Exploiting Redundancies to Enhance Schedulability in Fault-Tolerant and Real-Time Distributed Systems

Wei Luo, Xiao Qin, *Member, IEEE*, Xian-Chun Tan, Ke Qin, and Adam Manzanares

***Abstract*—In the past decades, distributed systems have been widely applied to real-time applications, most of which have fault-tolerance requirements to assure high reliability. Due to the stringent space constraints of real-time systems, the issue of schedulability becomes a major concern in the design of fault-tolerant and real-time distributed systems. Most existing real-time and fault-tolerant scheduling algorithms, which are based on the primary–backup scheme for periodic real-time tasks, introduce unnecessary redundancies by aggressively using active-backup copies. To solve this problem, we propose two novel fault-tolerant techniques, which are seamlessly integrated with fixed-priority-based scheduling algorithms. These techniques leverage redundancies to enhance schedulability in fault-tolerant and real-time distributed systems. Our fault-tolerant techniques make use of the primary–backup scheme to tolerate permanent hardware failures. The first technique (referred to as Tercos) terminates the execution of active-backup copies, when corresponding primary copies are successfully completed. Tercos is designed to reduce scheduling lengths in fault-free scenarios to enhance schedulability by virtue of executing portions of active-backup copies in passive forms. The second technique (referred to as Debus) uses a deferred-active-backup scheme to further minimize schedule lengths to improve the schedulability performance. Debus schedules active-backup copies as late as possible, while terminating active-backup copies when their primary copies are completed. Experimental results show that, compared with existing algorithms in literature, Tercos can significantly improve schedulability by up to 17.0% (with an average of 9.7%). Furthermore, empirical results reveal that Debus can enhance schedulability over Tercos by up to 12% (with an average of 7.8%).**

***Index Terms*—Distributed systems, fault tolerance, rate-monotonic (RM) algorithm, real-time task scheduling, primary-backup copy.**

I. Introduction

With the ever-increasing reliance on distributed systems for a variety of real-time applications like avionics, automated manufacturing, and nuclear plant control systems, it is imperative to develop dependable distributed systems delivering an array of real-time services. In real-time distributed systems, failing to produce correct results in a timely manner may cause catastrophic consequences. Hardware and software can cause failures in an unpredictable way in distributed systems; therefore, it is of paramount importance to provide fault tolerance for real-time distributed systems.

A large number of real-time distributed applications are comprised of a set of periodic tasks running on an array of computational nodes or processors. Scheduling periodic tasks as to guarantee that their deadlines are met is a challenging research issue. Real-time distributed systems may have some extra constraints, e.g., stringent space and weight constraints. Reducing the space and weight of an avionics system can conserve power consumption and thus exemplifies the space and weight constraints. Consequently, it is desirable to minimize the number of necessary processors used to execute real-time tasks without violating deadlines.

The primary–backup scheme plays an important role in achieving fault tolerance in real-time distributed systems. In this approach, each task has two versions allocated to two different processors. An acceptance test is employed to check the correctness of schedules [17]–[19], [21], [22]. There are three variants of the primary–backup scheme: 1) the active-backup copy [23], [24]; 2) passive-backup copy [17]–[19], [21], [22]; and 3) primary–backup copy overlapping techniques [25], [26]. In the active-backup copy schemes, if a primary copy (or a backup copy) finishes before its corresponding backup copy (or primary copy), the backup copy (or the primary copy) will not be terminated and deallocated, thereby leading to wasted processor time. To overcome this problem, Tatsuhiro *et al.* proposed a technique to reduce redundancies by employing active-backup copies [26]. The task model used in their study is constructed for aperiodic and nonpreemptive tasks; thus, their approach is inadequate for periodic and preemptive tasks.

In this paper, we address the redundancy problems introduced by active-backup copies of periodic and preemptive tasks. It is challenging to tackle this problem in the context of periodic and preemptive tasks running in distributed systems, because tasks may be preempted by high-priority tasks, resulting in different response times in different instances of a task.

The major contribution of this paper includes two novel real-time fault-tolerant techniques integrated with fixed-priority scheduling algorithms to exploit redundancies for enhancing schedulability in fault-tolerant and real-time distributed systems. The primary–backup approach is employed by both scheduling techniques to tolerate permanent processor failures.

Manuscript received May 4, 2007; revised January 22, 2008. First published February 20, 2009; current version published April 17, 2009. This work was supported in part by the U.S. National Science Foundation under Grants CCF-0742187, CNS-0757778, CNS-0831502, OCI-0753305, DUE-0621307, and DUE-0830831, by the National Natural Science Foundation of China under Project Grant 60573125, and in part by Auburn University under a startup grant. This paper was recommended by Associate Editor B. Veeravalli.

W. Luo, X.-C. Tan, and K. Qin are with the Department of Information System, China Ship Development and Design Center, Wuhan 430064, China (e-mail: free_xingezi@ 163.com).

X. Qin and A. Manzanares are with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849 USA (e-mail: xqin@auburn.edu; acm0008@auburn.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCA.2009.2013192

The first technique (referred to as Tercos) terminates the execution of active-backup copies when their primary copies are successfully completed. Tercos can reduce schedule lengths in fault-free scenarios to enhance schedulability by executing active-backup copies in passive forms. Although Tercos aims at reducing the redundancies of active-backup copies, redundant active-backup copies may be aggressively executed in parallel with their corresponding primary copies. To further improve the performance of real-time distributed systems in terms of schedulability, we developed a second scheduling technique (referred to as Debus), which makes use of a deferred-active-backup scheme. Unlike the Tercos scheme, Debus schedules passive-backup copies as late as possible while terminating the execution of active-backup copies when corresponding primary copies are completed. The major difference between Tercos and Debus is that Tercos is a passive way of eliminating redundancies of backup copies, while Debus is a proactive approach that takes advantage of deferring the execution of backup copies.

This paper is organized as follows. Related works are briefly discussed in Section II. Section III presents a system model and some assumptions. The Tercos and Debus techniques are described in Sections IV and V, respectively. Simulation experiments and the performance analysis are presented in Section VI. Finally, Section VII concludes this paper by summarizing the main contributions of this paper and commenting on the future directions of this work.

II. Related Work

In the past two decades, various scheduling algorithms have been proposed to support real-time systems. For example, many scheduling algorithms were designed and implemented to schedule periodic real-time tasks running in uniprocessor or multiprocessor systems. Liu and Layland developed the well-known rate-monotonic scheduling (RMS) algorithm for preemptively scheduling periodic task sets running on a single processor [1]. Liu and Layland derived a least upper bound of processor utilization to check the schedulability of a set of periodic tasks on a single processor [1]. Joseph and Pandya investigated the completion time test (CTT) for checking the schedulability of a set of fixed-priority tasks on a single processor. The CTT is stronger than the previous schedulability test method proposed by Liu and Layland [2].

It is worth noting that the RM algorithm is becoming an industrial standard because of its simplicity, and it is relatively straightforward to implement. Dhall and Liu proposed the RM first-fit (RMFF) algorithm, which is a major extension of the RM algorithm [3]. RMFF can be used to generate real-time schedules for multiprocessor systems. Moreover, Burchard *et al.* addressed the issue of assigning real-time tasks to multiprocessor systems [4].

Studies of real-time distributed systems reveal a number of challenges, including load balancing [13], resource management [14], [36], security [15], [33], availability [34], cooperative systems [35], scheduling mechanisms [37], and fault tolerance [18]–[22]. Fault tolerance is an inherent requirement of modern real-time systems, which can be implemented in both hardware and software. Multiple replicas of real-time tasks can be executed on different hardware components to achieve fault tolerance [5]. Fault tolerance can be implemented by N-version programming or recovery blocks [6]. In addition, hardware- and software-based fault-tolerant techniques can be integrated to provide hybrid fault-tolerant approaches for real-time distributed systems [7].

Fault-tolerant scheduling is an attractive avenue to achieving high reliability in uniprocessor and multiprocessor real-time systems. One of the first fault-tolerant scheduling mechanisms for uniprocessor real-time systems was developed by Liestman and Campbell [8]. Their algorithm can generate optimal schedules where passive replications are employed to tolerate software failures. Ghosh *et al.* [9] proposed an algorithm that considers the re-execution of faulty tasks to tolerate transient faults, and the algorithm is based on the RMS priority assignment policy [1]. An earliest deadline first (EDF)-based scheduling approach, which takes the effects of transient faults into account, was designed and implemented by Liberato *et al.* [10]. Their main idea is to simulate the EDF scheduler and to use slack times for executing task recoveries provided that fault patterns, or the maximum number of faults per task, are known *a priori*. Another EDF-based scheduling approach to supporting fault-tolerant systems was investigated by Caccamo and Buttazzo [11]. Their task model consists of instance skippable and fault-tolerant tasks. Primary jobs are scheduled online to provide high-quality service, while backup jobs are scheduled offline to provide acceptable services. Lima and Burns proposed an appropriate schedulability analysis using response time analysis [12]. An optimal priority assignment algorithm can be used in combination with their schedulability analysis to facilitate system fault resilience [13]. Some researchers also investigated the power management issues in the fault-tolerant real-time systems [31].

Fault-tolerant scheduling is an efficient way of achieving fault tolerance in real-time distributed systems. Krishna and Shin proposed a dynamic programming algorithm for multiprocessors [16]; in their algorithm, backup copies are ensured to be efficiently embedded with primary schedules. Scheduling algorithms along with fault-tolerant scheduling algorithms designed for real-time distributed systems fall into two major camps: static [17]–[19] and dynamic scheduling [20]–[22]. Many fault-tolerant scheduling algorithms leverage the primary–backup scheme to tolerate processor failures. In the primary–backup approach, each task has two versions allocated to two different processors. The acceptance test is a widely adopted approach to checking the correctness of schedules [17]–[19], [21], [22]. Three variants of the primary–backup approach include: 1) the active-backup-copy-based schemes [23], [24]; 2) the passive-backup-copy-based schemes [17]–[19], [21], [22]; and 3) the primary–backup-copy overloading techniques [25], [26].

In the active-backup-copy-based schemes, the primary and backup copies of each task are executed in parallel on two processors [23], [24]. For example, Bertossi *et al.* extended the well-known RMFF assignment algorithm. Their algorithm assigns RM priorities to all task copies, which are allocated to the first processor that can accommodate the task copies [27]. The active-backup-based schemes exhibit the advantages of requiring no synchronization between two copies and imposing no constraints on the execution times of tasks. However, it is recognized that processor times required by tasks in the active-backup-based approaches are doubled when compared with the passive-backup-based schemes. In contrast, the passive-backup-copy-based schemes only execute the backup copy of a task if its primary copy fails to pass the acceptance test [17]–[19], [21], [22]. The backup copy of a task can be deallocated from its schedule if the task's primary copy is successfully finished. More importantly, the passive-backup-copy-based schemes can take advantage of the *backup copy overloading* technique. This overloading technique allows passive-backup copies assigned to different processors to be overlapped on the same process to tolerate a single processor failure. However, the passive-backup-copy-based schemes have a shortcoming of tight timing constraints. The primary–backup-copy overlapping technique allows the primary copy and backup copies to be overlapped in execution times [25], [26]. This technique, which can exploit the advantages of the aforementioned two schemes, is envisioned as a compromise between the other two. Nevertheless, the common drawback of the aforementioned fault-tolerant scheduling schemes is that they merely support a single type of backup copy.

Much attention has been paid to system schedulability improvements in fault-tolerant distributed systems. Al-Omari *et al.* studied a schedulability enhancing technique (called primary–backup overloading), in which the primary copy of each task can overlap in time with the backup copy of another task on a processor. This technique inherently introduces redundancies due to active-backup copies [30]. Tatsuhiro *et al.* proposed an approach to reducing redundancies imposed by backup copies [26]. However, the task models used in their studies only address the issue of aperiodic and nonpreemptive tasks. To bridge the technology gap in fault-tolerant scheduling, in this study, we exploit the redundancies of active-backup copies to improve the schedulability of fault-tolerant distributed systems running periodic real-time tasks. In this paper, we make use of the dual priority scheme [28] to postpone the execution of active-backup copies by identifying spare capacities of RMS.

III. System Model

In this section, we describe a system model of real-time distributed systems. The system model is composed of a set of processors (the terms processors and computational nodes will be used interchangeably throughout this paper) as well as a set of real-time primary copy tasks along with a set of corresponding backup copies of the real-time tasks running in the distributed system. In this study, we consider real-time distributed systems where processors accessing their local memory modules are connected to one another via an interconnection network. Formally, a real-time distributed system model is composed of a set $\Gamma = \{\tau_1, \tau_2, \tau_3, \ldots, \tau_N\}$ of tasks in addition to a set $\Omega = \{P_1, P_2, \ldots, P_M\}$ of processors executing the task set. The ith periodic preemptive task τ_i is characterized by two parameters: period T_i and execution time C_i. A new instance of task τ_i is generated every T_i time units, and the jth instance of τ_i will be arriving at time $(j-1)T_i$. Without loss of generality, we assume that the deadline of each instance is at the end of its period, i.e., the jth instance must be completed before time jT_i. A periodic task is said to be *feasible* if all its instances can be transmitted before their corresponding deadlines. In this paper, we apply the RM algorithm to schedule tasks allocated to any processor in the distributed system.

A set $B\Gamma = \{\beta_1, \beta_2, \beta_3, \ldots, \beta_N\}$ of backup copies of periodic tasks are introduced to make fault tolerance possible. Note that $\beta_i = (D_i, T_i)(i = 1, 2, \ldots, N)$ is the backup copy with respect to the ith task τ_i. D_i is the execution time of β_i, and T_i denotes the period of β_i. We assume that the backup and primary copies of a task are identical, i.e., $D_i = C_i$. This assumption does not limit the applications of our approach to cases where backup and primary copies of a task are different. In our system model, each backup copy may be in one of two states: passive-backup copy or active-backup copy. We assign a task's primary copy before assigning its backup copy regardless of the backup copy's status form. The status forms of backup copies are formally determined by the following:

$$Status(\beta_i) = \begin{cases} \text{passive}, & B_i > D_i \\ \text{active}, & B_i \leq D_i \end{cases}$$

$$\text{where } P(\tau_i) = P_j,\ P(\beta_i) = P_k, \text{ and } B_i = T_i - R(i,j). \quad (1)$$

$R(i, j)$ in (1) denotes the worst case response time or WCRT of τ_i on processor P_j [4]. B_i denotes the recovery time for β_i, i.e., the time left after the execution of τ_i. $P(\tau_i)$ or $P(\beta i)$ denotes the processor on which τ_i or β_i is scheduled on. For ease of presentation, γ_i represents a primary copy or a backup copy, i.e., $\gamma_i = \tau_i$ or β_i. Equation (1) indicates that an active-backup copy must be running in parallel with its primary copy, whereas a passive-backup copy only needs to be executed if its primary copy fails.

The failure characteristics of processors are given in the following. Note that similar failure models can be found in the literature (see, for example, [18] and [27]).

1) Processors fail in a fail–stop manner. A processor has either ceased functioning or is operational, and a faulty processor cannot cause incorrect behavior in any nonfaulty processor.
2) All nonfaulty processors in a real-time distributed system can communicate with each other through message passing.
3) The failure of a processor P_f can be effectively detected by the remaining nonfaulty processors after the failure; the failure is determined before the closest task completion time of a task scheduled on P_f.

As for a periodic task with a primary copy τ_i and a corresponding passive-backup copy β_i, β_i is informed of the completion of τ_i at every occurrence of the periodic task by receiving a message from a processor running τ_i in the jth period (i.e., $[jT_i, (j+1)T_i]$) upon τ_i's completion time. This message can be very small, since the message usually contains the indexes of the primary task τ_i and the sender and receiver processors. In the worst case, when the message is not received by β_i within a certain due time, the passive back copy is immediately scheduled by assuming that a failure on the processor running τ_i occurs. In general, processor failure detections inevitably introduce marginal overheads due to the latency of delivering the message from τ_i to β_i. With the current off-the-shelf communication technology in place, such overhead can be measured on the order of a few microseconds. Therefore, this overhead can be easily included in the execution time of primary copies of periodic tasks. Except for the short message sent from τ_i to β_i, there is not any other form of synchronization between the primary copy τ_i and backup copy β_i.

A k-timely-fault-tolerant (k-TFT) schedule is defined as a schedule in which no task deadlines are missed, despite k arbitrary processor failures [32]. Our goal in this study is to generate 1-TFT schedules by employing processor and task redundancies in the scheduling algorithm. To achieve this goal, we focused on scenarios where only one processor may encounter failures at any instant of time, meaning that a second processor cannot fail before the first failed processor is recovered.

To facilitate the description of our algorithms, we introduce the following notation. Note that the similar notation was used in [27].

1) Sets $Primary(P_j)$ and $Backup(P_j)$ represent primary and backup copies assigned to processor P_j, i.e., $Primary(P_j) = \{\tau_i | P(\tau_i) = P_j\}$ and $Backup(P_j) = \{\beta_i | P(\beta_i) = P_j\}$.
2) A set $active(P_j)$ includes active-backup copies assigned to processor P_j, i.e., $active(P_j) = \{\beta_i | \beta_i \in Backup(P_j), Status(\beta i) = \text{active}\}$.
3) A set $passiveRecover(P_j, P_f)$ contains all the passive-backup copies that P_j must start scheduling when a failure of P_f is detected, i.e., $passiveRecover(P_j, P_f) = \{\beta_i \mid \beta_i \in Backup(P_j), \quad Status(\beta_i) = \text{passive}, P(\tau_i) = P_f\}$.
4) A set $activeRecover(P_j, P_f)$ denotes active copies assigned to P_j with primary copies assigned to P_f. This set contains all the active-backup copies that processor P_j must keep scheduling when P_f fails, i.e., $activeRecover(P_j, P_f) = \{\beta_i | \beta_i \in Backup(P_j), Status(\beta_i) = \text{active}, P(\tau_i) = P_f\}$.
5) $Recover(P_j, P_f)$ gives the union between sets $passiveRecover(P_j, P_f)$ and $activeRecover(P_j, P_f)$, i.e., $Recover(P_j, P_f) = passiveRecover(P_j, P_f) \cup activeRecover(P_j, P_f)$.

IV. TERCOS STRATEGY

A. Motivation

Existing algorithms like fault-tolerant RMFF (FTRMFF) make use of the WCRT analysis to check the schedulability of a task set and to determine the status of backup copies [27]. Although FTRMFF strives to schedule as many backup copies as possible to be executed in passive forms, active-backup copies still exhibit considerable unnecessary task redundancies when primary copies are successfully finished prior to their backup copies. For example, consider four real-time periodic tasks along with their backup copies, e.g., $\tau_1 = \beta_1 = (2, 4)$, $\tau_2 = \beta_2 = (2, 5)$, $\tau_3 = \beta_3 = (5, 9)$, and $\tau_4 = \beta_4 = (3, 15)$.

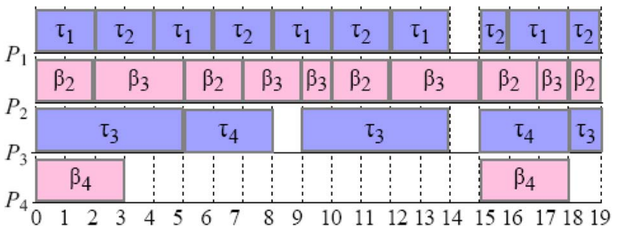

Fig. 1. Illustration of a redundancy imposed by an active-backup copy.

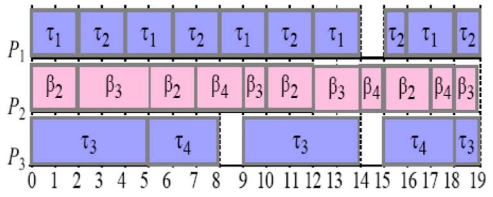
P_1
P_2
P_3
τ_1 τ_2 β_2 β_3 β_4 τ_3 τ_4
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Fig. 2. Segmentation of active-backup copies in a fault-free scenario.

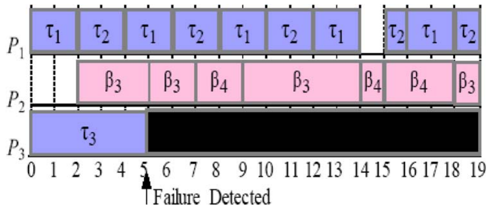

Fig. 3. Segmentation of active-backup copies when a failure occurs.

The FTRMFF algorithm generates a schedule where the primary and backup copies are allocated to nodes in a real-time distributed system as follows: $Primary(P_1) = \{\tau_1, \tau_2\}$, $backup(P_1) = \{\Phi\}$, $Primary(P_2) = \{\Phi\}$, $backup(P_2) = \{\beta_1, \beta_2, \beta_3\}$, $Primary(P_3) = \{\tau_3, \tau_4\}$, $backup(P_3) = \{\Phi\}$, $Primary(P_4) = \{\Phi\}$, $backup(P_4) = \{\beta_4\}$, and $Status(\beta_1) =$ passive, $Status(\beta_2) =$ active, $Status(\beta_3) =$ active, and $Status(\beta_4) =$ active. Fig. 1 shows a task schedule in a time interval between time 0 and 19.

It should be noted from Fig. 1 that, in a fault-free scenario, task τ_3 finishes its execution prior to its backup β_3. Specifically, at time slot 5, τ_3 has already been completed, whereas β_3 only has been running for three time slots. Nevertheless, β_3 continues executing despite the successful completion of τ_3, thereby wasting two time slots. The unnecessary redundancy introduced by β_3 can be eliminated by deallocating β_3 from its node immediately after the execution of τ_3. Fig. 2 shows that the schedulability of the distributed system can be improved by taking advantage of the free time slots left for such a deallocation. Only in the case where processor P_3 fails (e.g., at time slot 5), the remaining part of β_3 continues its execution without the deallocation (see Fig. 3). As a result, the *Tercos* strategy can reduce the number of processors needed to schedule the whole task set without adversely affecting the real-time and fault-tolerant constraints of the distributed system. In other words, *Tercos* improves schedulability over the FTRMFF algorithm.

B. Description of Tercos

Fig. 4 shows that the WCRT of τ_i is $R(i, 1)$ and the WCRT of β_i is $BR(i, 2)$. Note that $BR(i, 2)$ is larger than $R(i, 1)$,

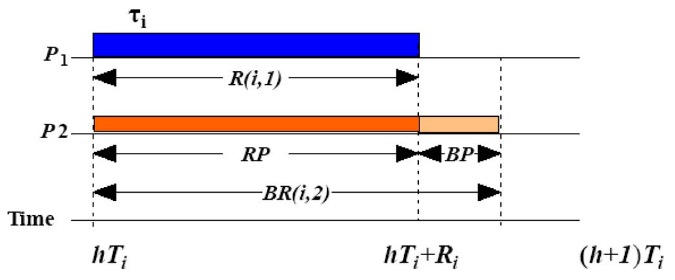

Fig. 4. Illustration of Tercos technique.

because in *Tercos*, the WCRTs of primary copies are always less than or equal to the WCRTs of the corresponding backup copies (see Section IV-C). The execution of β_i is comprised of two parts: a redundant part (referred to as *RP*) and a backup part (referred to as *BP*). The RP of a backup copy executes concurrently with its primary copy, while the BP of a backup copy only executes when its primary copy encounters failures. The BP will be deallocated when the primary copy successfully completes its execution. Thus, we can execute the BP of active-backup copies in passive forms, which can take advantage of backup copy deallocations and overlapping techniques to enhance system schedulability. However, the implementation of *Tercos* is somewhat nontrivial for two reasons. First, the *RP* of β_i can be preempted by other tasks with higher priorities, and therefore, we must take high-priority tasks into account while scheduling the RP. Second, deallocating the *BP* of β_i implies that the BP must be executed under the condition that τ_i encounters a failure, which in turn, increases the processor load of P_2 when other processors fail.

Hereafter, we use $Redundant(\beta_i)$ and $Backup(\beta_i)$ to denote the execution time of the *RP* and *BP* of β_i, respectively. Thus, $Redundant(\beta_i) = C_i - Backup(\beta_i)$. Note that all the task copies are assigned following the decreasing order of RM priorities (see Section IV-C). $Redundant(\beta_i)$ is equal to free time slots in the time interval $[0, R(i,1)]$ before assigning β_i. Suppose that $Idle(t)$ is the amount of free time that is in a period of time [0, t]; then, $Redundant(\beta_i) = Idle(R(i,1))$. $Idle(R(i,1))$ can be computed by the following two steps. First, we add a virtual task $\overline{\tau} = (\overline{T} = R(i,1), \overline{C})$ with the lowest priority among all the tasks that have been allocated to processor P_2. Second, we calculate the maximal value of $\overline{C}$ such that task $\overline{\gamma}$ meets its deadline. Thus, the value of $Idle(t)$ can be determined using

$$Idle\,(t) = \max\{\overline{C} | \overline{\gamma} \text{ is schedule}\}. \tag{2}$$

The finishing time t of task $\overline{\gamma}$ is given by

$$t = \overline{C} + \sum_{\gamma_j \in hep(\overline{\gamma})}^{\gamma_j \text{ is Primary Copy}} \left\lceil \frac{t}{T_j} \right\rceil * C_i + \sum_{\gamma_j \in hep(\overline{\gamma})}^{status(\gamma_j) = \text{active}} \left\lceil \frac{t}{T_j} \right\rceil * Redundant(\gamma_j). \tag{3}$$

A schedulability test can be carried out using the finishing time t computed by (3). Thus, if t is less than or equal to $R(i,1)$, then $\overline{\gamma}$ is schedulable. Otherwise, $\overline{\gamma}$ is not schedulable. It is to be noted that we have to consider the interference of task γ_j on itself. Hereinafter, we denote the set of tasks with a higher or equal priority than that of task γ_j by $hep(\gamma_j)$. Equation (3) can be derived by a recurrence on the value t. The range of values to check from 0 to t can be computed by a bisection search (Please refer to [29] for effective upper and lower bounds). Using the technique, we can easily obtain $Redundant(\beta_i) = Idle(R(i,1))$.

C. Task Allocation Strategy

Now, we present our task allocation strategy. Although a similar allocation strategy can be found in [27], a major difference is present in the schedulability test for tasks allocated to a processor. We present three task assignment conditions followed by the main scheduling heuristic policies.

In order to assign a task copy to a processor, the Tercos algorithm deals with two cases for each task copy, depending on its status form (i.e., primary, active backup, or passive backup). In the first scenario, no processor failure occurs, and in the second case, one processor fails during the course of task executions. In each scenario, Tercos has to test the schedulability of a task set on a single processor by means of the Tercos fault-tolerant CTT (TFT-CTT).

1) Primary task copy. To allocate a primary task τ_i to P_j.
   a) Fault-free case: The task set $\sigma = Primary(P_j) \cup active(P_j) \cup \{\tau_i\}$ must be schedulable.
   b) Possible failure of another processor P_f: The task set $\sigma = Primary(P_j) \cup Recover(P_j, P_f) \cup \{\tau_i\}$ must be schedulable as well.
2) Active-backup copy. To allocate an active-backup copy β_i to P_j, assuming that the primary copy τ_i is assigned to processor P_f.
   a) Fault-free case: The task set $\sigma = Primary(P_j) \cup active(P_j) \cup \{\beta_i\}$ must be schedulable.
   b) Processor P_f fails: The task set $\sigma = Primary(P_j) \cup Recover(P_j, P_f) \cup \{\beta_i\}$ must be schedulable. Note that, if any processor other than P_f is fault free and P_f fails, then β_i can be deallocated without being executed.
3) Passive-backup copy. To allocate a passive-backup task β_i to P_j, assuming that the primary copy τ_i is assigned to processor P_f. Because the passive-backup copy is executed only when P_f fails, there is only one case to be considered.
   a) Processor P_f fails: The task set $\sigma = Primary(P_j) \cup Recover(P_j, P_f) \cup \{\beta_i\}$ must be schedulable.

Before task copies are allocated, both primary and backup copies are sorted by an increasing order of periods (Note that the priority of a copy is equal to the inverse of its period). A tie between primary copy τ_i and its backup copy β_i is broken by giving a higher priority to β_i. Without loss of generality, each task copy has an initial priority at a lower level and a unique promoted priority i, where $1 \leq i \leq 2n$, at an upper level (Note that τ_1 has the highest priority 1 and β_N has the lowest priority $2n$). Thus, tasks are assigned to processors in accordance to the following order:

$$\tau_1, \beta_1, \tau_2, \beta_2, \ldots, \tau_N, \beta_N. \tag{4}$$

Assigning task copies in the decreasing order of priorities greatly simplifies the implementation of our algorithms, because tasks assigned later will not affect the WCRTs of tasks previously allocated. It is very convenient to determine the status form of a backup copy because the WCRT of its primary copy is unchanged during the process of task allocation and scheduling.

We apply the "best-fit" policy to assign and schedule tasks to processors; thus, each task is assigned to the "best" processor that can accommodate it. If the tasks cannot fit in any processor, a new extra processor will be added. It is worth noting that the best processors have different meanings for primary and backup copies. The three scheduling policies are listed as follows.

1) To assign a primary task copy to a processor on which WCRT is the smallest. This policy has the goal of reserving long recovery times for backup copies.
2) To assign a backup task copy to a processor on which it can be executed in the passive form.
3) If the backup task copy has to be executed in the active form, the backup copy is allocated to a processor on which $Redundant(\beta_i)$ is minimized.

D. Schedulability Tests

Joseph and Pandya derived a necessary and sufficient schedulability criterion as follows [4].

Theorem 1: Given a set $\Gamma = \{\tau_1, \tau_2, \tau_3, \ldots, \tau_N\}$ of tasks that are sorted in decreasing order of priorities (the earlier the deadline of a task, the higher the priority of the task), we can calculate the WCRT R_i of the ith task using

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \tag{5}$$

where C_i is the execution time of task τ_i. If the response time R_i of task τ_i is smaller than the deadline T_i of τ_i, i.e., $R_i \leq T_i$, then all periodic instances of task τ_i will have their deadlines guaranteed under all task phasings. This schedulability is called CTT.

Although Theorem 1 provides a means of efficient schedulability analysis, it is based on the conventional RMS model. It is believed that the existing schedulability analysis is not applicable to the task model constructed in our study. Therefore, we extend the CTT to the TFT-CTT based on our fault-tolerant model.

To facilitate the description of the following theorems, we introduce an important notation. $\gamma_{\max c}$ is a task copy to be currently assigned. It is intuitive that the priority of $\gamma_{\max c}$ is lower when compared to task copies that have been allocated and scheduled. $hp(\gamma_{\max c})$ represents a set of tasks whose priorities are higher than that of $\gamma_{\max c}$ on a processor where $\gamma_{\max c}$ is about to be allocated. Similarly, $hep(\gamma_{\max c})$ represents a set of tasks with priorities higher than or equal to that of $\gamma_{\max c}$ on a processor where $\gamma_{\max c}$ will be assigned, i.e., $hep(\gamma_{\max c}) = hp(\gamma_{\max c}) \cup \{\gamma_{\max c}\}$.

Theorem 2: Suppose that $\gamma_{\max c}$ is a primary copy that is currently being assigned. $hp(\gamma_{\max c})$ is a set of tasks with higher priorities than $\gamma_{\max c}$ on processor P_k. In the case of no faulty processors, the WCRT of $\gamma_{\max c}$, $R_{\mathrm{FR}}(\gamma_{\max c})$, is calculated by

$$\begin{aligned} R_{\mathrm{FR}}(\gamma_{\max c}) = & \sum_{\tau_i \in hep(\gamma_{\max c})} \left\lceil \frac{R_{\mathrm{FR}}(\gamma_{\max c})}{T_i} \right\rceil * C_i \\ & + \sum_{\beta_i \in hep(\gamma_{\max c})}^{status(\beta_i)=\text{active}} \left\lceil \frac{R_{\mathrm{FR}}(\gamma_{\max c})}{T_i} \right\rceil \\ & * Redundant(\beta_i), \\ & \text{if } \gamma_{\max c} \text{ is a primary copy.} \end{aligned} \tag{6}$$

If $R_{\mathrm{FR}}(\gamma_{\max c})$ is smaller than or equal to $T_{\max c}$, then $\gamma_{\max c}$ is schedulable in fault-free scenarios. Otherwise, $\gamma_{\max c}$ is unschedulable. This schedulability test is called *Tercos* fault-free fault-tolerant CTT (or TFR-FTCTT for short).

Proof: Recall that all task copies are allocated and scheduled in the decreasing order of RM priorities. Thus, all the tasks in task set $hp(\gamma_{\max c})$ have already been allocated and are schedulable. To prove that $\gamma_{\max c}$ is schedulable, we simply need to show that $R_{\mathrm{FR}}(\gamma_{\max c})$ is smaller than $\gamma_{\max c}$'s deadline. As per Theorem 1, $\gamma_{\max c}$ is the task copy with the longest response time, and its WCRT is equal to the workload of $hep(\gamma_{\max c})$ during the time interval $R_{\mathrm{FR}}(\gamma_{\max c})$. In the case of a fault-free scenario, $hep(\gamma_{\max c})$ contains primary copies and active-backup copies allocated to processor P_k, i.e., $hep(\gamma_{\max c}) = Primary(P_k) \cup active(P_k) \cup \{\gamma_{\max c}\}$. To derive the value of $R_{\mathrm{FR}}(\gamma_{\max c})$, we have to consider the following two cases.

Case 1) Each primary copy τ_i is executed once during each period T_i, meaning that τ_i will execute $\lceil t/T_i \rceil$ times during time interval [0, t]. The total workload of a primary copy on processor P_k is $\lceil t/T_i \rceil^* Ci$, and therefore, the workload contributed by primary copies to $R_{\mathrm{FR}}(\gamma_{\max c})$ is expressed by the first item on the right-hand side of (6).

Case 2) Each active-backup copy is executed in a similar way as its primary copy counterpart. An active-backup copy is an RP of its primary copy. Hence, the workload contribution of active-backup copies to $R_{\mathrm{FR}}(\gamma_{\max c})$ is computed by the second item on the right-hand side of (6).

Considering the previous two cases, we can derive the value of $R_{\mathrm{FR}}(\gamma_{\max c})$ using (6). If $\gamma_{\max c}$ is an active-backup copy (see Section IV-B), then its schedulability is guaranteed in a fault-free scenario. Thus, we only need to deal with cases where $\gamma_{\max c}$ is a primary copy. Consequently, we show that, if $R_{\mathrm{FR}}\ (\gamma_{\max c}) \leq T_{\max c}$, then $\gamma_{\max c}$ is schedulable because its deadline is guaranteed. □

Theorem 3: Suppose that $\gamma_{\max c}$, a primary copy or any form of backup copy, is a task copy currently being assigned and scheduled. $hp(\gamma_{\max c})$ is a set of tasks with higher priorities than that of $\gamma_{\max c}$ on P_k in the presence of P_j's failures. The WCRT of $\gamma_{\max c}$, $R_{\text{FO}}(\gamma_{\max c})$, is calculated by

$$R_{\text{FO}}(\gamma_{\max c}) = \sum_{\tau_i \in hep(\gamma_{\max c})} \left\lceil \frac{R_{\text{FO}}(\gamma_{\max c})}{T_i} \right\rceil * C_i + \sum_{\beta_i \in hep(\gamma_{\max c})} \Phi FT\left(i, R_{\text{FO}}(\gamma_{\max c})\right) \quad (7)$$

where $\Phi FT(i,t)$ and $DeadL(\gamma_{\max c})$ are shown at the bottom of the page.

If $R_{\text{FO}}(\gamma_{\max c})$ satisfies $R_{\text{FO}}(\gamma_{\max c}) \le DeadL(\gamma_{\max c})$, then $\gamma_{\max c}$ is schedulable, i.e., $\gamma_{\max c}$ can meet its deadline in the presence of any processor failures. Otherwise, $\gamma_{\max c}$ is unschedulable. This schedulability test is called Tercos fault-occurrence fault-tolerant CTT (TFO-FTCTT).

Proof: To show that $\gamma_{\max c}$ is schedulable, we need to prove that $R_{\text{FO}}(\gamma_{\max c})$ is less than $\gamma_{\max c}$'s fault-tolerant deadline. In the case where a processor failure occurs on processor $P_j (j \ne k)$, $hep(\gamma_{\max c})$ may contain primary copies and all types of backup copies, i.e., $hep(\gamma_{\max c}) = Primary(P_k) \cup Recover(P_k, P_j)$. $R_{\text{FO}}(\gamma_{\max c})$ can be derived by considering the following two cases.

Case 1) Each primary copy τ_i is executed once during its period T_i. During the time interval $[0,t]$, τ_i is executed $\lceil t/T_i \rceil$ times. The total execution time during $[0,t]$ is $\lceil t/T_i \rceil^* Ci$.

Case 2) Let us consider a backup copy β_i. The function $\Phi FT(i,t)$ gives the overall execution time of β_i during time interval $[0,t]$. If β_i is a passive-backup copy, then the first request of β_i must be executed within a recovery time interval $[0, B_{(i-1)}]$, which is shorter than T_i, while the subsequent requests of β_i will be executed once every period T_i. Thus, if $t \le B_{(i-1)}$, only one request of β_i will be executed in [0, t]. Otherwise, one request of β_i has to be executed within $[0, B_{(i-1)}]$, and $\lceil (t - B_{(i-1)})/T_i \rceil$ requests must be executed within $[B_{(i-1)}, t]$. Hence, $\Phi(i,t) = \lceil (t - B_{(i-1)})/T_i \rceil + 1$, when $t > B_{(i-1)}$. If β_i is an active copy, then it performs in a similar way as a passive-backup copy. The active copy β_i must finish its execution of *BP* in the recovery time $B_{(i-1)}$. The execution time of β_i's next requests is C_i.

Equation (7) can be derived from the previously mentioned two cases. Now, let us prove the correctness of (9). If $\gamma_{\max c}$ is a primary copy, the deadline is the period of $\gamma_{\max c}$, i.e., $T_{\max c}$. If β_i is a passive-backup copy, however, the first instance of β_i may be less than or equal to $B_{(\max c-1)}$. If β_i is an active-backup copy, we first should guarantee the schedulability of β_i after a processor failure, i.e., the schedulability of $Backup(\beta_i)$. We also have to check the schedulability of β_i in the subsequent instances. Therefore, the fault-tolerant deadline of β_i in its first invocation is $T_{\max c} + B_{(\max c-1)}$. Thus, the fault-tolerant deadline of $\gamma_{\max c}$ can be written as (9). Consequently, we prove that, if $R_{\text{FO}}(\gamma_{\max c}) \le DeadL(\gamma_{\max c})$, $\gamma_{\max c}$ is schedulable. □

E. Algorithm Description

We have delineated the scheduling strategy and schedulability test criterion in Sections IV-C and IV-D. Now, we present our proposed scheduling algorithm as follows.

Algorithm 1: the *Tercos* algorithm

```
(1) Sort task copies in the increasing order of RM priorities: $\tau_1, \beta_1, \tau_2, \beta_2, \ldots, \tau_N, \beta_N$. Set number $M$ of required processors to 1, namely, $M = 1$;
(2) Repeat the following steps for $i = 1, 2, \ldots, N$;
    (2.1) Assign primary copy $\tau_i$ to a processor $P_j$ on which $\tau_i$ is schedulable and $\tau_i$'s response time is minimized. That is, $\sigma = Primary(P_j) \cup active(P_j) \cup \{\tau_i\}$ passes the TFR-FTCTT, and for all the processors $P_f (f = 1, \ldots, M; f \ne j)$, $\sigma = Primary(P_j) \cup Recover(P_j, P_f) \cup \{\tau_i\}$ passes the TFO-FTCTT. If such a candidate processor does not exist, then increase $M$ by 1 and assign $\tau_i$ to $P(\tau_i) = P_M$. Determine the status form of $\beta_i$ according to (1).
    (2.2) If $Status(\beta_i) =$ passive, then assign $\beta_i$ to the first processor $P_f$ on which $\beta_i$ is schedulable. Thus, identify the first processor $P_f (f \ne j)$, $\sigma = Primary(P_f) \cup Recover(P_f, P_j) \cup \{\beta_i\}$ can pass the TFO-FTCTT, and set $P(\beta_i) = P_f$. If no existing processor can accommodate $\beta_i$, then increase $M$ by 1 and assign $\beta_i$ to $P_M = P(\beta_i)$;
    (2.3) If $Status(\beta_i) =$ active, then assign $\beta_i$ to a processor $P_f$ on which $\beta_i$ is schedulable and the backup
```

$$\Phi FT(i,t) = \begin{cases} Backup(\beta_i), & \text{if } status(\beta_i) = \text{active and } t \le B_{(i-1)} \\ Backup(\beta_i) + \lceil (t - B_{(i-1)})/T_i \rceil * C_i, & \text{if } status(\beta_i) = \text{active and } t > B_{(i-1)} \\ C_i, & \text{if } status(\beta_i) = \text{passive and } t \le B_{(i-1)} \\ \left(\left\lceil \left(t - B_{(i-1)}\right)/T_i \right\rceil + 1 \right) * C_i, & \text{if } status(\beta_i) = \text{passive and } t > B_{(i-1)} \end{cases} \quad (8)$$

$$DeadL(\gamma_{\max c}) = \begin{cases} T_{\max c}, & \gamma_{\max c} \text{ is a primary copy} \\ B_{(\max c-1)}, & \gamma_{\max c} \text{ is a passive-backup copy} \\ B_{(\max c-1)}, & \gamma_{\max c} \text{ is } \left(\text{active and for the first } R_{\text{FO}}(\gamma_{\max c}) \le B_{(\max c-1)}\right) \\ T_{\max c} + B_{(\max c-1)}, & \gamma_{\max c} \text{ is } \left(\text{active and for the first } R_{\text{FO}}(\gamma_{\max c}) > B_{(\max c-1)}\right) \end{cases} \quad (9)$$

time is minimized. That is, $\sigma = Primary(P_f) \cup active(P_f) \cup \{\beta_i\}$ passes the *TFR-FTCTT*, and for all processors $P_f(f = 1, \ldots, M; f \neq j)$, $\sigma = Primary(P_j) \cup Recover(P_j, P_f) \cup \{\beta_i\}$ passes the *TFO-FTCTT*. If no such processor exists, then increase M by 1 and assign β_i to $P_M = P(\beta_i)$;

(3) For all the required processors, determine a feasible schedule for assigned task copies;

(4) Return the number of required processors M.

The Tercos algorithm performs two major phases, namely, a static task assignment phase and a processor scheduling phase. In the first phase, Tercos judiciously assigns primary and backup copies to processors in a way to reduce the number of processors needed to schedule periodic tasks. This phase is performed in Steps 1)–2) in Algorithm 1. In the second phase (see Step 3) in Algorithm 1), all assigned task copies are scheduled according the fixed-priority scheduling algorithm. After the task assignment and processor scheduling are handled by Steps 1)–3), Step 4) returns the number of required processors necessary to satisfy the timing constraints of the primary and backup copies of periodic tasks.

Theorem 4: Let N be the number of primary copies of periodic tasks in a real-time distributed system and M be the number of required processors to execute the task copies. The time complexity of the *Tercos* task assignment algorithm is $O(M^{2*}N^2)$.

Proof: The time complexity of the *TFR-CTT* or *TFO-CTT* is $O(N)$. To assign a primary or backup copy, Tercos has to examine at most M processors. It is imperative to conduct the *TFR-CTT* or *TFO-CTT* to do the schedulabilty analysis for each processor in the worst case. Hence, the time complexity of the schedulability analysis on each processor is $O(N^*M^*M)$. Therefore, the time complexity of the *Tercos* assignment algorithm is $O(N^*M^*M)^*O(N) = O(M^{2*}N^2)$. □

V. Debus Strategy

A. Motivation

Existing algorithms like FTRMFF make use of the WCRT analysis to check the schedulability of a task set and to determine the status of backup copies [27]. Although FTRMMF strives to schedule as many backup copies as possible to be executed in passive forms, active-backup copies still exhibit considerable unnecessary task redundancies when primary copies are successfully finished prior to their backup copies. For example, consider four real-time periodic tasks along with their backup copies, e.g., $\tau_1 = \beta_1 = (2, 4)$, $\tau_2 = \beta_2 = (2, 5)$, $\tau_3 = \beta_3 = (5, 9)$, and $\tau_4 = \beta_4 = (3, 15)$. The FTRMFF algorithm generates a schedule where the primary and backup copies are allocated to nodes in a real-time distributed system as follows: $Primary(P_1) = \{\tau_1, \tau_2\}$, $backup(P_1) = \{\Phi\}$, $Primary(P_2) = \{\Phi\}$, $backup(P_2) = \{\beta_1, \beta_2, \beta_3\}$, $Primary(P_3) = \{\tau_3, \tau_4\}$, $backup(P_3) = \{\Phi\}$, $Primary(P_4) = \{\Phi\}$, $backup(P_4) = \{\beta_4\}$, and $Status(\beta_1) = $ passive, $Status(\beta_2) = $ active, $Status(\beta_3) = $ active, and $Status(\beta_4) = $ active. Fig. 5 shows a task schedule in a time interval between time 0 and 19.

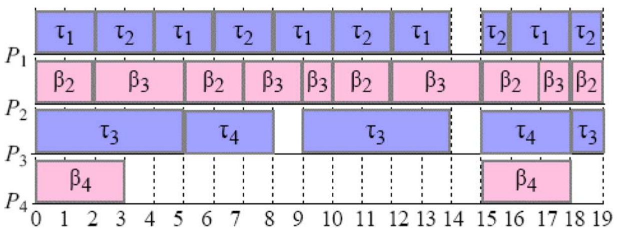

Fig. 5. Illustration of a redundancy imposed by an active-backup copy.

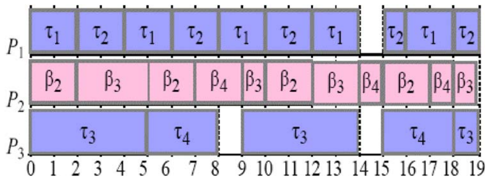

Fig. 6. Segmentation of active-backup copies in a fault-free scenario.

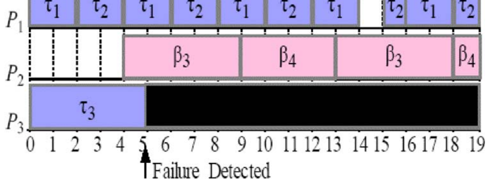

Fig. 7. Segmentation of *deferred-active-backup copy* when any failure occurs.

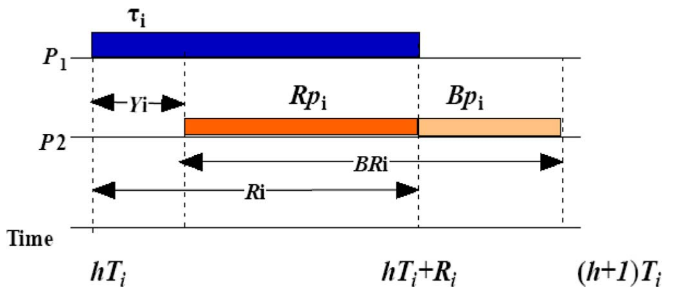

Fig. 8. Illustration of deferred-active-backup copies when any failure occurs.

If *Tercos* is used to allocate and schedule the four tasks, the number of required processors is four. In contrast, the *Debus* strategy postpones the execution of β_3 for four time units while terminating the execution of β_3 when its primary copy successfully completes the execution. In doing so, β_4 can be scheduled on P_2 (see Fig. 6) in fault-free scenarios. If P_3 fails at the end of τ_3' execution (i.e., at time slot 5), then β_3 can continue running without being terminated (see Fig. 7). Consequently, *Debus* reduces the number of required processors from four to three.

B. Description of Debus

Fig. 8 shows that Debus delays the execution of β_i by Y_i time units. Let the WCRTs of τ_i and β_i be R_i and BR_i, respectively. Note that BR_i is always larger than R_i, and this fact is determined by our task assignment strategy (see Section V-D). Divided by R_i, the execution of β_i is separated into two parts: *BP* and *RP*. *RP* executes in parallel with the primary copy, whereas *BP* is executed after its primary copy fails in producing correct results before the deadline. The implementation of *Debus* is challenging, because Debus has to precisely determine how much time the execution of active-backup copies should be delayed.

Again, we use $Redundant(\beta_i)$ and $Backup(\beta_i)$ to denote the execution time of *RP* and *BP* for β_i. Thus, $Redundant(\beta_i) = C_i - Backup(\beta_i)$. It should be noted that not all active-backup copies can be executed in schedules made by *Debus*. This is because, when a primary copy fails to produce correct results before its deadline, the recovery time of the primary copy may be occupied by copies of other tasks with higher priorities. This leads to an insufficient amount of time for the corresponding backup copy to be executed. We now introduce the third type of backup copy—*deferred*-active-backup copies, i.e., $status(\beta_i) = deferred$-active. The status of active-backup copies that cannot make use of the *Debus* technique are active, i.e., $status(\beta_i) =$ active. The following notation is used throughout Section V.

1) $deferred\text{-}active(P_j) = \{\beta_i \mid \beta_i \in Backup(P_j)$, $status(\beta_i) = deferred$-active$\}$.
2) $deferred\text{-}activeRecovery(P_j, P_f) = \{\beta_i \mid \beta_i \in Backup(P_j)$, $Status(\beta_i) = deferred$-active, $P(\tau_i) = Pf\}$
3) $DRecover(P_j, P_f) = passiveRecover(P_j, P_f) \cup activeRecover(P_j, P_f) \cup deferred\text{-}activeRecovery(P_j, P_f)$.

Lemma 1: Suppose that σ is a set of task copies on processor P_j when P_i fails. Then, $\sigma = Primary(P_j) \cup DRecover(P_j, P_i)$, and the *cumulative workload* $W(t, \sigma)$ on processor P_j required by σ during [0, t] is

$$W(t,\sigma) = \sum_{\tau_i \in \sigma} \left\lceil \frac{t}{T_i} \right\rceil * C_i + \sum_{\beta_i \in \sigma} \Phi T(i,t) \qquad (10)$$

where $\Phi T(i,t)$ is shown at the bottom of the page.

Proof: When P_i encounters failures, tasks copies running on P_j include all the primary copies that have been assigned to P_j and all the backup copies whose primary copies have been assigned to P_i. Thus, $\sigma = Primary(P_j) \cup DRecover(P_j, P_i)$. The $W(t, \sigma)$ can be derived from two cases.

Case 1) Each primary copy τ_i is executed in the same way as that described in Theorems 2 and 3. Thus, the workload contributed by primary copies to $W(t, \sigma)$ is calculated by the first item on the right-hand side of (10).

Case 2) Now, we consider each backup copy β_i. The function $\Phi FT(i, t)$ gives the total execution time of the instances of β_i during time interval [0, t]. If β_i is an active copy, then it performs as a primary copy. If β_i is a passive-backup copy, then the first instance of β_i must be executed within recovery time $[0, B_{(i-1)}]$, which is shorter than T_i. The next instance of β_i will be executed once every period T_i. If $t \le B_{(i-1)}$, there is only one instance of β_i to be executed in [0, t]. Otherwise, there is one instance of β_i to be executed within $[0, B_{(i-1)}]$ and $\lceil (t - B_{(i-1)})/T_i \rceil$ instances to be executed within $[B_{(i-1)}, t]$. Hence, $\Phi T(i, t) = (\lceil (t - B_{(i-1)})/T_i \rceil + 1)^* C_i$, when t is larger than $B_{(i-1)}$. If β_i is a passive-backup copy, the BP of β_i must be executed within a recovery time $[0, B_{(i-1)}]$. The next instance of β_i will be executed once every period T_i. If $t \le B_{(i-1)}$, there is only one backup time of β_i to be executed in [0, t]. Otherwise, there is only one backup time of β_i to be executed within $[0, B_{(i-1)}]$ and $\lceil (t - B_{(i-1)})/T_i \rceil$ instances to be executed within $[B_{(i-1)}, t]$. Thus, $\Phi T(i, t) = Backup(i - 1) + \lceil (t - B_{(i-1)})/T_i \rceil^* Ci$, when t is larger than $B_{(i-1)}$.

Considering the aforementioned two cases, we can derive $W(t, \sigma)$ using (10). □

With Lemma 1 in place, we can calculate $Backup(\beta_i)$, which is the amount of free time in a period of time $[0, B_i]$, i.e., $Backup(\beta_i) = Idle(B_i)$. Here, $Idle(t)$ can be computed by the following two steps. First, we add a virtual task $\overline{\gamma} = (\overline{T} = Bi, \overline{C})$ with the lowest priority among all tasks that have been assigned on processor P_j. Second, we determine the maximal value of $\overline{C}$ such that task $\overline{\gamma}$ meets its deadline. Formally, we have

$$Idle\,(t) = \max\{\overline{C} \mid \overline{\gamma} \text{ is schedulable}\}. \qquad (12)$$

The finishing time t for task $\overline{\gamma}$ can be written as

$$t = W\left(B_{\max c},\, \sigma \cup \{\overline{\gamma}\}\right). \qquad (13)$$

Moreover, if $t \le Bi$, then $\overline{\gamma}$ is schedulable; else, $\overline{\gamma}$ is unschedulable. Using this strategy, we can easily obtain $Backup(\beta_i) = Idle(Bi)$.

C. Employing the Dual Priority Scheduling Strategy

Debus is an effective algorithm to reduce redundancies imposed by active-backup copies. However, the implementation of

$$\Phi T(i,t) = \begin{cases} \lceil t/T_i \rceil^* C_i, & \text{if } status(\beta_i) = \text{active} \\ C_i, & \text{if } status(\beta_i) = \text{passive and } t \le B_{(i-1)} \\ \left(\left\lceil \left(t - B_{(i-1)}\right)/T_i \right\rceil + 1\right)^* C_i, & \text{if } status(\beta_i) = \text{passive and } t > B_{(i-1)} \\ Backup(i-1), & \text{if } status(\beta_i) = deferred\text{-active and } t \le B_{(i-1)} \\ Backup(i-1) + \left\lceil \left(t - B_{(i-1)}\right)/T_i \right\rceil^* C_i, & \text{if otherwise} \end{cases} \qquad (11)$$

Debus is nontrivial. In this study, we leverage the well-known dual priority scheduling technique to delay the executions of active-backup copies, thereby forcing active-backup copies to be executed in passive forms.

In the dual priority scheduling scheme [28], we assume that there are three unique priority bands: *Upper*, *Medium*, and *Lower*. Hard real-time tasks execute at either an upper or lower priority band. Upon release, each task is set by default to the lower priority band. At a fixed time offset from release, the priority of the task is promoted to the upper band. At run time, other tasks with soft deadlines are assigned priorities in the medium priority band. Hence, soft real-time tasks are executed in preference to hard real-time tasks that are yet to undergo priority promotion, and the execution of periodic tasks are delayed without adversely affecting the schedulability of periodic tasks.

In our model, each hard real-time task has an initial priority in the lower level and a unique promoted priority i, where $1 \leq i \leq 2n$ in the upper level (Note that 1 is the highest and $2n$ is the lowest priority in the upper band). The lower priorities have $2n$ priority levels, and the assignment of lower band priorities can be arbitrary.

In the Debus strategy, each active-backup copy β_i has a priority promotion delay $Y_i (0 \leq Y_i \leq D_i)$ measured relative to its release. Upon the release of a task, its priority is set to a lower band priority. After Y_i time units, its priority is promoted to priority i (in the upper band).

Calculating Y_i for a backup copy β_i is the key issue in implementing the *Debus* algorithm. R. Davis and A. Wellings proved that the dual priority scheduling technique is equivalent to the fixed-priority scheduling algorithm with offsets, i.e., with $O_i = Y_i$ [28]. Thus, the computation of $Y_{\max c}$ (suppose that $\beta_{\max c}$ is the current $deferred$-active-backup copy to be assigned) is given by

$$Y_{\max c} = \max\{\mathrm{y} \geq 0 \mid Idle_i(R_{\max c}) - Idle_i(y) \geq Redundant(\beta_{\max c})\} \quad (14)$$

where $Idle_i(t)$ is the amount of computation time a task at priority level i could perform during the time interval [0, t]. The optimal value of $Y_{\max c}$ can be computed by a dichotomic search between values y, 0, and $R_{\max c} - Redundant(\beta_{\max c})$. Here, $Idle(t)$ can be derived in a similar way described in Section IV-B. To calculate $Idle(t)$, Lemma 2 is proved in the following.

Lemma 2: Let ω be a set of task copies on processor P_i in fault-free scenarios. Thus, $\omega = Primary(P_i) \cup active(P_i) \cup deferred\text{-}active(P_i)$, and the cumulative workload $Q(t,\omega)$ on P_i required by σ during time interval [0, t] is expressed as

$$Q(t,\omega) = \sum_{\tau_j \in \omega} \left\lceil \frac{t}{T_j} \right\rceil^* C_j + \sum_{\beta_j \in \omega}^{status(\beta_j)=\text{active}} \left\lceil \frac{t}{T_j} \right\rceil^* C_j + \sum_{\beta_j \in \omega}^{status(\beta_j)=deferred\text{-active}} \left\lceil \frac{t-Y_j}{T_j} \right\rceil^* Redundant(\beta_j). \quad (15)$$

Proof: In fault-free scenarios, task copies running on P_i includes all primary copies, active-backup copies, and all deferred-active-backup copies that have been assigned to P_i. That is, $\omega = Primary(P_i) \cup active(P_i) \cup deferred\text{-}active(P_i)$. To derive $Q(t,\omega)$, let us consider the following three cases.

Case 1) Each primary copy τ_i is executed in the way described in Theorems 2 and 3. Therefore, the workload contribution of primary copies to $Q(t,\omega)$ is calculated by the first item on the right-hand side of (15).

Case 2) Each active-backup copy is executed in the same way as its primary copy; therefore, the workload contributed by active-backup copies to $Q(t,\omega)$ is expressed as the second item on the right-hand side of (15).

Case 3) Each deferred-active-backup copy is executed in a similar way as its primary copy. The main difference is that, during period T_i, each execution time is only the RP of its execution time. Therefore, the workload imposed by deferred-active-backup copies is written as the third item on the right-hand side of (15).

Considering the aforementioned three cases, we can calculate $Q(t,\omega)$ using (15). □

Lemma 2 suggests a means of computing $Idle(t)$ in two phases. First, a virtual task $\overline{\gamma} = (\overline{T} = t, \overline{C})$ is added, and the virtual task has the lowest priority among all the tasks that have been assigned on processor P_i. Next, the maximal value of $\overline{C}$ is determined in such a way that task $\overline{\gamma}$ meets its deadline. Thus, we have

$$Idle\,(t) = \max\{\overline{C} \mid \overline{\gamma} \text{ is schedulable}\}. \quad (16)$$

The finishing time f for task $\overline{\gamma}$ is written as

$$f = Q\,(t, \omega \cup \{\overline{\gamma}\})\,. \quad (17)$$

If f is smaller than or equal to t, then $\overline{\gamma}$ is schedulable. Otherwise, $\overline{\gamma}$ is not schedulable. If we are unable to determine $Y_{\max c}$ satisfying $R_{\max c} - Redundant(\beta_{\max c})$, then $\beta_{\max c}$ cannot be successfully allocated to a processor as a $deferred\text{-}active$ backup copy.

D. Schedulabilty Test

In an effort to calculate $Y_{\max c}$, the schedulability of $deferred\text{-}active$ backup copies is checked assuming no processor failures. Consequently, we only need to perform the schedulability test for primary and active-backup copies when systems experience no processor failures.

Let $hp(\gamma_{\max c})$ denote a set of all tasks with higher priorities than that of $\gamma_{\max c}$. Let $hep(\gamma_{\max c})$ be a set of all tasks with priorities higher than or equal to $\gamma_{\max c}$'s priority. Thus, we have $hep(\gamma_{\max c}) = hp(\gamma_{\max c}) \cup \{\gamma_{\max c}\}$.

Theorem 4: Let $\gamma_{\max c}$ be a primary copy or an active-backup copy currently being assigned and scheduled. The

WCRT of $\gamma_{\max c}$, $R_{\mathrm{FR}}(\gamma_{\max c})$, is calculated by

$$R_{\mathrm{FR}}(\gamma_{\max c}) = Q\left(R_{\mathrm{FR}}(\gamma_{\max c}), hep(\gamma_{\max c})\right). \qquad (18)$$

If $R_{\mathrm{FR}}(\gamma_{\max c})$ is smaller than or equal to $T_{\max c}$, then $\gamma_{\max c}$ is schedulable in fault-free scenarios. Otherwise, $\gamma_{\max c}$ is not schedulable. This schedulability test is called Debus fault-free fault-tolerant CTT or DFR-CTT for short.

Proof: To prove that $\gamma_{\max c}$ is schedulable, we need to show that $R_{\mathrm{FR}}(\gamma_{\max c})$ is less than $\gamma_{\max c}$'s deadline. As per Theorem 1, $\gamma_{\max c}$ is a task copy with the longest response time, and its WCRT is equal to the workload of tasks in $hep(\gamma_{\max c})$ during time interval $R_{\mathrm{FR}}(\gamma_{\max c})$. In the case of a fault-free scenario, $hep(\gamma_{\max c})$ contains primary copies, active-backup copies, and deferred-active-backup copies on processor P_k along with $\gamma_{\max c}$, i.e., $hep(\gamma_{\max c}) = Primary(P_k) \cup active(P_k) \cup deferred\text{-}active(P_k) \cup \{\gamma_{\max c}\}$. Thus, $R_{\mathrm{FR}}(\gamma_{\max c})$ can be calculated using (18).

The deadline of $\gamma_{\max c}$ is the period of $\gamma_{\max c}$, i.e., $T_{\max c}$, when $\gamma_{\max c}$ is a primary copy or an active-backup copy. It is evident that, if $R_{\mathrm{FR}}(\gamma_{\max c}) \le T_{\max c}$, then $\gamma_{\max c}$ can be completed before its deadline, which in turn proves that $\gamma_{\max c}$ is schedulable. □

Theorem 5: Let $\gamma_{\max c}$ be a primary copy or any form of a backup copy currently being assigned and scheduled. $hp(\gamma_{\max c})$ represents a set of tasks with higher priorities than that of $\gamma_{\max c}$ on processor P_k in the presence of P_j's failures. The WCRT of $\gamma_{\max c}$, $R_{\mathrm{FO}}(\gamma_{\max c})$, is derived from (19), as shown at the bottom of the page.

If $R_{\mathrm{FO}}(\gamma_{\max c})$ is smaller than or equal to $DeadL(\gamma_{\max c})$, then $\gamma_{\max c}$ is schedulable in the presence of any processor failure. Otherwise, $\gamma_{\max c}$ is not schedulable. This schedulability test is called DFR-CTT.

Proof: To show that $\gamma_{\max c}$ is schedulable, we need to prove that $R_{\mathrm{FO}}(\gamma_{\max c})$ is less than $\gamma_{\max c}$'s fault-tolerant deadline. In case of processor failures in P_j, we have $hep(\gamma_{\max c}) = Primary(P_k) \cup DRecover(P_k, P_j) \cup \{\gamma_{\max c}\}$. $R_{\mathrm{FO}}(\gamma_{\max c})$ can be derived from (19).

Now, let us prove the correctness of (20). If $\gamma_{\max c}$ is a primary or an active-backup copy, the deadline is the period of $\gamma_{\max c}$, i.e., $T_{\max c}$. If β_i is a passive-backup copy, the first instance of β_i is less than or equal to $B_{(\max c-1)}$. If β_i is a deferred-active-backup copy, the value of $Backup(\beta_i)$ can guarantee the schedulability of β_i in case of a processor failure. Nevertheless, we still need to check the schedulability of β_i's subsequent instances. Therefore, the fault-tolerant deadline of β_i's first instance is $T_{\max c} + B_{(\max c-1)}$. Thus, the fault-tolerant deadline of $\gamma_{\max c}$ can be written as (20). Consequently, we prove that, if $R_{\mathrm{FO}}(\gamma_{\max c}) \le DeadL(\gamma_{\max c})$, then $\gamma_{\max c}$'s deadline is guaranteed, and $\gamma_{\max c}$ is schedulable. □

E. Task Assignment Strategy

The task assignment strategy of *Debus* is identical to that of *Tercos*. The task assignment strategy of *Debus* is following the best-fit policy. When real-time tasks cannot be accommodated by any processor, an extra processor will be added. The scheduling heuristics, task assignment algorithm, and time complexity can be found in Section IV.

VI. Performance Evaluation

We evaluate the performance of our algorithms through extensive simulations. To reveal performance improvements gained by our algorithms, we first compare *Tercos* with FTRMFF [27]. Next, we compare *Debus* with *Tercos* to show the strengths of the *Debus* algorithm.

A. Experimental Platform

Large task sets with periodic tasks are generated according to the following parameters.

1) Periods of tasks (T_i). A value randomly generated from the interval [0, 500].
2) Execution time of the ith task. A value taken from a random distribution in the interval $0 < C_i \le \alpha T_i$, where the parameter $\alpha = \max(C_i/T_i)(i = 1, \ldots, N)$, which represents the maximum load occurring in the task set on all processors. Three values are chosen for α, namely, 0.2, 0.5, and 0.8. The three values represent a real-time system with a low, medium, and heavy workload, respectively.
3) Size of a task set (L). A value selected from a set [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000].

Varying parameters L and α, we generate 30 different task sets. For the chosen n and α, each experiment is repeated 30 times, and the average result is calculated.

The performance metric in our experiments is the number of processors required to assign a set of tasks. In the results of the experiments, we denote the number of processor used by FTRMFF as N, M is the number of processors required by *Tercos*, and T is the number of processors used by *Debus*. Our simulation source code was written in Visual C++ 6.0 and run on Pentium 4.0 2.9 G with 512-M RAM.

B. Performance Comparisons Between FTRMFF and Tercos

First, we compare FTRMFF and *Tercos* with respect to schedulability. Figs. 9 and 10 shows the impact of the task set size on the schedulability of the two algorithms. It is clear that both M and N proportionally increase with the task size L.

$$R_{\mathrm{FO}}(\gamma_{\max c}) = W\left(R_{\mathrm{FO}}(\gamma_{\max c}), hep(\gamma_{\max c})\right) \qquad (19)$$

$$DeadL(\gamma_{\max c}) = \begin{cases} T_{\max c}, & \gamma_{\max c} \text{ is } (Primary \text{ or } active) \\ B_{(\max c-1)}, & \gamma_{\max c} \text{ is passive} \\ T_{\max c} + B_{(\max c-1)}, & \gamma_{\max c} \text{ is } \left(deferred\text{-}active \text{ and for the first } R_{\mathrm{FO}}(\gamma_{\max c}) > B_{(\max c-1)}\right) \end{cases} \qquad (20)$$

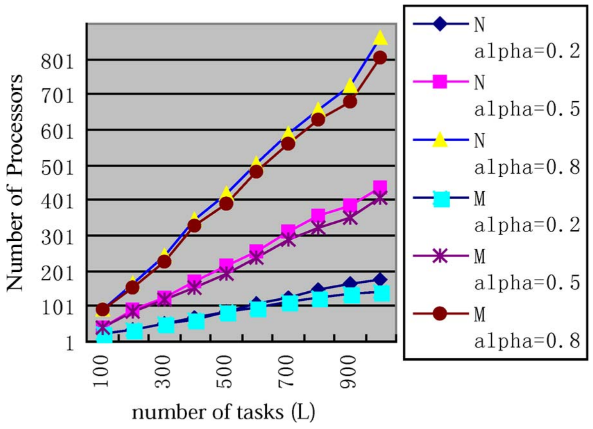

Fig. 9. Schedulability of Tercos and FTRMFF when L is increasing.

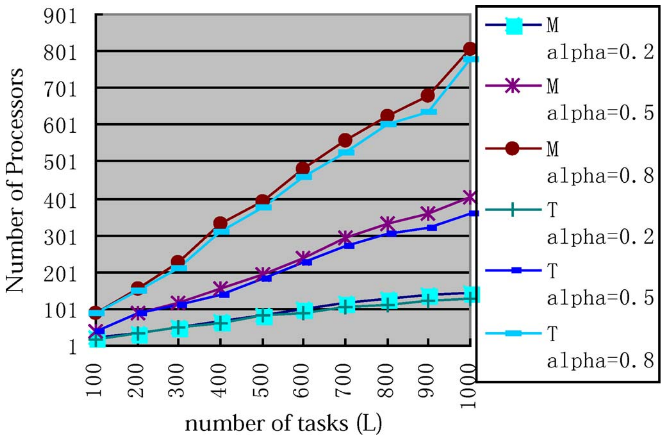

Fig. 11. Schedulability of Tercos and FTRMFF when L is increasing.

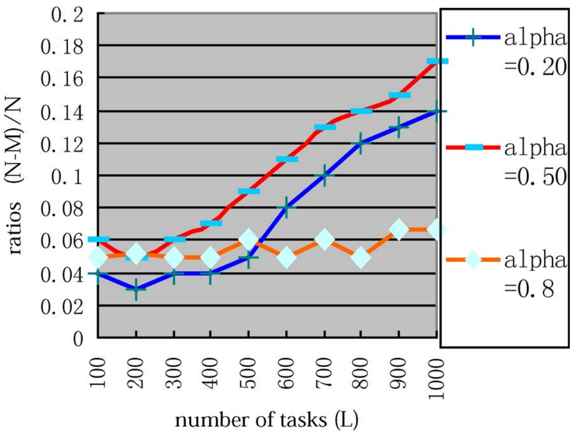

Fig. 10. $(N - M)/N$ shows the savings in the number processors required when L is increasing.

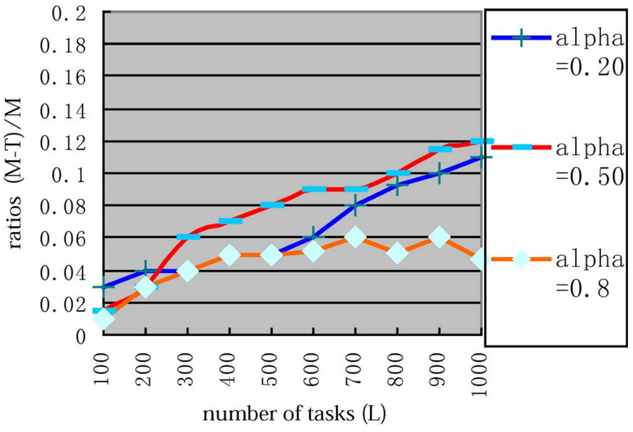

Fig. 12. $(M - T)/M$ shows the savings in the number processors required when L is increasing.

This is because, when the number of tasks increases, more processors are required to guarantee the deadlines of the tasks. Fig. 9 shows that *Tercos* provides remarkable savings in the number of required processors, because *Tercos* attempts to reduce redundancies introduced by active-backup copies when primary copies are successfully completed.

Moreover, Fig. 10 shows the value of $(N - M)/N$, which gives the *ratio of savings in processors* provided by *Tercos*. It is observed that the ratio of savings in processors reaches its peak when α is set to 0.5. We attribute this to the fact that, when $\alpha = 0.5$, the WCRTs of most primary copies are nearly half of their deadlines. Therefore, we can make the WCRTs of most back copies longer, which in turn results in shortened RPs. It is to be noted that the advantage of *Tercos* over FTRMFF is smallest when α is set to 0.8. This is mainly because the WCRTs of both primary and backup copies are relatively large, meaning that the overlapping of primary and backup copies is small. This means that there are few RPs which are required for *Tercos* to improve the ratio of savings in processors.

In summary, *Tercos* can reduce the number of required processors by up to 17.0% (with an average of 9.7%). Hence, we can conclude that *Tercos* improves schedulability over FTRMFF by reducing unnecessary redundancies.

C. Performance Comparisons Between Debus and Tercos

In this experiment, we compare *Tercos* and *Debus* in terms of schedulability. Fig. 11 shows the impact of the task set size on the schedulability measured as the number of required processors. Fig. 11 shows that M and T proportionally increase with the increase in task size L. These results are consistent with those shown in Fig. 9. Fig. 12 shows that *Debus* provides significant savings in the number of required processors, because *Debus* can further reduce task redundancies by delaying the execution of active-backup copies.

Furthermore, Fig. 12 shows the value of $(M - T)/M$, which is the *ratio of savings in required processors* provided by *Debus*. It is noticed that the savings in the number of processors reaches its peak when α is set to 0.5. Again, this finding can be explained by the fact that, when $\alpha = 0.5$, the WCRTs of most primary copies are nearly half of their deadlines. Consequently, a vast majority of active-backup copies are postponed to be executed as $deferred$-active-backup copies. It is observed that the advantage that *Debus* has over *Tercos* is smallest when α is set to 0.8. This phenomenon can be explained by the fact that, when α is large, the WCRTs of primary copies are very long, which lead to small recovery times of backup copies. Hence, the effectiveness of $deferred$-active-backup copies becomes insignificant. Fig. 12 also shows that the size of the task set has no apparent impact on the ratio $(M - T)/M$ when $\alpha = 0.8$; we can also attribute this result to the fact that fewer parts of the active-backup copies can be executed as passive status.

Compared with *Tercos*, *Debus* reduces the number of required processors by up to 12% (with an average of 7.8%). Thus, the experimental results show that *Debus* significantly improves schedulability over *Tercos* by employing the concept of $deferred$-active-backup copies.

VII. Summary and Future Work

In recognition that achieving high schedulability is one of the main design objectives of fault-tolerant and real-time distributed systems, we first addressed the limitations of conventional fault-tolerant real-time scheduling algorithms for periodic and preemptive real-time tasks. Next, we designed two efficient scheduling strategies with a goal to improve the schedulability of fault-tolerant and real-time distributed systems running preemptive and periodic tasks. The first strategy or *Tercos* makes an effort to avoid task redundancies by terminating the executions of active-backup copies as long as their corresponding primary copies can be successfully completed. The second scheme or *Debus* further enhances the schedulability of Tercos by delaying executions of active-backup copies. Empirical results show that, compared with existing algorithms in the literature, *Tercos* significantly improves system schedulability by up to 17.0% (with an average of 9.7%), whereas *Debus* is able to improve the system schedulability over *Tercos* by up to 12% (with an average of 7.8%).

Future studies in this emerging field include the following points. First, we will extend the *Debus* algorithm by further eliminating task redundancies to boost system schedulability. Second, we will be developing an advanced version of *Debus* by taking task precedence constraints into account.

Acknowledgment

The authors are grateful to the anonymous referees for their insightful suggestions and comments for improving the quality of this paper.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
[2] M. H. Klein, J. P. Lehoczky, and R. Rajkumar, "Rate-monotonic analysis for real-time industrial computing," *Computer*, vol. 27, no. 1, pp. 24–33, Jan. 1994.
[3] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem," *Oper. Res.*, vol. 26, no. 1, pp. 127–140, Jan./Feb. 1978.
[4] A. Burchard, J. Liebeherr, and S. H. Son, "New strategies for assigning real-time tasks to multiprocessor systems," *IEEE Trans. Comput.*, vol. 44, no. 12, pp. 1429–1443, Dec. 1995.
[5] M. Weber, "Operating systems enhancement for a fault-tolerant dual processor structure for the control of an industrial process," *Softw. Pract. Exper.*, vol. 17, no. 5, pp. 345–350, May 1985.
[6] M. R. Lyu, *Handbook of Software Reliability Engineering*. New York: McGraw-Hill, 1996.
[7] B. W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*. New York: Addison-Wesley, 1989.
[8] L. Liestman and R. H. Campbell, "A fault-tolerant scheduling problem," *IEEE Trans. Softw. Eng.*, vol. SE-12, no. 11, pp. 1089–1095, Nov. 1986.
[9] S. Ghosh, R. Melhem, and D. Mosse, "Fault-tolerant rate-monotonic scheduling," in *Proc. IFIP Int. Conf. Dependable Comput. Critical Appl.*, 1997, pp. 149–181.
[10] F. Liberato, R. Melhem, and D. Mosse, "Tolerance to multiple transient faults for aperiodic tasks in hard real-time systems," *IEEE Trans. Comput.*, vol. 49, no. 9, pp. 906–914, Sep. 2000.
[11] M. Caccamo and G. Buttazzo, "Optimal scheduling for fault-tolerant and firm real-time systems," in *Proc. Int. Conf. Real-Time Comput. Syst. Appl.*, Oct. 27–29, 1998, pp. 223–231.
[12] D. A. Lima and A. Burns, "An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems," *IEEE Trans. Comput.*, vol. 52, no. 10, pp. 1332–1346, Oct. 2003.
[13] X. Qin, H. Jiang, and D. R. Swanson, "Dynamic load balancing for I/O-intensive tasks on heterogeneous systems," in *Proc Int. Conf. High Performance Comput.*, 2003, pp. 300–309.
[14] E. N. Huh, L. R. Welch, B. A. Shirazi, and C. D. Cavanaugh, "Heterogeneous resource management for dynamic real-time systems," in *Proc. Int. Workshop Heterogeneous Comput.*, 2000, pp. 287–296.
[15] T. Xie and X. Qin, "Scheduling security-critical real-time applications on clusters," *IEEE Trans. Comput.*, vol. 55, no. 7, pp. 864–879, Jul. 2006.
[16] C. M. Krishna and K. Shin, "On scheduling tasks with a quick recovery from failure," *IEEE Trans. Comput.*, vol. C-35, no. 5, pp. 448–455, May 1986.
[17] X. Qin, H. Jiang, and D. R. Swanson, "An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems," in *Proc. Int. Conf. Parallel Process.*, Vancouver, BC, Canada, Aug. 2002, pp. 360–368.
[18] W. Luo, F. M. Yang, L. P. Pang, and X. Qin, "Fault-tolerant scheduling based on periodic tasks for heterogeneous systems," in *Proc. 3rd Int. Conf. Autonomic Trusted Comput.*, 2006, pp. 571–580.
[19] X. Qin and H. Jiang, "A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems," *Parallel Comput.*, vol. 32, no. 5/6, pp. 331–356, Jun. 2006.
[20] X. Qin and H. Jiang, "A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs on heterogeneous clusters," *J. Parallel Distrib. Comput.*, vol. 65, no. 8, pp. 885–900, Aug. 2005.
[21] G. Manimaran, C. Siva, and R. A. Murthy, "Fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 11, pp. 1137–1152, Nov. 1998.
[22] S. Ghosh, R. Melhem, and D. Mosse, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 3, pp. 272–284, Mar. 1997.
[23] C. H. Yang, G. Deconinck, and W. H. Gui, "Fault-tolerant scheduling for real-time embedded control systems," *J. Comput. Sci. Technol.*, vol. 19, no. 2, pp. 191–202, Mar. 2004.
[24] H. Liu and S. M. Fei, "A fault-tolerant scheduling algorithm based on EDF for distributed control systems," *Chin. J. Comput.*, vol. 14, no. 8, pp. 1371–1378, 2003.
[25] R. Al Omari, A. K. Somani, and G. Manimaran, "An adaptive scheme for fault-tolerant scheduling of soft real-time tasks in multiprocessor systems," *J. Parallel Distrib. Comput.*, vol. 65, no. 5, pp. 595–608, May 2005.
[26] T. Tatsuhiro, K. Yoshiaki, and K. Tohru, "A new fault-tolerant scheduling technique for real-time multiprocessors systems," in *Proc. Int. Workshop Real-Time Comput. Syst. Appl.*, 1995, pp. 197–202.
[27] A. A. Bertossi, L. V. Mancini, and F. Rossini, "Fault-tolerant rate-monotonic first-fit scheduling in hard-real-time systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 9, pp. 934–945, Sep. 1999.
[28] R. Davis and A. Wellings, "Dual priority scheduling," in *Proc. 16th IEEE Real-Time Syst. Symp.*, Pisa, Italy, 1995, pp. 100–109.
[29] G. Bernat and A. Burns, "A specification and analysis of weakly hard real-time systems [Dissertation]," Dept. de Ciencies Matematiques Informatica, Universitat de les Illes Balears, Palma, Spain, Jan. 1998.
[30] R. Al-Omari, A. K. Somani, and G. Manimaran, "A new fault-tolerant technique for improving the schedulability in multiprocessor real-time systems," in *Proc. Int. Parallel Process. Symp.*, San Francisco, CA, Apr. 2001.
[31] R. Melhem, D. Mosse, and E. Elnozahy, "The interplay of power management and fault recovery in real-time systems," *IEEE Trans. Comput.*, vol. 53, no. 2, pp. 217–231, Feb. 2004.
[32] Y. Oh and S. H. Son, "Scheduling real-time tasks for dependability," *J. Oper. Res. Soc.*, vol. 48, no. 6, pp. 629–639, 1997.
[33] S. Song, Y.-K. Kwok, and K. Hwang, "Trusted job scheduling in open computational grids: Security-driven heuristics and a fast genetic algorithms," in *Proc. Int. Symp. Parallel Distrib. Process.*, 2005, pp. 33–40.

[34] X. Qin and T. Xie, "An availability-aware task scheduling strategy for heterogeneous systems," *IEEE Trans. Comput.*, vol. 57, no. 2, pp. 188–199, Feb. 2008.

[35] Y. Y. Du, C. J. Jiang, and M. C. Zhou, "Modeling and analysis of real-time cooperative systems using PetriNets," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 37, no. 5, pp. 643–654, Sep. 2007.

[36] A. Janiak, M. Y. Kovalyov, and M. Marek, "Soft due window assignment and scheduling on parallel machines," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 37, no. 5, pp. 614–620, Sep. 2007.

[37] N. Garg, D. Grosu, and V. Chaudhary, "Antisocial behavior of agents in scheduling mechanisms," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 37, no. 6, pp. 946–954, Nov. 2007.

Wei Luo received the B.Sc. degree in computer science from Wuhan University of Technology, Wuhan, China, in 2002 and the M.Sc. and Ph.D. degrees in computer science from Huazhong University of Science and Technology, Wuhan, in 2005 and 2008, respectively.

Currently, he is a Researcher with the Department of Information System, China Ship Development and Design Center, Wuhan. His research interests are reliability-aware scheduling, cluster and fault-tolerant computing, parallel and distributed systems, and real-time/embedded systems.

Xiao Qin (S'99–M'04) received the B.Sc. and M.Sc. degrees in computer science from Huazhong University of Science and Technology, Wuhan, China, in 1996 and 1999, respectively, and the Ph.D. degree in computer science from the University of Nebraska, Lincoln, in 2004.

He is an Assistant Professor with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL. Prior to joining Auburn University in 2007, he was an Assistant Professor with New Mexico Institute of Mining and Technology, Socorro, for three years. His research interests include parallel and distributed systems, storage systems, fault tolerance, real-time computing, and performance evaluation. His research is supported by the U.S. National Science Foundation, Auburn University, and Intel Corporation.

He had served as a Subject Area Editor of the IEEE Distributed Systems Online (2000–2001). He has been on the program committees of various international conferences, including the IEEE Cluster, the IEEE International Performance Computing and Communications Conference, and the International Conference on Parallel Processing.

Xian-Chun Tan received the B.Sc. degree in mechanics and mathematics from Jining University, Jining, China.

Currently, he is a Research Fellow with the Department of Information System, China Ship Development and Design Center, Wuhan, China. His research interests are reliability-aware scheduling, cluster and fault-tolerant computing, parallel and distributed systems, and real-time/embedded systems.

Ke Qin received the B.Sc. degree in electronic information from the China Ship Engineering University, Wuhan, China.

Currently, he is a Research Fellow with the Department of Information System, China Ship Development and Design Center, Wuhan. His research interests are reliability-aware scheduling, cluster and fault-tolerant computing, parallel and distributed systems, and real-time/embedded systems.

Adam Manzanares received the B.S. degree in computer science from New Mexico Institute of Mining and Technology, Socorro, in 2006. Prior to August 2007, he was a Master's student at the University of Colorado, Boulder. He is currently working toward the Ph.D. degree in the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL.

His research interests focus on storage systems, cluster computing, distributed systems, and wireless networks.