QoS-Aware Fault-Tolerant Scheduling for Real-Time Tasks on Heterogeneous Clusters

Xiaomin Zhu, *Member, IEEE*, Xiao Qin, *Senior Member, IEEE*, and Meikang Qiu, *Senior Member, IEEE*

Abstract—Fault-tolerant scheduling plays a significant role in improving system reliability of clusters. Although extensive fault-tolerant scheduling algorithms have been proposed for real-time tasks in parallel and distributed systems, quality of service (QoS) requirements of tasks have not been taken into account. This paper presents a fault-tolerant scheduling algorithm called QAFT that can tolerate one node's permanent failures at one time instant for real-time tasks with QoS needs on heterogeneous clusters. In order to improve system flexibility, reliability, schedulability, and resource utilization, QAFT strives to either advance the start time of primary copies and delay the start time of backup copies in order to help backup copies adopt the passive execution scheme, or to decrease the simultaneous execution time of the primary and backup copies of a task as much as possible to improve resource utilization. QAFT is capable of adaptively adjusting the QoS levels of tasks and the execution schemes of backup copies to attain high system flexibility. Furthermore, we employ the overlapping technology of backup copies. The latest start time of backup copies and their constraints are analyzed and discussed. We conduct extensive experiments to compare our QAFT with two existing schemes—NOQAFT and DYFARS. Experimental results show that QAFT significantly improves the scheduling quality of NOQAFT and DYFARS.

Index Terms—Heterogeneous clusters, real-time, scheduling, fault tolerance, quality of service (QoS), heuristic.

1 Introduction

Heterogeneous clusters, consisting of a diverse computers interconnected by high-speed networks, have been adopted as an efficient high-performance computing platforms for computing-intensive [1] and data-intensive applications [2], [3]. Besides, real-time application development and deployment have been conducted on heterogeneous clusters, in which the correctness of the systems depends not only on the logic results of computation, but also on the time instants at which these results are produced [4]. Examples of real-time systems include automated flight control systems, systems for monitoring patients in critical condition, signal processing systems, etc. Real-time systems, in which missing a deadline may be catastrophic, are called hard real-time systems, an example being automated flight control systems. Another category is called soft real-time systems, like multimedia systems, where nothing catastrophic happens if a deadline is missed [5].

In addition to timeliness requirements, quality of service (QoS) requirements must be addressed in various hard and soft real-time systems. It should be noted that the QoS in this paper refers to the task quality after proceeded. For example, Atdelzater et al. studied an automated flight control system [6] utilized to fly a simulated model of an F-16 flight aircraft. In this system, all the fight control tasks including Guidance, Controller, Slow Navigation, Fast Navigation, and Missile Control need to be completed within deadlines. In order to improve the stability of the system, each task selects different QoS levels by varying its period or execution time. Different QoS levels provide different flight qualities. More details of the automated flight control system can be found in [7]. Another example, taken from [41], is a real-time signal processing system, in which signal data can be processed using different algorithms. For example, a wide range of algorithms can be employed for decoding block turbo codes [8], [9], [10]. High-complexity algorithms can guarantee that signal processing has a higher QoS level (higher data accuracy) at the expense of processing time, but low-complexity algorithms produce the opposite.

Noticeably, the aforementioned QoS-aware real-time systems must then incorporate inherent high reliability features. Since the automated flight control system is used in the military battle field, the system must ensure that each task is executed within its deadline even in the presence of hardware or software faults. For the signal processing system, although some tasks missing their deadlines may not result in disaster, the outdated or half-baked processed data may be useless for users, especially in the field of modern information battle. Therefore, the system must guarantee its functional and timing correctness even when faults occur. Consequently, providing a fault-tolerant mechanism for such systems is mandatory due to the critical nature of tasks in the systems.

Motivation. Growing evidence shows that scheduling is an efficient approach to achieving high performance of applications in parallel systems like clusters. A wide variety of scheduling algorithms have been developed to provide fault tolerance for clusters supporting real-time applications in the past decade (see, for example, [11], [12]). Many QoS-based scheduling algorithms were proposed for real-time applications in distributed systems in addition to parallel systems [13], [14], [15].

- *X. Zhu is with the Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha, Hunan 410073, P.R. China. E-mail: xmzhu@nudt.edu.cn.*
- *X. Qin is with the Department of Computer Science and Software Engineering, 3101E Shelby Center, Auburn University, Auburn, AL 36849-5347. E-mail: xqin@auburn.edu.*
- *M. Qiu is with the Department of Electrical and Computer Engineering, University of Kentucky, Lexington, KY 40506. E-mail: mqiu@engr.uky.edu.*

Manuscript received 26 Sept. 2010; revised 25 Jan. 2011; accepted 1 Mar. 2011; published online 17 Mar. 2011.
Recommended for acceptance by J. Wu.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2010-09-0537.
Digital Object Identifier no. 10.1109/TC.2011.68.

Unfortunately, to the best of our knowledge, no work has been done on fault-tolerant scheduling for real-time tasks with QoS requirements. It is a challenge to design and implement novel QoS-aware fault-tolerant scheduling algorithms for real-time tasks running on clusters in general and heterogeneous clusters specifically. This challenge is motivation to integrate fault tolerance with QoS-based scheduling by developing a dynamic fault-tolerant scheduling algorithm or QAFT for aperiodic, independent, real-time, and QoS-aware tasks on heterogeneous clusters.

Contributions. The major contributions of this study are summarized as follows:

- For the first time, we develop a novel QoS-aware fault-tolerant model, which extends the conventional primary-backup (PB) fault-tolerant model.
- We design a new dynamic real-time QoS-aware fault-tolerant scheduling algorithm, QAFT, to support heterogeneous clusters.
- We demonstrate that, by considering heterogeneous features of clusters, we can design a real-time dynamic scheduling algorithm that significantly improves the scheduling quality of conventional scheduling algorithms for heterogeneous clusters.

The rest of this paper is organized as follows: Section 2 reviews related work in literature. Section 3 presents the system model with fault tolerance and QoS requirements. Section 4 describes the QAFT algorithm and the main principles behind it. Section 5 gives simulation experiments and performance analysis. Section 6 concludes the paper with a summary and future work.

2 Related Work

A large number of fault-tolerant techniques have been developed because of the importance of fault tolerance in safety-critical real-time systems. For example, Weber studied a fault-tolerant technique using replicated hardware components [16]. Kim and Damm proposed a hybrid approach to integrating software checks at the end of hardware computation cycles [17]. To achieve fault masking of permanent hardware faults, one can use redundant concurrent tasks to carry out synchronous or asynchronous computations.

Scheduling is a popular way to facilitate fault tolerance by allocating multiple copies of tasks on different processors [18]. Efficient fault-tolerant scheduling algorithms are capable of improving the schedulability, reliability, and flexibility of computer systems. Unfortunately, many practical instances of scheduling problems have been found to be NP-complete [22]. These problems motivate us in order to investigate heuristic algorithms to address the scheduling issues.

Fault-tolerant scheduling algorithms can be either static (i.e., offline) or dynamic (i.e., online). In static algorithms, assignments of tasks to nodes and the time at which the tasks start to execute are determined a priori [11], [23]. Static fault-tolerant scheduling algorithms are suitable for periodic tasks [18]. However, aperiodic tasks whose arrival times are not known a priori must be scheduled by dynamic fault-tolerant scheduling algorithms [24], [25], [26]. Our work is focused on scheduling aperiodic and independent real-time tasks. Nevertheless, our approach can be applied to dependent tasks because tasks having precedence constraints are equivalent to independent tasks by modifying ready times and deadlines of dependent tasks [18]. Moreover, fault-tolerant scheduling algorithms can be classified into two groups: preemptive scheduling [27], [28] and nonpreemptive scheduling [25], [26], [32], [33]. In a preemptive scheduling approach, an executing task can be preempted by another task. In a nonpreemptive scheduling mechanism, running tasks cannot be interrupted during their executions and a task can be run only after the running tasks are completed. Although preemptive scheduling is able to achieve high system utilization, in many real-world cases, hardware devices and software configuration make preemptions either impossible or prohibitively expensive. Nonpreemptive scheduling, on the other hand, has the advantages of accurate response time analysis, ease of implementation, no synchronization overhead, and reduced stack memory requirements [19], [20], [33]. Nonpreemptive scheduling has proven to be beneficial in many applications, such as multimedia applications [21]. Therefore, in this study, we focus on nonpreemptive scheduling.

Increasing attention has been directed toward dynamic fault-tolerant scheduling algorithms using the primary-backup model (or PB in short). In the PB model, two copies of one task are scheduled on two different nodes, and an acceptance test is used to check the correctness of schedules [11]. Ghosh et al. proposed two techniques called deallocation and overloading, to improve schedulability while providing fault tolerance with a low overhead. Multiple backup copies in the overloading scheme may overlap in the same time slot on the same processor. The deallocation scheme is the reclamation of resources reserved for backup copies when the corresponding primary copies are completed successfully [29]. Manimaran et al. extended the algorithm proposed in [29] by: 1) considering resource constraints among tasks and, 2) partitioning processors into groups to tolerate more than one failure at a time [18]. Al-Omari et al. studied a PB overloading technique which allows the primary copy of one task to overlap with the backup copy of another task to achieve high schedulability [30]. One of the common characteristics in the aforementioned algorithms is that the backup copy of a task is permitted to execute only if a fault occurs in the primary task. This strategy is referred to as the *passive backup-copy* scheme, in which real-time tasks must have enough laxity to restart their backup copies. Thus, the laxity is larger than the computation time of tasks. An important assumption in the passive backup-copy schemes is that the laxity of a task must be at least twice as large as its computation time. However, this assumption is unrealistic in practice, particularly when real-time systems are heavily loaded.

Unlike the passive backup-copy scheme, the *active backup-copy* scheme is adequate for tasks with small laxities. For example, Tsuchiya et al. proposed a technique in which two copies of each task are concurrently executed with different start times [31]. Yang et al. studied a fault-tolerant scheduling algorithm where the two copies of a task were executed simultaneously in order to improve schedulability [32]. Al-Omari et al. investigated an adaptive scheme that controlled the overlap interval between the primary copy and backup copy of a task based on the primary-fault probability and the task's laxity [33]. Although these methods overcome the drawbacks of the passive backup-copy scheme, the existing scheduling solutions are still inadequate for a heterogeneous computing environment.

Luo et al. proposed a dynamic and reliability-driven real-time fault-tolerant scheduling algorithm—DYFARS—in heterogeneous systems [34]. DYFARS considers both active and passive backup copies, thereby providing high system flexibility. However, the DYFARS algorithm does not take the QoS requirements of real-time tasks into account while supporting fault tolerance.

In this paper, we pay our attention on the issues of nonpreemptive fault-tolerant scheduling of real-time aperiodic tasks with QoS needs on heterogeneous clusters. The new dynamic algorithm allows a heterogeneous cluster to tolerate one node's failure. Given a dynamically changing load, our algorithm adaptively switches between the active backup-copy scheme and passive backup-copy scheme. To achieve high flexibility and schedulability, the algorithm maximizes the QoS benefits of real-time tasks by adjusting the tasks' QoS levels.

3 Scheduling Model

In this section, we will introduce the models, notions, and terminology used in this paper.

3.1 Fault Model

The fault-tolerance problem addressed in this study is outlined as a fault model with the following three features [11]:

- At one time instant, only one node can be failed. Before another node is failed, those failures of primary copies of tasks on the fault node can be successfully executed by their backup copies.
- Faults can be transient or permanent and are independent, i.e., a fault on one node will not affect the other nodes.
- There exists a fault-detection mechanism, such as acceptance tests, to detect node failures. The scheduler will not schedule tasks to a known faulty node any more.

This fault model can be easily extended to tolerate multiple nodes' faults in a large-scale system with an extremely large number of nodes. The extension can be processed by the following two steps. First, the large set of nodes are divided into several small groups. Second, the fault model outlined above is employed in each group [30].

3.2 Scheduler Model

A scheduler can be implemented in a distributed or a centralized manner. In the distributed scheduler model, tasks arrive independently at each local scheduler, which produces schedules in parallel with other schedulers. In the centralized scheduler model, all tasks arrive at a central node called scheduler, from which the tasks are distributed to nodes in a cluster for further execution.

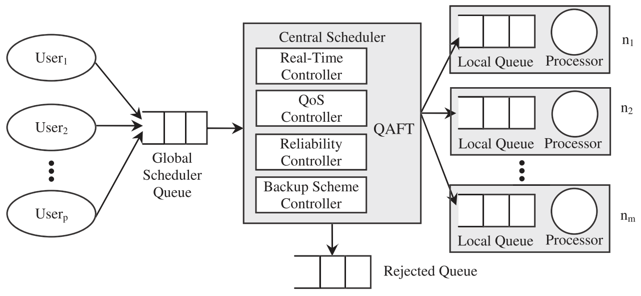


Fig. 1. Scheduler model.

In this study, we focus on the centralized scheduler model because centralized schedulers have two attractive features compared with distributed schedulers. First, it is straightforward to design a centralized fault-tolerant scheduler using a backup scheduler that concurrently executes with the primary scheduler. Second, the implementation of a centralized scheduler is simpler and easier than that of a distributed scheduler [37]. Therefore, we use in this study the centralized scheduler model (see Fig. 1) like those described in the literature [4], [11], [40], [42], [43].

3.3 Task Model

We consider a set $T = \{t_1, t_2, \dots, t_n\}$ of real-time tasks that are independent and aperiodic. Since the primary-backup technique is used in our fault-tolerant scheduling scheme, each task t_i has two copies (i.e., t_i^P and t_i^B) executed on two different nodes.

A heterogeneous cluster in this study is composed of a set $N = \{n_1, n_2, \dots, n_m\}$ of heterogeneous nodes with different processing powers connected by high-speed interconnects such as Myrinet and InfiniBand.

Given a task $t_i \in T$, we denote the arrival time and deadline of task t_i as a_i and d_i, respectively. Let f_i^P and f_i^B be the finish times of primary copy t_i^P and backup copy t_i^B, respectively. l_i^P denotes the laxity of t_i^P, i.e., $l_i^P = d_i - f_i^P$. Let $EST^P = (est_{ij}^P)_{n\times m}$ be an earliest start time matrix of the tasks' primary copies, where element est_{ij}^P denotes the earliest start time of t_i^P on node n_j. Likewise, $LST^B = (lst_{ij}^B)_{n\times m}$ is a latest start time matrix of the tasks' backup copies, where element lst_{ij}^B denotes the latest start time of t_i^B on node n_j. $n(t_i^P)$, and $n(t_i^B)$ represents the nodes to which t_i^P and t_i^B are allocated, respectively.

Let $Z^P = (z_{ij}^P)_{n\times m}$ be a binary matrix, where z_{ij}^P equals 1 if and only if t_i^P has been allocated to node n_j, otherwise $z_{ij}^P = 0$. Similarly, $Z^B = (z_{ij}^B)_{n\times m}$ is also a binary matrix, in which an element z_{ij}^B equals to 1, if and only if, t_i^B has been assigned to n_j, otherwise z_{ij}^B equals 0. Consequently, $n(t_i^P) = j \Leftrightarrow z_{ij}^P = 1$ and $n(t_i^B) = j \Leftrightarrow z_{ij}^B = 1$. After t_i^P is allocated to n_j, if n_j does not fail until t_i^P is successfully finished, then $o_{ij}^P = 1$, or $o_{ij}^P = 0$. If t_i^P fails to be finished successfully, but its corresponding backup copy t_i^B can be successfully finished, then $o_{ij}^B = 1$, else $o_{ij}^B = 0$.

The QoS requirement of a task is specified by users or application developers. We model QoS requirements using a QoS level set $Q = \{q_1, q_2, \dots, q_k\}$, where $q_1 < q_2 < \cdots < q_k$. In this paper, we assume all the tasks belong to a single application (e.g., tasks in a real-time signal processing application). Thus, these tasks share the same QoS requirement scope. X_i^P and X_i^B represent all possible schedules for primary copy t_i^P and backup copy t_i^B, respectively. $x_i^P \in X_i^P$ is a scheduling decision of t_i^P. Similarly, $x_i^B \in X_i^B$ is a scheduling decision of t_i^B. The QoS level of t_i^P adopting the scheduling decision x_i^P can be represented by $q(x_i^P)$. As such, $q(x_i^B)$ is the QoS level of t_i^B using the scheduling decision x_i^B. x_i^P and x_i^B are feasible scheduling decisions if 1) $f_i^P \leq d_i$ and $f_i^B \leq d_i$; 2) $q_1 \leq q(x_i^P) \leq q_k$ and $q_1 \leq q(x_i^B) \leq q_k$. Let $e_{ij}(q(x_i^P))$ and $e_{ik}(q(x_i^B))$ denote the execution time of t_i^P using QoS level $q(x_i^P)$ on node n_j and the execution time of t_i^B using QoS level $q(x_i^B)$ on node n_k, respectively. We assume that execution times of tasks can be estimated and given a priori. This is a commonly used assumption in scheduling research [35]. It is a reasonable assumption since execution times can be estimated by code profiling and statistical prediction techniques (see, for example, [38], [39]). One of our scheduling objectives (see (1)) is to obtain the maximal QoS benefit, i.e., to maximize the QoS levels of all accepted tasks under timing constraints.

$$QB(X_i^P, X_i^B) = \max_{x_i^P \in X_i^P, x_i^B \in X_i^B} \{SQL/SST\}, \tag{1}$$

where $SQL = \sum_{j=1}^m \sum_{i=1}^n z_{ij}^P q(x_i^P) o_{ij}^P + \sum_{j=1}^m \sum_{i=1}^n z_{ij}^B q(x_i^B) o_{ij}^B$ and $SST = \sum_{j=1}^m \sum_{i=1}^n z_{ij}^p o_{ij}^p + z_{ij}^B o_{ij}^B$.

Recall that backup copies may be active or passive. Our scheme adaptively decides the backup-copy mode (i.e., active or passive) based on the laxities of primary copies. Let $s(t_{ij}^B)$ denote backup-copy mode of t_i^B on node n_j. The backup-copy mode can be expressed as

$$s(t_{ij}^B) = \begin{cases} passive, & \text{if } l_i^P \geq e_{ij}\big(q(x_i^B)\big), \\ active, & \text{else.} \end{cases} \tag{2}$$

3.4 Reliability Model

A reliability model can be used to quantitatively evaluate a system's level of fault tolerance [11], [34]. Reliability is defined as the probability that none of real-time tasks fail, even in the presence of hardware failures. Reliability cost is a very important metric for a system's reliability. We assume that fault arrival rate is constant and the distribution of the fault count for any fixed time interval is approximated using a Poisson distribution, which means the reliability cost can then be defined as [11], [34], [36]:

$$rc = \sum_{j=1}^m \sum_{i=1}^n \lambda_j z_{ij} e_{ij}, \tag{3}$$

where λ_j denotes the failure rate of node n_j. z_{ij} represents the allocation of task t_i on node n_j, $z_{ij} = 1$ means t_i is allocated to n_j, else $z_{ij} = 0$. Also, e_{ij} represents the execution time of task t_i on node n_j. It is worth noting that the above reliability model assumes that nodes in a cluster are fault-free, hence the model is unable to estimate the reliability of the cluster when one node fails.

The reliability-cost model neither reflects any QoS requirement of real-time tasks, nor does it consider the execution scheme of backup copies employing the PB technique. To further enhance the reliability of the real-time clusters, we propose an improved model that incorporates the PB fault-tolerant technology, QoS requirements, and execution schemes of backup copies. Equation 4 represents the reliability cost contributed by primary copies; (5) gives the reliability cost of backup copies.

$$rc(Z^P) = \sum_{j=1}^m \sum_{i=1}^n \lambda_j z_{ij}^P o_{ij}^P e_{ij}\big(q(x_i^P)\big), \tag{4}$$

$$\begin{aligned} rc(Z^B) = &\sum_{j=1}^m \sum_{i=1, o_{ij}^P=1, s(t_{ij}^B)=active}^n \lambda_j z_{ij}^B o_{ij}^B re_{ij}^B \\ &+ \sum_{j=1}^m \sum_{i=1, o_{ij}^P=0}^n \lambda_j z_{ij}^B o_{ij}^B e_{ij}\big(q(x_i^B)\big), \end{aligned} \tag{5}$$

where re_{ij}^B represents the real execution time of backup copy t_i^B on node n_j. re_{ij}^B depends not only on the execution scheme of t_i^B, but also on the execution result of t_i^P. Specifically, re_{ij}^B can be calculated from (14) below.

$$re_{ij}^B = \begin{cases} 0, & \text{if } o_{ij}^P = 1 \wedge s(t_{ij}^B) = passive, \\ (0, e_{ij}\big(q(x_i^B)\big)], & \text{if } o_{ij}^P = 1 \wedge s(t_{ij}^B) = active, \\ e_{ij}\big(q(x_i^B)\big), & \text{if } o_{ij}^P = 0. \end{cases} \tag{6}$$

The reliability cost of a set of real-time tasks on a heterogeneous cluster can be derived from (4) and (5) as

$$rc = rc(Z^P) + rc(Z^B). \tag{7}$$

The reliability r of a cluster with respect to a set of real-time tasks is expressed as (8). A similar reliability model can be found in [36].

$$r = e^{-rc}. \tag{8}$$

3.5 Scheduling Principles

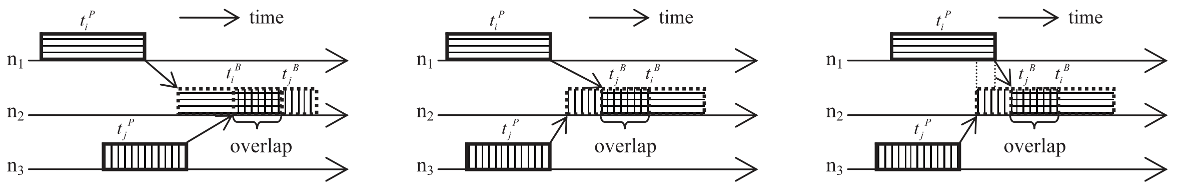
Equation 7 suggests that our scheduling algorithm has to aim at minimizing reliable cost to improve system reliability, indicating that nodes offering small reliability cost should be chosen in task allocations. Our algorithm follows the following two scheduling principles:

- Given the same execution time, the algorithm should schedule tasks on nodes with low failure rates.
- For a group of nodes with the same failure rate, the algorithm needs to assign tasks in order to the nodes providing short execution times.

Thus, tasks should be allocated to computing nodes with powerful processing capacities and lower failure rates to improve the cluster's reliability.

4 Fault-Tolerant Scheduling Algorithm QAFT

In this section, we present an efficient *QoS-aware fault-tolerant scheduling algorithm* (QAFT) for real-time, independent, aperiodic tasks with QoS requirements on heterogeneous clusters. QAFT efficiently considers the QoS needs, system reliability, system resource utilization, and schedulability. To facilitate the presentation of the QAFT algorithm, it is necessary to introduce some properties.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 2. Scenarios of case 1.

Property 1. *A system can tolerate one node's permanent failure if and only if the primary copy and backup copy of a real-time task are allocated to two different nodes:*

$$\forall t_i \in T, n\big(t_i^P\big) \neq n\big(t_i^B\big). \tag{9}$$

Property 1 indicates that the primary copy and backup copy of a real-time task cannot be allocated to the same node, or when the node meets a fault, both copies will not be successfully finished, thus failing to realize fault tolerance.

Property 2. *A task's QoS level is determined by its start time, execution time, and deadline, that is:*

$$\forall t_i \in T, e_{ij}\big(q\big(x_i^P\big)\big) \leq d_i - est_{ij}^P, \tag{10}$$

$$\forall t_i \in T, e_{ik}\big(q\big(x_i^B\big)\big) \leq d_i - lst_{ik}^B, \tag{11}$$

where $j \neq k$. Equations 10 and 11 indicate that if a real-time task can be accepted, its primary copy and backup copy must satisfy timing constraints. Also, the primary copy and backup copy of a real-time task may select different QoS levels, which will greatly improve the system's flexibility. However, in traditional real-time fault-tolerant scheduling algorithms, if a backup copy cannot be processed as the same execution time as its corresponding primary copy, the real-time task will be rejected. Thus, the flexible selection of QoS levels for primary copies and backup copies in our fault-tolerant scheduling algorithm efficiently improves the schedulability.

In order to save the system resource, the backup copy of a real-time task should try to employ passive scheme. If a backup copy has to adopt active scheme, its execution time should be as little as possible. Therefore, the primary copy should be scheduled as early as possible, but the backup copy is just opposite, which means the backup copy should be scheduled as late as possible.

Property 3. *The earliest start time est_{ij}^P of a primary copy t_i^P on node n_j must satisfy the following two constraints:*

- *Node n_j has an idle time slot sufficient to accommodate t_i^P.*
- *The finish time of t_i^P must be earlier than or equal to the deadline of t_i, i.e., $f_i^P \leq d_i$.*

Without loss of generality, before est_{ij}^P can be computed, it is assumed that tasks $t_{i1}, t_{i2}, \ldots, t_{iq}$ have been allocated to node n_j. Note that these tasks can be primary copies or backup copies. The idle time slots on n_j are $[0, s_{i1}]$, $[f_{i1}, s_{i2}], \ldots, [f_{i(q-1)}, s_{iq}], [f_{iq}, \infty]$. In order to get est_{ij}^P, all idle time slots must be scanned from left to right. Consequently, the first idle time slot $[f_{ik}, s_{i(k+1)}]$ that satisfies the following inequality is chosen:

$$s_{i(k+1)} - max\{a_i, f_{ik}\} \geq e_{ij}\big(q\big(x_i^P\big)\big). \tag{12}$$

Thus, the earliest start time est_{ij}^P of t_i on n_j is determined as follows:

$$est_{ij}^P = max\{a_i, f_{ik}\}. \tag{13}$$

As far as the computation for the latest start time of backup copy t_i^B is concerned, we do not necessarily select an idle time slot to accommodate t_i^B. This is because in our fault-tolerant scheduling algorithm, multiple backup copies may overlap in the same time slot on the same node.

We can now discuss the latest start time of a task's backup copy. For any backup copy t_i^B on node n_j, the latest start time lst_{ij}^B can be computed as

$$lst_{ij}^B = \begin{cases} st_{xj} - e_{ij}(q(x_i^B), & \text{if } d_i \geq st_{xj}, \\ d_i - e_{ij}(q(x_i^B), & \text{else.} \end{cases} \tag{14}$$

where st_{xj} denotes the the start time of task t_x that executes following t_i^B and cannot overlap with t_i^B. It should be noted that the latest start time needs to satisfy some constraints while computing. Now, we consider four cases.

Case 1. $\exists t_i \in T, t_j \in T$, if $s(t_{i2}^B) = passive$, $s(t_{j2}^B) = passive$, $n(t_i^P) \neq n(t_j^P)$, $n(t_i^B) = n(t_j^B)$, and t_i^B is allocated first. Fig. 2 shows scenarios of this case.

From the scenarios, we can observe that if the two backup copies are both passive schemes, there is no constraint for computing the latest start time of t_j^B.

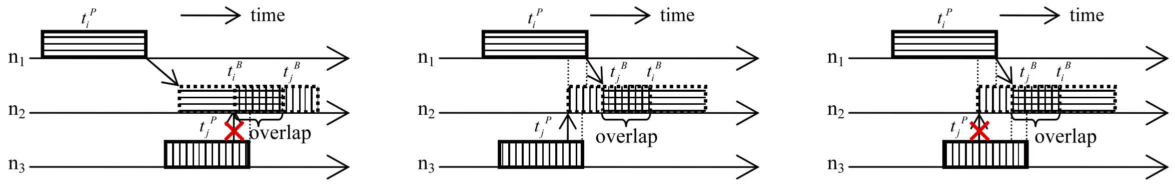
Case 2. $\exists t_i \in T, t_j \in T$, if $s(t_{i2}^B) = passive$, $s(t_{j2}^B) = active$, $n(t_i^P) \neq n(t_j^P)$, $n(t_i^B) = n(t_j^B)$, and t_i^B is allocated first. Fig. 3 illustrates three scenarios of this case.

In scenario 1 of case 2, t_j^B cannot start at the timing instant illustrated in this scenario, because if node n_1 fails before f_i^P, then t_i^B should execute. Since t_i^B and t_j^B overlap with each other on the same node n_2, and t_j^B employs active scheme, this will result in t_i^B and t_j^B executing simultaneously in some time slot. Thus, the system is not able to realize fault tolerance.

Similarly, the system cannot realize fault tolerance in scenario 3 if t_j^B starts to execute at the timing instant shown in this scenario.

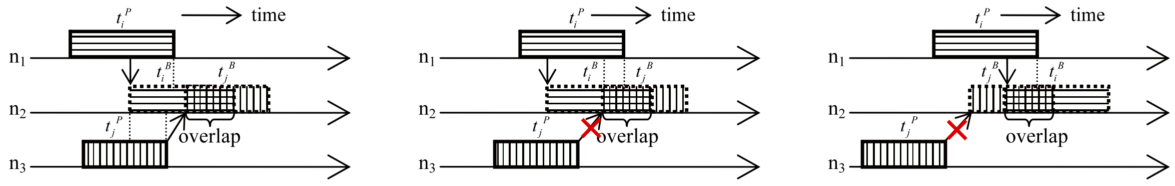
Case 3. $\exists t_i \in T, t_j \in T$, if $s(t_{i2}^B) = active$, $s(t_{j2}^B) = passive$, $n(t_i^P) \neq n(t_j^P)$, $n(t_i^B) = n(t_j^B)$, and t_i^B is allocated first. Fig. 4 illustrates three scenarios of this case.

Scenarios 2 and 3 in this case show that t_j^B cannot start executing at the illustrated start time. The reason is as follows: If n_3 fails before f_j^P, t_j^B must execute. Since t_i^B and



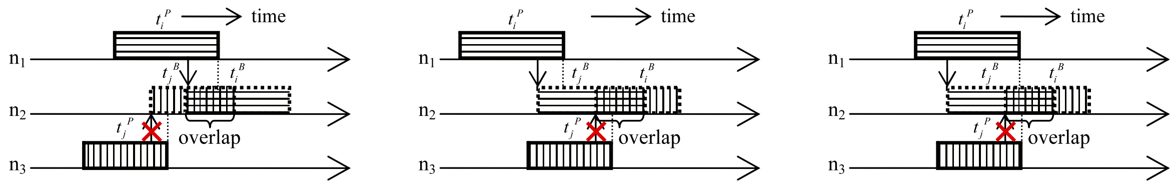
(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 3. Scenarios of case 2.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 4. Scenarios of case 3.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 5. Scenarios of case 4.

t_j^B overlap on the same node n_2, and t_i^B executes using active scheme, which makes t_i^B and t_j^B execute at the same time in some time slot. As a result, the fault tolerance cannot be achieved.

Case 4. $\exists t_i \in T, t_j \in T$, if $s(t_{i2}^B) = active$, $s(t_{j2}^B) = active$, $n(t_i^P) \neq n(t_j^P)$, $n(t_i^B) = n(t_j^B)$, and t_i^B is allocated first. Fig. 5 depicts three scenarios of this case.

The three scenarios in this case all cannot realize fault tolerance if t_j^B starts to execute at the timing instant portrayed in these scenarios. This can be explained that node n_1 or n_3 fails, which may result in t_i^B and t_j^B executing at the same time in some slot.

Theorem 1. *Backup copy t_j^B is able to overlap with backup copy t_i^B, and t_i^B is allocated first, if the overlapping start time of t_i^B and t_j^B is later than the finish time of t_i^P and t_j^P.*

Proof. Suppose the overlapping start time of t_i^B and t_j^B can be earlier than the finish time of t_i^P or t_j^P. If the node with t_i^P or t_j^P fails, which is possible in order to make t_i^B and t_j^B execute simultaneously in some time slots like the scenarios in cases 2-4, then it means that the assumption is incorrect, which completes the proof for this Theorem. □

The pseudocode of primary copy allocation in QAFT algorithm is outlined in Fig. 6.

At the beginning, the primary copy t_i^P is destined for maximal QoS level (see line 2). In order to make the corresponding backup copy t_i^B employ passive scheme, t_i^P should execute as earlier as possible to compute the earliest start time est_{ij}^P of t_i^P on node n_j (see line 6). If t_i^P can be allocated to n_j, it must finish within its deadline (see lines 7-8). If t_i^P can execute on several nodes, it can select the node which makes the system reliability maximal. Line 9 is used to calculate the reliability. It is worth noting that, at this point, backup copy t_i^B has not been scheduled yet, so its reliability cost of t_i^B is set to zero. If some nodes make the system have identical reliability to accommodate t_i^P, the node on which the start time of t_i^P is earliest must be selected. This operation strives to make t_i^B use passive scheme, or it strives to make the simultaneous execution

```
1. for each new task $t_i$, schedule the primary copy
     $t_i^P$ do
2.   $find \leftarrow FALSE$; $q_m \leftarrow q_k$; $selectedNode \leftarrow NULL$;
3.   while $q_m != q_1$ do
4.     $r \leftarrow 0$; $est^P \leftarrow \infty$;
5.     for each node $n_j$ in the system do
6.       Calculate the earliest start time $est_{ij}^P$;
7.       if $est_{ij}^P \leq d_i - e_{ij}(q_m)$ (Property 1) then;
8.         $find \leftarrow TRUE$;
9.         Calculate system reliability $r_j$ by Eq. (8);
10.          if $r_j > r$ or $r_j == r$ && $est_{ij}^P < est^P$ then
11.            $r \leftarrow r_j$; $est^P \leftarrow est_{ij}^P$; $selectedNode \leftarrow n_j$;
12.          end if
13.        end if
14.      end for
15.      if $find == FALSE$ then
16.        Degrade one QoS level of $q_m$: $q_m \leftarrow q_{m-1}$;
17.      else
18.        break;
19.      end if
20.    end while
21.    if $find == FALSE$ then
22.      Reject the primary copy $t_i^P$;
23.    else
24.      $n(t_i^P) \leftarrow selectedNode$;
25.    end if
26. end for
```

Fig. 6. The pseudocode of primary copy allocation in QAFT.

time of t_i^P and t_i^B minimal (see lines 10-12). If employing a higher QoS level cannot satisfy the timing constraint of t_i^P on any node, degrade one QoS level and select a feasible node again. Otherwise, exist the loop (see lines 15-19). If line 23 can be executed, it indicates t_i^P cannot finish using minimal QoS level within its deadline on any node, thus it rejects t_i^P or allocates t_i^P (see lines 21-25).

The objectives of primary copy allocation in QAFT is to allocate primary copies to the node which makes the system reliability maximal, and to make the QoS levels of primary copies maximal within the timing constraints.

Now, we will evaluate the time complexity of primary allocation in QAFT as shown below:

Theorem 2. *The time complexity of primary copy allocation in QAFT is $O(kmq)$, where m is the number of nodes in a cluster, k is the number of QoS levels, and q is the number of waiting tasks in a node.*

Proof. To obtain the earliest start time est_{ij}^P of primary copy t_i^P on node n_j, the time complexity is $O(q)$ (see line 6). The time complexity of calculating the system reliability r_j after t_i^P being allocated to node n_j is $O(1)$ (see line 9). For other lines between lines 6 and 15, they only consume $O(1)$. Degrading one QoS level will take a constant time $O(1)$. The time complexity of allocating t_i^P to n_j is a constant $O(1)$. Thus, the time complexity of primary copy allocation in QAFT is calculated as follows: $O(k)(O(m(O(q)))) = O(kmq)$. □

The pseudocode of backup-copy allocation in QAFT algorithm is illustrated in Fig. 7.

```
1. for each new task $t_i$ whose primary copy $t_i^P$ has been
   allocated, schedule the backup copy $t_i^B$ do
2.   $find \leftarrow FALSE$; $q_m \leftarrow q_k$;
3.   while $q_m != q_1$ do
4.     $r \leftarrow 0$; $lst^B \leftarrow 0$; $selectedNode \leftarrow NULL$;
5.     for each node $n_j$ except $n(t_i^P)$ in the
       system do
6.       $s(t_{ij}^B) \leftarrow active$;
7.       Calculate the latest start time $lst_{ij}^B$ by Eq. (14)
         based on Theorem 1;
8.       if $lst_{ij}^B \le d_i - e_{ij}(q_m)$ (Property 1) then
9.         $find \leftarrow TRUE$;
10.        if $lst_{ij}^B >= f_i^P$ then
11.          $s(t_{ij}^B) \leftarrow passive$;
12.          Compute system reliability $r_j$ by Eq. (8);
13.          if $r_j > r$ or $r_j == r \&\& lst_{ij}^B > lst^B$ then
14.            $r \leftarrow r_j$; $lst^B \leftarrow lst_{ij}^B$; $selectedNode \leftarrow n_j$;
15.          end if
16.        else
17.          if $lst_{ij}^B > lst^B$ then
18.            $lst^B \leftarrow lst_{ij}^B$; $selectedNode \leftarrow n_j$;
19.          end if
20.        end if
21.      end if
22.    end for
23.    if $find == FALSE$ then
24.      Degrade one QoS level of $q_m$: $q_m \leftarrow q_{m-1}$;
25.    else
26.      break;
27.    end if
28.  end while
29.  if $find == FALSE$ then
30.    Reject primary copy $t_i^P$ and backup copy $t_i^B$;
31.  else
32.    $n(t_i^B) \leftarrow selectedNode$;
33.  end if
34. end for
```

Fig. 7. The pseudocode of backup-copy allocation in QAFT.

The goal of backup allocation in QAFT is to allocate backup copies to the node, which makes the system reliability maximal within the timing constraints. Meanwhile, it also strives to make backup copies use active scheme and enhance their QoS levels.

First, the backup copy t_i^B is destined for maximal QoS level (see line 2). In order to make t_i^B employ passive scheme, t_i^B should begin to execute as late as possible, thus we can calculate the latest start time lst_{ij}^B of t_i^B on node n_j (see line 7). If t_i^B can be allocated to n_j, t_i^B must satisfy Theorem 1 and finish within its deadline (see lines 8-9). If the latest start time of t_i^B is later than or equal to the finish time of t_i^P, t_i^B is capable of employing passive scheme (see lines 10-11). As a result, t_i^B does not need to use active scheme on other nodes any more. If t_i^B can execute using active scheme on some nodes, select the node on which the allocation make the system reliability maximal (see lines 13-15). If the latest start time of t_i^B is earlier than the finish time of t_i^P, t_i^B must execute by active scheme. In this condition, the simultaneous execution time of t_i^P and t_i^B should be as little as possible, so select the node on which t_i^B has the latest start time (see lines 17-19). If employing higher QoS level cannot satisfy timing constraint of t_i^P and Theorem 1 on any nodes, degrade one QoS level and select feasible node again (see lines 23-27). If line 29 can be executed, which indicates t_i^B cannot satisfy timing constraint or Theorem 1 on any nodes using minimal QoS level, reject t_i^B and its corresponding primary copy t_i^P, then otherwise allocate t_i^B (see lines 29-33).

Theorem 3. *The time complexity of backup-copy allocation in QAFT is $O(kmq)$, where m is the number of nodes in the system, k is the number of QoS levels, and q is the number of waiting tasks in a node.*

Proof. To obtain the latest start time lst_{ij}^B of backup copy t_i^B on node n_j, the time complexity is $O(q)$ (see line 7). The time complexity of calculating the system reliability r_j is $O(1)$ after t_i^B is allocated to node n_j (see line 12). For other lines between lines 5 and 22, they only consume $O(1)$. Degrading one QoS level will take a constant time $O(1)$. The time complexity of allocating t_i^B to n_j is a constant $O(1)$. Thus, the time complexity of the backup-copy allocation in QAFT is calculated as follows: $O(k)(O(m(O(q)))) = O(kmq)$. □

Theorems 2 and **3** intuitively show that the running time of the algorithm is proportional to the number of computing nodes, the number of QoS levels, and system load (i.e., the number of waiting tasks in a node).

5 Performance Evaluation

In this section, we present several groups of experimental results obtained from extensive simulations to evaluate the performance of QAFT. A competitive advantage of conducting simulation experiments is that performance evaluation on a large-scale distributed system can be accomplished without additional hardware cost [4]. To show performance improvements gained by QAFT, we compare it with DYFARS, which is lately proposed in the literature [34], and NOQAFT, a nonoverlapping QoS-aware fault-tolerance scheduling algorithm. The DYFARS and NOQAFT algorithms are briefly described as follows:

- DYFARS: In a process of allocating each primary copy, DYFARS chooses a node offering the minimal reliability cost. Given a backup copy, DYFARS schedules the backup copy as a passive copy, and if the backup copy has to be an active copy during the scheduling process, DYFARS makes an effort to schedule the active backup copy to a node that provides the minimal system reliability cost.
- NOQAFT: This is a baseline scheduling algorithm developed for comparison purposes. NOQAFT is a variant of the QAFT algorithm. The difference between QAFT and NOQAFT is the fact that NOQAFT does not consider the overlapping of backup copies. The goal of introducing NOQAFT is to observe the performance gap between NOQAFT and QAFT in order to evaluate the effectiveness of applying the backup-copy overlapping technology.

To make the comparison fair, we slightly modify DYFARS in such a way that it arbitrarily picks a QoS level within the QoS range of tasks. Though DYFARS is intended to schedule real-time tasks with QoS requirements, it makes no effort to maximize the QoS level of tasks.

In our experiments, we compare QAFT, NOQAFT, and DYFARS with the following metrics:

- *Guarantee Ratio* (*GR*), defined as: GR = Total number of tasks guaranteed to meet their deadlines/Total number of tasks ×100%.
- *QoS Level Average* (*QLA*) is used to test the QoS levels of accepted tasks.
- *Reliability Cost* in a time unit (*RC*) to measure the system reliability.
- *Overall System Performance* (*OSP*) defined as a product of guarantee ratio, QoS level average, and reliability cost in a time unit, $OSP = GR \times QLA \times exp(-RC)$), which is like the definition in paper [4].

5.1 Simulation Method and Parameters

The experimental parameters in our simulation studies are similar as those used in the literature [42], [43].

- In order to present the node heterogeneity, we use g_j, a positive real number, to represent the node power of node n_j. Parameters $powerAverage$ and $powerSpan$ denote the average processing power of all nodes and variable scope taking $powerAverage$ as center, respectively. g_j is uniformly distributed between $powerAverage - powerSpan$ and $powerAverage + powerSpan$.
- We employ h_i, a positive real number, to denote the hardness of task t_i. The bigger the value of h_i is, the longer the execution time of t_i. Parameters $hardnessAverage$ and $hardnessSpan$ denote the average hardness of all tasks and variable scope taking $hardnessAverage$ as center, respectively. Similarly, the value h_i is uniformly distributed between $hardnessAverage - hardnessSpan$ and $hardnessAverage + hardnessSpan$.
- Because our scheduling model is a general one, without losing generality, we only need to give a general definition of QoS level. Suppose that $0 \leq q(x_i^P) \leq 1$ is the current QoS level of t_i^P, and $0 \leq q(x_i^B) \leq 1$ is the current QoS level of t_i^B. $q(x_i^P)$ and $q(x_i^B)$ are in $[0, 0.1, 0.2, \ldots, 0.9, 1]$.
- The execution time matrix $E = (e_{ij})_{n\times m}$ can be classified into two classes: consistent and inconsistent. For an consistent matrix E, if node n_x has a shorter execution time than node n_y for task t_k, then the same is true for any task t_i. For an inconsistent matrix E, if node n_x has a shorter execution time than node n_y for task t_k, then the same is not true for the other task, t_i. In our study, the matrix E belongs to the consistent type. The execution time e_{ij} of task t_i on node n_j can be generated as: $e_{ij} = q(x_i) \times baseTime \times (h_i/g_j)$. More precisely, $e_{ij}(q(x_i^P)) = q(x_i^P) \times baseTime \times (h_i/g_j)$, and $e_{ij}(q(x_i^B)) = q(x_i^B) \times baseTime \times (h_i/g_j)$. Parameter $baseTime$ is a random positive real number.
- The deadline d_i of task t_i is chosen as follows: $d_i = a_i + \max\{e_{ij}\} + baseDeadline$ [11], where $baseDeadline$ is a random positive real number. $baseDeadline$ determines whether the tasks have loose deadlines or not.
- The arrival rate a_i is described as: $a_i = a_{i-1} + intervalTime$, where $intervalTime$ is a random positive real number, and $a_0 = 0$.
- The failure rate of node n_j is uniformly distributed and the time unit is $10^{-7}/h$.

TABLE 1
Parameters for Simulation Studies

Parameter	Value(Fixed)-(Varied)
Node Number	(64)-(4, 16, 32, 64, 128, 256)
Task Number	(2048)
$powerAverage$	(700)
$powerSpan$	(360)-(160, 200, 240, 280, 320, 360, 400)
$hardnessAverage$	(300)
$hardnessSpan$	(120)-(40, 80, 120, 160, 200, 240, 280)
$baseDeadline$	(360)-(360, 540, 720, 900, 1080, 1260, 1440)
$baseTime$	(60)
$intervalTime$	(1)-(1, 2, 3, 4, 5, 6, 7)
$errRate$	(1.2-2.0)

Table 1 gives the simulation parameters and their values.

5.2 Performance Impact of Node Number

In this section, we present a group of experimental results to observe the performance comparison of QAFT, NOQAFT, and DYFARS with respect to the impact of node number.

Fig. 8 illustrates the performance impact of node number. Fig. 8a shows that with the increase of node number, the guarantee ratios of QAFT, NOQAFT, and DYFARS all get increased. This is because the system node power is enhanced with the number of nodes being increased, which leads to more real-time tasks being accommodated. In addition, from Fig. 8a, the guarantee ratios of QAFT and NOQAFT turn out to be higher than that of DYFARS. This result can be attributed to the fact that DYFARS cannot adaptively adjust tasks' QoS levels, making some tasks unable to be accepted. However, NOQAFT and

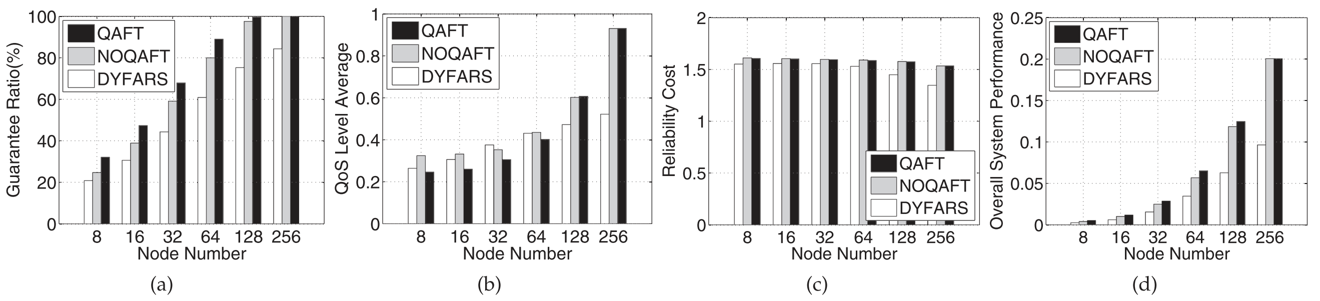


Fig. 8. Performance impact of node number.

QAFT are able to increase the guarantee ratios by degrading the QoS levels of some tasks when the system is in heavy load. Thereby, NOQAFT and QAFT have excellent flexibility. Furthermore, we observe that the guarantee ratio of QAFT always remains higher than that of NOQAFT. The reason is that QAFT adopts backup-copy overlapping technology based on NOQAFT, which enhances the utilization of system resource.

From Fig. 8b, it is found that the QoS level of QAFT is lower when the node number is small. This is because the system is in heavy load when the system has less nodes. QAFT is able to degrade the QoS levels of accepted real-time tasks in order to increase the guarantee ratio. Also, when the node number increases, the QoS level of QAFT increases accordingly, which indicates that the system processing power increases as the node number increases. When the node number is less than or equal to 64, QAFT can guarantee higher QoS levels than NOQAFT. The reason is that QAFT improves the schedulability by degrading the QoS levels within an acceptable QoS range for real-time tasks. Fig. 8b also reveals that the highest QoS level average is about 0.5. This can be attributed to the fact that DYFARS randomly selects QoS levels for coming real-time tasks. Those tasks with higher QoS levels are easily rejected for longer execution time when the system is in heavy load. However, when the system is in light load, it can accept tasks with higher QoS levels leading to an increased QoS level average. Since DYFARS arbitrarily selects QoS levels for real-time tasks from 0.1 to 1, the QoS level average is close to 0.5 when the system is in light load, making is possible to accept more tasks. For QAFT and NOQAFT, they can adaptively adjust tasks' QoS levels by the system load. Thus, the QoS level average can be 1 when the node number is large enough.

Fig. 8c depicts the reliability cost of QAFT, NOQAFT, and DYFARS. The reliability of them decreases with the increase of node number. The change of DYFARS is obvious, because the failure rate of nodes is uniformly distributed. When the node number increases, the number of nodes with higher reliability increases, but the task number is constant. This leads to more tasks being allocated to the nodes with higher reliability. Hence, the reliability cost decreases gradually. However, the reliability little, because QAFT and NOQAFT can adaptively increase tasks' QoS levels to use system resource. In addition, Fig. 8c shows that the reliability cost of DYFARS is the smallest, and that of NOQAFT is the largest. This is because DYFARS makes more tasks execute on nodes with higher reliability, whereas QAFT and NOQAFT allocate tasks to nodes with relatively lower reliability for optimal utilization of system resource, which is the reason that QAFT and NOQAFT can provide higher guarantee ratio and higher QoS level average. Meanwhile, NOQAFT has a higher reliability cost than QAFT because QAFT employs the backup-copy overlapping technology which enhances the system utilization of nodes with higher reliability, making more tasks execute on themselves.

The overall system performances of QAFT, NOQAFT, and DYFARS are shown in Fig. 8d. QAFT outperforms NOQAFT and DYFARS by 14.8 and 86 percent on average, respectively.

5.3 Performance Impact of Node Heterogeneity

In this section, we carry out a group of experiments to observe the performance impact of node heterogeneity on QAFT, NOQAFT, and DYFARS. Parameter $powerSpan$ represents the node heterogeneity. If $powerSpan$ increases, the system heterogeneity is enlarged.

The experimental results are shown in Fig. 9. Fig. 9a demonstrates that when $powerSpan$ varies from 160 to 400, QAFT always has a higher guarantee ratio than NOQAFT and DYFARS, whether the system is heterogeneous or not. The reason for this is that QAFT employs the backup-copy overlapping technology that improves the system resource utilization. In addition, we observe from Fig. 9a that the guarantee ratios of QAFT and NOQAFT increase slightly because tasks are preferentially allocated to nodes with higher processing power (nodes with higher processing power have better reliability) when the system heterogeneity is changed. Nodes with higher processing power will then be further enhanced in terms of processing power. Thus improving the guarantee ratio on these nodes. In contrast, the nodes with lower processing power are just the opposite. However, the adaptivity, with respect to QoS levels of QAFT and NOQAFT, counteracts the negative impact. So far as DYFARS is concerned, it cannot adaptively adjust the QoS levels of real-time tasks allocated to nodes with lower processing power, and the total processing power is invariable, thus guaranteeing the ratio of DYFARS experiences no obvious changes.

It can be found from Fig. 9b that the QoS level average of QAFT is lower than that of NOQAFT and DYFARS due to the fact that QAFT adaptively degrades the tasks' QoS levels, so as to improve the schedulability.

Fig. 9c shows that there is no obvious impact of node heterogeneity on QAFT, NOQAFT, and DYFARS because

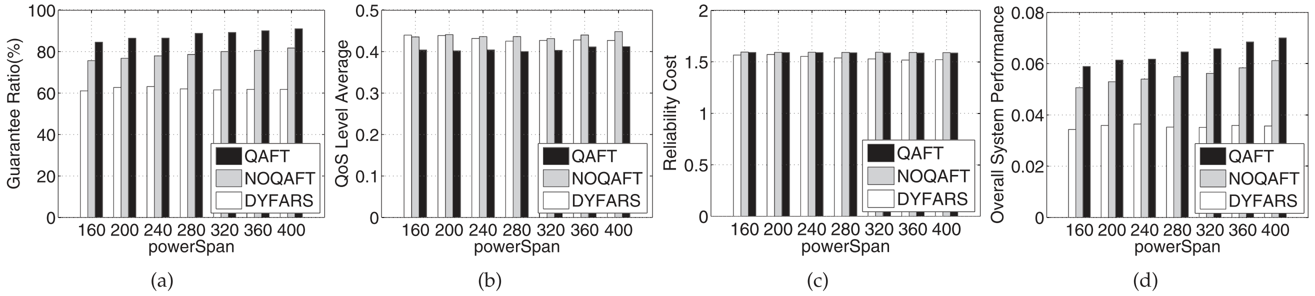


Fig. 9. Performance impact of node heterogeneity.

all algorithms sufficiently consider node heterogeneity and system reliability cost while scheduling.

Fig. 9d shows the overall system performances of QAFT, NOQAFT, and DYFARS. QAFT outperforms NOQAFT and DYFARS in terms of overall system performance by 16.2 and 81 percent, respectively.

5.4 Performance Impact of Task Heterogeneity

This section observes the performance impacts of task heterogeneity on QAFT, NOQAFT, and DYFARS through a series of experiments. Parameter *hardnessSpan* stands for the variance in direct proportion of task heterogeneity, which increases as task heterogeneity increases and decreases as task heterogeneity decreases. Fig. 10 shows the result.

Fig. 10a shows that QAFT always has a higher guarantee ratio than NOQAFT and DYFARS, while *hardnessSpan* ranges from 40 to 280. This means that whether the same granularity or relative difference is shared or not, QAFT displays better performance. Also, there is no performance degradation with the growth of difference in granularity. The reason why QAFT shows higher guarantee ratio than NOQAFT and DYFARS is the same as the reason described in the previous section. QAFT adopts the backup-copy overlapping technology, which increases utilization efficiency, and owns higher guarantee ratio while scheduling. Fig. 10 illustrates that the guarantee ratios of both DYFARS and NOQAFT increase as task heterogeneity increases. In this instance, as task heterogeneity increases, the execution time of some small-granularity tasks decreases, and its counterpart of large-granularity increases. As for DYFARS, shorter execution time of small-granularity tasks contributes to its acceptance; execution time, however, of large-granularity tasks cannot be accepted whatever the situation is. Thus, longer execution time of large-granularity tasks does not influence its guarantee ratio. Therefore, on the whole, an increase in task heterogeneity contributes to an increase in guarantee ratio of scheduling. For NOQAFT, however, shorter execution time of small-granularity tasks contributes to its acceptance. Large-granularity tasks can be accepted by adjusting QoS levels, which does not have a significant influence of guarantee ratio. Owing to QAFT's making use of backup-copy overlapping technology, execution time change of small-granularity tasks does not lay significant influence on guarantee ratio, but large-granularity can be accepted by adjusting QoS levels. This does not lay significant influence on guarantee ratio and no obvious change of guarantee ratio takes place.

Based on Fig. 10b, we can see that as task heterogeneity increases, the QoS level averages of QAFT and NOQAFT increase. This is because the execution time of some small-granularity tasks decreases and the execution time of some large-granularity tasks increases with the increase of task heterogeneity. For tasks with relatively short execution time, QAFT and NOQAFT can heighten QoS levels by adaptive adjusting, but for tasks with relatively long execution time, QoS level's heightening is limited. Thus, on the whole, due to the adoption of QoS adapting, QAFT can make full use of system resources to increase QoS levels.

Fig. 10c shows that almost no reliablility cost change of QAFT, NOQAFT, and DYFARS can be found with change of task heterogeneity. The reason why no significant influence takes place is that all of the three algorithms take system reliability cost into full consideration and incorporate it into scheduling algorithms.

Fig. 10d shows the overall system performance of QAFT, NOQAFT, and DYFARS as task heterogeneity changes. We come to the conclusion that regardless of the change in the task heterogeneity, QAFT always has the best performance and outperforms NOQAFT and DYFARS by 13.6 and 87 percent on average, respectively.

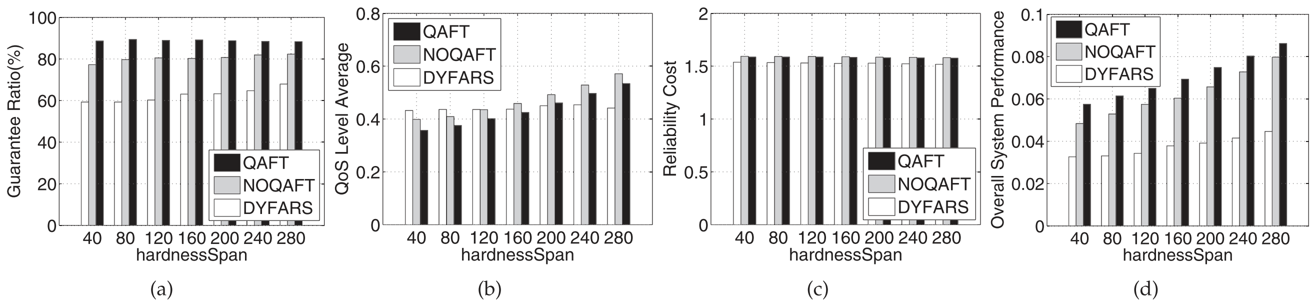


Fig. 10. Performance impact of task heterogeneity.

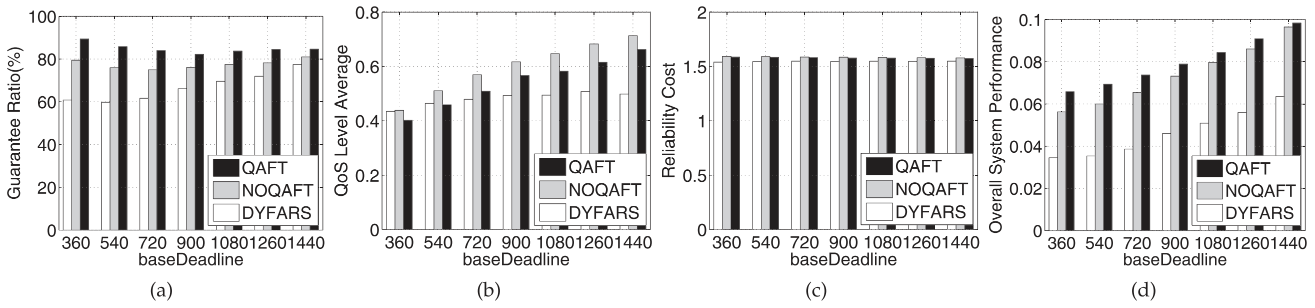


Fig. 11. Performance impact of task deadline.

5.5 Performance Impact of Task Deadline

This section illustrates the performance impact of task deadline on QAFT, NOQAFT, and DYFARS. Parameter $baseDeadline$ varies from 360 to 1,440 with step 180.

Fig. 11 illustrates the experimental results. Fig. 11a shows the change trends with respect to task deadline of QAFT, NOQAFT, and DYFARS. DYFARS basically keeps the ascending trend because the tasks' deadlines are prolonged, causing the system to have loose time to accommodate more tasks. Thus, the guarantee ratio is increased. From Fig. 11a, we observe that when the parameter $baseDeadline$ is 720, NOQAFT is with the minimal guarantee ratio, and QAFT has a minimal guarantee ratio when the value of $baseDeadline$ is 900. Therefore, QAFT and NOQAFT employ the adaptive method in terms of QoS level adjusting, the QoS level is enhanced preferentially resulting in longer execution time of accepted tasks. This condition will lead to some tasks not being accepted due to a later start time. As a result, the guarantee ratio is lightly degraded. However, when deadlines become looser, tasks can be accepted, despite higher QoS levels. Hence, the guarantee ratios of QAFT and NOQAFT increase.

Fig. 11b depicts that the QoS level of DYFARS keeps increasing until the QoS level value approaches 0.5 because DYFARS randomly selects QoS levels for real-time tasks from 0.1 to 1. The QoS level average is close to 0.5 when the system accepts more tasks. In contrast, the QoS levels of QAFT and NOQAFT always increase even though all the while their guarantee ratios do not increase. We can explain this that the looser deadlines are able to make QAFT and NOQAFT enhance tasks' QoS levels further.

Observed from Fig. 11c, the reliability costs of QAFT, NOQAFT, and DYFARS have no obvious change when deadlines become looser. This can be attributed to the fact that the three algorithms take the reliability cost into consideration while scheduling.

Fig. 11d shows that QAFT outperforms NOQAFT and DYFARS with respect to overall system performance by 9.5 and 76 percent on average, respectively.

5.6 Performance Impact of Arrival Rate

This section shows the performance impact of arrival rate on QAFT, NOQAFT, and DYFARS. Parameter $intervalTime$ varies from 1 to 7 with the 1 step.

The experimental results are depicted in Fig. 12. Fig. 12a shows that when the value of $intervalTime$ is smaller, real-time tasks arrive quickly. This may result in some tasks arriving later and violating their deadlines due to their later start time. With the increase of $intervalTime$, the arrival rate of real-time tasks becomes slower, so less tasks are waiting on nodes, leading to tasks having earlier start time. Consequently, the guarantee ratios of QAFT, NOQAFT, and DYFARS are increased. From Fig. 12a, it also can be observed that QAFT has a higher guarantee ratio than NOQAFT and DYFARS. This is because QAFT is capable of adaptively adjusting tasks' QoS levels and employing backup-copy overlapping technology, so as to sufficiently utilize the system resource. Thus, more tasks can be accommodated. However, DYFARS cannot dynamically change tasks' QoS levels, and NOQAFT does not develop the backup-copy overlapping technology, therefore their guarantee ratios are not as high as those of QAFT.

Fig. 12b depicts that the QoS level averages of QAFT and NOQAFT increase when the value of $intervalTime$ varies from 1 to 7. This explains why the system load becomes lighter when tasks arrive slowly. Thereby, the system has more spare time to process more tasks and provides higher QoS levels for them. Besides, the QoS level average of QAFT is higher than that of NOQAFT. We attribute this result to the fact that QAFT employs the backup-copy overlapping

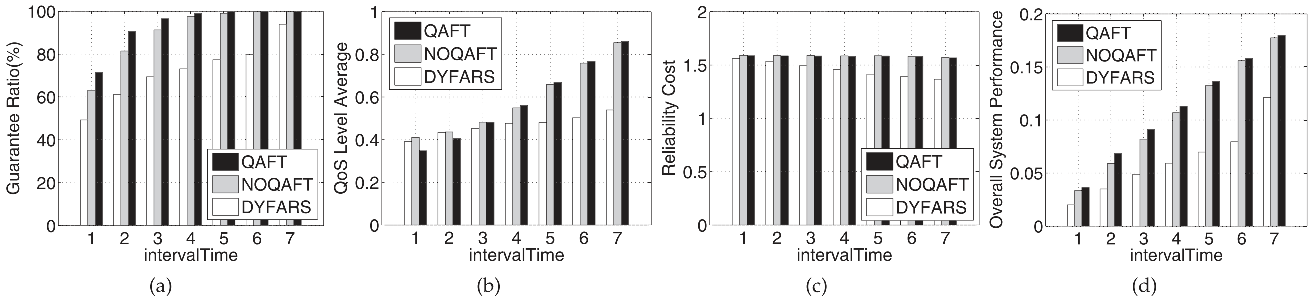


Fig. 12. Performance impact of arrival rate.

technology, thus more system resource can be gained to enhance the accepted tasks' QoS levels. The QoS level average of DYFARS increases slightly with the increase of $intervalTime$, but the maximal value is about 0.5. This is because DYFARS randomly selects tasks' QoS levels and cannot adaptively adjust them.

It should be noted that QAFT and NOQAFT first guarantee schedulability, then strive to enhance tasks' QoS levels. For instance, in Fig. 12a, the guarantee ratios of QAFT and NOQAFT basically approach 1 (the values of $intervalTime$ are 4, 5, 6, and 7) with no obvious variation, but in Fig. 12b, the QoS level averages of QAFT and NOQAFT increase obviously on the same values of $intervalTime$.

Fig. 12c shows that the reliability cost of DYFARS decreases with the increase of $intervalTime$ because, when tasks arrive slowly, these tasks are inclined to be allocated to those nodes with higher reliability. On the contrary, we find that the variation with respect to the reliability costs of QAFT and NOQAFT is small. The reason is that QAFT and NOQAFT constantly guarantee the sufficient utilization of all nodes. Thus, the impact is not obvious.

From Fig. 12d, we can see that QAFT shows superiority to NOQAFT and DYFARS in terms of overall system performance by 6.8 and 85 percent on average, respectively.

6 Conclusions and Future Work

This paper presents an efficient QoS-aware fault-tolerant scheduling algorithm—QAFT—that schedules independent real-time tasks tolerating hardware failures in a heterogeneous cluster. The fault-tolerant capability is incorporated into the QAFT algorithm by implementing the primary-backup model. To sufficiently utilize system resources, QAFT employs the backup-copy overlapping technology, striving to advance the start time of primary copies and delay the start time of backup copies, within timing constraints. QAFT enhances system flexibility because QAFT adaptively adjusts the QoS levels of real-time tasks based on the dynamic changing of system load. More importantly, QAFT improves the reliability of heterogeneous clusters by assigning tasks to nodes offering low reliability cost.

The QAFT algorithm is the first of its kind reported in the literature; it comprehensively addresses the issues of fault tolerance, reliability, real-time, QoS requirements, and heterogeneity. To evaluate the performance of QAFT, we conduct extensive simulations to compare QAFT with the two existing scheduling algorithms—NOQAFT and DYFARS. The experimental results show that compared with the existing solutions, QAFT significantly improves the schedulability and QoS levels of real-time tasks on heterogeneous clusters.

The following issues will be addressed in our future studies: First, we will extend our fault-tolerant scheduling model to multidimensional computing resources including memory, network bandwidth, and data storage. Second, we will combine the communication and dispatching times into QAFT to make the scheduling results more precise. Third, we plan to implement QAFT in a scheduling mechanism of a heterogeneous cluster.

Acknowledgments

The authors are grateful to the anonymous referees for their insightful suggestions and comments. This research was supported by the National Basic Research Program of China under Grant No. 6136101, the National High-Tech Research and Development Plan of China under Grant No. 2008AA7070412, the US National Science Foundation (NSF) under Grants CCF-0845257 (CAREER), CNS-0917137 (CSR), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0831502 (CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), DUE-0830831 (SFS), Auburn University under a startup grant and a gift (No. 2005-04-070) from the Intel Corporation, as well as NSFC 61071061 and the University of Kentucky Start Up Fund.

References

[1] K. Hwang and Z. Xu, *Scalable Parallel Computing: Technology, Architecture, Programming.* McGraw-Hill, 1998.

[2] A. Goller and F. Leberl, "Radar Image Processing with Clusters of Computers," *IEEE Aerospace and Electronics Systems Magazine,* vol. 24, no. 1, pp. 18-22, Jan. 2009.

[3] H.Y. Chang, K.C. Huang, C.Y. Shen, S.C. Tcheng, and C.Y. Chou, "Parallel Computation of a Weather Model in a Cluster Environment," *J. Computer-Aided Civil and Infrastructure Eng.,* vol. 16, no. 5, pp. 365-373, Sept. 2001.

[4] T. Xie and X. Qin, "Scheduling Security-Critical Real-Time Applications on Clusters," *IEEE Trans. Computers,* vol. 55, no. 7, pp. 864-879, July 2006.

[5] C.M. Krishna and K.G. Shin, *Real-Time Systems.* McGraw-Hill, 2001.

[6] T.F. Atdelzater, E.M. Atkins, and K.G. Shin, "QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control," *IEEE Trans. Computers,* vol. 49, no. 11, pp. 1170-1183, Nov. 2000.

[7] S. Liden, "The Evolution of Flight Management Systems," *Proc. IEEE/AIAA 13th Digital Avionics Systems Conf. (DASC '94),* pp. 157-169, Oct. 1994.

[8] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near Optimal Decoding of Product Codes," *Proc. IEEE Global Telecomm. Conf. (GLOBECOM '94),* pp. 339-343, Nov./Dec. 1994.

[9] Z. Chi, L. Song, and K.K. Parhi, "A Study on the Performance, Complexity Tradeoffs of Block Turbo Decoder Design," *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS '01),* vol. 4, pp. 65-68, May 2001.

[10] P. Adde and R. Pyndiah, "Recent Simplifications and Improvements in Block Turbo Codes," *Proc. Second Int'l Symp. Trubo Codes and Related Topics (ISTCRT '00),* pp. 133-136, Sept. 2000.

[11] X. Qin and H. Jiang, "A Novel Fault-Tolerant Scheduling Algorithm for Precedence Constrained Tasks in Real-Time Heterogeneous Systems," *J. Parallel Computing,* vol. 32, no. 5, pp. 331-356, Aug. 2006.

[12] A. Amin, R.A. Ammar, and S.S. Gokhale, "An Efficient Method to Schedule Tandem of Real-Time Tasks in Cluster Computing with Possible Processor Failures," *Proc. Eighth IEEE Int'l Symp. Computers and Comm. (ISCC '03),* pp. 1207-1212, June 2003.

[13] F. Harada, T. Ushio, and Y. Nakamoto, "Adaptive Resource Allocation Control for Fair QoS Management," *IEEE Trans. Computers,* vol. 56, no. 3, pp. 344-357, Mar. 2007.

[14] L. He, S.A. Jarvis, and D.P. Spooner, "Dynamic Scheduling of Parallel Jobs with QoS Demands in Multiclusters and Grids," *Proc. Fifth IEEE/ACM Int'l Workshop Grid Computing (Grid '04),* pp. 402-409, Nov. 2004.

[15] A. Doğan and F. Özgüner, "Scheduling of a Meta-Task with QoS Requirements in Heterogeneous Computing Systems," *J. Parallel and Distributed Computing,* vol. 66, no. 2, pp. 181-196, Feb. 2006.

[16] M. Weber, "Operating-System Enhancements for a Fault-Tolerant Dual-Processor Structure for the Control of an Industrial Process," *Software: Practice and Experience,* vol. 17, no. 5, pp. 345-350, May 1987.

[17] K.H. Kim and A. Damm, "Fault-Tolerance Approaches in Two Experimental Real-Time Systems," *Proc. Seventh IEEE Workshop Real-Time Operating Systems and Software (RTOSS '90)*, pp. 94-98, May 1990.

[18] 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, Nov. 1998.

[19] R. Jejurikar and R. Gupta, "Energy Aware Non-Preemptive Scheduling for Hard Real-Time Systems," *Proc. 17th Euromicro Conf. Real-Time Systems (ECRTS '05)*, pp. 21-30, July 2005.

[20] W. Li, K. Kavi, and R. Akl, "A Non-Preemptive Scheduling Algorithm for Soft Real-Time Systems," *Computers and Electrical Eng.*, vol. 33, no. 1, pp. 12-29, Jan. 2007.

[21] S. Dolev and A. Keizelman, "Non-Preemptive Real-Time Scheduling of Multimedia Tasks," *J. Real-Time Systems*, vol. 17, no. 1, pp. 23-39, July 1999.

[22] J.D. Ullman, "NP-Complete Scheduling Problems," *J. Computer and System Sciences*, vol. 10, no. 3, pp. 384-393, Oct. 1975.

[23] C.C. Han, K.G. Shin, and J. Wu, "A Fault-Tolerant Scheduling Algorithm for Real-Time Periodic Tasks with Possible Software Faults," *IEEE Trans. Computers*, vol. 52, no. 3, pp. 362-372, Mar. 2003.

[24] O. González, H. Shrikumar, J.A. Stankovic, and K. Ramamritham, "Adaptive Fault Tolerance and Graceful Degradation under Dynamic Hard Real-Time Scheduling," *Proc. 18th IEEE Real-Time Systems Symp. (RTSS '97)*, pp. 79-89, Dec. 1997.

[25] M. Naedele, "Fault-Tolerant Real-Time Scheduling under Execution Time Constraints," *Proc. Sixth Int'l Conf. Real-Time Computing Systems and Applications (RTCSA '99)*, pp. 392-395, Dec. 1999.

[26] Q. Zheng, B. Veeravalli, and C.K. Tham, "On the Design of Fault-Tolerant Scheduling Strategies Using Primary-Backup Approach for Computational Grids with Low Replication Costs," *IEEE Trans. Computers*, vol. 58, no. 3, pp. 380-393, Mar. 2009.

[27] Y.S. Hong and H.W. Goo, "A Fault-Tolerant Scheduling Scheme for Hybrid Tasks in Distributed Real-Time Systems," *Proc. Third IEEE Workshop Software Technologies for Future Embedded and Ubiquitous Systems (SEUS '05)*, pp. 3-6, May 2005.

[28] P. Mejia-Alvarez and D. Mosse, "A Responsiveness Approach for Scheduling Fault Recovery in Real-Time Systems," *Proc. Fifth IEEE Symp. Real-Time Technology and Applications (RTAS '99)*, pp. 4-13, June 1999.

[29] S. Ghosh, R. Melhem, and D. Mossé, "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.

[30] R. Al-Omari, A.K. Somani, and G. Manimaran, "Efficent Overloading Technique for Primary-Backup Scheduling in Real-Time Systems," *J. Parallel and Distributed Computing*, vol. 64, no. 5, pp. 629-648, May 2004.

[31] T. Tsuchiya, Y. Kakuda, and T. Kikuno, "A New Fault-Tolerant Scheduling Technique for Real-Time Multiprocessor Systems," *Proc. Second Int'l Workshop Real-Time Computing Systems and Applications (RTCSA '95)*, pp. 197-202, Oct. 1995.

[32] C.H. Yang, G. Deconinec, and W.H. Gui, "Fault-Tolerant Scheduling for Real-Time Embedded Control Systems," *J. Computer Science and Technology*, vol. 19, no. 2, pp. 191-202, Feb. 2004.

[33] 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 and Distributed Computing*, vol. 65, no. 5, pp. 595-608, May 2005.

[34] W. Luo, J. Li, F. Yang, G. Tu, L. Pang, and L. Shu, "DYFARS: Boosting Reliability in Fault-Tolerant Heterogeneous Distributed Systems through Dynamic Scheduling," *Proc. Eighth ACIS Int'l Conf. Software Eng., Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD '07)*, pp. 640-645, Aug. 2007.

[35] T.D. Braun, H.J. Siegel, and N. Beck, et al., "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *J. Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810-837, June 2001.

[36] S. Srinivasan and N.K. Jha, "Safety and Reliability Driven Task Allocation in Distributed Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 10, no. 3, pp. 238-251, Mar. 1999.

[37] X. Qin and H. Jiang, "A Dynamic and Reliability-Driven Scheduling Algorithm for Parallel Real-Time Jobs Executing on Heterogeneous Clusters," *J. Parallel and Distributed Computing*, vol. 65, no. 8, pp. 885-900, Aug. 2005.

[38] H.J. Siegel, H.G. Dietz, and J.K. Antonio, "Software Support for Heterogeneous Computing," *The Computer Science and Eng. Handbook*, CRC Press, 1997.

[39] A.A. Khokhar, V.K. Prasanna, M.E. Shaaban, and C.L. Wang, "Heterogeneous Computing: Challenges and Opportunities," *Computer*, vol. 26, no. 6, pp. 18-27, June 1993.

[40] M. Qiu and E.H.-M. Sha, "Cost Minimization while Satisfying Hard/Soft Timing Constraints for Heterogeneous Embedded Systems," *ACM Trans. Design Automation of Electronic Systems*, vol. 14, no. 2, pp. 1-30, Mar. 2009.

[41] X. Zhu and P. Lu, "Study of Scheduling for Processing Real-Time Communication Signals on Heterogeneous Clusters," *Proc. Ninth Int'l Symp. Parallel Architectures, Algorithms, and Networks (I-SPAN '08)*, pp. 121-126, May 2008.

[42] X. Zhu and P. Lu, "Multi-Dimensional Scheduling for Real-Time Tasks on Heterogeneous Clusters," *J. Computer Science and Technology*, vol. 24, no. 3, pp. 434-446, Mar. 2009.

[43] X. Zhu and P. Lu, "A Two-Phase Scheduling Strategy for Real-Time Applications with Security Requirements on Heterogeneous Clusters," *Computers and Electrical Eng.*, vol. 35, pp. 980-993, Nov. 2009.

Xiaomin Zhu (M'10) received the BS and MS degrees in computer science from Liaoning Technical University, China, in 2001 and 2004, respectively, and the PhD degree in computer science from Fudan University, China, in 2009. He is currently an assistant professor in the School of Information System and Management at National University of Defense Technology, China. His research interests include cluster computing, fault-tolerant computing, green computing, and performance evaluation. He is a member of the IEEE, the IEEE Communication Society, and the ACM.

Xiao Qin (S'99-M'04-SM'09) received the BS and MS degrees in computer science from Huazhong University of Science and Technology in 1992 and 1999, respectively. He received the PhD degree in computer science from the University of Nebraska-Lincoln in 2004. He is currently an associate professor in the Department of Computer Science and Software Engineering at Auburn University. Prior to joining Auburn University in 2007, he had been an assistant professor with New Mexico Institute of Mining and Technology (New Mexico Tech) for three years. He won an NSF CAREER award in 2009. His research is supported by the US National Science Foundation (NSF), Auburn University, and Intel Corporation. He has been on the program committees of various international conferences, including IEEE Cluster, IEEE MSST, IEEE IPCCC, and ICPP. His research interests include parallel and distributed systems, storage systems, fault tolerance, real-time systems, and performance evaluation. He is a member of the ACM and a senior member of the IEEE.

Meikang Qiu (SM'07) received the BE and ME degrees from Shanghai Jiao Tong University, China. He received the MS and PhD degrees of computer science from the University of Texas at Dallas, in 2003 and 2007, respectively. He had worked at Chinese Helicopter R&D Institute and IBM. He is currently an assistant professor of ECE at the University of Kentucky. He has published more than 100 peer reviewed papers, including 35 journal papers. He has been on various chairs and TPC members for many international conferences. He served as the Program Chair of IEEE EmbeddCom '09 and EM-Com '09. He received Air Force Summer Faculty Award 2009. He won three best paper awards (IEEE Embedded and ubiquitous Computing (EUC '09), IEEE/ACM GreenCom '10, and IEEE CSE '10) and one best paper nomination. His research interests include embedded systems, computer security, and wireless sensor networks. He is a senior member of the IEEE.