test is established. This test is more complex than the utilization based approach but has better performance. The online computation of RRFTGS is low so it is suitable for hard real time systems.

We consider one fault model in this paper, which means before recovery from a fault no fault would emerge. This assumption is wildly used in the research area of fault-tolerant scheduling[5-7,11]. We also assume that a fault processor can recover functioning immediately. This assumption conforms to the characteristic of transient fault[3,13]. If the period of a fault is not negligible or permanent, a spare processor is necessary for maintaining fault tolerance.

System model and overview of RRFTGS

The hard real time system studied in our work is composed of a sporadic task set of n tasks. The hardware platform has m identical processors. Task τ_i ($i=1,2,\cdots,n$) has one primary copy $P_i\{T_i, D_i, C_i\}$ and one backup copy $B_i\{T_i, D_i, E_i, O_i\}$. We use the subscript to indicate the priority of a task. A lower subscript represents a higher priority. We also determine that a primary copy has higher priority than corresponding backup copy. T_i is the minimum interval between two successive release. D_i is the relative deadline. C_i and E_i are worst case execution time of primary copy and backup copy respectively. O_i is the time offset between a backup's release and activation time. In the rest of this paper we use r_i to represent the release time of a job. $R_i^{nofault}$, $R_i^{selffault}$ and $R_i^{highfault}$ stand for the response time of τ_i under the condition of (1) no-fault (there is no fault in the systems), (2) self-fault (the task which is being tested encounters a fault) and (3) high-fault (a high priority task encounters a fault) respectively. We also use $E_i^{partial}$ to represent the worst case overlapping time between a primary copy and its backup.

The primary and backup copy release a job respectively at the same time (time-triggered or event-triggered). The job released by primary copy is ready immediately while the backup job hangs for O_i time. The scheduler selects jobs by priority order to run on right functioning processors. If no fault happens, the backup is cancelled when corresponding primary responses correctly despite of it is already running or not. The primary is cancelled when encounters a fault and its backup will be put into ready statues. Other backups are cancelled at the same time. After the backup responses all tasks resumes to normal behavior at the next release.

RRFTGS includes online and offline portion. The online portion is the scheduling procedure described above. The offline portion includes schedulability test and calculation of O_i, which are detailed in next two sections. The schedulability test is based on response time analysis approach. By calculating the interference caused by high priority tasks, we can get sufficient conditions.

Schedulability test

During the design period of hard real time system, the timing constrains have to be guaranteed. In this section we set up the schedulability test for RRFTGS. Two parameters O_i and $E_i^{partial}$ are used in schedulability test which are calculated after the task is determined schedulable. How to get these two parameters is described in section 4. In this section we assume these are known already. We assume that τ_k is the task being tested and τ_f is the task encountering a fault. In RRRFTGS three conditions have to be considered which are (1) no-fault (there is no fault in the systems), (2)

self-fault (the task which is being tested encounters fault) and (3) high-fault (a high priority task encounters a fault). A task should pass the schedulability test of all three conditions before it is conformed schedulable.

1.1 The upper bound of workload of a task without faults

First we consider a task τ_i which does not have faults and give the equations for calculating the upper bound of its workload W in any time interval of length L. Fig. 1 and Fig. 2 show the worst case scenario of P_i and B_i's workload when there is a carry-in (CI) job or non-carry-in (NC) job respectively. R_i is the upper bound of response time of primary copy, where $R_i = \max\left(R_i^{nofault}, R_i^{highfault}\right)$.

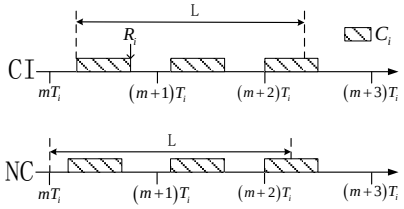

Fig. 1 the upper bound of workload of a primary copy

The upper bound of workload of P_i in an interval of length L can be calculated by equation (1) and (2).

$$W_{P_i}^{CI}(L) = \left\lfloor \frac{L + R_i - C_i}{T_i} \right\rfloor \times C_i + \min\left(L + R_i - C_i - \left\lfloor \frac{L + R_i - C_i}{T_i} \right\rfloor \times T_i, C_i \right), \quad (1)$$

$$W_{P_i}^{NC}(L) = \left\lfloor \frac{L}{T_i} \right\rfloor \times C_i + \min\left(L - \left\lfloor \frac{L}{T_i} \right\rfloor \times T_i, C_i \right). \quad (2)$$

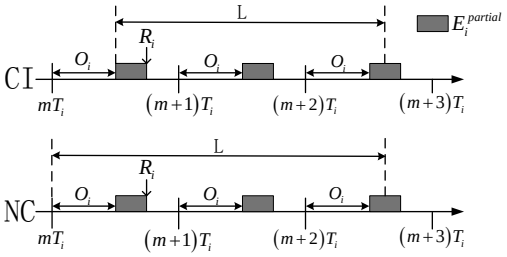

Fig. 2 the upper bound of workload of a backup copy

The upper bound of workload of B_i in an interval of length L can be calculated by equation (3) and (4).

$$W_{B_i}^{CI}(L) = \left\lfloor \frac{L + O_i}{T_i} \right\rfloor \times E_i^{partial} + \min\left(L - \left\lfloor \frac{L + O_i}{T_i} \right\rfloor \times T_i, E_i^{partial} \right), \quad (3)$$

$$W_{B_i}^{NC}(L) = \left\lfloor \frac{L}{T_i} \right\rfloor \times E_i^{partial} + \min\left(L - \left\lfloor \frac{L}{T_i} \right\rfloor \times T_i - O_i, E_i^{partial} \right). \quad (4)$$

In the rest of this section, we will discuss the upper bound of total interference of high priority tasks during one execution of task τ_k and give the schedulability test for all 3 conditions.

1.2 No-fault

If there is no fault in the system, B_k is cancelled immediately when P_k responses. So the response time of P_k is the response time of τ_k. We assume one job of P_k gets most interference so that it responses as late as possible. The problem window is the time interval from the release time of that job r_k to the response time $R_k^{nofault}$. We assume $r_k = 0$, then the length of the problem window is $R_k^{nofault}$. Letting $L = R_k^{nofault}$ in equation (1)～(4) we can get the upper bound of workload

of P_i and B_i which are $W_{P_i}^{CI}\left(R_k^{nofault}\right)$, $W_{P_i}^{NC}\left(R_k^{nofault}\right)$, $W_{B_i}^{CI}\left(R_k^{nofault}\right)$ and $W_{B_i}^{NC}\left(R_k^{nofault}\right)$. The upper bound of interference $I_{P_i}^{CI}\left(R_k^{nofault}\right)$, $I_{P_i}^{NC}\left(R_k^{nofault}\right)$, $I_{B_i}^{CI}\left(R_k^{nofault}\right)$, $I_{B_i}^{NC}\left(R_k^{nofault}\right)$ and their difference $I_{P_i}^{DIFF}\left(R_k^{nofault}\right)$, $I_{B_i}^{DIFF}\left(R_k^{nofault}\right)$ can be calculated by equation (5) and (6).

$$I_{ver}^{cond}\left(R_k^{nofault}\right)=\min\left(W_{ver}^{cond}\left(R_k^{nofault}\right),R_k^{nofault}-C_k+1\right), \tag{5}$$

$$I_{ver}^{DIFF}\left(R_k^{nofault}\right)=I_{ver}^{CI}\left(R_k^{nofault}\right)-I_{ver}^{NC}\left(R_k^{nofault}\right), \tag{6}$$

where $cond\in\{CI,NC\}$, $ver\in\{P_i,B_i\}$.

The sum of interference on P_k caused by high priority tasks in problem window is calculated by equation (7).

$$I_{k-sum}^{nofault}\left(R_k^{nofault}\right)=\sum_{i=1}^{k-1}\left(I_{P_i}^{NC}\left(R_k^{nofault}\right)+I_{B_i}^{NC}\left(R_k^{nofault}\right)\right)+sum_{\max\_m-1}\left(I_{P_i\cup B_i}^{DIFF}\left(R_k^{nofault}\right)\right), \tag{7}$$

where $sum_{\max\_m-1}\left(I_{P_i\cup B_i}^{DIFF}\left(R_k^{nofault}\right)\right)$ is the sum of largest $m-1$ (m is the amount of processors) interference difference.

Starting from $R_k^{nofault}(1)=C_k$, we calculate equation (8) iteratively until $R_k^{nofault}(n+1)=R_k^{nofault}(n)$, where $R_k^{nofault}(n)$ is the response time of P_k under the condition of no-fault. If $R_k^{nofault}\le D_k$, τ_k is schedulable.

$$R_k^{nofault}(n+1)=C_k+\left\lfloor\frac{I_{k-sum}^{nofault}\left(R_k^{nofault}(n)\right)}{m}\right\rfloor \tag{8}$$

1.3 Self-fault

When τ_k encounters a fault, i.e. its primary copy P_k encounters a fault, the backup copy B_k will replace P_k. So under this condition we are supposed to test the schedulability of B_k. When a fault occurs, all backups except for B_k are cancelled and B_k is activated right away. So if we push that instant as late as possible other backups will produce more interference and there is less time left for B_k. On the other hand the fault of P_k can occur no later than its no-fault response time $R_k^{nofault}$. Consequently we assume the fault occurs at $R_k^{nofault}$. During the interval $\left[r_k,R_k^{selffault}\right)$, high priority primaries produce interference constantly. While high priority backups only produce interference during $\left[r_k,R_k^{nofault}\right)$. Consider a special condition that the backup responses before its primary. This is possible if the backup has low accuracy and executes for a very short period of time. In this case the high priority primary and backup all produce interference during $\left[r_k,R_k^{selffault}\right)$. So letting $L=R_k^{selffault}$ in equation (1) and (2) we can get the upper bound of workload of a high priority primary copy P_i, which are $W_{P_i}^{CI}\left(R_k^{selffault}\right)$ and $W_{P_i}^{NC}\left(R_k^{selffault}\right)$. The upper bound of interference of P_i $I_{P_i}^{CI}\left(R_k^{selffault}\right)$ and $I_{P_i}^{NC}\left(R_k^{selffault}\right)$ and their difference $I_{P_i}^{DIFF}\left(R_k^{selffault}\right)$ are calculated by equation (9) and (10).

$$I_{P_i}^{cond}\left(R_k^{selffault}\right)=\min\left(W_{P_i}^{cond}\left(R_k^{selffault}\right),R_k^{selffault}-E_k+1\right), \tag{9}$$

where $cond\in\{CI,NC\}$.

$$I_{P_i}^{DIFF}\left(R_k^{selffault}\right)=I_{P_i}^{CI}\left(R_k^{selffault}\right)-I_{P_i}^{NC}\left(R_k^{selffault}\right). \tag{10}$$

Letting $L=\min\left(R_k^{selffault},R_k^{nofault}\right)$ in equation (3) and (4) we can get the upper bound of workload of a high priority backup copy B_i, which are $W_{B_i}^{CI}\left(\min\left(R_k^{selffault},R_k^{nofault}\right)\right)$ and $W_{B_i}^{NC}\left(\min\left(R_k^{selffault},R_k^{nofault}\right)\right)$. The upper bound of interference of B_i $I_{B_i}^{CI}\left(\min\left(R_k^{selffault},R_k^{nofault}\right)\right)$ and $I_{B_i}^{NC}\left(\min\left(R_k^{selffault},R_k^{nofault}\right)\right)$ and their difference $I_{B_i}^{DIFF}\left(\min\left(R_k^{selffault},R_k^{nofault}\right)\right)$ are calculated by equation (11) and (12).

$$I_{B_i}^{cond}\left(\min\left(R_k^{selffault},R_k^{nofault}\right)\right)=\min\left(W_{B_i}^{cond}\left(\min\left(R_k^{selffault},R_k^{nofault}\right)\right),\min\left(R_k^{selffault},R_k^{nofault}\right)-E_k+1\right),\quad(11)$$

where $cond\in\{CI,NC\}$.

$$I_{B_i}^{DIFF}\left(\min\left(R_k^{selffault},R_k^{nofault}\right)\right)=I_{B_i}^{CI}\left(\min\left(R_k^{selffault},R_k^{nofault}\right)\right)-I_{B_i}^{NC}\left(\min\left(R_k^{selffault},R_k^{nofault}\right)\right).\quad(12)$$

P_k only release one job in problem window. So its upper bound of interference is $I_{P_k}=\min\left(C_k,R_k^{selffault}-E_k+1\right)$. The sum of interference on B_k is calculated by equation (13).

$$\begin{aligned}I_{k-sum}^{selffault}\left(R_k^{selffault}\right)&=I_{P_k}+\sum_{i=1}^{k-1}\left(I_{P_i}^{NC}\left(R_k^{selffault}\right)+I_{B_i}^{NC}\left(\min\left(R_k^{selffault},R_k^{nofault}\right)\right)\right)\\&+sum_{\max\_m-1}\left(I_{P_i}^{DIFF}\left(R_k^{selffault}\right),I_{B_i}^{DIFF}\left(\min\left(R_k^{selffault},R_k^{nofault}\right)\right)\right)\end{aligned},\quad(13)$$

where $sum_{\max\_m-1}\left(I_{P_i}^{DIFF}\left(R_k^{selffault}\right),I_{B_i}^{DIFF}\left(\min\left(R_k^{selffault},R_k^{nofault}\right)\right)\right)$ is the sum of largest $m-1$ (m is the amount of processors) interference difference.

Starting from $R_k^{selffault}(1)=E_k$, we calculate equation (14) iteratively until $R_k^{selffault}(n+1)=R_k^{selffault}(n)$, where $R_k^{selffault}(n)$ is the response time of B_k under the condition of self-fault. If $R_k^{selffault}\le D_k$, τ_k is schedulable.

$$R_k^{selffault}(n+1)=E_k+\left\lfloor\frac{I_{k-sum}^{selffault}\left(R_k^{selffault}(n)\right)}{m}\right\rfloor\quad(14)$$

1.4 High-fault

If a high priority task τ_f ($f<k$) encounters a fault, which means in problem window $\left[r_k,R_k^{highfault}\right)$ a job of P_f encounters a fault, the job released by B_f in the same period of τ_f has to response. The upper bound of interference of high priority tasks are affected by the assumption of the instant t_{fault} when the fault occurs: (1) if t_{fault} coincides with the upper bound of no-fault response time of P_f, P_f produces most interference in problem window, (2) if the responding job of B_f fully resides in the problem window, B_f produces most interference in problem window, (3) if t_{fault} coincides with $R_k^{highfault}$, all high priority backups produce most interference. It is very difficult to find the t_{fault} which maximums the total interference. Thus we calculate the upper bound of interference of P_i, B_i, P_f and B_f respectively then add them up as the upper bound of total interference. In this way we get a safe but a little pessimistic upper bound.

Letting $L=R_k^{highfault}$ in equation (1)～(4) we can get the upper bound of workload of P_i and B_i ($i\ne f$) in problem window, which are $W_{P_i}^{CI}\left(R_k^{highfault}\right)$, $W_{P_i}^{NC}\left(R_k^{highfault}\right)$, $W_{B_i}^{CI}\left(R_k^{highfault}\right)$ and $W_{B_i}^{NC}\left(R_k^{highfault}\right)$. The upper bounds of interference of P_i and B_i $I_{P_i}^{CI}\left(R_k^{highfault}\right)$, $I_{P_i}^{NC}\left(R_k^{highfault}\right)$, $I_{B_i}^{CI}\left(R_k^{highfault}\right)$, $I_{B_i}^{NC}\left(R_k^{highfault}\right)$ and their interference $I_{P_i}^{DIFF}\left(R_k^{highfault}\right)$, $I_{B_i}^{DIFF}\left(R_k^{highfault}\right)$ are calculated by equation (15) and (16).

$$I_{ver}^{cond}\left(R_k^{highfault}\right)=\min\left(W_{ver}^{cond}\left(R_k^{highfault}\right),R_k^{highfault}-C_k+1\right),\quad(15)$$

$$I_{ver}^{DIFF}\left(R_k^{highfault}\right)=I_{ver}^{CI}\left(R_k^{highfault}\right)-I_{ver}^{NC}\left(R_k^{highfault}\right),\quad(16)$$

where $cond\in\{CI,NC\}$, $ver\in\{P_i,B_i\}$.

If t_{fault} coincides with the upper bound of no-fault response time of P_f, the upper bound of workload of P_f: $W_{P_f}^{CI}\left(R_k^{highfault}\right)$ and $W_{P_f}^{NC}\left(R_k^{highfault}\right)$, the upper bound of interference: $I_{P_f}^{CI}\left(R_k^{highfault}\right)$ and $I_{P_f}^{NC}\left(R_k^{highfault}\right)$, and their difference $I_{P_f}^{DIFF}\left(R_k^{highfault}\right)$ are also calculated by equation (1)～(2) and equation (15)～(16).

One job of B_f in problem window needs to response. Considering a special case of $T_f > R_k^{highfault}$, we have equation (17)～(19) to calculate the upper bound of workload and interference of B_f by assuming the first job of B_f in problem window responses.

$$W_{B_f}^{CI}\left(R_k^{highfault}\right)=\begin{cases}\min\left(E_f, R_k^{highfault}\right) & \left\lfloor \frac{A}{T_f} \right\rfloor \le 1 \\ E_f+\left\lfloor \frac{A}{T_f}-1 \right\rfloor \times E_f^{partial}+ & \\ \min\left(A-\left\lfloor \frac{A}{T_f} \right\rfloor \times T_f, E_f^{partial}\right) & else\end{cases} \tag{17}$$

where $A = R_k^{highfault} + R_f^{selffault} - E_f$.

$$W_{B_f}^{NC}\left(R_k^{highfault}\right)=\begin{cases}\min\left(E_f, R_k^{highfault}-O_f\right) & \left\lfloor \frac{R_k^{highfault}}{T_f} \right\rfloor \le 1 \\ E_f+\left\lfloor \frac{R_k^{highfault}}{T_f}-1 \right\rfloor \times E_f^{partial}+ & \\ \min\left(R_k^{highfault}-\left\lfloor \frac{R_k^{highfault}}{T_f} \right\rfloor \times T_f-O_f, E_f^{partial}\right) & else\end{cases} \tag{18}$$

$$I_{B_f}^{cond}\left(R_k^{highfault}\right)=\min\left(W_{B_f}^{cond}\left(R_k^{highfault}\right), R_k^{highfault}-C_k+1\right) \tag{19}$$

$$I_{B_f}^{DIFF}\left(R_k^{highfault}\right)=I_{B_f}^{CI}\left(R_k^{highfault}\right)-I_{B_f}^{NC}\left(R_k^{highfault}\right) \tag{20}$$

where $cond \in \{CI, NC\}$.

The upper bound of interference of P_k in problem window can be calculated by equation (21).

$$I_{k-sum}^{highfault}\left(R_k^{highfault}\right)=I_{P_f}^{NC}\left(R_k^{highfault}\right)+I_{B_f}^{NC}\left(R_k^{highfault}\right)+\sum_{i=1,i\neq f}^{k-1}\left(I_{P_i}^{NC}\left(R_k^{highfault}\right)+I_{B_i}^{NC}\left(R_k^{highfault}\right)\right)+ sum_{\max\_m-1}\left(I_{P_i\cup B_i\cup P_f\cup B_f}^{DIFF}\left(R_k^{highfault}\right)\right) \tag{21}$$

where $sum_{\max\_m-1}\left(I_{P_i\cup B_i\cup P_f\cup B_f}^{DIFF}\left(R_k^{highfault}\right)\right)$ is the sum of largest $m-1$ (m is the amount of processors) interference difference.

Starting from $R_k^{highfault}(1)=C_k$, we calculate equation (22) iteratively until $R_k^{highfault}(n+1)=R_k^{highfault}(n)$, where $R_k^{highfault}(n)$ is the response time of P_k under the condition of high-fault. If $R_k^{highfault} \le D_k$, τ_k is schedulable.

$$R_k^{highfault}(n+1)=C_k+\left\lfloor \frac{I_{k-sum}^{highfault}\left(R_k^{highfault}(n)\right)}{m} \right\rfloor \tag{22}$$

Run the high-fault schedulability test for all τ_f ($f<k$). If τ_k is schedulable when any one of high priority tasks encounters a fault, we can determine τ_k is schedulable under the condition of high-fault.

The calculation of O_i and $E_i^{partial}$

In the schedulability test described in last section, we use two unknown parameters: O_i and $E_i^{partial}$. In this section we give the detail on how to calculate them. The procedure of testing schedulability of a task set goes from the highest priority task to the lowest. The calculation of O_i and $E_i^{partial}$ are performed after τ_i is determined schedulable and based on the response time of τ_i.

By the testing procedure of section 4.2 we can get the upper bound of response time of B_i under self-fault condition. This response time has considered the worst case scenario which means

whenever B_i is ready it responses no later than $R_i^{selffault}$ from that instant. If we push the ready instant backwards $D_i - R_i^{selffault}$ from r_i, the response time of B_i is no later than D_i. So we can get

$$O_i = D_i - R_i^{selffault} \quad (23)$$

We assume that when B_i is ready it gets a processor and keeps running without interruption until corresponding primary P_i responses. At that time B_i is cancelled. In this case B_i consumes most resources. So we have

$$E_i^{partial} = \max\left(R_i^{nofault} - O_i, 0\right) \quad (24)$$

Simulation results

In this section we use randomly generated task sets to evaluate the performance of RRFTGS. For comparison we choose GS and FTGS-1 algorithm. GS is the global scheduling algorithm without fault tolerance capacity. FTGS-1 is a special form of the fault-tolerant global scheduling algorithm proposed by Pathan et al.[10] which only use passive backups. In this paper we make $f = 1$ in Pathan's algorithm which stands for one fault assumption. The performance metric is the ratio m/U which reflects processor demanding of each algorithm. m is the lowest amount of processors which makes the task set schedulable. U is the total utilization of the task set.

The task set generation algorithm is similar as that in [5-7]. The number of tasks in a task set n is at most 500. The utilization bound is a. The parameters of an individual task are generated as follow. The period T is uniformly distributed in $[1,500]$. The worst case execution time C is a random value in $[1, aT]$. The relative deadline $D = T$. For generating backup we simply use an identical copy of primary.

We set a as 0.2 and 0.5 respectively for each experiment. The priority assignment algorithms are TkC and OPA which have good performance in non-fault-tolerant cases[14]. The minimum demanding of processors m is determined by searching the possible space $[\lceil U \rceil, 2n]$ from the lowest. Repeat the experiment for 30 times with each group of parameters and use the average value as the result. The simulation results are shown in Fig. 3 and Fig. 4.

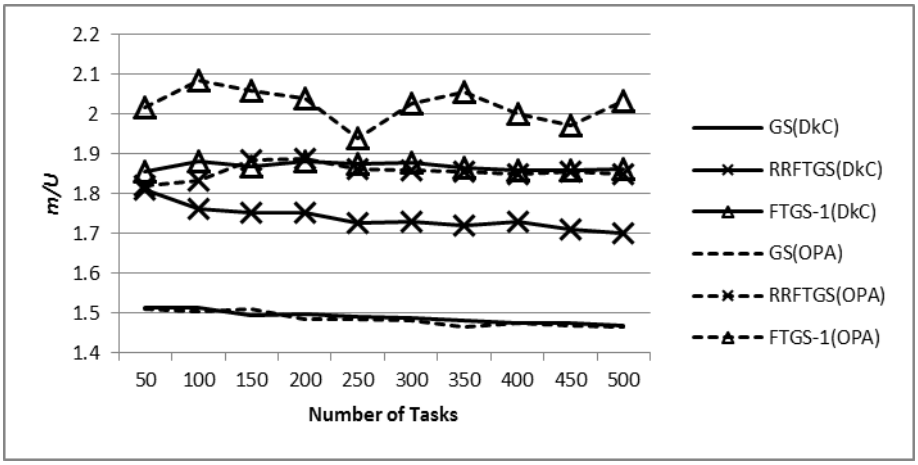

Fig. 3 ratio m/U of different algorithms when a=0.2

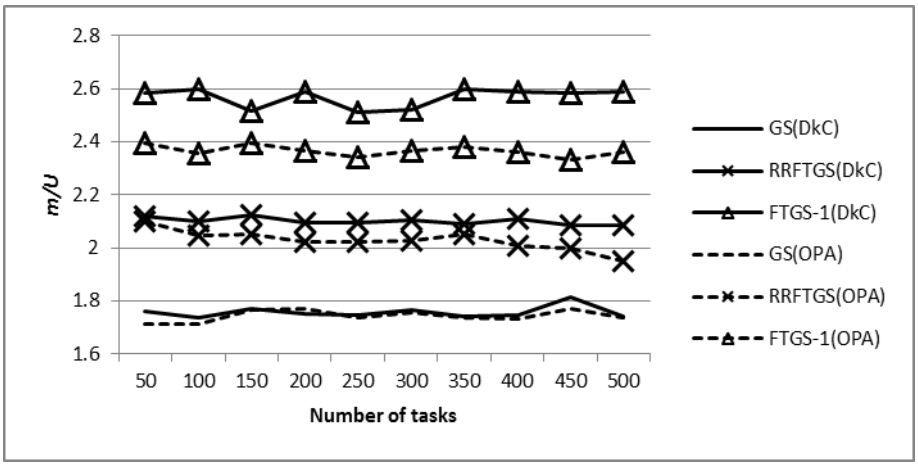

Fig. 4 ratio m/U of different algorithms when a=0.5

From Fig. 3～Fig. 4, we can see that RRFTGS has lower m/U ratio than FTGS-1 for both priority assignment algorithms. This result shows that RRFTGS has better performance than the approach that only use passive backups when $a \le 0.5$. If $a > 0.5$ passive backup only approach is no longer feasible. RRFTGS is still feasible in this case but the cost is high, as shown in Fig. 5.

There is an obvious trend that with the utilization bound increases the cost for fault tolerance rises up significantly. The reason is that it is difficult to schedule a high utilization backup as passive. When a high utilization primary responses usually there is little time left till its deadline. A portion of the backup should execute before its primary responses for the sake of schedulability.

Conclusion

In this paper we have proposed a fault tolerant global scheduling algorithm RRFTGS for multiprocessor hard real time systems as well as its schedulability test. RRFTGS uses both active and passive backups and determines by itself which one should be active backup or the other way. Comparing with the passive only approach, RRFTGS has better performance when utilization bound is less than 0.5. Another advantage of RRFTGS is that it can deal with task sets with utilization bound higher than 0.5 while the passive only approach can't.

For simplicity this paper assumes that a primary copy and its backup have no synchronization. However if we take into account the synchronization that actually exists the processor demanding for fault tolerance would decrease since overestimation of interference will be reduced. This is our research in the future.

References

[1] Mottaghia M. H., Zarandi H. R. DFTS: A dynamic fault-tolerant scheduling for real-time tasks in multicore processors. *Microprocessors and Microsystems*, 38(1), pp. 88-97, 2014.

[2] Galashi O. A., Mohammadi K., Gosheblagh R. O. Hybrid redundancy approach to increase the reliability of FPGA based speed controller core for high speed train. *Journal of Electronics* , 31(3), pp. 256 - 266, 2014.

[3] Baumann R. Soft errors in advanced computer systems. *IEEE Design & Test of Computers*, 22(3), pp. 258-266, 2005.

[4] Krishna C. M. Fault-tolerant scheduling in homogeneous real-time systems. *ACM Computing Surveys*, 46(4), pp. 48:1 - 48:34, 2014.

[5] Bertossi A. A. , Mancini L.V. , Rossini F. Fault-tolerant rate-monotonic first-fit scheduling in hard-real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(9), pp. 934 - 945, 1999.

[6] Bertossi A. A., Mancini L. V., Menapace A. Scheduling Hard-Real-Time Tasks with Backup Phasing Delay. Proceedings of 10th IEEE International Symposium on Distributed Simulation and Real-Time Applications, 2006: 107-118.

[7] Zhu P., Yang F. M., Tu G., et al. Feasible fault-tolerant scheduling algorithm for distributed hard real-time system. *Journal of Software*, 23(4), pp. 1010 - 1021, 2012.
[8] Chen H. M., Luo W., Wang W., et al. A novel real-time fault-tolerant scheduling algorithm based on distributed control systems. Proceedings of 2011 International Conference on Computer Science and Service System , 2011: 80-83.
[9] Berten V., Goossens J., Jeannot E. A probabilistic approach for fault tolerant multiprocessor real-time scheduling. Proceedings of 20th International Parallel and Distributed Processing Symposium , 2006: 152-160
[10] Pathan R. M., Jonsson J. FTGS: Fault-Tolerant Fixed-Priority Scheduling on Multiprocessors. Proceedings of IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, 2011: 1164-1175.
[11] Samala A. K., Mallb R., Tripathy C. Fault tolerant scheduling of hard real-time tasks on multiprocessor system using a hybrid genetic algorithm. *Swarm and Evolutionary Computation*, 14, pp. 92 - 105 , 2014.
[12] Guan N., Stigge M., Wang Y., et al. New Response Time Bounds for Fixed Priority Multiprocessor Scheduling. Proceedings of Real-Time Systems Symposium, 2009: 387-397
[13] Koren I., Krishna C. M. *Fault-Tolerant Systems*, Morgan Kaufmann, 2007.
[14] Davis R. I., Burns A. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1), pp. 1-40, 2011.