On the Design of Fault-Tolerant Scheduling Strategies Using Primary-Backup Approach for Computational Grids with Low Replication Costs

Qin Zheng, *Member*, *IEEE*, Bharadwaj Veeravalli, *Senior Member*, *IEEE*, and Chen-Khong Tham, *Member*, *IEEE*

Abstract—Fault-tolerant scheduling is an imperative step for large-scale computational Grid systems, as often geographically distributed nodes cooperate to execute a task. By and large, *primary-backup* approach is a common methodology used for fault tolerance wherein each task has a primary copy and a backup copy on two different processors. For independent tasks, the backup copy can overload with other backup copies on the same processor, as long as their corresponding primary copies are scheduled on different processors. However, for dependent tasks, precedence constraint among tasks must be considered when scheduling backup copies and overloading backups. In this paper, we first identify two cases that may happen when scheduling dependent tasks with primary-backup approach. For one of the cases, we derive two important constraints that must be satisfied. Further, we show that these two constraints play a crucial role in limiting the schedulability and overloading efficiency of backups of dependent tasks. We then propose two strategies to improve schedulability and overloading efficiency, respectively. We propose two algorithms, called the Minimum Replication Cost with Early Completion Time (MRC-ECT) algorithm and the Minimum Completion Time with Less Replication Cost (MCT-LRC) algorithm, to schedule backups of independent jobs and dependent jobs, respectively. Algorithm MRC-ECT is shown to guarantee an optimal backup schedule in terms of replication cost for an independent task, while MCT-LRC can schedule a backup of a dependent task with minimum completion time and less replication cost. We conduct extensive simulation experiments to quantify the performance of the proposed algorithms and strategies.

Index Terms—Grid computing, directed acyclic graphs, independent tasks, primary-backup, fault-tolerance.

1 Introduction

Grid computing has emerged as the next-generation parallel and distributed computing methodology that aggregates dispersed heterogeneous resources for solving various kinds of large-scale parallel applications in science, engineering, and commerce [1]. These applications may be submitted by Grid users dynamically via Grid middleware and may have tight deadline requirements. Further, these applications may consist of independent jobs (for example, parameter sweep applications) and dependent jobs modeled by directed acyclic graphs (DAGs) which may consist of tens, hundreds, or even thousands of interdependent component tasks [2]. Running applications in such environments is susceptible to a wide range of failures as revealed by a recent survey [3] with real users on fault treatments in the Grid. Failures or error conditions due to the inherently unreliable nature of the Grid environment include hardware failures (e.g., host crash and network partition), software errors (e.g., memory leak and numerical exception), and other sources of failures (e.g., machine rebooted by the owner, network congestion, and excessive CPU load) [2]. As Grids are much more complex and heterogeneous than traditional computing systems, in a grid, one can discover a failure in a grid processor about what he/she could never know its hardware platform model has existed [3]. However, the issue of handling failures in the Grid services model as represented by the Open Grid Services Infrastructure (OGSI) and its Globus toolkit 3 (GT3) implementation remains largely unexplored [4]. This paper considers the problem of fault-tolerant scheduling of independent and dependent jobs arriving dynamically with deadline requirements in computational Grid, against processor failures. This problem is an NP-complete problem as given an application consisting of a number of jobs, either independent or dependent, and a set of heterogeneous resources, the problem of scheduling the jobs into the resources subject to some criteria (without fault tolerance) is a well-known NP-complete problem [5].

- *Q. Zheng is with the Advanced Computing Programme, Institute of High Performance Computing, 1 Science Park Road, #01-01 The Capricorn, Singapore Science Park II, Singapore 117528. E-mail: qinzheng.sg@gmail.com.*
- *B. Veeravalli and C.-K. Tham are with the Computer Networks and Distributed Systems (CNDS) Laboratory, Department of Electrical and Computer Engineering, National University of Singapore, 4 Engineering Drive 3, Singapore 117576. E-mail: {elebv, eletck}@nus.edu.sg.*

Manuscript received 24 Sept. 2006; revised 5 June 2007; accepted 26 Feb. 2008; published online 15 Sept. 2008.
Recommended for acceptance by C.-L. Wang.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0383-0906.
Digital Object Identifier no. 10.1109/TC.2008.172.

1.1 Related Works

In [6], fault-tolerant approaches are classified into 1) embed fault-tolerant mechanisms within the middleware software layer as in systems like Condor [7] or Legion [8] or 2) embed fault-tolerance mechanisms within algorithms. Works in [2],

[9], and [4] belong to the first category. Primary-backup approach, also called passive replication strategy, belongs to the second category. It was first studied in [10] where one server is selected as the primary and all the others are backups. If the primary fails, then a *failover* occurs and one of the backups takes over. In this paper, we consider a different approach where a backup is scheduled for each primary and they are located on two different processors. In this approach, a backup is executed when its primary cannot complete execution due to processor failure. It does not require fault diagnosis and is guaranteed to recover all affected tasks by processor failure. This approach is very useful for Grid where fault diagnosis is very difficult as one can discover a failure in a grid processor about what he/she could never know its hardware platform model has existed [3]. Most works using the primary-backup approach [11], [12], [13], [14], [15] consider scheduling of independent tasks. Backup overloading is introduced in [11] to reduce replication cost of independent tasks which allows scheduling backups of multiple primaries on the same or overlapping time interval on a processor. However, there are no algorithms in the literature that can guarantee to find an optimal schedule for backups of independent tasks in terms of replication cost. In this paper, we propose an algorithm that is guaranteed to find an optimal backup schedule for each independent task. In [16] and [17], to schedule tasks with precedence constraint, "strong primary copy" is considered which is assumed to *never* encounter the case where a primary cannot receive results from all its predecessors. As a result, their algorithm *eFRD* considers only direct predecessors of a task when scheduling and cannot tolerate failures of its other predecessors. In this paper, we consider all predecessors of a task, and thus, a task can always receive results from all its predecessors even if any of these predecessors fails.

1.2 Motivations

Our work is motivated by the need of efficient algorithms that can schedule both independent and dependent jobs on computational Grids so that jobs can run successfully even when processor failure occurs. Besides the need to deal with processor failure, grid applications, which are, in general, distributed, heterogeneous multitask applications, should be able to handle failures sensitive to the context of their own component tasks. For example, for DAGs where component tasks have dependencies among themselves, failure occurs when a task is unable to receive results from any of the tasks that it has dependency on. Therefore, precedence constraint among tasks must be considered while scheduling backup copies and overloading them. Scenarios where scheduling of backups of DAGs are allowed and scenarios that should be forbidden to enable fault tolerance must be clearly identified. The effect of this constraint on schedulability and overloading efficiency of backups must be analyzed. Consequently, strategies to improve both of them need to be developed. For both kinds of tasks, efficient algorithms must be developed which can schedule their backups with minimum completion time and minimum replication cost. To the best of our literature knowledge, this study is first of its kind to collectively consider all types of tasks, which arise in real-life situation, toward a fault-tolerant scheduling in Grid systems.

1.3 Scope of This Work and Our Contributions

The scope of our work is restricted to designing fault-tolerant scheduling algorithms and to evaluate their performance for all three categories of jobs—*independent*, *dependent*, and *hybrid* tasks—that are suitable for computational Grid systems. We will not be considering system level issues such as, *how communication is effected* and on the *underlying protocols for communications*. Within the scope of this formulation, our contribution in this work is fivefold. First, we analyze constraints that must be satisfied for backup scheduling and overloading of dependent tasks. Second, two strategies are proposed which uses information on maximum fault recovery time to improve schedulability; and interleaving technique to improve overloading efficiency, respectively. Third, we develop an algorithm which can determine the earliest possible start time for a backup of a dependent task. Fourth, two separate algorithms are developed which can schedule backups with minimum replication cost and minimum completion time, respectively. Finally, to calculate replication cost of a backup for both kinds of tasks, we develop an efficient algorithm.

The remainder of this paper is organized as follows: Section 2 describes the system model and states the problem. Sections 3, 4, and 5 consider dependent tasks. Constraints in backup scheduling and overloading are discussed in Section 3. Consequently, strategies to improve backup scheduling and overloading are proposed in Section 4. An algorithm which determines the earliest possible start time of a backup under these constraints and with the proposed strategies is described in Section 5. Section 6 presents the proposed algorithms for scheduling independent tasks and dependent tasks. Simulation results are discussed in Section 7. Section 8 provides concluding remarks and discusses future work.

2 System Model and Problem Statement

2.1 Task Model

Both independent jobs and DAGs comprise a number of tasks. E is a set of edges that defines the precedence relationships among tasks of a DAG. Each job has three attributes: arrival time t_a, deadline t_d, and execution time for each task. The Grid system considered has M heterogeneous processors. The execution time of a task on processors is heterogeneous and we use $t_e(j)$ to denote the execution time of a task j on a processor P_e. Each task has a primary and a backup. The execution time of the primary and the backup could be different depending on the processors they are scheduled on.

Both kinds of jobs are real-time, aperiodic, and nonpreemptive. Aperiodic jobs are those whose arrival times are not known a priori and therefore must be scheduled *dynamically* when they arrive. Nonpreemptive jobs are those that cannot be interrupted during execution and must finish to completion. Each task cannot be divided further for parallel processing, and thus must be scheduled in its entirety on a processor.

2.2 Fault Model

Tasks will fail when the processor where they are located fails due to hardware faults. The faults can be transient or permanent and are assumed to be independent. The

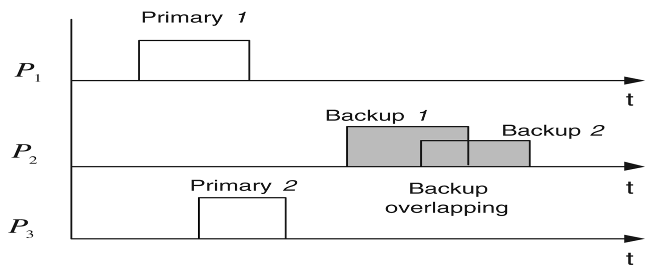


Fig. 1. Backup overlapping between two tasks.

maximum number of processors that are expected to fail at any instant of time is assumed to be one. For each task, the backup is scheduled after its primary. However, it is not required that backups of all tasks in a job must be scheduled after primaries of all tasks in a job. There exists a fault-detection mechanism such as fail-signal and acceptance test to detect processor and task failures [11], [2]. If a failure is detected in the primary, the backup will execute.

Backup overloading is defined as scheduling backups of multiple primaries on the same or overlapping time interval on a processor. Backup overlapping is shown in Fig. 1 where two primary copies are scheduled on processors 1 and 3 and their backups are scheduled in an overlapping manner on processor 2. The following are the conditions under which backups can be overloaded on a processor [11], [13]:

1. Backups scheduled on a processor can overload only if their primaries are scheduled on different processors. Therefore, although several backups may be overlapped on a processor, at most, one of them needs to be executed under the single processor failure model. The case that several overlapped backups execute concurrently will not happen.
2. At most, one of these primaries is expected to encounter a fault. This is to ensure that, at most, one backup is required to be executed among the overloaded backups.
3. At most, one version of a task is expected to encounter a fault. In other words, if the primary of a task fails, its backup always succeeds. This condition is guaranteed by the assumption that the minimum required value of mean time to failure (MTTF)[1] is always greater than or equal to the maximum task execution time in a primary-backup approach.

For DAGs, we assume the minimum required value of MTTF is always greater than or equal to the maximum job execution time. *Maximum failure recovery time* which denotes the maximum amount of time to recover a processor failure in the system is also an important parameter and we will discuss it later. When a primary finishes execution, its backup is deleted which is called *"resource reclaiming."* Resource reclaiming is also invoked when the primary completes earlier than its estimated execution time. Resource reclaiming is necessary so that the backup slot can be released timely for new tasks.

1. MTTF is defined as the expected time for which the system operates before the first failure occurs.

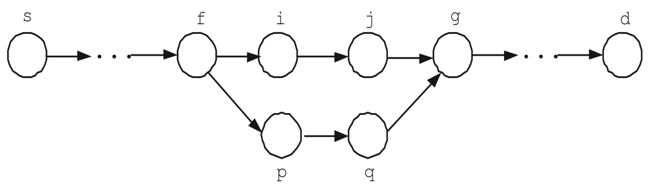


Fig. 2. A Directed Acyclic Graph (DAG).

2.3 Problem Statement

We now formally state the problem to be tackled. *Given a Grid system, task and fault model described above, we seek a schedule for primary and backup for each task, so that the response time and replication cost of a task are minimized, while both copies meet the deadline.*

3 Constraints in Scheduling and Backup Overloading of Dependent Tasks

For independent tasks, scheduling of backups is independent and backups can overload as long as their primaries are scheduled on different processors. However, backup scheduling and overloading of dependent tasks are nontrivial and additional constraints need to be identified. In this section, we discuss these constraints and find that they limit backup schedulability and overloading efficiency significantly.

A DAG is shown in Fig. 2 where arrow denotes precedence relationships among tasks. In the following, we consider scheduling primary and backup of task j. We first discuss how to schedule task j considering only one direct predecessor task i. Then, we discuss how to schedule j considering all its predecessors. Finally, we discuss the constraint in backup overloading.

3.1 Scheduling a Task Considering One Direct Predecessor

We consider scheduling task j with one direct predecessor i. Let $t^{P,s}(j)$ and $t^{P,f}(j)$ denote the start and finish time of primary of j. Let $t^{B,s}(j)$ and $t^{B,f}(j)$ denote the start and finish time of backup of j. Obviously, task j can only start execution after receiving result from task i. Therefore, primary of j must start after primary of i finishes executing. That is

$$t^{P,s}(j) > t^{P,f}(i). \tag{1}$$

However, a key question to be considered is that does primary of j have to start after backup of i? That is

$$Case1 : t^{P,s}(j) > t^{B,f}(i). \tag{2}$$

This case is shown in Fig. 3a and is referred to as Case1. In this case, primary of j can always receive results from i even if primary of i fails. However, response time and schedulability of j are constrained. Therefore, we allow primary of j to start before backup of i finishes. That is

$$Case2 : t^{P,s}(j) < t^{B,f}(i). \tag{3}$$

This case is referred as Case2 and is shown in Fig. 3b. In Case2, if primary of i fails, primary of j cannot receive results from backup of i which is still executing. Consequently, backup of j must be able to receive result and execute. To make it possible, the following two conditions must be

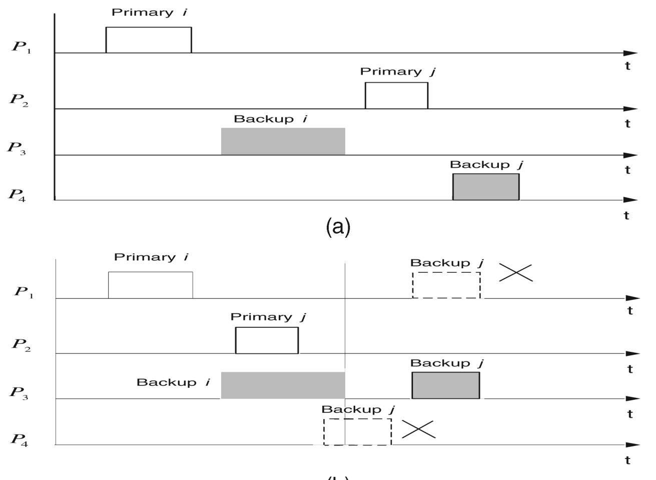


Fig. 3. Scheduling task j consider only one direct predecessor task i. (a) Case1. (b) Case2.

satisfied for Case2. Let $P_P(j)$ and $P_B(j)$ denote the two processors where primary and backup of j are scheduled, respectively. We have

$$Condition1 : t^{B,s}(j) > t^{B,f}(i), \tag{4}$$

$$Condition2 : P_B(j) \neq P_P(i). \tag{5}$$

Lemma 1. *Backup of task j can only start after backup of task i finishes and must not be scheduled on the processor where primary of i is located.*

Proof. First, we prove that backup of task j can only start after backup of task i finishes (Condition1). Condition1 must be satisfied as otherwise, when $P_P(i)$ fails, both primary and backup of j cannot receive result from backup of i which is still executing. The task cannot be completed. Next, we prove that backup of task j must not be scheduled on the processor where primary of i is located (Condition2). Condition2 must be satisfied as otherwise, when $P_P(i)$ fails, backup of j will also fail while primary of j cannot receive result from backup of i. This task cannot be completed. □

The significance of this lemma is that it specifies the earliest possible start time and processors on which backup of task j can be scheduled, with only one direct predecessor task i. For example, as shown in Fig. 3b, backup of j cannot be scheduled on processor P_1 or on processors P_3 and P_4 before $t^{B,f}(i)$. Therefore, backup of j can only be scheduled on processor P_3 or P_4 after $t^{B,f}(i)$.

In both Case1 and Case2, a special case is that primaries of tasks i and j are scheduled on the same processor. For Case2, if this special case happens, backup of task j can be scheduled subject to only Condition1. This is because Condition2 is always satisfied as backup of task j must not be scheduled on the processor where its primary is located. In the following, we will not distinguish this special case when scheduling backups.

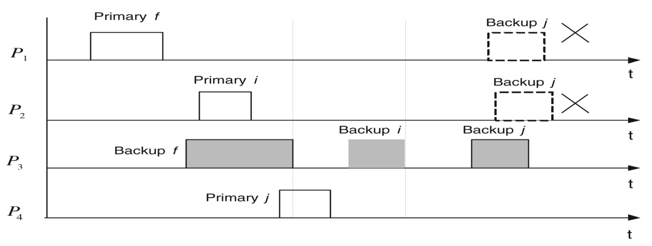


Fig. 4. Scheduling task j considering predecessors tasks f and i.

3.2 Scheduling a Task Considering All Its Predecessors

We now consider the case of scheduling primary and backup of task j with all its predecessors. Let the set of predecessors of j be denoted as set $S_p(j)$. Also, we denote the set of its direct predecessors as set $S_{dp}(j)$. Let u be a task in $S_{dp}(j)$. Primary of task j must start after primaries of all tasks in $S_{dp}(j)$ finish. That is

$$t^{P,s}(j) > \max_{u \in S_{dp}(j)} \{t^{P,f}(u)\}. \tag{6}$$

After primary of j is scheduled, we can compare its start time with finish time of backups of tasks in $S_{dp}(j)$ to determine whether their relationships satisfy Case1 or Case2. Task j satisfies Case1 relationship with all its direct predecessors *if*

$$t^{P,s}(j) > \max_{u \in S_{dp}(j)} \{t^{B,f}(u)\}. \tag{7}$$

If the above equation is satisfied, primary of j can always receive results from all its direct predecessors and backup of j can be scheduled with no additional constraints. Otherwise, task j must satisfy Case2 relationship with some of its direct predecessors. We first illustrate using the following example. In Fig. 4, we consider scheduling task j with its predecessors tasks f and i as shown in the DAG in Fig. 2. Tasks f and i as well as primary of task j are scheduled as shown in Fig. 4. It can be observed that tasks f and i satisfy Case2 relationship and tasks i and j also satisfy Case2 relationship. Furthermore, backup of j need to be scheduled after backup of i and cannot be scheduled on processors P_1 and P_2, where primaries of f and i are located. Otherwise, when processor 1 or 2 fails, backup of j also fails while primary of j cannot receive result. This DAG cannot be completed. Therefore, backup of j can only be scheduled on processor 3 after backup of i finishes.

We are now ready to present important conditions that must be satisfied when scheduling backup of task j if j satisfies Case2 relationship with any direct predecessor. Let $S^2_{dp}(j)$ denote the set of tasks in $S_{dp}(j)$ that satisfy Case2 relationship with j. Let $S^2_p(j)$ denote the set of tasks that comprises $S^2_{dp}(j)$ and tasks in $S_p(j)$ which have a series of only Case2 (no Case1) relationship with j. Let u be a task in $S^2_{dp}(j)$. $S^2_p(j)$ can be determined by the following equation:

$$S^2_p(j) = \left\{S^2_{dp}(j)\right\} \bigcup \left\{ \bigcup_{u \in S^2_{dp}(j)} \left\{S^2_p(u)\right\} \right\}. \tag{8}$$

$S^2_p(j)$ includes all tasks in $S^2_{dp}(j)$ and for each task u in $S^2_{dp}(j)$, all tasks in $S^2_p(u)$. We will discuss it in detail in

Lemma 4. Let $P^2_{dp}(j)$ and $P^2_p(j)$ denote the set of processors that primaries of tasks in $S^2_{dp}(j)$ and $S^2_p(j)$ are located, respectively. For instance, in the above example, tasks f and i are in $S^2_p(j)$ as task i is in $S^2_{dp}(j)$ and Case2 relationship exists between tasks f and i and tasks i and j. Consequently, $P_P(f)$ and $P_P(i)$ are in $P^2_p(j)$. We have

$$ConditionG1 : t^{B,s}(j) > \max_{u \in S^2_{dp}(j)} \{t^{B,f}(u)\}, \qquad (9)$$

$$ConditionG2 : P_B(j) \notin P^2_p(j). \qquad (10)$$

Lemma 2. *Backup of task j can only start after $\max_{u \in S^2_{dp}(j)}\{t^{B,f}(u)\}$ and must not be scheduled on processors in $P^2_p(j)$.*

Proof. First, we prove backup of task j can only start after $\max_{u \in S^2_{dp}(j)}\{t^{B,f}(u)\}$ (ConditionG1). We prove by contradiction. Suppose there is a task u in $S^2_{dp}(j)$ and $t^{B,s}(j) < t^{B,f}(u)$. When $P_P(u)$ fails, backup of task j cannot receive result from backup of u. This task cannot be completed. Contradiction happens. Therefore, backup of j must start after the maximum finish time of backups of all task in $S^2_{dp}(j)$.

Next, we prove backup of task j must not be scheduled on processors in $P^2_p(j)$ (ConditionG2). We prove by contradiction. Suppose that backup of task j is scheduled on processor P_e in $P^2_p(j)$. When processor P_e fails, backup of j will also fail. As a result, task j cannot be completed. Contradiction happens. Therefore, backup of task j must not be scheduled on processors in $P^2_p(j)$.□

The significance of this lemma is that it specifies the earliest possible start time and processors on which backup of task j can be scheduled, with all its predecessor. In the following Lemma 3, we will prove that $S^2_{dp}(j)$ is sufficient to determine $t^{B,s}(j)$. In Lemma 4, we will show that in order to determine $P^2_p(j)$ in ConditionG2, $S^2_p(j)$ does not need to have tasks that have a mixed series of Case1 and Case2 relationships with j.

Lemma 3. *$S^2_{dp}(j)$ is sufficient to determine $t^{B,s}(j)$.*

Proof. Let u be a task in $S^2_{dp}(j)$. Let task a be a direct predecessor of u. We first prove that no matter Case1 or Case2 relationship exists between a and u, backup of u always starts after backup of a finishes. For Case1, backup of u always starts after backup of a finishes as primary of u always starts after backup of a finishes. For Case2, backup of u always starts after backup of a finishes (Condition1). Therefore, backup of a task always starts after backup of its direct predecessor finishes. The same procedure can be applied to direct predecessors of task a and so on. Therefore, backups of direct predecessors of j always start after backups of other predecessors of j. It is sufficient to use $S^2_{dp}(j)$ to determine $t^{B,s}(j)$. □

The significance of this lemma is as follows: Lemma 3 attempts to derive the time after which backup of task j can start so that it can get all results from its predecessors even when processor fault happens. It is proved that it is sufficient to consider not all predecessors but only direct predecessors to determine this time. As a result, when dependent tasks arrive, algorithms can determine the earliest possible start time of backups in a much faster way.

Lemma 4. *$S^2_p(j)$ is sufficient to determine the set of processors that backup of task j cannot be scheduled on.*

Proof. We prove by contradiction. Suppose that tasks have a mixed series of Case1 and Case2 relationships with task j must be considered. As a result, backup of j cannot be scheduled on processors where primaries of these tasks are located. Otherwise, when one of these processors fails, both primary and backup of task j will not be able to complete. Let u denote one such task. Between u and j, there must exist at least two tasks satisfying Case1 relationship. Let these two tasks be a and b. Assume that primary of task u and backup of task j are scheduled on a processor and this processor fails. As a result, primary of task u and backup of task j both fail. However, no matter what relationship exists between tasks u and a, primary of task b is guaranteed to receive results from a. Therefore, primary of j will receive results and task j can be completed. Contradiction happens. □

The significance of this lemma is as follows: Lemma 4 attempts to derive the set of processors on which backup of task j cannot be scheduled so that it can get all results from its predecessors even when processor fault happens. It proves that it is sufficient to consider only predecessors with a series of Case2 relationships with j to determine this set. As a result, when dependent tasks arrive, algorithms can determine the set of processors that backups can be scheduled on in a much faster way. We will show the importance and use of the above two lemmas in the design of our strategies and algorithms.

3.3 Backup Overloading of Dependent Tasks

We have the following generic claim for backup overloading for the case of dependent tasks.

Theorem 1. *Backup overloading is not allowed among tasks with precedence relationships.*

Proof. Recall the following conclusion in the proof of Lemma 4: backup of a task always starts after finish time of its direct predecessors. Therefore, a task will never be able to overload its backup with backups of its direct predecessors. Similarly, backup of a task will never be able to overload with other predecessors as their backups finish even before direct predecessors of this task. Therefore, a task is never able to overload its backup with its predecessor. Similarly, this task is never able to overload its backup with tasks that have dependencies on it. Therefore, backup overloading is not allowed among tasks with precedence relationships. □

The theorem clearly shows that backup overloading in a DAG is constrained. As a result, its replication cost is high. Note that tasks without precedence can overload backups, such as tasks j and q in Fig. 2. However, as to be shown in the next section, backup overloading among tasks without precedence relationships is also limited.

4 Design of Strategies for Efficient Backup Schedulability and Overloading

In the previous section, we analyzed that scheduling backup and backup overloading are constrained for dependent tasks. In this section, we propose two strategies to improve schedulability and overloading efficiency of backups, respectively. Finally, necessary fault recovery requirements for dependent tasks are discussed.

4.1 Improving Backup Schedulability Based on Maximum Failure Recovery Time

In this strategy, we relax ConditionG2 by exploiting the information on maximum failure recovery time of a processor. This strategy is referred to as Improving Backup Schedulability Based on Maximum Failure Recovery Time (IBS-MFRT). When maximum failure recovery time (denoted as t_{mfr}) in the system is known, tasks in $S_p^2(j)$ whose primaries are at least t_{mfr} away from backup of task j can be exempted. This is because even when one of them fails, the fault must be recovered before backup of task j starts and will not affect the execution of j. Therefore, less processors are in $P_p^2(j)$. Let u be a task in $S_p^2(j)$ and P_e be a processor in $P_p^2(j)$. Tasks in $S_p^2(j)$ that are located on processor P_e can be determined and the latest finish time of primaries of these tasks (denoted as $t_e^{P,lf}(j)$) can be determined using the following equation:

$$t_e^{P,lf}(j) = \max_{u \in S_p^2(j), P_P(u)=P_e} \{t^{P,f}(u)\}. \quad (11)$$

ConditionG2 can be relaxed to ConditionG2R, that is, backup of task j can be scheduled on processor P_e in $P_p^2(j)$ if

$$ConditionG2R : t^{B,s}(j) > t_e^{P,lf}(j) + t_{mfr}, P_e \in P_p^2(j). \quad (12)$$

It implies that backup of task j can be scheduled on processors in $P_p^2(j)$ after certain time. Recall that previously without exploring information on maximum failure recovery time, backup of j cannot be scheduled on any processor in $P_p^2(j)$. This strategy improves schedulability of backup of j and will be particularly useful for fault tolerance of large-scale dependent jobs consisting of a large number of tasks each of which need to be executed for a long period of time.

4.2 Interleaving Technique

We have earlier shown via Theorem 1 that backup overloading is not possible for tasks with precedence relationships. Although tasks without precedence (such as j and q in Fig. 2) or tasks from different DAGs can overload backups, its efficiency is limited due to the two general conditions which inhibit backups of these tasks from being scheduled before certain time and on a certain set of processors. On the other hand, independent tasks can be scheduled with much more flexibility. This motivates us to overload backups of independent tasks to backups of dependent tasks, which is referred to as the *interleaving technique*. This technique has a profound effect in improving the overall performance of the system in terms of resource utilization. Of course, this can be carried out only when there are independent tasks arriving to the system. As a result of this interleaving, the overall replication cost in the system is reduced.

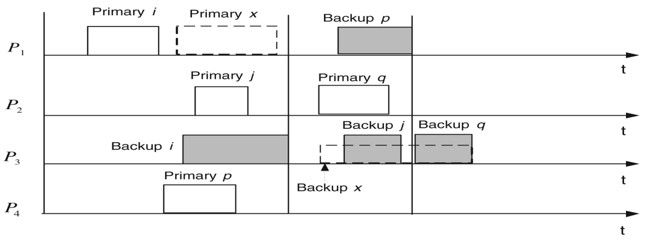


Fig. 5. Backup overloading between tasks without precedence relationships in a DAG and interleaving technique.

We use the following example to illustrate this idea. This example is designed to illustrate two fundamental aspects of this scheduling problem. In the first part, we will show that the overloading of backups for tasks without precedence relationships is also limited. In the second part, we will show how interleaving clearly aids to resolve this situation. In Fig. 5, we first consider scheduling task q where tasks i and j are scheduled as in Fig. 3 and task p and primary of task q are scheduled as shown in Fig. 5. Backup of q cannot be scheduled before backup of p and on processor P_4, as p and q satisfy Case2 relationship. As a result, it can only be scheduled on processor 1 or 3 after $t^{B,f}(p)$. Recall that backup of task j can only be scheduled on processor 3 or 4 after $t^{B,f}(i)$. Therefore, backup of q and j can only overload if they are both scheduled on processor 3 and after $t^{B,f}(p)$ or $t^{B,f}(i)$ whichever is larger. However, in the example, they will not be able to overload as backup of j is scheduled before $t^{B,f}(p)$ which is larger (rescheduling of tasks is not considered in this paper). Therefore, overloading of backups of tasks without precedence relationships is also constrained.

Now, we consider scheduling an independent task x using interleaving technique. Assume backup of task q is scheduled as shown in the figure. After primary of x is scheduled, there is no constraint on scheduling its backup which can overload with backups of both tasks q and j effectively as shown in the figure.

4.3 Fault Recovery Operations for Dependent Tasks

Recovery operations for independent tasks are straightforward where a backup executes if its primary fails. Therefore, we focus on dependent tasks and especially recovery operations required by a task if primary of one of its predecessors fails. We consider three tasks f, i, and j as shown in Fig. 2. Suppose the primary of task f fails. Consequently, the backup of task f executes. Recovery operations required by task i depend on whether Case1 or Case2 relationships exists between tasks f and i. For Case1 relationship, no operations are required and the primary of i will receive the result from the backup of f and execute. For Case2 relationship, the primary of task i will not be able to receive result and its backup need to be activated which will wait for result and execute instead. Next, we consider recovery operations required by task j, which depend on relationships between these three tasks. If Case1 relationship exists between tasks f and i, then no operations are required by task j as the primary of task i is executed. Otherwise, as the backup of task i is executed, recovery operations required by task j also depend on relationship

```
1. if $t^{P,s}(j) > \max_{u \in S_{dp}(j)} \{t^{B,f}(u)\}$ then
2.         for $P_e \neq P_P(j)$ do
3.                 $t_e^{B,eps}(j) \leftarrow t^{P,f}(j)$.
4. else
5.         for $P_e \neq P_P(j)$ do
6.                 $t_e^{B,eps}(j) \leftarrow \max\left\{t^{P,f}(j), \max_{u \in S_{dp}^2(j)} \{t^{B,f}(u)\}\right\}$.
7.         for $P_e \in P_p^2(j)$ and $P_e \neq P_P(j)$ do
8.                 if $t_e^{B,eps}(j) < t_e^{P,lf}(j) + t_{mfr}$ then
9.                         $t_e^{B,eps}(j) \leftarrow t_e^{P,lf}(j) + t_{mfr}$.
```

Fig. 6. Algorithm to determine the earliest possible start time.

between tasks i and j. For Case1 relationship, no operations are required while for Case2 relationship, the backup of task j will be activated. Similarly, recovery operations required by direct successors of task j can be determined and so on and so forth.

5 Algorithm to Determine the Earliest Possible Start Time

In the previous two sections, we discussed the constraints on the start time of backups of dependent tasks. In this section, we present an algorithm to determine their earliest possible start time on all processors. A pseudocode description of this algorithm is given in Fig. 6. Let P_e denote a processor besides processor $P_p(j)$ and $t_e^{B,eps}(j)$ denote the earliest possible start time of backup of task j on processor P_e. If primary of task j satisfies Case1 relationship with all its direct predecessors (7), its backup only needs to start after its primary and the earliest possible start time $t_e^{B,eps}(j)$ is $t^{P,f}(j)$. Otherwise, its backup must start after backups of all tasks in $S_{dp}^2(j)$ (ConditionG1) and $t_e^{B,eps}(j)$ is $\max\{t^{P,f}(j), \max_{u \in S_{dp}^2(j)}\{t^{B,f}(u)\}\}$. Furthermore, for processors in $P_p^2(j)$, its backup must start after the time specified by ConditionG2R and $t_e^{B,eps}(j)$ is $\max\{t^{P,f}(j), \max_{u \in S_{dp}^2(j)}\{t^{B,f}(u)\}, t_e^{P,lf}(j) + t_{mfr}\}$.

5.1 Complexity Analysis

We now analyze the complexity of this algorithm. Let $N_{dp}(j)$ denote the number of direct predecessors of task j. Let $N_{dp}^2(j)$ denote the number of direct predecessors of task j that satisfy Case2 relationship with j. Let $N_p^2(j)$ denote the number of tasks in $S_p^2(j)$ which satisfy a or a series of Case2 relationships with j.

The algorithm first takes $O(N_{dp}(j))$ to determine $\max_{u \in S_{dp}(j)}\{t^{B,f}(u)\}$. If (7) is satisfied, it takes $O(M)$ to assign $t^{P,f}(j)$ to $t_e^{B,eps}(j)$ for all processors. Otherwise, its backup must start after backups of all tasks in $S_{dp}^2(j)$ (ConditionG1) and $\max_{u \in S_{dp}^2(j)}\{t^{B,f}(u)\}$ can be determined in $O(N_{dp}^2(j))$. Then, it takes $O(M)$ to determine $t_e^{B,eps}(j)$ for all processors.

For lines 7 to 9, $P_p^2(j)$ can be determined after $S_p^2(j)$ is determined (8) which takes $O(N_{dp}^2(j))$. The latest finish time of primaries of tasks in $S_p^2(j)$ ($t_e^{P,lf}(j)$) for processors in $P_p^2(j)$ can be determined in two steps. In the first step, it takes $O(N_p^2(j))$ to map tasks in $S_p^2(j)$ to processors in $P_p^2(j)$. In the second step, it takes $O(N_p^2(j))$ to compare and determine the latest finish time on processors in $P_p^2(j)$. Then, the algorithm takes $O(N_p^2(j))$ to update $t_e^{B,eps}(j)$ for processors in $P_p^2(j)$. Therefore, the worst-case complexity for lines 7 to 9 is $O(N_{dp}^2(j) + N_p^2(j))$ which is $O(N_p^2(j))$ as $N_p^2(j)$ is larger or at least equal to $N_{dp}^2(j)$.

Therefore, the overall worst-case complexity to determine the earliest possible start time for a dependent task is $O(N_{dp}(j) + N_{dp}^2(j) + M + N_p^2(j))$ which is $O(N_{dp}(j) + M + N_p^2(j))$ as $N_{dp}(j)$ is larger or at least equal to $N_{dp}^2(j)$.

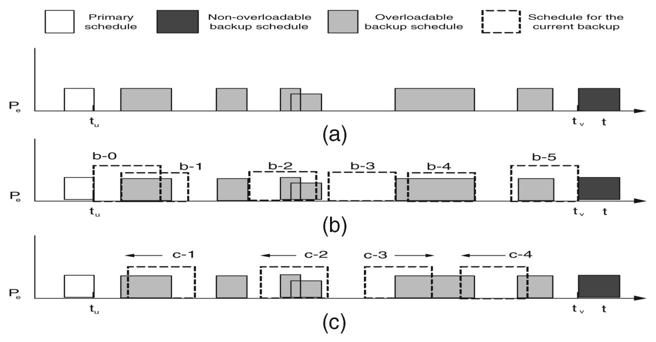

Primary schedule
Non-overloadable backup schedule
Overloadable backup schedule
Schedule for the current backup
P_e
t_u
t_v
t
(a)
b-0
b-1
b-2
b-3
b-4
b-5
(b)
c-1
c-2
c-3
c-4
(c)

Fig. 7. Schedule a backup on an interval: (a) overloadable backup schedules, (b) boundary schedules, and (c) other possible schedules.

6 The Proposed Fault-Tolerant Scheduling Algorithm for Backups

In this section, we present two algorithms for scheduling backups of independent tasks and dependent tasks, respectively. For independent tasks, we minimize replication cost of each backup as long as the deadline is met. The objective is to improve resource utilization. For dependent tasks, we minimize completion time of each backup and the objective is to reduce job rejection. By minimizing completion time of a task, tasks that have dependencies on it can start early so that the possibility of meeting the job deadline is increased.

6.1 Backup Schedules

After the earliest possible start time for a backup on all processors is determined, the time window that this backup can be scheduled on all processors is determined which is between this time and its deadline. Primary schedules and nonoverloadable backup schedules that are scheduled on the time window can be identified. These schedules partition the time window into a number of intervals and backup can only be scheduled on these intervals. One such interval is shown in Fig. 7a which is between t_u and t_v, where t_u is the end time of the primary schedule and t_v is the start time of the nonoverloadable backup schedule. There only exist overloadable backup schedules on each interval which can overload each other as shown in the figure. Note that these backup schedules could be scheduled for independent tasks or dependent task as interleaving technique is allowed.

Now, we address the problem of how to schedule backup of task j in an interval. We first consider special cases for scheduling backup of task j (dashed rectangle) where its start time and/or finish time collide with boundaries of the interval or boundaries of overloadable backup schedules. These special cases are referred to as boundary schedules and are labeled as b-0 to b-5 in Fig. 7b. Other possible cases are labeled as c-1 to c-4 in Fig. 7c where start time and/or finish time of backup of task j fall onto overloadable backup schedules and/or idle periods.

Lemma 5. *There exist only four possible cases c-1 to c-4 as in Fig. 7c for scheduling a backup of task j other than boundary schedules.*

Proof. As start time and/or finish time of backup of task j do not collide with boundaries (boundary schedules), they can only fall onto overloadable backup schedules (denoted as O-period) and/or idle periods (denoted as I-period). Therefore, there exist only four cases where start time and finish time fall onto different periods (c-1 and c-3); or they both fall onto I-periods (c-2); or they both fall onto O-periods (c-4). □

While traditional approach like As Soon As Possible (ASAP) [18], [19] or As Late As Possible (ALAP) [20] can be used to schedule the primary, they are not suitable to schedule the backup as the backup can overload with existing backup schedules, and thus, nonboundary schedules like cases c-1 to c-4 must be considered. As a result, sampling must be used to determine the start time of the backup and the replication cost. However, the complexity of this approach is not fixed and depends on the sampling rate. In the following sections, we present our proposed algorithms which do not need sampling when scheduling backups.

6.2 Boundary Schedules and Replication Cost

Replication cost is defined as the actual percentage of time needed to be scheduled for the backup besides all overloaded periods with existing backups. Let $t_e^O(j)$ denote the amount of time that backup of task j can overload with other backups on processor P_e. Replication cost to schedule backup of j on P_e is defined as

$$C_e^R(j) = \frac{t_e(j) - t_e^O(j)}{t_e(j)}. \qquad (13)$$

Lemma 6. *Nonboundary schedules always have higher replication cost or the same replication cost but later completion time than boundary schedules.*

Proof. We consider four nonboundary schedules c-1 to c-4 one by one. For case c-1, it can be moved to the left until its start time or finish time reaches boundaries. Case c-1 becomes boundary schedule b-1 if its start time reaches the left boundary, b-4 if its finish time reaches the right boundary, or b-0 when left boundary of the interval (t_u) is reached. In this process, the amount of overloading $t_e^O(j)$ is always increasing and start time becomes earlier. Therefore, case c-1 always has higher replication cost than its corresponding boundary schedules. Similarly, case c-2 can be moved to the left until its start time or finish time reaches boundaries which becomes boundary schedules b-2, b-4, or b-0. In this process, the amount of overloading $t_e^O(j)$ remains the same but start time becomes earlier. Case c-3 can be moved to the right until its start time or finish time reaches boundaries which becomes boundary schedules b-1, b-4, or b-5 when right boundary of the interval (t_v) is reached. In this process, the amount of overloading $t_e^O(j)$ is always increasing. Case c-4 can be moved to the left until its start time or finish time reaches boundaries which becomes boundary schedules b-1, b-3, or b-0. In this process, the amount of overloading $t_e^O(j)$ remains the same but start time becomes earlier. Therefore, the four nonboundary schedules c-1 to c-4 will always have higher replication cost or the same replication cost but later completion time than boundary schedules. □

The significance of this lemma is as follows: It proves that to minimize replication cost, it is sufficient to consider only boundary schedules for backups. Without this lemma, cases c-1 to c-4 must be considered and sampling must be used as it is continuous. As a result, the complexity is not fixed and depends on the sampling rate.

In the following two sections, we present the proposed algorithms for scheduling backups of independent tasks and dependent tasks, respectively. Both algorithms consider only boundary schedules to reduce replication cost. Therefore, sampling is not required and the complexity of these two algorithms is fixed and becomes much lower. As a result, when tasks arrive dynamically, our proposed algorithms can determine schedules for backups in a much faster way.

6.3 Algorithm for Scheduling Backups of Independent Jobs

We refer to our algorithm as Minimum Replication Cost with Early Completion Time (MRC-ECT). For all processors besides the one where the primary is scheduled on, boundary schedules within the time window are considered and their replication cost is compared. The boundary schedule which has minimum replication cost (denoted as $C^R(j)$) is chosen where

$$C^R(j) = \min_{1 \le e \le M, P_e \ne P_P(j)} \{C_e^R(j)\}. \qquad (14)$$

In case a tie happens, the boundary schedule which can complete earliest is selected. A pseudocode description of the algorithm is given in Fig. 8. Let t_l and t_r denote the start time and completion time of an existing schedule, respectively. The algorithm first considers boundary schedules of the time window. Then, all existing schedules within or overlapping with the time window are examined one by one and their respective boundary schedules are considered. Algorithm MRC-ECT invokes a subroutine BoundarySchedule which takes start time of the backup (denoted as t_s) as input parameter. This subroutine records the current boundary schedule if it is eligible and is better than the recorded (best) schedule. Specifically, the current boundary schedule is better if it has lower replication cost or the same replication cost but earlier completion time than the recorded (best) schedule. Algorithms to determine schedule eligibility and replication cost will be discussed later in this section.

Theorem 2. *MRC-ECT is guaranteed to find an optimal backup schedule in terms of replication cost for a task.*

Proof. From Lemma 6, it is proved that the optimal schedule in terms of replication cost must be boundary schedules. As MRC-ECT considers all boundary schedules when scheduling a backup, it guarantees to find the optimal schedule. □

```
1. for $P_e \neq P_P(j)$ do
2.     invoke BoundarySchedule with parameter $t_e^{B,eps}(j)$ and $deadline - t_e(j)$, respectively.
3.     for existing schedules within or overlapping with the time window do
4.         if the existing schedule is primary or non-overloadable backup schedules then
5.             invoke BoundarySchedule with parameter $t_l - t_e(j)$ and $t_r$, respectively.
6.         else // the existing schedule is an overloadable backup schedule
7.             invoke BoundarySchedule with parameter $t_l - t_e(j)$, $t_l$, $t_r - t_e(j)$, and $t_r$, respectively.
8.
9. BoundarySchedule($t_s$)
10.     if ScheduleEligible($t_s$)=$True$ then
11.         $cost \leftarrow$ ReplicationCost($t_s$).
12.         if $cost$ is less than $C^R(j)$ or they are equal and $t_s + t_e(j)$ is less than $t^{B,f}(j)$ then
13.             $P_B(j) \leftarrow P_e$.
14.             $C^R(j) \leftarrow cost$.
15.             $t^{B,f}(j) \leftarrow t_s + t_e(j)$.
```

Fig. 8. Algorithm MRC-ECT.

```
1. for $P_e \neq P_P(j)$ do
2.     $t_s \leftarrow deadline$.
3.     if ScheduleEligible($t_e^{B,eps}(j)$)=$True$ then
4.         $t_s \leftarrow t_e^{B,eps}(j)$.
5.     else
6.         while $t_l$ of existing schedules (in non-decreasing order) is not later than $t^{B,f}(j)$ or $t_s + t_e(j)$ do
7.             if the existing schedule is primary or non-overloadable backup schedules then
8.                 invoke ScheduleEligible with parameter $t_l - t_e(j)$ and $t_r$, in this order.
9.             else // the existing schedule is an overloadable backup schedule.
10.                invoke ScheduleEligible with parameter $t_l - t_e(j)$, $t_l$, $t_r - t_e(j)$, and $t_r$, in non-decreasing order.
11.        if both $t^{B,f}(j)$ and $t_s + t_e(j)$ are later than $deadline$ then
12.            if ScheduleEligible($deadline - t_e(j)$)=$True$ then
13.                $t_s \leftarrow deadline - t_e(j)$.
14.    $cost \leftarrow$ ReplicationCost($t_s$).
15.    if $t_s + t_e(j)$ is less than $t^{B,f}(j)$ or they are equal and $cost$ is less than the recorded cost then
16.        $P_B(j) \leftarrow P_e$.
17.        $t^{B,f}(j) \leftarrow t_s + t_e(j)$.
18.        the recorded cost $\leftarrow cost$.
```

Fig. 9. Algorithm MCT-LRC.

We use the following example to illustrate the difference between MRC-ECT and ASAP and ALAP. Consider scheduling the independent task x in Fig. 5. ASAP will schedule backup of task j immediately after its primary while ALAP will schedule backup of j so that it will complete right before the deadline. On the other hand, MRC-ECT attempts to schedule backup of j with minimum replication cost by maximizing overloading and early completion time. When deadline of j is after backup of q, it can be scheduled as shown in the figure which clearly minimizes replication cost. Note that postponing it any further will delay its completion time without reducing replication cost while pushing it forward will affect overloading efficiency. This algorithm derives its name by this property.

6.4 Algorithm for Scheduling Backups of Dependent Jobs

We refer to our algorithm as Minimum Completion Time with Less Replication Cost (MCT-LRC). For all processors besides the one where the primary is scheduled on, boundary schedules within the time window are considered and the boundary schedule which can complete earliest is chosen. In case a tie happens, the boundary schedule which has minimum replication cost is selected. A pseudocode description of the algorithm is given in Fig. 9. The algorithm first considers the left boundary schedules of the time window. Then, all existing schedules[2] (sorted in nondecreasing order of start time) within or overlapping with the time window are examined one by one, while the earliest boundary schedule of an existing schedule completes not later than the recorded (best) schedule or the schedule already found on the current processor (start at t_s). For each existing schedule, its boundary schedules are considered in the order of their start time and if one boundary schedule is eligible, remaining boundary schedules are exempted. If no eligible schedule has been found, the algorithm considers the right boundary schedule of the time window. Finally, the algorithm calculates replication cost of the (local) earliest schedule on the current processor and records it if it can complete earlier than the recorded schedule or they complete simultaneously but the local schedule has lower replication cost.

2. Note that all the existing schedules are already available as a sorted list based on their start time and hence will not contribute to the complexity.

6.5 Schedule Eligibility

A schedule is *eligible* if it is within the time window and does not overlap with any primary schedule or nonoverloadable backup schedule. Specifically, there must not exist any primary schedule or nonoverloadable backup schedule that starts and/or ends in this schedule or contains this schedule.

6.6 Computation of Replication Cost

If a schedule is eligible, there must exist only overloadable backup schedules, if any, that overlap with the current schedule. These overloadable backup schedules can be classified into three categories based on their relationships with the current schedule. Schedules in the first category

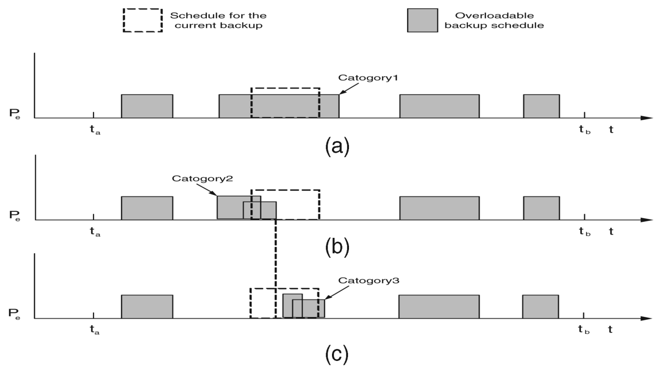


Fig. 10. Calculate replication cost of the current backup schedule. (a) Category1 overloadable backup schedules. (b) Category2 overloadable backup schedules. (c) Category3 overloadable backup schedules.

contain the current schedule, as shown in Fig. 10a. Schedules in the second category start before and end within the current schedule, as shown in Fig. 10b. Schedules in the third category start within but can end within or after the current schedule, as shown in Fig. 10c.

A pseudocode description of the algorithm to calculate replication cost of the current backup schedule is given in Fig. 11. Let t_l and t_r denote the start and completion time of an overloadable backup schedule, respectively. Let $t_e^{O,t}$ denote the time till the current backup schedule can overload with overloadable backup schedules. Let t_e^{NO} denote the amount of time that the current backup schedule cannot overload with any overloadable backup schedule.

The amount of time that overloading is not possible t_e^{NO} is initialized to zero and $t_e^{O,t}$ is initialized to t_s which is the start time of the current backup schedule. Overloadable backup schedules within or overlapping with the time window are browsed one by one in order of their start time. Schedules in Category1 and Category2 are examined first. If any Category1 schedule exists, the current schedule can overload its entirety and thus $t_e^{O,t}$ is assigned $t^{B,f}$ which is finish time of the current backup schedule; and break. As a result, t_e^{NO} becomes 0 and replication cost is 0. For Category2 schedules, $t_e^{O,t}$ is determined by the latest finish time of these schedules and is updated accordingly.

Category3 schedules are then examined and they need to be considered if their finish time is later than $t_e^{O,t}$. If their start time is also later than $t_e^{O,t}$, such as the first Category3 schedule shown in Fig. 10c, there exist a period of time that cannot be overloaded. Note that the current schedule will not be able to overload with later Category3 schedules as they start even later. Therefore, nonoverloadable period t_e^{NO} need to be updated. $t_e^{O,t}$ is also updated accordingly. After update, if $t_e^{O,t}$ is later or equal to $t^{B,f}$ which means the current schedule can overload with the Category3 schedule till the end, $t^{B,f}$ is assigned to $t_e^{O,t}$ and break. One such Category3 schedule is the second Category3 schedule shown in Fig. 10c. After all schedules are examined, nonoverloadable period t_e^{NO} is updated to account for the period between $t_e^{O,t}$ and $t^{B,f}$. Finally, replication cost of the current backup schedule is returned which is the ratio of the nonoverloadable amount over its execution time.

```
1.  $t_e^{NO} \leftarrow 0$, $t_e^{O,t} \leftarrow t_s$. /*Initialization*/
2.  Overloadable backup schedules are browsed in non-decreasing
    order of their start time.
3.          case Category1
4.              $t_e^{O,t} \leftarrow t^{B,f}$; break.
5.          case Category2
6.              if $t_r > t_e^{O,t}$ then
7.                  $t_e^{O,t} \leftarrow t_r$.
8.          case Category3
9.              if $t_r > t_e^{O,t}$ then
10.                 if $t_l > t_e^{O,t}$ then
11.                     $t_e^{NO} \leftarrow t_e^{NO} + t_l - t_e^{O,t}$.
12.                 $t_e^{O,t} \leftarrow t_r$.
13.                 if $t_e^{O,t} \geq t^{B,f}$ then
14.                     $t_e^{O,t} \leftarrow t^{B,f}$; break.
15. $t_e^{NO} \leftarrow t_e^{NO} + t^{B,f} - t_e^{O,t}$.
16. return $t_e^{NO}/t_e$.
```

Fig. 11. Algorithm to calculate replication cost.

6.7 Complexity Analysis

In this section, we analyze the complexity of our algorithms MRC-ECT and MCT-LRC. Suppose we consider scheduling a backup of task j. Let $N_{tw_e}(j)$ denote the number of existing schedules that are within or overlapping with the time window for scheduling task j on a processor P_e. Let $N_{bs_e}(j)$ denote the maximum number of existing schedules that are within or overlapping with any boundary schedule within the time window. Let $N_{tw}(j)$ denote the maximum number of schedules that are within or overlapping with the time window on any processor. Let $N_{bs}(j)$ denote the maximum number of schedules that are within or overlapping with a boundary schedule on all processors, where $N_{bs}(j) = \max_{P_e \neq P_P(j)}\{N_{bs_e}(j)\}$.

Schedule eligibility can be determined in $O(N_{bs_e}(j))$ as there are maximum $N_{bs_e}(j)$ schedules to be considered for a backup schedule. Replication cost can be calculated in $O(N_{bs_e}(j))$ as there are maximum $N_{bs_e}(j)$ overloadable backup schedules to be considered for a backup schedule. As for each existing schedule within a time window, both eligibility and replication cost need to be determined in the worst case, its complexity is $O(N_{bs_e}(j))$. As both algorithms need to examine all existing schedules in the worst case, their complexity is $O(N_{tw_e}(j) \cdot N_{bs_e}(j))$. Therefore, the overall complexity of algorithms MRC-ECT and MCT-LRC to schedule a backup in the Grid system is $O(M \cdot N_{tw}(j) \cdot N_{bs}(j))$.

Note that $N_{bs}(j)$ is usually much smaller than $N_{tw}(j)$. Furthermore, even scheduling a primary using ASAP takes $O(M \cdot N_{tw}(j))$ while the time taken to schedule a backup using ASAP depends on sampling rate and is not fixed.

7 Performance Study

In this section, we evaluate the performance of our algorithms and strategies. We first present performance metrics and simulation parameter used. Then, experimental results are presented where jobs are independent, dependent, or hybrid. We use ASAP to schedule primaries of both independent and dependent tasks. MRC-ECT and MCT-LRC are used to schedule backups of independent and dependent tasks, respectively. We compare with algorithm eFRD [17] and the approach where both primaries and backups are scheduled using ASAP. ALAP is not used for comparison as it tries to schedule backups of tasks right before the deadline, and thus, not suitable for dependent tasks.

7.1 Performance Metrics

For our formulation and scope of study, the metrics that are natural to consider are job rejection ratio, average replication cost, average backup response time, and average reliability.

Job rejection ratio is defined as the percentage of jobs that are rejected. A job is rejected if primary or backup of any its task cannot be scheduled before deadline. This metric reflects the ability of an algorithm in scheduling dynamic jobs under deadline requirements.

Replication cost of a job is defined as the ratio of weighted sum of replication cost of its tasks over cumulative execution time of its tasks. This metric reflects the ability of a fault-tolerant algorithm in minimizing processor time used for backup.

Response time of a job is defined as the amount of time it takes to execute its tasks since its arrival. In failure-free situations, this time depends on the latest completion time of primaries of its tasks. When failure happens, this time depends on the latest completion time of backups of its tasks, which is referred to as backup response time. This metric reflects the ability of a fault-tolerant algorithm in minimizing completion time for jobs when failure occurs.

Reliability is defined as the percentage of jobs that can be recovered when a processor failure occurs. This metric reflects the ability of an algorithm in fault tolerance.

7.2 Simulation Parameters

Our simulation was done by implementing a discrete-event simulator where the events driving the simulation are the arrival and completion of a job as well as occurrence of processor faults as in [11]. Job arrival is assumed to follow Poisson process with rate λ and the execution time of a job is assumed to be exponentially distributed with a mean of $1/\mu$. Obviously, workload is defined as λ/μ [13]. In our experiments, 100,000 DAGs are made to arrive at a Grid comprising a number of processors. The numbers of processors chosen are 16, 80, 400, and 2,000. These processors have heterogeneous processing speeds, which is assumed to be uniformly distributed in the range [1.0, 10.0]. The *computational time* of a task on a processor is the ratio of its execution time over processing speed of this processor as mentioned in [21].

Each DAG has a number of tasks (denoted as n) randomly chosen from the set {20, 40, 60, 80, 100}. For each DAG, a random graph is generated with n nodes and graph connectivity randomly chosen assuming a uniform distribution from 1 percent to 100 percent (fully connected). The execution time of a task is assumed to be uniformly distributed with a mean of t_{exe}/n, where t_{exe} is the execution time of the job it belongs to. The deadline of a job is also assumed to be uniformly distributed with a mean $t_a + \eta \cdot \frac{2 \cdot t_{exe}}{5.5}$, where t_a is the arrival time of this job, η is a parameter less than 1.0, and 5.5 is the mean processing speed. Similar assumption is made in [12] and [13] for independent tasks without taking processing speed into account. In our experiments, η is varied between 0.2 and 0.3 to show its effect on performance.

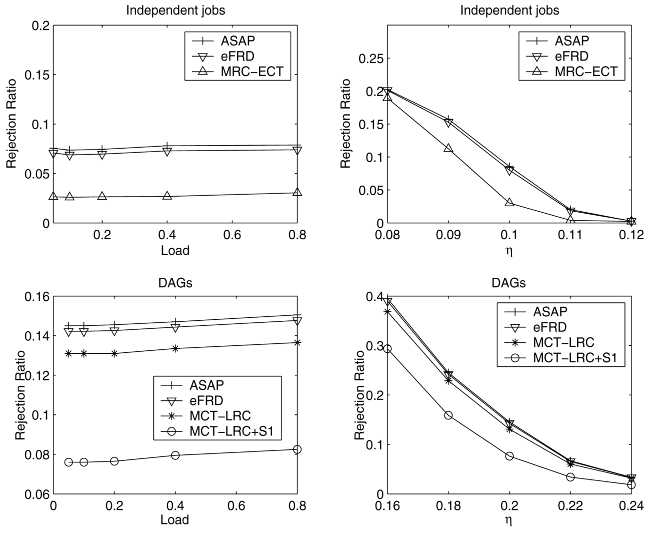


Fig. 12. Rejection ratio of independent jobs and DAGs.

7.3 Results and Discussions

The results are organized in five parts. The first three parts present results on rejection ratio, replication cost, and response time for independent and dependent tasks. The fourth part presents results on effect of connectivity of dependent tasks on performance. The last part presents results on hybrid jobs. In order to obtain stable and accurate results with a confidence of 95 percent, each of our experiments is repeated 25 times and we take an average measure. In the following, the results on a Grid comprising 16 processors are shown and similar trends on performance are observed on Grids with 80, 400, and 2,000 processors.

7.3.1 Rejection Ratio

In Fig. 12, we observe that our algorithms MRC-ECT and MCT-LRC perform better than algorithms ASAP and eFRD for independent and dependent jobs, respectively. This is because our algorithms consider boundary schedules for backups which have less replication cost; furthermore, MRC-ECT is guaranteed to find an optimal schedule for each backup in terms of replication cost. The rejection ratio of DAGs is much higher than that of independent jobs due to precedence constraint. Our strategy based on Maximum Failure Recovery Time (denoted as S1 in the figure) can reduce rejection ratio significantly for DAGs by improving backup schedulability. While rejection ratio increases slowly when load increases, it decreases sharply when η increases as deadline is relaxed. When η is large enough, almost all jobs can be scheduled successfully and algorithms exhibit similar performance. On the other hand, when η is very small, a number of jobs cannot be scheduled within the deadline by all algorithms.

In Fig. 13, we focus on our algorithms, and it can be observed that most of jobs are rejected because backups cannot be scheduled before deadline for both kinds of jobs. This is because the primary is scheduled before its backup and sometimes there may not exist time slots before its deadline on two different processors. As a result, the backup cannot be scheduled. Furthermore, for DAGs, the two constraints

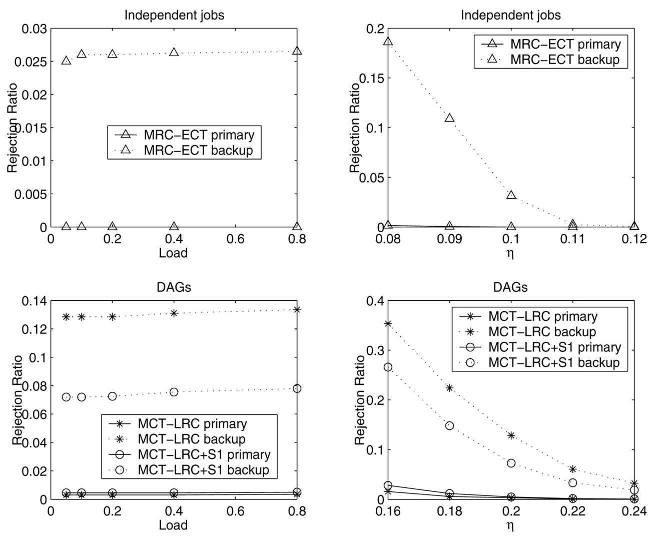


Fig. 13. Rejection ratio of primaries and backups of independent jobs and DAGs.

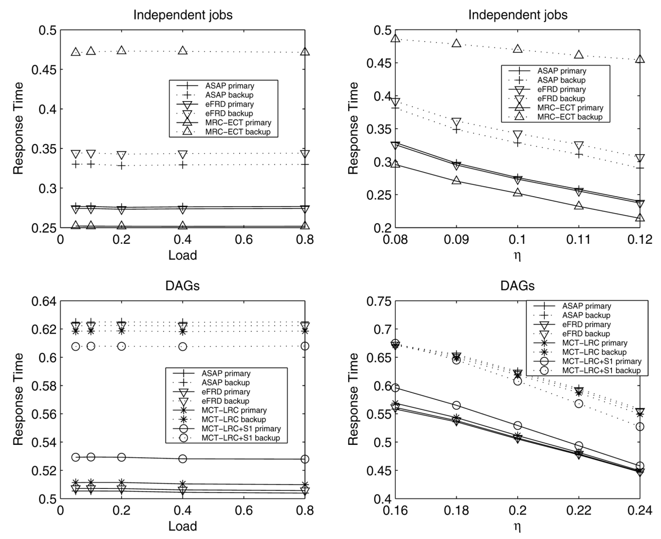


Fig. 15. Response time of independent jobs and DAGs.

discussed in Section 3 specify that the backup must be scheduled after certain time and cannot be scheduled on certain processors. In either case, the backup may not be able to be scheduled before the deadline. Finally, we observe that our strategy based on Maximum Failure Recovery Time can reduce rejection caused by backups significantly for DAGs.

7.3.2 Replication Cost

In Fig. 14, we observe that our algorithm MRC-ECT, which is guaranteed to find an optimal schedule in terms of replication cost, performs better than algorithms ASAP and eFRD significantly for independent jobs. MCT-LRC performs better than algorithms ASAP and eFRD for DAGs. Replication cost of DAGs is much higher than that of independent jobs, and the reason has been discussed in Sections 3.3 and 4.2. We can also observe that replication cost increases slowly when load increases, but sharply when η increases. The latter is because when η is smaller, more jobs have to be scheduled in a period of time and thus overloading efficiency of their backups is improved leading to less replication cost.

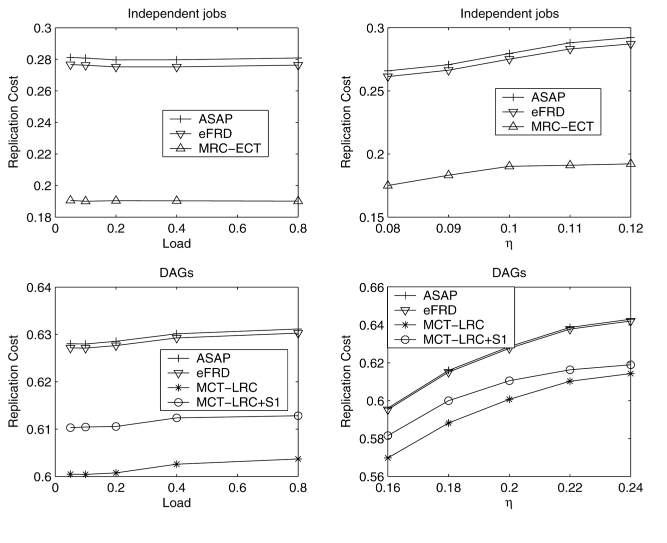


Fig. 14. Replication cost of independent jobs and DAGs.

7.3.3 Response Time

In Fig. 15, it can be observed that response time of backups is longer than that of primaries for both kinds of jobs. For independent jobs, algorithms ASAP and eFRD have shorter backup response time than algorithm MRC-ECT which takes replication cost priority over completion time. However, algorithm MRC-ECT performs better than algorithms ASAP and eFRD for primary response time. This is because in MRC-ECT backup schedules are overloaded more tightly, and thus, more idle slots are available for primaries. For DAGs, our algorithms and algorithms ASAP and eFRD perform closely for both primary and backup response times. We can observe that response time keeps unchanged when load increases, but decreases sharply when η increases. The latter is because when η is smaller, more jobs are scheduled in a period of time and primaries and backups of an arriving job may have to start later as earlier slots are occupied by existing schedules.

7.3.4 Effect of DAG Connectivity

In Fig. 16, we consider the performance of DAGs with different graph connectivity with respect to η. Algorithm MCT-LRC and strategy1 are used. It can be observed that when graph connectivity is high, the performance degrades in terms of rejection ratio, replication cost, and response time. This is because when graph connectivity is high, a task generally has more predecessors which makes the earliest possible start time of its primary and backup become later and reduces its backup overloading efficiency. Grid users can use this figure to participate the performance of DAGs submitted by them with information on graph connectivity and deadline requirements.

7.3.5 Hybrid Jobs

In Fig. 17, we consider hybrid jobs which comprising 40 percent independent jobs and 60 percent DAGs. Algorithm MRC-ECT is used for independent jobs and

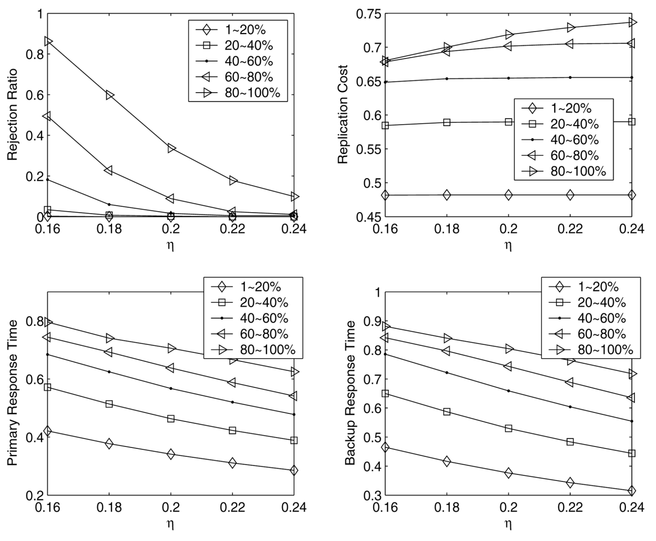


Fig. 16. Effect of graph connectivity on rejection ratio, replication cost, primary, and backup response time of DAGs.

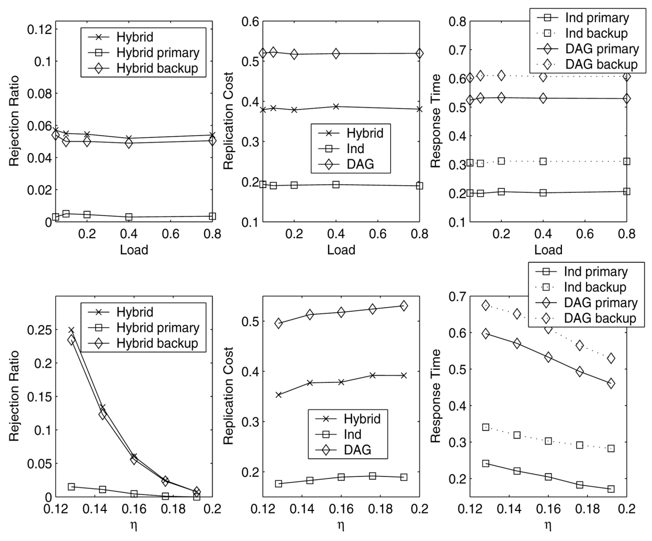


Fig. 17. Rejection ratio, replication cost, and response time of hybrid jobs.

algorithm MCT-LRC and S1 are used for DAGs. Interleaving technique (denoted as S2) is used for both kinds of jobs. The range of η displayed is determined based on η and percentage of the two kinds of jobs. We will compare the performance of hybrid jobs to that of independent jobs and DAGs. It can be observed that the rejection ratio of hybrid jobs is between that of independent jobs and DAGs. Similarly, most of jobs are rejected because backups are not available. Replication cost of hybrid jobs is between that of independent jobs and DAGs. We can observe that our interleaving technique can reduce the replication cost of DAGs significantly (about 10 percent), without compromising the replication cost of independent jobs. Response time of DAGs in hybrid jobs is similar to the case of only DAGs. However, the response time of independent jobs in hybrid jobs is significantly better than that of only independent jobs. This is because while primaries and backups of DAGs cannot be scheduled on some earlier slots due to precedence constraint, primaries and backups of independent jobs can occupy these slots leading to reduced response time.

8 CONCLUSIONS

In this paper, for Grid systems, we addressed the problem of fault-tolerant scheduling of independent, dependent, and hybrid jobs. We analyzed the impact of precedence constraints on scheduling and overloading of backups of dependent tasks and showed that this constraint indeed limits *schedulability and overloading efficiency* significantly. We have proposed and analyzed algorithms for each task category to improve the performance. Our algorithms for independent and dependent tasks (MRC-ECT and MCT-LRC, respectively) do not require sampling, as required by traditional algorithms to perform backup overloading. The time complexity of our algorithms is shown to be polynomial and can schedule backups in a much faster way. While algorithm MRC-ECT is guaranteed to find an optimal backup schedule for an independent task, it was shown that MCT-LRC can schedule a backup of a dependent task with minimum completion time and less replication cost.

Through extensive simulation experiments, we observed that our algorithms perform better than algorithms ASAP and eFRD for both kinds of jobs in terms of rejection ratio and replication cost. Our strategy exploring information on maximum failure recovery time can improve schedulability of backups of dependent tasks significantly leading to reduced job rejection ratio. When jobs are hybrid, our interleaving technique can reduce the replication cost of DAGs considerably while not compromising the replication cost of independent jobs. Furthermore, the response time of independent jobs is reduced significantly.

Our work can be adopted by Globus toolkit in handling failures for independent, dependent, and hybrid jobs in Grid systems. Immediate extensions to this work could be to conduct experimental works correlating with the parameter settings of the proposed algorithms. One may consider Grid systems where computing and data resources are geographically dispersed in different administrative domains with local scheduling entity. One may bring in issues that are akin to communication protocols to implement our schemes in real-life environments. Finally, cooperative computing being plausible on Grid systems owing to large resource availability, it would be interesting to fine-tune our strategies to take care of such cooperative computations.

ACKNOWLEDGMENTS

This work was supported in part by Grant 052 015 0024/R-263-000-350-592 from the National Grid Office via A*Star, Singapore.

REFERENCES

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a Future Computing Infrastructure.* Morgan Kaufmann, 2004.
[2] S. Hwang and C. Kesselman, "A Flexible Framework for Fault Tolerance in the Grid," *J. Grid Computing,* vol. 1, pp. 251-272, 2003.

[3] R. Medeiros, W. Cirne, F. Brasileiro, and J. Sauve, "Faults in Grids: Why Are They So Bad and What Can Be Done About It?" *Proc. Fourth Int'l Workshop Grid Computing (GRID),* 2003.

[4] X. Zhang, D. Zagorodnov, M. Hiltunen, K. Marzullo, and R.D. Schlichting, "Fault-Tolerant Grid Services Using Primary-Backup: Feasibility and Performance," *Proc. IEEE Int'l Conf. Cluster Computing (CLUSTER '04),* pp. 105-114, 2004.

[5] M.J. Gonzalez, "Deterministic Processor Scheduling," *ACM Computing Surveys,* vol. 9, no. 3, pp. 173-204, 1997.

[6] A. Iamnitchi and I. Foster, "A Problem-Specific Fault-Tolerance Mechanism for Asynchronous, Distributed Systems," *Proc. Int'l Conf. Parallel Processing (ICPP),* 2000.

[7] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Cluster Computing,* vol. 5, no. 3, 2002.

[8] A. Natrajan, M. Humphrey, and A. Grimshaw, "Grids: Harnessing Geographically-Separated Resources in a Multi-Organisational Context," *Proc. High Performance Computing Systems,* 2001.

[9] B. Lee and J.B. Weissman, "Dynamic Replica Management in the Service Grid," *Proc. 10th IEEE Symp. High Performance Distributed Computing (HPDC),* 2001.

[10] N. Budhiraja, K. Marzullo, F.B. Schneider, and S. Toueg, "Primary-Backup Protocols: Lower Bounds and Optimal Implementations," *Proc. Third IFIP Conf. Dependable Computing for Critical Applications (DCCA),* 1992.

[11] S. Ghosh, R. Melhem, and D. Mosse, "Fault-Tolerance through Scheduling of Aperiodic Tasks in Hard Real-Time Multiprocessor Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 3, pp. 272-284, Mar. 1997.

[12] G. Manimaran and C.S.R. Murthy, "A Fault-Tolerant Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems and Its Analysis," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 11, pp. 1137-1152, 1998.

[13] R. AI-Omari, A.K. Somani, and G. Maninaran, "A New Fault-Tolerant Technique for Improving Schedulability in Multiprocessor Real-Time Systems," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* 2001.

[14] J.H. Abawajy, "Fault-Tolerant Scheduling Policy for Grid Computing Systems," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS),* 2004.

[15] Y. Oh and S.H. Son, "Scheduling Real-Time Tasks for Dependability," *J. Operational Research Soc.,* vol. 48, no. 6, pp. 629-639, 1997.

[16] X. Qin, H. Jiang, and D. Swanson, "An Efficient Fault-Tolerant Scheduling Algorithm for Real-Time Tasks with Precedence Constraints in Heterogeneous Systems," *Proc. Int'l Conf. Parallel Processing (ICPP),* 2002.

[17] X. Qin and H. Jiang, "A Novel Fault-Tolerant Scheduling Algorithm for Precedence Constrained Tasks in Real-Time Heterogeneous Systems," *Parallel Computing,* vol. 32, nos. 5/6, pp. 331-356, 2006.

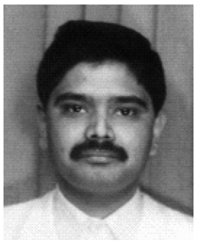
[18] P.G. Paulin and J.P. Knight, "Force Directed Scheduling for the Behavioral Synthesis of Asics," *IEEE Trans. Computer-Aided Design,* vol. 8, no. 6, pp. 661-679, June 1989.

[19] C. Tseng and D. Siewoirek, "Automated Synthesis of Data Paths in Digital Systems," *IEEE Trans. Computer-Aided Design,* vol. 5, no. 3, pp. 379-395, July 1986.

[20] P. Marwedel, "A New Synthesis Algorithm for the Mimola Software System," *Proc. 23rd Design Automation Conf. (DAC '86),* pp. 271-277, 1986.

[21] L. He, S.A. Jarvis, D.P. Spooner, H. Jiang, D.N. Dillenberger, and G.R. Nudd, "Reliability Driven Task Scheduling for Heterogeneous Systems," *Proc. IASTED Int'l Conf. Parallel and Distributed Computing and Systems (PDCS '03),* pp. 465-470, 2003.

Qin Zheng received the BEng degree in information engineering from Xi'an Jiaotong University, Xi'an, China, in 2001 and the PhD degree in electrical and computer engineering from the National University of Singapore, in 2006. He then worked as a research fellow in the Department of Electrical and Computer Engineering, National University of Singapore until November 2007 when he joined the Advanced Computing Programme, Institute of High Performance Computing, Agency for Science, Technology and Research (A*STAR), Singapore. His research interests include scheduling and pricing in Grid computing, and routing and survivability in MPLS and WDM optical networks. He is a member of the ACM, the IEEE, and the IEEE Computer Society.



Bharadwaj Veeravalli received the BS degree in physics from Madurai-Kamaraj University, India, in 1987 and the master's degree in electrical communication engineering and the PhD degree from the Indian Institute of Science, Bangalore, India, in 1991 and 1994, respectively. He did his postdoctoral research in the Department of Computer Science, Concordia University, Montreal, in 1996. He is currently with the Department of Electrical and Computer Engineering, Communications and Information Engineering (CIE) Division, National University of Singapore, as a tenured associate professor. His main stream research interests include multiprocessor systems, Cluster/Grid computing, scheduling in parallel and distributed systems, bioinformatics and computational biology, and multimedia computing. He is one of the earliest researchers in the field of divisible load theory (DLT). He has published more than 85 papers in high-quality international journals and conferences. He had successfully secured several externally funded projects. He has coauthored three research monographs in the areas of PDS, Distributed Databases (competitive algorithms), and Networked Multimedia Systems, in the years 1996, 2003, and 2005, respectively. He had guest edited a special issue on Cluster/Grid Computing for *International Journal of Computers and Applications (IJCA)*, USA journal in 2004. He is currently on the editorial board of the *IEEE Transactions on Computers*, the *IEEE Transactions on SMC-A*, and *IJCA*, USA, as an associate editor. He had served as a program committee member and as a session chair in several international conferences and had been recently invited to contribute to Multimedia Encyclopedia, Kluwer Academic Publishers, 2005. His academic profile, professional activities, main stream and peripheral research interests, research projects and collaborations, and most recent list of publications can be found in http://cnds.ece.edu.sg/elebv. He is a senior member of the IEEE and the IEEE Computer Society.

Chen-Khong Tham received the MA and PhD degrees in electrical and information sciences engineering from the University of Cambridge, United Kingdom. He is an associate professor in the Department of Electrical and Computer Engineering (ECE), National University of Singapore (NUS), and a senior scientist and the department head of the Networking Protocols Department, Institute for Infocomm Research (I2R), Singapore. He is the program manager of a research program on UWB-enabled Sentient Computing (UWB-SC) funded by A*STAR Singapore. His research interests include coordinated quality-of-service (QoS) management in wired and wireless computer networks and distributed systems, such as wireless sensor networks. He held a 2004/05 Edward Clarence Dyason Universitas21 Fellowship at the University of Melbourne. He is a member of the IEEE.