

# An adaptive scheme for fault-tolerant scheduling of soft real-time tasks in multiprocessor systems

R. Al-Omari<sup>a,1</sup>, A.K. Somani<sup>b</sup>, G. Manimaran<sup>b,\*</sup>

<sup>a</sup>*Eclipz Nest Pervasive, System Group, IBM Austin, TX, USA*

<sup>b</sup>*Department of Electrical and Computer Engineering, Iowa State University, 3219 Coover Hill, Ames IA 50011, USA*

Accepted 28 September 2004

## Abstract

The scheduling of real-time tasks with primary-backup-based fault-tolerant requirements has been an important problem for several years. Most of the known scheduling schemes are non-adaptive in nature meaning that they do not adapt to the dynamics of faults and task's parameters in the system. In this paper, we propose an adaptive fault-tolerant scheduling scheme that has a mechanism to control the *overlap interval* between the primary and backup versions of tasks such that the overall performance of the system is improved. The overlap interval is determined based on the observed fault rate and task's soft laxity. We also propose a new performance index, called *SR index*, that integrates schedulability (*S*) and reliability (*R*) into a single metric. To evaluate the proposed scheme, we have conducted analytical and simulation studies under different fault and deadline scenarios, and found that the proposed adaptive scheme adapts to system dynamics and offers better *SR index* than that of the non-adaptive schemes.

© 2005 Elsevier Inc. All rights reserved.

**Keywords:** Primary-backup scheduling; Real-time systems; Schedulability–reliability tradeoff; Performance index; Multiprocessors

## 1. Introduction

Real-time systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced [12]. Multiprocessors and multicomputer systems are emerging as a powerful computing means for real-time applications due to their capability for high performance and reliability. In most real-time applications, there is a need for reliable execution of tasks even in the presence of faults. Reliable (fault-tolerant) execution of a task is usually achieved by scheduling multiple versions of the task [1–6,9,11,14,15].

Primary-backup (PB) scheduling is an important fault-tolerant approach in which two versions of a task are scheduled on two different processors and an *acceptance test* is used to check the correctness of the result [3–5,9,11,14]. The three variants [5] of this approach are: primary-backup exclusive (PB-EXCL) [3,9], primary-backup concurrent (PB-CONCUR) [5,15], and primary-backup overlap (PB-OVER) [5].

In PB-EXCL approach, the primary and backup versions of a task are excluded in time as well as in space (i.e., scheduled on two different processors in a non-overlapping manner). The backup version is executed only if the output of the primary fails the acceptance test, otherwise it (backup) is deallocated from the schedule. This approach uses fewer resources if faults rarely occur, because backups are mostly deallocated. Nevertheless, it requires the *execution interval* (defined as the interval between start time of primary and finish time of backup) of a task to be at least twice that of the task's computation time. However, the execution interval

\* Corresponding author. Fax: +1 5152948432.

E-mail addresses: [alomari@us.ibm.com](mailto:alomari@us.ibm.com) (R. Al-Omari), [arun@iastate.edu](mailto:arun@iastate.edu) (A.K. Somani), [gmani@iastate.edu](mailto:gmani@iastate.edu) (G. Manimaran).

<sup>1</sup> This work was done when the author was at Iowa State University.

of a task is critical in soft real-time systems. The lower the execution interval, the higher the utility of the task output.

In PB-CONCUR approach, the primary and the backup versions of a task are executed concurrently. In this approach, the *consumption time* (defined as the total processor time used by a task for its execution) of a task is always twice that of the task's computation time irrespective of the fault rate in the system. Nevertheless, the execution interval for the task is always equal to its computation time. However, the consumption time is inversely proportional to the "processor utilization efficiency" (defined in Section 3.2). The lower the consumption time, the better the schedulability. The PB-OVER is a flexible approach wherein the primary and the backup versions of a task are scheduled to overlap in execution. This approach has the potential to exploit the advantages of the other two approaches if the overlap interval is suitably adapted.

It is evident from the above discussion that the PB approaches have a trade-off between the percentage of tasks being scheduled and the total utility of their output to the system. This trade-off depends on the primary fault probability in the system and task's laxity. For example, the PB-EXCL offers lower consumption time than PB-CONCUR when the fault probability is low; PB-CONCUR offers less execution interval for the task than PB-EXCL when the fault probability is high. Thus, the PB-CONCUR offers better performance when the fault probability is high and tasks have tight laxity. On the other hand, when the fault probability is low or tasks have sufficient laxity the PB-EXCL offers better performance. It can be noted that the reduction in both consumption time and execution interval will contribute towards better performance.

This trade-off can be captured by an adaptive scheduling scheme that controls the overlap interval between the versions based on the primary fault probability and task's laxity. Therefore, in this paper, we propose an adaptive PB scheduling scheme that makes use of an estimate of the primary fault probability and task's laxity to control the degree of overlap between its (task) versions. The adaptive scheme operates between two extremes: behaves like PB-EXCL approach when this interval is zero, behaves like PB-CONCUR approach when the interval is the execution time of the task. There exists some adaptive fault-tolerant scheduling algorithms (e.g., [4]) wherein the task specifies to the scheduler the adaptiveness strategy. Clearly, such an algorithm does not enjoy the benefits of feedback-based adaptiveness wherein the scheduler adapt with the system and tasks dynamics.

The rest of the paper is structured as follows. In Section 2, task, scheduler, and fault models are stated. In Section 3, we introduce the proposed adaptive scheduling scheme, the fault monitoring system, and the performance index that is used to evaluate the proposed schemes. In Section 4 we analytically compare the performance of the non-adaptive and adaptive PB-based fault-tolerant schemes. In Section 5, we propose an adaptive scheduling algorithm. In Section 6, we present the simulation studies of the PB fault-tolerant

approaches. Finally, in Section 7, some concluding remarks are made.

## 2. System model

In this section, we state the task, scheduler, and fault models.

### 2.1. Task model

1. The system has soft real-time aperiodic tasks. Each task  $T_i$  has the attributes arrival time ( $a_i$ ), ready time ( $r_i$ ), worst-case computation time ( $c_i$ ), a relative soft deadline ( $d_i^s$ ), and a relative firm deadline ( $d_i^f$ ).
2. Each task  $T_i$  has two versions, namely primary version ( $Pr_i$ ) and backup version ( $Bk_i$ ). The versions have identical attributes.
3.  $d_i^s = k_1 \times c_i$  and  $d_i^f = k_2 \times c_i$ , where  $k_2 \geq k_1$ . The smallest value of  $k_2$  is assumed to be 2 in order to ensure that the primary and the backup versions of each task can be scheduled within the firm deadline in the case of PB-EXCL.
4. Tasks are non-preemptable, i.e., when a task starts execution on a processor, it finishes to its completion. We do not consider preemptive scheduling in multiprocessor systems because it requires complex synchronization between primary and backup versions of a task. Moreover, the issue of resource access control for shared resources complicates the synchronization issue further. These are the reasons why most of the fault-tolerant scheduling algorithms (e.g., [3–5,9,14]) for multiprocessor systems assume a non-preemptive scheduling model, as adopted in this paper.

### 2.2. Scheduler model

In our dynamic multiprocessor scheduling, all the tasks arrive at a central processor called the scheduler, from where they are distributed to other processors in the system for execution. These processors are identical and connected through a shared medium. The communication between the scheduler and the processors is through dispatch queues. Each processor has its own dispatch queue as shown in Fig. 1. The scheduler runs in parallel with the processors, scheduling the newly arrived tasks and periodically updating the dispatch queues. A fault monitoring system periodically observes the fault rate in the system using a feedback loop. This fault rate is used to estimate the primary fault probability for the tasks in the system.

### 2.3. Fault model

We assume that each processor, except the scheduler, may fail due to hardware fault which results in tasks failure. We

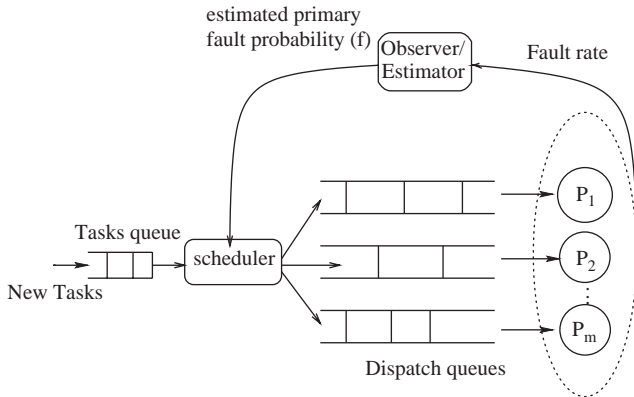


Fig. 1. An adaptive scheduler model.

further assume that the scheduler is made fault-tolerant by executing it on more than one processor or by using any other fault-tolerance technique. The faults can be transient or permanent and are independent of each other. Each independent fault results in failing of only one processor. The following assumptions form the fault model.

**Assumption 1.** The fault rate in the system changes with time. For real-time systems operating in unpredictable environments, fault rate is not known a priori. However, system performance can be specified under a set of representative fault rate profiles borrowed from control theory [8]; namely, the *step fault rate* and the *ramp fault rate*. In the context of real-time systems, the step fault rate represents the worst-case fault rate variation, and the ramp fault rate represents a nominal form of fault rate variation. A fault rate profile  $FR(t)$  is the system fault rate as a function of time. The fault rate profiles are defined as follows.

- *Step fault rate*  $SFR(t)$ : A fault rate profile that instantaneously jumps from a nominal fault rate  $FR_{nom}$  to fault rate  $FR_{max}$  and stay constant after the jump. The step fault rate is represented with a tuple  $SFR(FR_{nom}, FR_{max})$ .
- *Ramp fault rate*  $RFR(t)$ : A fault rate profile that increases linearly from the nominal fault rate to a specific level of fault rate during a time interval. Compared with the step fault rate, the ramp signal represents a less severe and more realistic fault rate variation scenario. The ramp fault rate  $RFR(t)$  is described with a tuple  $RFR(FR_{nom}, FR_{max}, T)$ , where  $FR_{nom}$  is the original fault rate,  $FR_{max}$  is the new fault rate, and  $T$  is the time it takes the fault rate to increase from  $FR_{nom}$  to  $FR_{max}$ .

**Assumption 2.** We assume that the  $(d_i^f - r_i) \forall T_i$  are much smaller than the typical mean time to failure (*MTTF*) value of the system. *MTTF* of the system is defined as the expected time for which the system operates before the first failure occurs. This assumption is used to enhance the probability of successfully executing the backup of a task, if its primary fails. Note that, there are always chances that both the primary and backup of a task can fail especially when

fault rate is high. This leads to system failure, in our work we assume that there is another mechanism that is activated if such a case occurs. Nevertheless, the functionality of the proposed techniques do not change if the backup fails but these techniques do not tolerate such a case.

**Assumption 3.** There exists fault-detection mechanisms such as fail-signal and acceptance test to detect processor and task failures, respectively. The scheduler will not schedule tasks to a known faulty processor.

### 3. Proposed adaptive scheduling scheme

Towards achieving the objective of enhancing the performance of the PB scheduling, we use a fault monitoring system to observe the fault rate in the system. The observed fault rate is used to estimate the fault probability of the tasks. This probability is used as a feedback to the scheduler to be used with the arriving tasks' laxity in determining the overlap interval between the versions of each task. Since the fault probability and task's soft laxity are used to adjust one of the scheduler parameters (i.e., the overlap interval), the scheduler becomes an adaptive scheduler.

Two variations of the adaptive scheme are proposed by varying the adaptation mechanism. The adaptation can be done in a continuous manner which leads to an approach called PB-OVER continuous (PB-OVER-CONT), or it can be in a discrete manner which leads to an approach called PB-OVER switch (PB-OVER-SWITCH). In PB-OVER-CONT, the overlap interval varies from no overlap to full overlap in a continuous manner as the fault probability varies from 0 to 1. In PB-OVER-SWITCH, the scheduler uses a threshold value of fault probability to switch from PB-CONCUR to PB-EXCL, i.e., if the probability is less than the threshold, the adaptive scheduler behaves like PB-EXCL, otherwise it behaves like PB-CONCUR. In PB-OVER-SWITCH, the threshold value is adapted with task's laxity.

#### 3.1. Fault monitoring system

A fault monitoring system periodically (with period  $p$ ) monitors the completion of tasks in the system. For each period the monitoring system counts the number of faulty primaries ( $n^f(t)$ ) and the total number of primaries completed ( $n(t)$ ) during that period.

According to the *frequency interpretation* probability concepts, the probability of an event (primary fail) is the proportion of the time that events of the same kind will occur in the long run. Hence, we define the primary fault probability ( $f$ ) as the ratio of the number of faulty primaries ( $n^f(t)$ ) to the total number of primaries ( $n(t)$ ) executed in a given interval  $[0, t]$ :  $f(t) = \frac{n^f(t)}{n(t)}$ . This probability is calculated every  $p$  time units, where  $p$  is the sample period for the monitor.

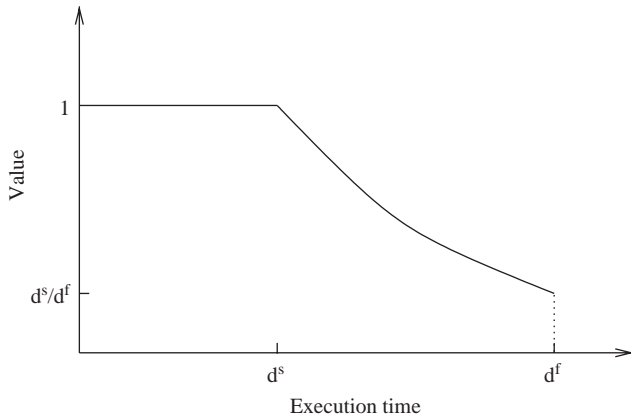


Fig. 2. The value function used to credit the logical result.

### 3.2. Performance index

To compare the PB fault-tolerant approaches, we propose a new performance index, called *schedulability–reliability (SR) index*, that captures the trade-off between the “processor utilization efficiency” of a task under a given fault-tolerant scheme versus the “value” offered by the task due to its successful execution.

For a task  $T_i$ , we define the *processor utilization efficiency* ( $UTIL_i$ ), of the task under a given PB scheme as the ratio of worst case execution time ( $c_i$ ) to the expected consumption time ( $\overline{PT}_i$ ).  $UTIL_i = c_i / \overline{PT}_i$ . We also define a value function ( $VAL_i$ ) that is used to credit the successful output from a task depending on its expected execution interval ( $\overline{ET}_i$ ). Fig. 2 shows the shape of the value function ( $VAL_i$ ) wherein the task contributes a value of one if it finishes before its soft deadline, a monotonically decreasing value if it finishes between its soft and firm deadlines, a value of zero if it misses its firm deadline. The monotonically decreasing task value is inversely proportional to its expected execution interval.

Thus, the tradeoff between processor utilization efficiency and the value offered is indirectly captured by the tradeoff between reducing  $\overline{ET}_i$  and reducing  $\overline{PT}_i$  of a task  $T_i$ . This is because reducing  $\overline{ET}_i$  increases the schedulability (and hence the “value” offered by the task) due to meeting more deadlines, but at the same time it could result in increase in  $\overline{PT}_i$  which results in reducing the schedulability. Similarly, reducing  $\overline{PT}_i$  increases schedulability due to less resource consumption, but at the same time it could decrease schedulability due to increase in  $\overline{ET}_i$  which results in missing of more deadlines.

$$VAL_i = \begin{cases} 1 & \text{for } \overline{ET}_i \leq d_i^s, \\ \frac{d_i^s}{\overline{ET}_i} & \text{for } d_i^s \leq \overline{ET}_i \leq d_i^f, \\ 0 & \text{for } \overline{ET}_i > d_i^f. \end{cases} \quad (1)$$

The performance index of a soft real-time system is usually measured as the sum of the values contributed by the

admitted tasks. For a given system capacity, there are two options: (i) admit a few tasks with a higher value for each task or (ii) admit a large number of tasks with a lower value for each task. Therefore, the  $SR_i$  index for a task ( $T_i$ ) is

$$SR_i = UTIL_i \times VAL_i. \quad (2)$$

The effect on the number of admitted tasks is inherently captured by the processor utilization efficiency of each task. The  $SR$  index for a set of  $n$  tasks that are admitted into the system is computed as  $SR = \frac{\sum_{i=1}^n SR_i}{n}$ .

## 4. Analysis of PB-based fault-tolerant approaches

In this section, we make certain assumptions about the system parameters in order to mathematically analyze and compare the effect of fault rate on the performance of non-adaptive and adaptive PB-based fault-tolerance approaches. Based on our analysis, we propose an adaptive fault-tolerant scheduling algorithm in the next section.

### 4.1. Assumptions

To analyze and compare the fault-tolerant approaches, we make the following assumptions:

1. The primary fault probability ( $f$ ) is estimated using the fault monitoring mechanism.
2. The worst case execution time for the tasks is fixed and is equal to  $c$ .
3. The soft deadline ( $d^s$ ) for a task is  $c$ , and its firm deadline ( $d^f$ ) is  $2c$ . The rationale for choosing these values for the deadlines is that these values represent the worst-case scenario (tight laxity) for the schemes, which helps in evaluating the schemes for the  $SR$  metric. Specifically, the PB-EXCL scheme does not have flexibility for starting the backup copy of a task. The effect of laxity is analyzed in Section 4.4.

### 4.2. Non-adaptive PB-base fault-tolerant approaches

In this section, we analyze the effect of the primary fault probability ( $f$ ) on the performance of the non-adaptive PB-base fault-tolerance approaches (PB-EXCL, PB-CONCUR, and PB-OVER).

#### 4.2.1. Primary-backup EXCLUSIVE (PB-EXCL)

In this approach, the primary and the backup versions of a task are excluded in space as well as in time in the schedule as shown in Fig. 3. From the figure, the expected execution interval  $\overline{ET}$  for the task is

$$\overline{ET} = 2cf + (1 - f)c = c(1 + f). \quad (3)$$

$\overline{ET}$  can take either of the following two values depending on whether the primary version of a task fails or not:

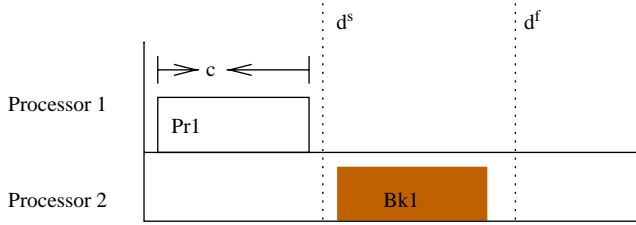


Fig. 3. Primary-backup exclusive (PB-EXCL).

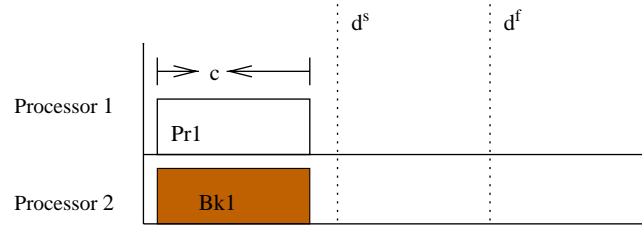


Fig. 4. Primary-backup concurrent (PB-CONCUR).

(i)  $\overline{ET} = 2c$  when the primary version of a task fails with probability  $f$  and the backup version succeeds; (ii)  $\overline{ET} = c$  when the primary version succeeds with probability  $1 - f$ . In the latter case, the backup is deallocated.

The expected consumption time ( $\overline{PT}$ ) for the task is given by

$$\overline{PT} = 2cf + (1 - f)c = c(1 + f). \quad (4)$$

Similarly,  $\overline{PT}$  can take either of the following two values depending on whether the primary version of the task fails or not: (i)  $\overline{PT} = 2c$  when the primary version of a task fails with probability  $f$  and the backup version succeeds; (ii)  $\overline{PT} = c$  when the primary version succeeds with probability  $1 - f$ . In the latter case, the backup version is deallocated.

From these two equations, it can be seen that both  $\overline{PT}$  and  $\overline{ET}$  increase linearly from  $c$  to  $2c$  as  $f$  varies from 0 to 1. Also, it can be observed that the PB-EXCL approach performs well when  $f = 0$  as  $\overline{ET} = \overline{PT} = c$ , which is the best achievable. However, this approach performs very poorly when  $f = 1$  as it doubles the execution interval for the task, i.e.,  $\overline{ET} = 2c$ .

The processor utilization efficiency in this approach is given by

$$UTIL_i = \frac{c}{(1 + f)c} = \frac{1}{(1 + f)}. \quad (5)$$

Since  $d_i^s = c$  and  $d_i^f = 2c$ , the value function for this approach is  $VAL_i = \frac{c}{(1+f)c} = \frac{1}{1+f}$ . The  $SR_i$  index for this approach is then

$$SR_i = \frac{1}{(1 + f)^2}. \quad (6)$$

#### 4.2.2. Primary-backup CONCURRENT (PB-CONCUR)

In this approach, the primary and the backup versions of tasks are executed concurrently as shown in Fig. 4. From the figure, the expected execution interval  $\overline{ET}$  for the task is given by

$$\overline{ET} = c. \quad (7)$$

This means that the execution interval for the task is constant and does not depend on  $f$ . Also,  $\overline{PT} = 2c$  and does not depend on  $f$ . From these two equations, the approach is favorable when  $f = 1$  as  $\overline{ET} = c$  and  $\overline{PT} = 2c$  for this case, which is the best that can be achieved. However, this

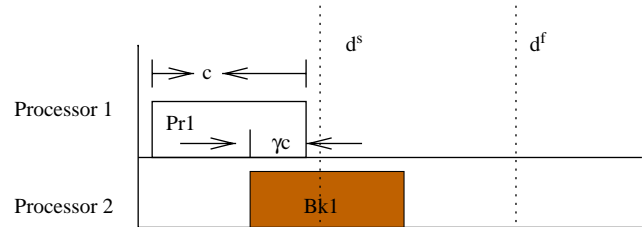


Fig. 5. Primary-backup overlap (PB-OVER).

approach is not favorable when  $f = 0$  as  $\overline{PT} = 2c$  which is twice that of PB-EXCL for the same case.

The processor utilization efficiency for this approach is given by

$$UTIL_i = \frac{c}{2c} = 0.5. \quad (8)$$

Since  $d_i^s = c$  and  $d_i^f = 2c$ , the value function for this approach is  $VAL_i = \frac{c}{c} = 1$ . The  $SR_i$  index for this approach is then

$$SR_i = 0.5 \quad (9)$$

#### 4.2.3. Primary-backup OVERLAP (PB-OVER)

In this approach, the primary and the backup versions of a task can overlap in execution by an amount equal to  $\gamma c$  as shown in Fig. 5.

From the figure, the expected execution interval  $\overline{ET}$  for the task is given by

$$\overline{ET} = f(c + (1 - \gamma)c) + (1 - f)c = cf(1 - \gamma) + c. \quad (10)$$

$\overline{ET}$  can take either of the following two values depending on whether the primary version of the task fails or not: (i)  $\overline{ET} = (c + (1 - \gamma)c)$  when the primary version fails with probability  $f$  and the backup version succeeds; (ii)  $\overline{ET} = c$  when the primary version succeeds with probability  $1 - f$ .

The expected consumption time  $\overline{PT}$  for the task is

$$\overline{PT} = 2cf + (1 - f)(c + \gamma c) = (1 - \gamma)cf + (1 + \gamma)c. \quad (11)$$

In this approach,  $\overline{PT}$  can take either of the following two values depending on whether the primary version of the task fails or not: (i)  $\overline{PT} = 2c$  when the primary version of a task

fails with probability  $f$  and the backup version succeeds; (ii)  $\overline{PT} = (1 + \gamma)c$  when the primary succeeds with probability  $1 - f$  since a processor time of  $(1 - \gamma)c$  is reclaimed due to deallocation of part of the backup version.

The processor utilization efficiency for this approach is given by

$$UTIL_i = \frac{c}{(1 + f + \gamma(1 - f))c} = \frac{1}{1 + f + \gamma(1 - f)}. \quad (12)$$

Since  $d_i^s = c$  and  $d_i^f = 2c$ , the value function for this approach is

$$VAL_i = \frac{c}{(1 + f(1 - \gamma))c} = \frac{1}{1 + f(1 - \gamma)}. \quad (13)$$

The  $SR_i$  index for this approach is then

$$SR_i = \frac{1}{(1 + f + \gamma(1 - f))(1 + f(1 - \gamma))}. \quad (14)$$

The most important property that is expected from PB-OVER is to combine the advantages of PB-EXCL and PB-CONCUR approaches. That is, when  $f = 0$ , the desirable values of  $\overline{ET}$  and  $\overline{PT}$  is  $c$ . When  $f = 1$ , the desirable values of  $\overline{ET}$  and  $\overline{PT}$  is  $c$  and  $2c$ , respectively.

#### 4.3. Adaptive PB-OVER fault-tolerant approaches

In this section, we propose two adaptive PB-OVER approaches by varying the adaptation mechanism: PB-OVER continuous (PB-OVER-CONT) and PB-OVER switch (PB-OVER-SWITCH).

##### 4.3.1. Primary-backup OVERlap CONTinuous (PB-OVER-CONT)

In PB-OVER-CONT, the overlap interval varies from no overlap to full overlap in a continuous manner as the fault probability varies from 0 to 1. From Eqs. (10) and (11), we found that this can be achieved by substituting  $\gamma = f$  which results in

$$\overline{ET} = (-f^2 + f + 1)c \quad \text{and} \quad \overline{PT} = (-f^2 + 2f + 1)c. \quad (15)$$

Fig. 6 shows the values of  $\overline{PT}$ , and  $\overline{ET}$  for the PB-OVER-CONT approach with varying  $f$ . From the figure, it can be seen that  $\overline{PT}$  increases quadratically from  $c$  to  $2c$  as  $f$  varies from 0 to 1.  $\overline{ET}$  increases quadratically from  $c$  to  $1.25c$  as  $f$  varies from 0 to 0.5 and then decreases quadratically from  $1.25c$  to  $c$  as  $f$  varies from 0.5 to 1.

Using the assumption in Section 4.1 the processor utilization efficiency for this approach is given by

$$UTIL_i = \frac{c}{(-f^2 + 2f + 1)c} = \frac{1}{-f^2 + 2f + 1}. \quad (16)$$

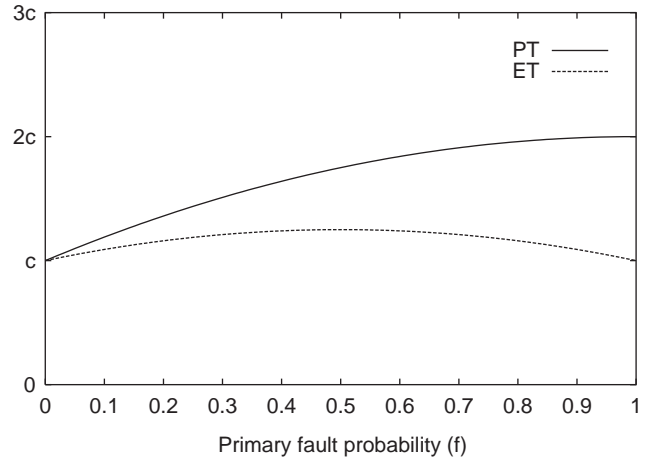


Fig. 6.  $\overline{PT}$ , and  $\overline{ET}$  for the PB-OVER-CONT approach.

Since  $d_i^s = c$  and  $d_i^f = 2c$ , the value function for this approach is

$$VAL_i = \frac{c}{(-f^2 + f + 1)c} = \frac{1}{(-f^2 + f + 1)}. \quad (17)$$

The  $SR_i$  index for this approach is then

$$SR_i = \frac{1}{(-f^2 + 2f + 1)(-f^2 + f + 1)}. \quad (18)$$

##### 4.3.2. Primary-backup OVERlap SWITCH (PB-OVER-SWITCH)

In PB-OVER-SWITCH, the scheduler switches from PB-EXCL to PB-CONCUR depending on the value of  $f$ . If  $f$  is less than a threshold  $f_0$ , then the PB-OVER-SWITCH approach behaves like PB-EXCL, else it behaves like PB-CONCUR. The threshold  $f_0$  is the value of  $f$  at which the schedulability–reliability index of PB-EXCL is equal to that of PB-CONCUR. Thus,  $\overline{ET}$  and  $\overline{PT}$  of the task become

$$\overline{ET} = \begin{cases} c \times (1 + f) & \text{for } 0 \leq f \leq f_0, \\ c & \text{for } f_0 < f \leq 1, \end{cases} \quad (19)$$

$$\overline{PT} = \begin{cases} c \times (1 + f) & \text{for } 0 \leq f \leq f_0, \\ 2c & \text{for } f_0 < f \leq 1, \end{cases} \quad (20)$$

For the given assumption the value of  $f_0$  is the value of  $f$  that satisfies  $\frac{1}{(1+f)^2} = 0.5$ , which is  $f = \sqrt{2} - 1$ .

Fig. 7 shows the values of  $\overline{PT}$ , and  $\overline{ET}$  of the PB-OVER-SWITCH approach for varying  $f$ . From the figure, it can be noted that  $\overline{PT}$  and  $\overline{ET}$  increase linearly from  $c$  to  $\sqrt{2}c$  as  $f$  varies from 0 to  $\sqrt{2} - 1$ . When  $f > \sqrt{2} - 1$ , the scheduler will switch to the PB-CONCUR approach causing  $\overline{PT}$  to jump to  $2c$  and  $\overline{ET}$  to jump to  $c$ .

Using the assumption in Section 4.1 the  $SR$  index for this approach is given by

$$SR_i = \begin{cases} \frac{1}{(1+f)^2} & \text{for } 0 \leq f \leq \sqrt{2} - 1, \\ 0.5 & \text{for } \sqrt{2} - 1 \leq f \leq 1. \end{cases} \quad (21)$$

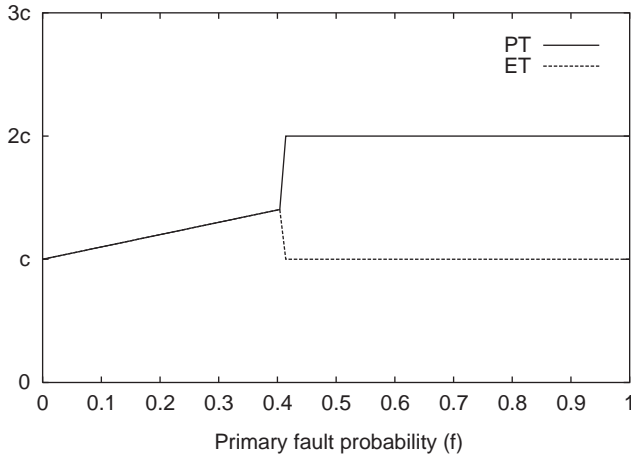


Fig. 7.  $\overline{PT}$ , and  $\overline{ET}$  for the PB-OVER-CONT approach.

Using the analytical equations, Figs. 8(a) shows the effect of primary fault probability on the *SR* index for all PB approaches. From the figures, it can be seen that the PB-CONCUR offers the best *SR* index when  $f > \sqrt{2} - 1$ . This is because in this region ( $f > \sqrt{2} - 1$ ) the probability of fault is high. Therefore, scheduling both versions of a task concurrently within the soft deadline will give the best *SR* index, which is 0.5. The *SR* index offered by the PB-OVER-CONT lies between the PB-CONCUR and PB-EXCL for all fault probability. Except for the interval ( $f \in [0.38, 0.46]$ ) where PB-OVER-CONT has the minimum *SR* index. The PB-EXCL offers the best *SR* index when  $f \leq \sqrt{2} - 1$  because the probability of backup deallocation is high in this interval. Since the PB-OVER-SWITCH behaves like PB-EXCL when  $f \leq \sqrt{2} - 1$  and like PB-CONCUR when  $f > \sqrt{2} - 1$ , it offers the best *SR* index for all values of  $f$ .

#### 4.4. Effect of task's soft laxity on performance

In this section, we first study the effect of task's soft laxity on the performance of the PB-based fault-tolerant approaches. Next, we propose a mechanism that allow the

fault probability threshold ( $f_0$ ) to be adapted with task's soft laxity.

In Section 4, we assumed that the task's soft deadline ( $d^s$ ) is equal to  $c$ . This assumption allows the PB-EXCL approach to schedule only one version of a task ( $Pr$ ) within its soft deadline and the other version ( $Bk$ ) must be scheduled after the soft deadline. This degrades the performance of the PB-EXCL approach as  $f$  increases since the correct output is always produced by the backup which has less value. When tasks have large soft deadline, then both the primary and the backup versions of the task can be scheduled in an exclusive manner within their soft deadline. Therefore, all PB-based fault-tolerant approaches offer the same output value for the finished tasks which is equal to one and does not depend on  $f$ . Hence, the *SR* index will be  $0.5$ ,  $\frac{1}{1+f}$ , and  $\frac{1}{-f^2+2f+1}$ , respectively for the PB-CONCUR, PB-EXCL, and the PB-OVER-CONT approaches (Fig. 8b).

Using those equations, Fig. 8b shows the effect of primary fault probability on the *SR* index for all PB approaches when tasks have large soft laxity. From the figures, we notice that the PB-EXCL offers the best *SR* index for all value of  $f$ . The PB-SWITCH offers the best *SR* index only when  $f < \sqrt{2} - 1$  and it offers the lowest *SR* index when  $f > \sqrt{2} - 1$ . This is because the threshold value ( $f_0$ ) that is used by the PB-SWITCH approach to switch between the PB-EXCL and PB-CONCUR approaches is constant ( $f_0 = \sqrt{2} - 1$ ) and does not change with changing task's soft deadline. In Section 4.3.2, the threshold  $f_0$  is defined as the value of  $f$  at which the *SR* index of PB-EXCL is equal to that of PB-CONCUR. Therefore, for the given task's deadline the value of  $f_0$  is the value of  $f$  that satisfies  $\frac{1}{(1+f)} = 0.5$ , which is  $f = 1$ . By using this new threshold value ( $f_0 = 1$ ), the PB-OVER-SWITCH always behaves like PB-EXCL approach which offers the best performance in this case.

It is evident from the above discussion that the threshold value ( $f_0$ ) have to be adapted with the task's soft laxity to be able to determine the correct overlap interval between the primary and the backup versions of each task that maximizes the performance of the system. To do so, the scheduler behaves as follows for each task ( $T_i$ ) that arrives

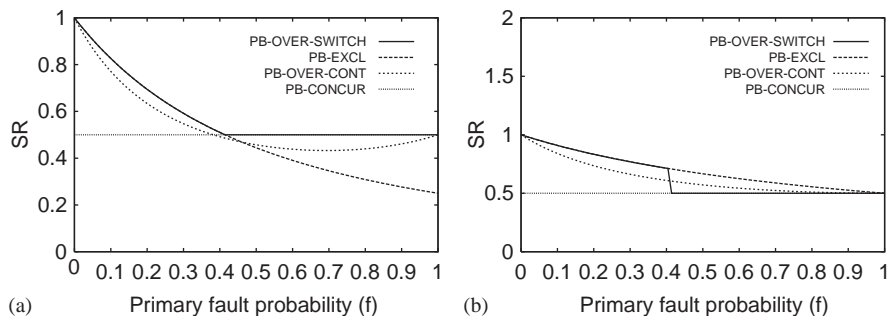


Fig. 8. *SR* index for the PB approaches: (a) small soft laxity and (b) large soft laxity.

in the system:

1. Schedules the primary ( $Pr_i$ ) version of the task.
2. If the primary was scheduled within the soft deadline, then

- (a) Determine the overlap interval ( $\sigma_i c_i$ ) between the primary and the backup versions of the task so that both versions will be scheduled within the soft deadline.

$$\sigma_i = \frac{c_i - (d_i^s - ft(Pr_i))}{c_i}, \quad (22)$$

where  $ft(Pr_i)$  is the relative finish time of the primary version.

- (b) If  $\sigma_i < 0$  then,  $\sigma_i = 0$ .

3. Else if the primary was scheduled to finish after the soft deadline, then  $\sigma_i = 1$ .

Using this value ( $\sigma_i$ ) the scheduler calculates the threshold value ( $f_0^i$ ) that is used by the PB-OVER-SWITCH approach to select the correct overlap interval (i.e., if ( $f < f_0^i$ ):  $\gamma = 0$ ?  $\gamma = 1$ ) between the primary and the backup versions for this task ( $T_i$ ). To calculate  $f_0^i$  for a given task  $T_i$  the scheduler uses the following equation:

$$f_0^i = (\sqrt{2} - 2)\sigma_i + 1. \quad (23)$$

The above equation is derived from the fact that  $f_0$  change from  $\sqrt{2} - 1$  to 1, when the task's soft laxity change from the case where the scheduler is only able to schedule one version of task within its soft deadline to the case where it able to schedule both version in an exclusive manner within the soft deadline.

## 5. The proposed adaptive dynamic scheduling algorithm

In this section, we first introduce the adaptive dynamic scheduling algorithm. Next, we discuss the issue of the primary-backup synchronization.

Unlike static scheduling of periodic tasks, dynamic scheduling of aperiodic tasks must be very simple because its overhead has serious effects on real-time processing. In this section, we present a heuristic algorithm that incorporates the two adaptive approaches for scheduling two versions of each task in such a way that the  $SR$  index of the system is maximized. The scheduler works as follows for each task  $T_i = (r_i, c_i, d_i^s, d_i^f)$  that arrives in the system:

1.  $Pr_i$  is scheduled first as follows:

- (a) Tries to find a free slot of length  $c_i$  between time  $r_i$  to time  $d_i^f$ . Since the scheduler tries to maximize the output value, it is appropriate to schedule  $Pr_i$  as early as possible so it will finish within the soft deadline ( $d_i^s$ ). Note that a heuristic search algorithm such as the Spring scheduling [13] can be used to find the best fit free slot.

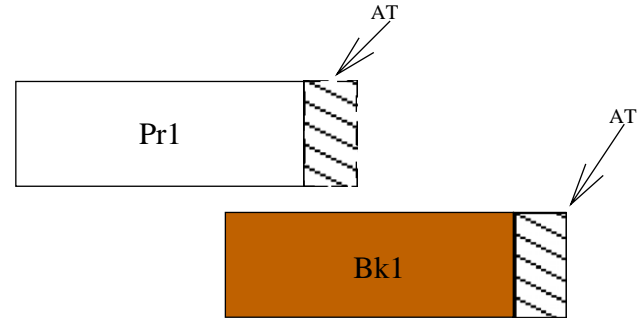


Fig. 9. Structure of primary and backup versions.

- (b) If  $Pr_i$  cannot be scheduled without overlapping any previously scheduled slot, task  $T_i$  is rejected.

2. If  $Pr_i$  is schedulable, then

- (a) Determines  $\sigma_i$  and calculates  $f_0^i$  as discussed in Section 4.4.
- (b) Uses the estimated fault probability ( $f$ ), to determine the overlap interval between the primary and backup versions of the task ( $\gamma c_i$ ) as discussed in Section 4.3.
- (c) Sets ready time ( $r_i^b$ ) for the backup version,  $r_i^b = ft(Pr_i) - \gamma c_i$ .
- (d) Sets processor( $Bk_i$ )  $\neq$  processor( $Pr_i$ ).

3.  $Bk_i$  is scheduled as follows:

- (a) Tries to find a free slot of length  $c_i$  between time  $r_i^b$  to time  $d_i^f$ . Since the scheduler tries to maximize the  $SR$  index, it is appropriate to schedule  $Bk_i$  as early as possible.
- (b) If  $Bk_i$  cannot be scheduled without overlapping any previously scheduled slot,  $Pr_i$  is unscheduled and task  $T_i$  is rejected.

### 5.1. Primary-backup synchronization

The primary and backup versions of a task need to synchronize each other to produce the correct result. A *flag* bit, which is initialized to 0, is used to indicate the success of writing a valid result by the primary or backup version. The structures of the primary and backup versions are shown in Fig. 9.

From Fig. 9, the primary version works as follows. It does the intended computation, then performs the acceptance test (AT). If AT succeeds and  $flag = 0$ , it writes the result, sets the  $flag = 1$ , and initiates a signal to the kernel to de-allocate the backup. The operation of the backup is similar to that of the primary except that it does not de-allocate the primary as shown in Fig. 10. The *lock* bit is used to ensure mutually exclusive access to the *flag* between primary and backup in the case of concurrent execution.

```

Synchronize(Task version)
{
  switch (Task version)
  case: Primary
    Acquire the lock.
    If (flag=0) then
      {Write the result.
       flag=1.
       De-allocate the backup. }
    Release the lock.
  case: Backup
    Acquire the lock.
    If (flag = 0) then
      {Write the result.
       flag=1. }
    Release the lock.
}

```

Fig. 10. Primary-backup synchronization.

## 6. Simulation studies

In this section, we first introduce the simulation model that is used to study the performance of the PB-based fault-tolerant approaches. Next, we compare the performance of the proposed new adaptive fault-tolerant approaches (PB-OVER-CONT, and PB-OVER-SWITCH) with the existing non-adaptive approaches (PB-EXCL, and PB-CONCUR).

### 6.1. Simulation model

A multiprocessor simulation of a soft real-time system was used to study the performance of the adaptive scheduling scheme. The parameters used in the simulation studies are given in Table 1. The tasks for the simulation are generated as follows:

1. The worst case computation times of primary versions are chosen uniformly between  $Min_c$  and  $Max_c$ .
2. The soft deadline ( $d_i^s$ ) of a task  $T_i$  is equal to  $k_1 \times c_i$ . Also, the firm deadline ( $d_i^f$ ) of a task  $T_i$  equal to  $k_2 \times c_i$ .
3. The inter-arrival time between tasks follows exponential distribution with mean  $\theta$ .
4. The task load  $L$  is defined as the expected number of task arrivals per mean service time and its value is approximately equal to  $\frac{C}{\theta}$ , where  $C$  is the mean computation time of the system. The mean computation time  $C$  has been calculated based on fault free system.

Table 1  
Simulation parameters

Parameter	Explanation	Value used
$Min_c$	Minimum computation time of tasks	2 s
$Max_c$	Maximum computation time of tasks	20 s
$k_1$	The factor that relate $d_i^s$ to $c_i$	1...5
$k_2$	The factor that relate $d_i^f$ to $c_i$	2...5
$f$	The primary fault probability	0...1
$L$	The system offered task load	1
$m$	Number of processors	6
$p$	Monitoring period	40 s

5. The backup versions are assumed to have identical characteristics of their primary versions.

The simulator has five components: source which generate tasks; a scheduler that makes admission/rejection decisions and determines the overlap interval on submitted tasks; a shared memory multiprocessor system that models the execution of the tasks; a monitor that periodically counts the number of finished faulty primaries; and an estimator that periodically estimates the primary fault probability.

### 6.2. Simulation results

In this section, we compare the performance of the proposed new adaptive fault-tolerant approaches (PB-OVER-CONT, and PB-OVER-SWITCH) with the existing non-adaptive approaches (PB-EXCL, and PB-OVER). Two experiments have been used to study the PB-based fault-tolerant approaches. In the first experiment (Experiments A) the fault rate is constant for each simulation run. In the second experiment (Experiments B) the fault rate, for each simulation run, is dynamically changed using the step and the ramp fault rate profiles (see Section 2.3).

#### 6.2.1. Experiments A: steady fault rate

Experiments A compares the performance of the proposed new adaptive fault-tolerant approaches with the existing non-adaptive approaches. The comparison is done using the simulation model. The  $SR$  index has been used as the performance metric. For each point in the performance plots (Figs. 11–13), the system was simulated with 20,000 tasks. This number of tasks has been chosen to have a 99% confidence interval within  $\pm 0.0035$  around each value of  $SR$ .

**6.2.1.1. Effect of the primary fault probability ( $f$ ) on  $SR$  index.** Figs. 11(a) and (b) show the effect of primary fault probability on the  $SR$  index for all PB approaches. Fig. 11(a) shows the behavior of the PB approaches when  $f$  varies for tasks that have relative soft deadline equals to  $2c_i$  and relative firm deadline equals to  $5c_i$ . Fig. 11(b) shows the case for tasks that have relative soft deadline equals to  $4c_i$  and relative firm deadline equals to  $5c_i$ .

From the figures, it can be seen that when the relative soft deadline is large (Fig. 11(b)) the PB-OVER-SWITCH and

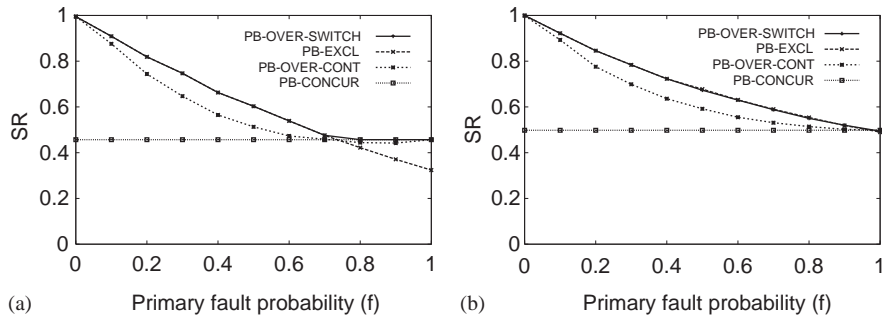


Fig. 11. Effect of primary fault probability ( $f$ ) on SR index: (a)  $d^s = 2c_i$ , and  $d^f = 5c_i$ , and (b)  $d^s = 4c_i$ , and  $d^f = 5c_i$ .

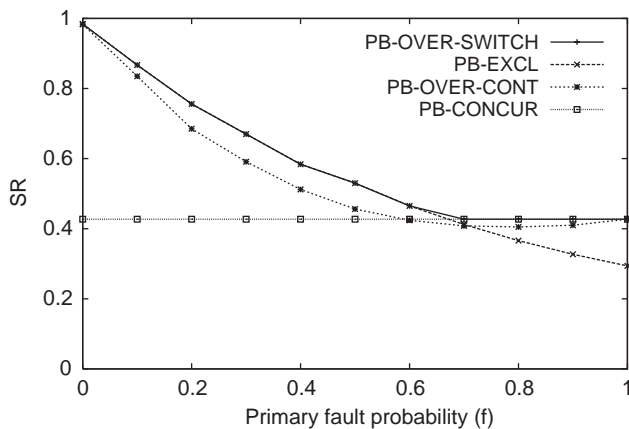


Fig. 12. Effect of primary fault probability ( $f$ ) on SR index.  $d^s \in [c_i, 3c_i]$ , and  $d^f \in [3c_i, 5c_i]$ .

the PB-EXCL approaches behave similar and offer the best SR index. This is because when  $d^s$  is large, the tasks have enough soft laxity to be scheduled in an exclusive manner within the soft deadline. The threshold value  $f_0$  used by the PB-OVER-SWITCH approach is approximately equal to 1. From Fig. 11(a), we can notice that the PB-EXCL approach has the highest SR index when  $f < 0.75$ , and the PB-CONCUR approach has the highest SR index when  $f > 0.75$ . The SR index offered by the PB-OVER-CONT lies between the PB-CONCUR and PB-EXCL for all values of primary fault probability. Since the PB-OVER-SWITCH behaves like PB-EXCL when  $f \leq 0.75$  and like PB-CONCUR when  $f > 0.75$ , it offers the best SR index for all values of  $f$ .

Fig. 12 shows the effect of primary fault probability on the SR index for all PB approaches. For Fig. 11, the task load was homogeneous (i.e., all the tasks have the same relative soft and firm laxity). For Fig. 12, the tasks soft deadline is chosen uniformly in the interval  $[c_i, 3c_i]$  and the task firm deadline is chosen uniformly in the interval  $[3c_i, 5c_i]$  in order to generate a non-homogeneous task load. From Fig. 12, we notice that the behavior of the PB-based fault-tolerant approaches stay the same as in the case of homogeneous task load.

**6.2.1.2. Effect of task's soft laxity on SR index.** Figs. 13(a) and (b) show the effect of tasks soft laxity on the SR index for all PB approaches. Fig. 13(a) shows the behavior of the PB approaches when  $d^s$  varies for the system in which the primary fault probability is 0.25 and tasks' relative firm deadline is  $5c_i$ . Fig. 13(b) shows the case for system in which the primary fault probability is 0.75 and tasks' relative firm deadline is also  $5c_i$ .

From the figures, it can be seen that when the primary fault probability ( $f$ ) is low (Fig. 13(a)) the PB-OVER-SWITCH and the PB-EXCL approaches behave similar and offer the best SR index for all values of  $d^s$ . This is because when  $f$  is small, the PB-OVER-SWITCH switch to an overlap interval equal to zero. The SR index offered by the PB-OVER-CONT lies between the PB-CONCUR and PB-EXCL for all value of  $d^s$ . From Fig. 13(b), we can notice that the PB-CONCUR and PB-OVER-SWITCH approaches have the highest SR index when  $d^s \leq 2$ . This is because, when  $f$  is high and tasks have small soft laxity then the threshold value that is used by the PB-OVER-SWITCH for each task is smaller than the fault probability in the system ( $f = 0.75$ ). Thus, PB-OVER-SWITCH behaves like PB-CONCUR for this region. We can notice that when  $2 \leq d^s \leq 3$ , the PB-OVER-SWITCH approach has the highest SR index. This is because, in this interval, the tasks have medium values of soft deadline. Thus PB-OVER-SWITCH maximizes the performance of the system by adapting task's threshold based on its soft deadline. Finally, we can notice, from Fig. 13(b), that the PB-EXCL and PB-OVER-SWITCH approaches have the highest SR index when  $d^s > 3$ .

### 6.2.2. Experiments B: dynamic fault rate

To capture the transient behavior of the PB fault tolerant approaches in response to fault rate variations, we use the instantaneous value for the schedulability–reliability ( $SR(t)$ ) index. The instantaneous  $SR(t)$  index is defined as the product of the instant processor utilization efficiency ( $UTIL(t)$ ) and the instant task value ( $VAL(t)$ ) at time  $t$ . In contrast, the average SR index is defined as the time average of the instantaneous  $SR(t)$  index for the entire run-time. Each point in the performance plots (Fig. 14) is the average value (with

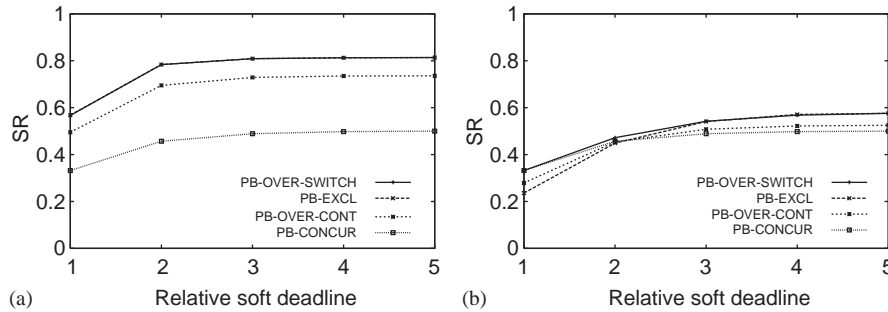


Fig. 13. Effect of task's soft laxity on  $SR$  index: (a)  $f = 0.25$ , and  $d^f = 5c_i$ , and (b)  $f = 0.75$ , and  $d^f = 5c_i$ .

95% confidence interval within  $\pm 0.019$  around the results) of 20 runs. In each run the system was simulated with 20,000 tasks.

**6.2.2.1. Step fault rate profile.** Fig. 14(a) shows the effect on  $SR(t)$  index for the four PB fault-tolerant approaches in response to the step fault rate  $SFR(FR_{0\%}, FR_{100\%})$ .  $FR_{0\%}$  is the fault rate that makes the primary fault probability equal to zero ( $f = 0$ ), and  $FR_{100\%}$  is the fault rate that makes the primary fault probability equal to one ( $f = 1$ ). Each time unit in the X-axes correspond to 40 seconds in the simulation time. The step fault rate was changed from  $FR_{0\%}$  to  $FR_{100\%}$  at  $t = 500$  time units.

From the figure, we see that the PB-CONCUR offers a constant  $SR(t)$  which does not change with  $t$ . This is because of its nature of always scheduling both the versions of a task concurrently within the soft deadline. The PB-EXCL offers  $SR(t)$  equals to one when  $t \leq 500$ , otherwise a  $SR(t)$  equals to 0.25 when  $t > 500$ . This is because for  $t < 500$  the probability of primary to fail is equal to zero ( $f = 0$ ) which will result in deallocating the backup versions. For  $t > 500$  the probability of primary to fail is equal to one which will result in executing the backups. The PB-OVER-CONT and PB-OVER-SWITCH behave similarly in response to step fault rate. They offer  $SR(t)$  equals to one for  $t \leq 500$ , otherwise a  $SR(t)$  equals to 0.5 for  $t > 500$ . Since the fault rate switch from  $FR_{0\%}$  to  $FR_{100\%}$  the two adaptive approaches will behave similar to PB-EXCL approach for the interval  $[0, 500]$ , and they will adapt after a short period ( $3p$ ) to behave similar to PB-CONCUR for the interval  $[500, 1000]$ .

From the figure, we see that the smallest value of  $SR(t)$  in the transient state ( $t \approx 500$ ) for the adaptive approaches is within 25% from its steady state value. This value is called the overshoot, which represents the worst-case transient performance of the system in response to the fault rate profile. Also, we can notice that the time taken for  $SR(t)$  to enter the steady state after the step fault rate profile is equal to  $3p$ . This time is called the settling time, which represents how fast the system can recover from a transient state. Since from the control theory point of view our sys-

tem is an open-loop system, the overshoot and the settling time are only vary with the monitoring period  $p$ . Reducing the monitoring period  $p$  will enhance both the overshoot value and the settling time. In contrast, reducing the monitoring period  $p$  will reduce the accuracy of estimating the fault probability which will effect the stability of the system. According to the frequency interpretation probability concepts, the probability of an event should be calculated in a long run. In this paper, we did not include the experiments for tuning the monitoring period  $p$  because of space limitation.

In summary, the average performance metrics of the PB fault-tolerant approaches in response to the step fault rate profile are listed in Table 2. From the table, we can see that adaptive PB fault-tolerant approaches offer the best average performance metrics ( $VAL \approx 1$ ,  $UTIL \approx 0.75$ ,  $SR \approx 0.75$ ) in response to the step fault rate profile. The PB-EXCL offers an average processor utilization efficiency ( $UTIL \approx 0.75$ ), and average schedulability-reliability index ( $SR \approx 0.62$ ) higher than that of PB-CONCUR in response to the step fault rate profile.

**6.2.2.2. Ramp fault rate profile.** Fig. 14(b) shows the effect on  $SR(t)$  for the four PB fault-tolerant approaches in response to the ramp fault rate  $RFR(FR_{0\%}, FR_{100\%}, 1000)$ . The ramp fault rate was increased from  $FR_{0\%}$  to  $FR_{100\%}$  during 1000 time units.

From the figure, it can be seen that the PB-CONCUR offers a constant  $SR(t)$  index which does not change with  $t$ . The PB-EXCL offers  $SR(t)$  that decreases quadratically from one to 0.25 as  $t$  varies from 0 to 1000. This is because the primary fault probability increases linearly from zero to one as  $t$  varies from 0 to 1000. The PB-OVER-CONT offers  $SR(t)$  that decreases quadratically from one to approximately 0.433 as  $t$  varies from 0 to approximately 694, and increases quadratically from 0.433 to 0.5 as  $t$  varies from 694 to 1000. This is because the overlap interval between the primary and the backup versions of tasks will vary from no overlap to full overlap as  $t$  varies from 0 to 1000. The PB-OVER-SWITCH behaves like PB-EXCL when  $t \leq 414$ , and like PB-CONCUR when  $t > 414$ . This occurs because

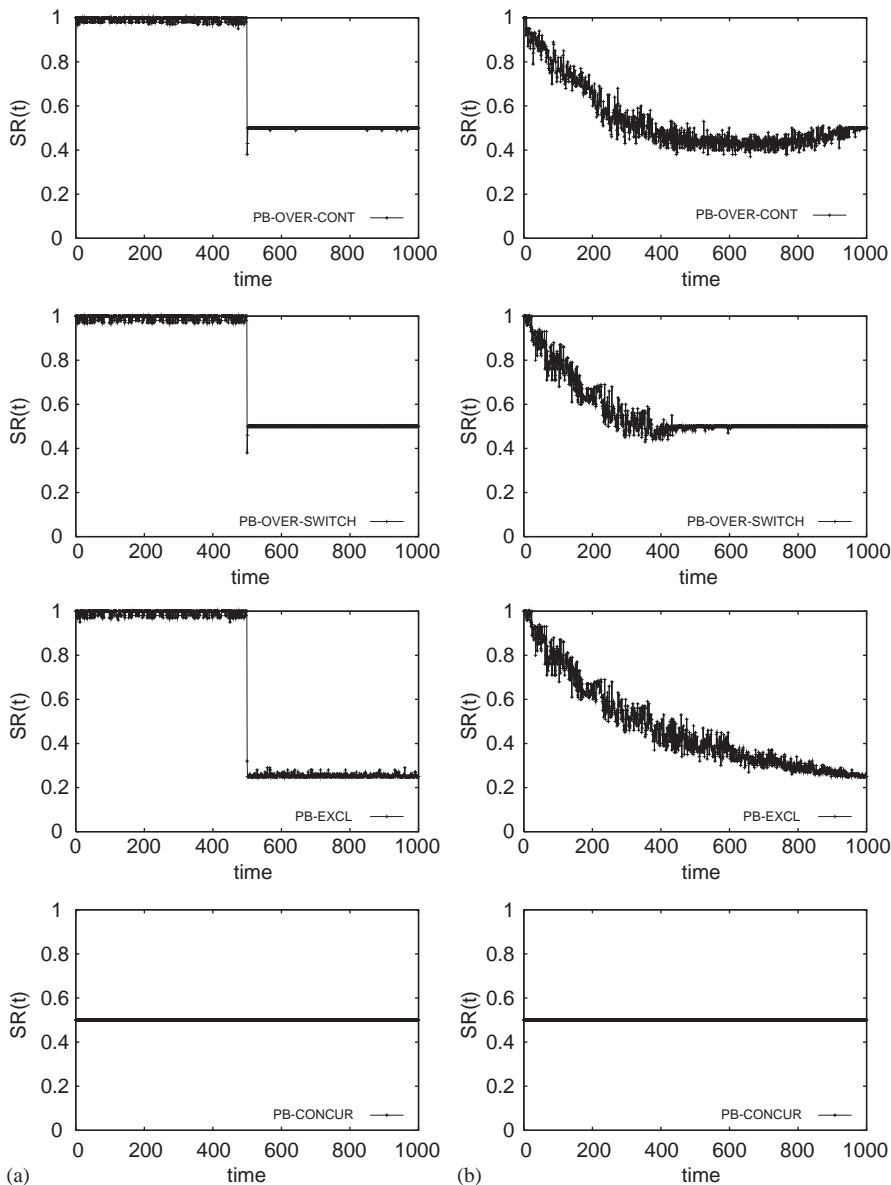


Fig. 14.  $SR(t)$  of the PB approaches in response to dynamic fault rate ( $d^s = c_i$ , and  $d^f = 2c_i$ ): (a) step fault-rate profile and (b) ramp fault-rate profile.

Table 2  
The average performance metrics of the PB approaches in response to dynamic fault rate

Approach	Step fault rate profile			Ramp fault rate profile		
	VAL	UTIL	SR	VAL	UTIL	SR
PB-EXCL	0.747	0.747	0.620	0.694	0.694	0.496
PB-CONCUR	1.000	0.500	0.500	1.000	0.500	0.500
PB-OVER-CONT	0.994	0.745	0.744	0.867	0.633	0.551
PB-OVER-SWITCH	0.995	0.745	0.7461	0.931	0.638	0.583

the primary fault probability  $f$  is greater than the threshold value ( $f_0 = \sqrt{2} - 1$ ) when  $t > 414$ .

In summary, the average performance metrics of the PB fault-tolerant approaches in response to the ramp fault rate

profile are listed in Table 2. From the table, we can see that PB-OVER-SWITCH approach offers the best average schedulability–reliability index ( $SR \simeq 0.58$ ) in response to the ramp fault rate profile. The PB-EXCL offers the best average processor utilization efficiency ( $UTIL \simeq 0.69$ ) in response to the ramp fault rate profile. The PB-CONCUR offers the best average output value ( $VAL \simeq 1$ ) in response to the ramp fault rate profile.

### 7. Conclusions

In this paper, we have considered the problem of scheduling of real-time tasks with PB-based fault-tolerant require-

ments in multiprocessor systems. We have proposed an adaptive fault-tolerant scheduling scheme for the problem. The scheme has a mechanism to control the overlap interval between the primary and backup versions of tasks in the schedule, in order to improve the overall performance of the system. The overlap interval is computed based on the estimated value of primary fault probability in the system and task's soft laxity. Two variants (PB-OVER-CONT and PB-OVER-SWITCH) of the adaptive scheme have been proposed and studied.

In PB-OVER-CONT, the overlap interval varies from no overlap to full overlap in a continuous manner as the fault probability varies from 0 to 1. In PB-OVER-SWITCH, the scheduler uses a threshold value of fault probability to switch from PB-CONCUR to PB-EXCL. This threshold value is adapted with task's soft laxity. We have also proposed a new metric, called schedulability–reliability (*SR*) index and conducted the following analytical and simulation studies:

- Analytical studies to quantify the effect of primary fault probability on three performance metrics, *viz.*, processor utilization efficiency, task value (utility), and *SR* index.
- Simulation studies to validate the above analytical results using the analytical assumptions.
- Simulation studies to quantify the effect of task's soft laxity in the performance of the PB-based fault-tolerant approaches.
- Simulation experiments to study the instantaneous behavior of  $SR(t)$  in response to dynamically changing fault rates. In particular, we have studied this behavior for step and ramp fault rate profiles. Our studies show that both the variants of the adaptive scheme exhibit a similar behavior for step fault rate profile, while for the ramp fault rate profile, PB-OVER-SWITCH performs better than PB-OVER-CONT.

In summary, our studies show that the proposed PB-OVER-SWITCH adaptive scheme always performs better than its adaptive and non-adaptive counterpart for the *SR* index metric.

Note that, extending the proposed schemes to periodic tasks with non-preemptive scheduling is straightforward as the primary and backup copies of each instance of periodic tasks have to be scheduled based on the estimated overlap interval. However, extending the scheme to preemptive task model in a multiprocessor system involves dealing with such issues as synchronization between primary and backup executions, which is a good candidate for further work.

## References

[1] M. Caccamo, G. Buttazzo, Optimal scheduling for fault-tolerant and firm real-time systems, in: Proceedings of the IEEE International Conference on Real-Time Computing Systems and Applications, 1998.

- [2] L. Chen, A. Avizienis, N-version programming: a fault tolerant approach to reliability of software operation, in: Proceedings of the IEEE Fault-Tolerant Computing Symposium, 1978, pp. 3–9.
- [3] S. Ghosh, R. Melhem, D. Mosse, Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems, *IEEE Trans. Parallel Distributed Systems* 8 (3) (March 1997) 272–284.
- [4] O. González, H. Shrikumar, J.A. Stankovic, K. Ramamritham, Adaptive fault-tolerance and graceful degradation under dynamic hard real-time scheduling, in: Proceedings of the IEEE Real-Time Systems Symposium, 1997.
- [5] I. Gupta, G. Manimaran, C. Siva Ram Murthy, Primary backup based fault-tolerant dynamic scheduling of object-based tasks in multiprocessor real-time systems, in: D.R. Avresky (Ed.), Dependable Network Computing, Kluwer Academic Publishers, Dordrecht, 1999.
- [6] C.M. Krishna, K.G. Shin, On scheduling tasks with quick recovery from failure, *IEEE Trans. Comput.* 35 (5) (May 1986) 448–455.
- [8] C. Lu, J. Stankovic, T. Abdelzaker, G. Tao, S. Son, M. Marley, Performance specifications and metrics for adaptive real-time systems, in: Proceedings of the Real-Time Systems Symposium, December 2000.
- [9] G. Manimaran, C. Siva Ram Murthy, A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis, *IEEE Trans. Parallel Distributed Systems* 9 (11) (November 1998) 1137–1152.
- [10] R.A. Omari, A.K. Somani, G. Manimaran, An adaptive scheme for fault-tolerant scheduling of soft real-time tasks in multiprocessor systems, in: Proceedings of the International Conference on High Performance Computing (HiPC), December 2001.
- [11] B. Randell, System structure for software fault-tolerance, *IEEE Trans. Software Eng.* 1 (2) (June 1975) 220–232.
- [12] K.G. Shin, P. Ramanathan, Real-time computing: a new discipline of computer science and engineering, *Proc. IEEE* 82 (1) (January 1994) 6–24.
- [13] J.A. Stankovic, K. Ramamritham, The Spring kernel: a new paradigm for real-time operating systems, *ACM SIGOPS Operating Systems Rev.* 23 (2) (January 1995) 77–83.
- [14] S. Tridandapani, A.K. Somani, U. Reddy, Low overhead multiprocessor allocation strategies exploiting system spare capacity for fault detection and location, *IEEE Trans. Comput.* 44 (7) (July 1995) 865–877.
- [15] F. Wang, K. Ramamritham, J.A. Stankovic, Determining redundancy levels for fault tolerant real-time systems, *IEEE Trans. Comput.* 44 (2) (February 1995) 292–301.



**Raéd Al-Omari** is an Advisory/Scientist Engineer in the System and Technology Group at IBM Inc, since July 2001. He received his BS degree in Electrical Engineering and MS degree in Computer Engineering from Jordan University of Science and Technology (JUST), in 1994, and 1997 respectively. He received his PhD degree in computer engineering from Iowa State University (ISU), in 2001. His research expertise is in the areas of Fault-tolerance, real-time systems, parallel and distributed systems, computer architecture, and hardware tracing in SMP. He is recipient of a distinction academic award from JUST in 19991-1992 and research excellence award from ISU in 2001. His biography is featured in the 25th edition of the National dean's list for 2002, and in the 57th and 58th Editions of Who's Who in America. His work in IBM now includes Hardware development for future UNIX system microprocessors.



**Arun K. Somani** is currently Jerry R. Junkins Endowed Chair Professor of Electrical and Computer Engineering at Iowa State University where he first served as David C. Nicholas Professor during 1997-2002. He earned his MSEE and PhD degrees in electrical engineering from the McGill University, Montreal, Canada, in 1983 and 1985, respectively. He worked as Scientific Officer for Govt. of India, New Delhi from 1974 to 1982 and as a faculty member at the University of Washington, Seattle, WA from 1985 to 1997. Professor Somani's

research interests are in the area of fault tolerant computing, computer interconnection networks, WDM-based optical networking, wireless communication, computer architecture, and parallel computer systems. He has taught courses in these areas and published more than 200 technical papers and has graduated more than 60 MS and 17 PhD students and currently supervising 11 graduate students. He is the chief architect of anti-submarine warfare system (developed for Indian navy), Proteus multicomputer system (developed for US coastal navy), and Meshkin fault-tolerant computer system (developed for the Boeing Company). He has been elected a Fellow of IEEE for his contributions to theory and applications of computer networks. He has served on several program committees of various conferences in his research areas was the General Chair of IEEE Fault Tolerant Computing Symposium - 1997 and Technical Program Committee Chair of International Conference on Computer Communications and Networks, 1999, and OPTICOMM 2003.



**G. Manimaran** is currently an Assistant Professor in the Department of Electrical and Computer Engineering at Iowa State University, since spring 1999. He received his Ph.D degree in Computer Science and Engineering from IIT Madras, India, in 1998. His research expertise is in the areas resource management in real-time systems, QoS routing and multicasting, and Internet infrastructure security. He has co-authored about 100 peer-reviewed research publications in international journals and conferences/workshops, of which

two conference/workshop papers received the best paper awards. He is a co-author of the text "Resource management in real-time systems and networks," MIT Press, 2001. He has served or as guest co-editor for special issues in IEEE Network (Jan/Feb. 2003), Journal of Systems and Software (2005), and Journal of High Speed Networks (2005). He has served as Program co-chair of the International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS) 2003 and its General co-chair in 2004. He is a founding co-chair of the Trusted Internet Workshop held in conjunction with HiPC. He has served as a member of technical program committees of several IEEE conferences. He is a member of the IEEE, IEEE Computer and Communication Societies, and ACM.