Available online at www.sciencedirect.com
SCIENCE DIRECT®
J. Parallel Distrib. Comput. 65 (2005) 885–900
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters

Xiao Qin[a,*], Hong Jiang[b]

[a]*Department of Computer Science, New Mexico Institute of Mining and Technology, 801 Leroy Place, Socorro, NM 87801-4796, USA*
[b]*Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588-0115, USA*

Received 21 February 2003; received in revised form 27 September 2004; accepted 10 February 2005
Available online 17 May 2005

Abstract

In this paper, a heuristic dynamic scheduling scheme for parallel real-time jobs executing on a heterogeneous cluster is presented. In our system model, parallel real-time jobs, which are modeled by directed acyclic graphs, arrive at a heterogeneous cluster following a Poisson process. A job is said to be feasible if all its tasks meet their respective deadlines. The scheduling algorithm proposed in this paper takes reliability measures into account, thereby enhancing the reliability of heterogeneous clusters without any additional hardware cost. To make scheduling results more realistic and precise, we incorporate scheduling and dispatching times into the proposed scheduling approach. An admission control mechanism is in place so that parallel real-time jobs whose deadlines cannot be guaranteed are rejected by the system. For experimental performance study, we have considered a real world application as well as synthetic workloads. Simulation results show that compared with existing scheduling algorithms in the literature, our scheduling algorithm reduces reliability cost by up to 71.4% (with an average of 63.7%) while improving schedulability over a spectrum of workload and system parameters. Furthermore, results suggest that shortening scheduling times leads to a higher guarantee ratio. Hence, if parallel scheduling algorithms are applied to shorten scheduling times, the performance of heterogeneous clusters will be further enhanced.

Keywords: Dynamic scheduling; Real-time; Parallel processing; Heterogeneous clusters; Cluster computing; Reliability cost; Performance evaluation

1. Introduction

Heterogeneous clusters have become widely used for scientific and commercial applications. These systems require a mixture of general-purpose machines, programmable digital machines, and application specific integrated circuits [33]. A heterogeneous cluster involves multiple heterogeneous modules that interact with one another to solve a problem [34,41]. In a heterogeneous cluster, applications comprise multiple subtasks that have diverse execution requirements. The subtasks must be assigned to machines and ordered for execution in such a way that the overall application execution time is minimized [18].

Recently, heterogeneous clusters have also been employed in real-time applications [43], in which the systems depend not only on results of computation, but also on time instants at which these results become available. The consequences of missing deadlines of *hard* real-time systems may be catastrophic, whereas such consequences for soft real-time systems are relatively less damaging. Examples of hard real-time applications include aircraft control, radar for tracking missiles, and medical electronics. On-line transaction processing systems are examples of soft real-time applications. In real-time applications, reliability is one of the most important issues. Due to the critical nature of jobs executed in many real-time systems, high reliability becomes an inherent requirement of such systems, and this is especially true for hard real-time applications.

* Corresponding author. Fax: +1 505 835 5587.
E-mail addresses: xqin@cs.nmt.edu (X. Qin), jiang@cse.unl.edu (H. Jiang).

0743-7315/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2005.02.003

Growing evidence shows that scheduling is a key factor in obtaining high reliability and performance in heterogeneous clusters supporting real-time applications. The objective of real-time scheduling is to map tasks onto machines and order their execution so that task precedence requirements are satisfied and a minimum schedule length, when attainable, is given. Besides achieving this conventional objective, the dynamic scheduling strategy proposed in this paper provides high reliability for non-preemptive, aperiodic, real-time jobs without any additional hardware cost. In particular, we have developed a framework of real-time scheduling by which parallel jobs are scheduled dynamically, as they arrive at a heterogeneous cluster. In this framework, a designated machine, called *scheduler*, is responsible for dynamically scheduling real-time jobs as they arrive, and dispatching them to other machines, called *processing elements*, to execute. The proposed algorithm takes into account dispatching and scheduling times in addition to reliability costs, and these factors have been neglected by most scheduling schemes that deal with real-time heterogeneous clusters. This approach is shown by our simulation studies to not only make real-time jobs more predictable and reliable, but also make the scheduling more realistic.

The paper is organized as follows. In Section 2, work reported in the literature that is the most relevant to our work is briefly described. The system and reliability models are presented in Section 3. Section 4 proposes a novel dynamic scheduling algorithm. Performance evaluation is presented in Section 5. Finally, Section 6 concludes the paper by summarizing the main contributions of this paper and commenting on future directions of this work.

2. Related work

Many scheduling algorithms have been proposed in the literature to support real-time systems. Real-time scheduling algorithms are classified into two categories: static (off-line) [1,15,21,24,27,29,32] and dynamic (on-line) [13,17,19,22,31,36]. Very recently, Palis addressed task-scheduling problems in the context of reservation-based real-time systems that provide quality of service guarantees [22]. Real-time tasks in Palis's scheduling framework are preemptive [22], whereas it is assumed in our scheduling model that real-time tasks are non-preemptive. Moreover, the algorithm proposed by Palis [22] as well as many other algorithms presented in [21,36] were designed for independent real-time tasks. In contrast, our proposed algorithm, like those described in [15,24,27,29], can schedule tasks with precedence constraints, which are represented by directed acyclic graphs (DAG). We recently extended non-real-time DAGs into real-time DAGs to study real-time scheduling of tasks with precedence constraints [24]. However, these algorithms, while considering precedence constraints, belong to the static category, limiting their applications to offline scheduling only. Furthermore, most of these real-time scheduling algorithms are designed for homogeneous systems, making them unsuitable for heterogeneous systems.

In the literature, parallel jobs have often been represented by DAGs [4,7,15,29,41]. Wu et al. proposed a runtime parallel incremental DAG scheduling approach [41]. Cosnard et al. developed a scheduling algorithm for a parameterized DAG, which first derives symbolic linear clusters and then assigns task clusters to machines [7]. As for distributed computing, a typical model is the fork-join paradigm [30], where main program thread runs on one processor and spawns a number of tasks from time-to-time. Sahni and Vairaktarakis addressed the scheduling problem in the fork-join paradigm, and developed efficient heuristics to obtain minimum finish time schedules for single-master processor and multiple-master systems [30]. The scheduling algorithms in these three studies, however, were also designed for homogeneous systems.

The studies in heterogeneous clusters reveal a number of challenges, which include load balancing [3,42,25], resource management [10] and scheduling [5,6,38]. The issue of scheduling on heterogeneous systems has been addressed in many papers [3,6,8,11,28,29,35,37]. It is suggested that minimizing a task's completion time leads to a minimal start time of the task [18,37]. Topcuoglu et al. studied two efficient and low-complexity heuristics for DAGs: the heterogeneous Earliest-Finish-Time (HEFT) algorithm and the Critical-Path-on-a-Machine (CPOP) algorithm [37]. Iverson and Özgüner proposed a matching and scheduling framework where multiple applications compete for computational resources on networks [11]. Maheswaran and Siegel investigated a dynamic matching and scheduling algorithm for heterogeneous system [18], whereas Beaumont proposed a static scheduling algorithm that based on a realistic model for heterogeneous networks of workstations. To consider reliability of different resources in a system while making scheduling decisions, Doğan and Özgüner introduced two cost functions that were incorporated into a matching and scheduling algorithm for tasks with precedence constraints [8]. As computational Grids have emerged as new platforms for high-performance computing, grids have become a new frontier for research in the area of scheduling and resource management. Arora et al. proposed a new scheduling algorithm for a generalized heterogeneous Grid environment [3]. Unfortunately, all these scheduling algorithms assumed that tasks are non-real-time. Non-real-time scheduling algorithms are unable to schedule real-time jobs efficiently, simply because they are not designed to meet the predictability requirement of real-time jobs.

Some work has been done to combine real-time computing with heterogeneous systems [10,29,31,38]. Tracy et al. addressed a real-time scheduling issue in heterogeneous systems [38]. Huh et al. proposed a solution for dynamic resource management problems in real-time heterogeneous systems [10]. Ranaweera and Agrawal developed a scalable scheduling scheme on heterogeneous systems to reduce the number of pipeline stages and the pipeline period of

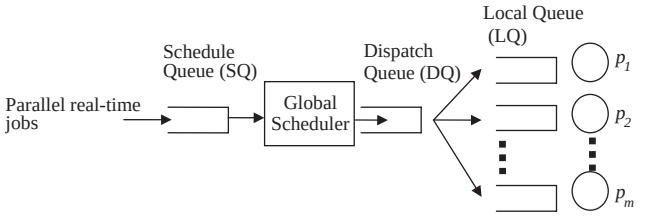

Fig. 1. The scheduler on model for dynamic scheduling of parallel real-time jobs in a heterogeneous cluster.

time- critical applications [29]. Santos et al. introduced a new real-time scheduling concept based on the hybrid deterministic/probabilistic analysis [31]. Although the above algorithms took both the real-time and heterogeneous issues into consideration, these algorithms did not consider reliability. We have proposed a real-time scheduling in heterogeneous systems, which can minimize the reliability cost of the systems [24]. While the scheduling algorithms developed in [24] were static in nature, algorithms studied in this paper, on the other hand, are dynamic.

To the best of our knowledge, scheduling and dispatching times are ignored by most dynamic non-real-time and real-time scheduling algorithms. To make real-time scheduling results more precise, scheduling and dispatching times have to be incorporated in dynamic scheduling algorithms. Therefore, our study takes a closer look at the impact of scheduling and dispatching times on scheduling performance (see Sections 5.4 and 5.5).

In this paper, we only focus on dynamic scheduling for real-time systems. For this reason, we have not discussed a diversity of scheduling strategies developed for non-real-time applications. However, Kwok and Ahmad provided classifications and detailed descriptions of various static scheduling approaches [16], and many other scheduling schemes have been introduced for parallel computing systems [9].

3. System and reliability models

In this section we describe a general system model for parallel applications running on a heterogeneous cluster. We then present a reliability model that captures the typical reliability characteristics of a cluster. Reliability cost in the reliability model is an important performance metric used throughout the rest of this study. This section ends by formulating scheduling and dispatching times that are considered important for performance of real-time applications in dynamic cluster-computing environments.

3.1. System model

Fig. 1 depicts the scheduler model in a heterogeneous cluster environment. This model is similar to the one described in [13,14,19,35], where a *global scheduler* works in concert with a Resource Manager. It is assumed that all parallel jobs, along with information provided by application programmers, are submitted to the global scheduler by a special user command. A *schedule queue* (*SQ*) for arriving jobs is maintained by the scheduler, which schedules real-time tasks of each job in SQ and places an accepted job in a *dispatch queue* (*DQ*) from which tasks of each accepted job are transmitted to designated machines, also called *processing elements* (*PEs*), for execution. The scheduler executes in parallel with PEs, each of which maintains a *local queue* (*LQ*) to which real-time tasks are transmitted from DQ. A parallel job is considered acceptable if all tasks in this job can be completed before their deadlines; otherwise, the job is rejected by the scheduler.

In a distributed scheduling scheme, an alternative approach to dynamic scheduling, jobs arrive independently at each local scheduler, which produces schedules in parallel with other schedulers. Compared with the distributed scheme, the centralized scheduling model has two attractive features. First, it is straightforward to provide the centralized scheduler with fault-tolerance, using a backup scheduler that concurrently executes with the primary scheduler. The backup scheduler independently determines whether or not the timing constraints of given jobs can be satisfied and stores the tasks of accepted jobs into the backup scheduler's DQ. Tasks in the backup scheduler's DQ will not be transmitted to the processing elements until a failure of the primary scheduler is detected. Second, implementation of a centralized scheduling model is simpler and easier than that of a distributed scheduling model. If schedulers in the distributed model are dedicated to scheduling, the computing power tends to be underutilized, especially when the schedulers are idle. On the other hand, if the schedulers are able to serve as processing elements when they have no job to schedule, it is difficult (if it is not impossible) to predict when the schedulers will be idle in a dynamic cluster environment. Therefore, the centralized scheduler is employed in our scheduler model. Nevertheless, and importantly, our proposed scheduling approach can also be implemented in a distributed scheduling scheme.

A parallel real-time job is modeled by a directed acyclic graph (DAG) $J = \{V, E\}$, where $V = \{v_1, v_2, \ldots, v_n\}$ represents a set of real-time tasks, and E represents a set of weighted and directed edges among real-time tasks. $e_{ij} = (v_i, v_j) \in E$ denotes a message transmitted from task v_i to v_j, and $|e_{ij}|$ is the volume of data transmitted between these tasks.

A heterogeneous cluster is modeled by a set $P = \{p_1, p_2, \ldots, p_m\}$ of machines, where p_i is a machine with local memory. Machines in the heterogeneous cluster are connected with one other by a high-speed network. A machine communicates with other machines through message passing, and the communication time between two tasks assigned to the same machine is assumed to be zero [26,27,37].

One challenging issue in improving performance of clusters lies in their heterogeneity. There are two essential reasons that a homogeneous cluster will eventually become a

heterogeneous cluster. First, most of machines in a cluster are commercially off-the-shelf products, which are likely to become outdated. Before predecessors become unusable, recently purchased machines will be added into the cluster. As a result, a heterogeneous cluster may consist of different types of machines with a broad range of both computing power and failure rate. Second, heterogeneous machines tend to be connected with each other by different types of high-speed networks, since a cluster will consist of outdated network from previous installation and new network that may have better communication performance.

The *computational heterogeneity* of a job is expressed as a function, $C : V \times P \rightarrow R$, which represents the execution time of each task on each available machine in a heterogeneous cluster [11,26,34], where c_{ij} denotes the execution time of task v_i on machine p_j. Likewise, the *communicational heterogeneity* of the job can be expressed by a function [11,26,34], $COM : E \times P \times P \rightarrow R$, in which the communication time for transferring a message e_{sr} from task v_s on machine p_i to task v_r on machine p_j is determined by $w_{ij}^{*}|e_{sr}|$ [6,26], where w_{ij}, the weight on the edge between p_i and p_j, represents the time for transmitting a message of unit length between the two machines. Thus, w_{ij} can be viewed as a measure of *communicational heterogeneity*.

3.2. Reliability model

The reliability model, which is similar as the one defined in [24,26,34], assumes that permanent failures occur according to a *Poisson* probability distribution and failures are mutually independent. Let X be an $m \times n$ binary matrix corresponding to a schedule, in which n tasks of a job are assigned to m processors. Element x_{ij} equals 1 if and only if v_i has been assigned to p_j; otherwise $x_{ij} = 0$.

A machine might fail during an idle time, but it is assumed that machines' failures during an idle time interval are not considered in our reliability model. The reason for this assumption is two-fold [24,34]. First, instead of affecting the system reliability, failures during an idle time merely make impact upon completion times of tasks. Second, a machine's failure during an idle period can be fixed by replacing the failed machine with a spare unit, meaning that such failures are not critical for reliability analysis.

The reliability cost of a task v_i on p_j is a product of p_j's failure rate λ_j and v_i's execution time on p_j. Thus, the reliability cost of a machine is the summation over reliability costs of all tasks assigned to that machine based on a given schedule. Given a vector of failure rates $\Lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)$, a specific schedule X, and a job J, the reliability cost of the machines of the cluster is defined as

$$RC_{\mathrm{PN}}(\Lambda, X, J) = \sum_{j=1}^{m} \sum_{i=1}^{n} (-\lambda_j x_{ij} c_{ij}). \quad (1)$$

Before estimating reliability cost of links connecting among machines, we introduce a set E_{kb}, containing all messages transmitted from p_k to p_b. Formally, E_{kb} is defined as

$$E_{kb} = \{(v_i, v_j) | e_{ij} > 0 \wedge x_{ik} = 1 \wedge x_{jb} = 1\}, \quad \forall 1 \leqslant k, \ \ b \leqslant m : k \neq b.$$

Let μ_{kb} be the failure rate of the link between p_k and p_b. The reliability cost of a message $e_{ij} \in E_{kb}$ is a product of μ_{kb} and $w_{kb}|e_{ij}|$. Therefore, e_{ij}'s reliability cost can be calculated as: $-\mu_{kb} x_{ik} x_{jb} w_{kb} |e_{ij}| = -\mu_{kb} w_{kb} |e_{ij}|$. Based on the definition of one message's reliability cost, the reliability cost of a link between p_k and p_b, denoted as $RC_{kb}(M, X, J)$, can be computed as a cumulative reliability cost of all messages assigned to this link. More precisely, $RC_{kb}(M, X, J)$ is obtained by the following equation, where M is an $m \times m$ matrix of failure rates for links.

$$RC_{kb}(M, X, J) = \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} [-\mu_{kb} x_{ik} x_{jb} (w_{kb} |e_{ij}|)]. \quad (2)$$

$RC_{\mathrm{LINK}}(M, X, J)$, the reliability cost of links in the system, can be derived from Eq. (2). Thus, $RC_{\mathrm{LINK}}(M, X, J)$ equals to the summation over all link's reliability cost, and therefore we have,

$$RC_{\mathrm{LINK}}(M, X, J) = \sum_{k=1}^{m} \sum_{b=1, b \neq k}^{m} R_{kb}(M, X, J). \quad (3)$$

We are now in a position to determine $RC(\Lambda, M, X, J)$, the heterogeneous cluster's reliability cost that is a summation of the reliability cost of machines and links. Hence, we obtain $RC(\Lambda, M, X, J)$ from Eqs. (1) and (3) as

$$RC(\Lambda, M, X, J) = RC_{\mathrm{PN}}(\Lambda, X, J) + RC_{\mathrm{LINK}}(M, X, J). \quad (4)$$

Given a cluster with the reliability cost as $RC(\Lambda, M, X, J)$, the reliability is given by Eq. (5):

$$\begin{aligned} &Reliability(\Lambda, M, X, J) \\ &\quad = \exp(-RC(\Lambda, M, X, J)) \\ &\quad = \exp(-RC_{\mathrm{PN}}(\Lambda, X, J)) \\ &\qquad \exp(-RC_{\mathrm{LINK}}(M, X, J)). \end{aligned} \quad (5)$$

Therefore, scheduling a task with larger execution time to a more reliable machine is a good approach to increase the system's overall reliability. For the convenience of reference in the rest of the paper, we summarize the notation of the system and reliability models in Table 1.

Table 1
Model parameters

Parameter	Explanation
c_{ij}	execution time of task v_i on machine p_j
e_{ij}	a message transmitted from task v_i to v_j
w_{ij}	time for transmitting a message of unit length between machine p_i and p_j
λ_i	failure rate of machine p_i
μ_{ij}	failure rate of a link between p_i and p_j
x_{ij}	$x_{ij} = 1$ if and only if task v_i has been assigned to machine p_j; otherwise $x_{ij} = 0$
RC	reliability cost of a heterogeneous cluster
RC_{PN}	reliability cost of machines in a cluster
RC_{LINK}	reliability cost of communication links in a cluster

3.3. Scheduling and dispatching times

In a dynamic scheduling environment, it takes a scheduler a certain amount of time to schedule a parallel job. This time can be significant if the number of tasks in the job is large. To the best of our knowledge, most dynamic real-time scheduling algorithms either assumes zero scheduling time or do not take scheduling time into account. In real clusters, a dynamic real-time scheduling algorithm that does not consider scheduling time may not be predictable. Therefore in this study we will incorporate scheduling time into the proposed scheme. To further improve the predictability of real-time scheduling, we also take the so-called dispatch time—the time it takes a scheduler to send real-time tasks of an accepted job from the DQ to the processing elements—into consideration.

Assume that v_i is a real-time task in job J_k, thus, $v_i \in V(J_k)$. Let $t_{\mathrm{dispatch}}(v_i)$ and $t_{\mathrm{schedule}}(J_i)$ denote the time overhead of dispatching task v_i from the scheduler to the processing element and the scheduling time for job J_i, respectively. It is assumed that the underlying network that connects the scheduler with the processing elements affords real-time communications [12], which is able to guarantee a given task v_i to be dispatched within time interval $t_{\mathrm{dispatch}}(v_i)$. $t^{\mathrm{DQ}}_{\mathrm{delay}}(v_i)$ denotes the queuing delay in DQ experienced by task v_i, and $t^{\mathrm{SQ}}_{\mathrm{delay}}(J_i)$ represents the queuing delay in SQ experienced by job J_i. Let $t^{\mathrm{job}}_{\mathrm{interval}}(J_k)$ be the inter-arrival interval between two consecutive jobs J_{k-1} and J_k. $t^{\mathrm{task}}_{\mathrm{interval}}(v_i)$ denotes the time interval between J_k's arrival at the scheduler and v_i's arrival at its target processing element. The reason why the derivation of $t^{\mathrm{task}}_{\mathrm{interval}}(v_i)$ is important and indispensable in a practical heterogeneous cluster is that a task v_i cannot start executing on a machine p_j until v_i arrives at p_j. Thus, the earliest start time of v_i on any processing machine, determined in expression (10) to be presented shortly, is less than or equal to $t^{\mathrm{task}}_{\mathrm{interval}}(v_i)$. The time interval $t^{\mathrm{task}}_{\mathrm{interval}}(v_i)$ consists of four time intervals, namely, queuing delay experienced in SQ, scheduling overhead incurred in job J_k, delay time experienced in DQ and the dispatch time. Therefore, $t^{\mathrm{task}}_{\mathrm{interval}}(v_i)$ can be defined as

$$t^{\mathrm{task}}_{\mathrm{interval}}(v_i) = t^{\mathrm{SQ}}_{\mathrm{delay}}(J_k) + t_{\mathrm{schedule}}(J_k) + t^{\mathrm{DQ}}_{\mathrm{delay}}(v_i) + t_{\mathrm{dispatch}}(v_i), \tag{6}$$

where $v_i \in V(J_k)$,

$$t^{\mathrm{DQ}}_{\mathrm{delay}}(v_i) = \sum_{v_j \in DQ} t_{\mathrm{dispatch}}(v_j) \quad \text{and} \quad t^{\mathrm{SQ}}_{\mathrm{delay}}(J_k) = \sum_{J_j \in SQ} t_{\mathrm{schedule}}(J_j)$$

$$= \begin{cases} 0 & \text{if } t^{\mathrm{SQ}}_{\mathrm{delay}}(J_{k-1}) + t_{\mathrm{schedule}}(J_{k-1}) \leqslant t^{\mathrm{job}}_{\mathrm{interval}}(J_k) \text{ with probability } p^s(J_k), \\ t^{\mathrm{SQ}}_{\mathrm{delay}}(J_{k-1}) + t_{\mathrm{schedule}}(J_{k-1}) - t^{\mathrm{job}}_{\mathrm{interval}}(J_k), & \text{otherwise with probability } 1 - p^s(J_k). \end{cases}$$

Let $p^s(J_k)$ denote the probability that there is no task currently queued in the scheduling queue. Thus, the probability of $t^{\mathrm{SQ}}_{\mathrm{delay}}(J_k)$ being equal to 0 is $p^s(J_k)$. For simplicity, we assume that the event represented by $p^s(J_k)$ is independent of other submitted jobs. However, our approach to calculating $t^{\mathrm{SQ}}_{\mathrm{delay}}(J_k)$ does not depend on this assumption. The probability $p^s(J_k)$ can be obtained either from experimental data or through profiling.

From the above equation, the following recursive expression can be obtained for $k \geqslant 2$.

$$t^{\mathrm{SQ}}_{\mathrm{delay}}(J_k) = (1 - p^s(J_k))(t^{\mathrm{SQ}}_{\mathrm{delay}}(J_{k-1}) + t_{\mathrm{schedule}}(J_{k-1}) - t^{\mathrm{job}}_{\mathrm{interval}}(J_k)). \tag{7}$$

Applying the above equation recursively $k-1$ times, we obtain

$$t^{\mathrm{SQ}}_{\mathrm{delay}}(J_k) = \sum_{j=1}^{k} \prod_{i=j}^{k} \{(1 - p^s(J_k))(t_{\mathrm{schedule}}(J_{j-1}) - t^{\mathrm{job}}_{\mathrm{interval}}(J_j))\}. \tag{8}$$

For future reference, we sumarize the notation for scheduling and dispatching times in Table 2.

4. Scheduling algorithms

4.1. Definitions and assumptions

To facilitate the presentation of the proposed algorithm, it is necessary to introduce some additional definitions

Table 2
Notation of scheduling and dispatching times

Notation	Explanation
$t_{\text{schedule}}(J_i)$	scheduling time for job J_i
$t_{\text{dispatch}}(v_i)$	time overhead of dispatching task v_i from the scheduler to its processing element
$t_{\text{delay}}^{\text{DQ}}(v_i)$	queuing delay in a *Dispatch Queue* experienced by task v_i
$t_{\text{delay}}^{\text{SQ}}(J_i)$	queuing delay in a *Schedule Queue* experienced by job J_i
$t_{\text{interval}}^{\text{job}}(J_k)$	inter-arrival interval between two consecutive jobs J_{k-1} and J_k
$t_{\text{interval}}^{\text{task}}(v_i)$	time interval between J_k's arrival at the scheduler and v_i's arrival at its processing element

mst(e):
Note: $e = (v_j, v)$, $mst(e_{r+1}) = \infty$, $mst(e_0) = 0$, $|e_0| = 0$, and $MQ_i = \{e_1, e_2, \ldots, e_r\}$ is the message queue containing all messages scheduled to the link.
1. **for** $(g = 0\ to\ r + 1)$ **do**/* Check the idle time slots */
2. **if** $mst_{ik}(e_{g+1}) - MAX\{mst(e_g) + w_{ik}^*|e_g|, ft(v_j)\} \geqslant w_{ik}^*|e|$ **then** /* If the idle time slots */
3. **return** $mst(e_g) + w_{ik}^*|e_g|, ft(v_j)$; /* can accommodate v, return the value */
4. **end for**
5. **return** ∞; /* No such idle time slots is found, mst is set to be ∞*/

and assumptions. Let $st(v_i)$, $ft(v_i)$ and $dt(v_i)$ be the start time, finish time, and deadline of task v_i, respectively. Our scheduling algorithms are devised to determine v_i's start time, which is subject to constraints: $ft(v_i) = st(v_i) + c_{ij}$ and $ft(v_i) \leqslant dt(v_i)$, where v_i is allocated to p_j.

Let $est^j(v_r)$ be v_r's earliest start time on p_j. $est^j(v_r)$ must satisfy the following three conditions:

(4.1a) It is later than the time when all messages from v_i's predecessors arrive at p_j,
(4.1b) It is later than the delay time $t_{\text{interval}}^{\text{task}}(v_i)$, and
(4.1c) Machine p_j has an idle time slot sufficient to accommodate v_i.

Before $est^j(v_r)$ is computed, it is assumed that, without loss of generality, tasks $v_{i1}, v_{i2}, \ldots, v_{iq}$ have been allocated to p_j. The idle time slots on p_j are $[0, st(v_{i1})]$, $[ft(v_{i1}), st(v_{i2})]$, $\ldots$, $[ft(v_{i(q-1)}), st(v_{iq})]$, $[ft(v_{iq}), \infty]$, and all idle time slots are scanned from left to right. Consequently, the first idle time slot $[ft(v_{ik}), st(v_{ik+1})]$ that satisfies the following inequality is chosen:

$$st(v_{i(k+1)}) - MAX\{eat^j(v_i), t_{\text{interval}}^{\text{task}}(v_i), ft(v_{ik})\} \geqslant c_{ij}). \tag{9}$$

Thus, the earliest start time is determined as follows:

$$est^j(v_i) = MAX\{eat^j(v_i), t_{\text{interval}}^{\text{task}}(v_i), ft(v_{ik})\}, \tag{10}$$

where $eat^j(v_i)$ is the earliest available time when all messages sent from v_i's predecessors arrive at p_j. The earliest available time $eat^j(v_i)$ is computed as follows. Recall that $D(v_i)$ is a set of messages from v_i's predecessors to v_i, $eat^j(v_i, e)$ denotes the earliest available time of task v_i if message e represents the only precedence constraint. Thus, we have:

$$eat^j(v_i) = \underset{e \in D(v_i)}{MAX}\{eat^j(v_i, e)\}, \tag{11}$$

where $eat^j(v_i, e)$ can be obtained from the earliest start time of message e, $mst(e)$, which depends on how the message is routed and scheduled on the links. Thus, a message is allocated to a link if the link has an idle time slot that is later than the sender's finish time and is large enough to accommodate the message. Before presenting the expression to calculate $eat^j(v_i, e)$, we outline below the algorithm to determine $mst(e)$.

As mentioned earlier, $eat^j(v_i, e)$, can be derived from $mst(e)$. More precisely, $eat^j(v_i, e)$, given in expression (11), is equal to the finish time of message e if v_i and its predecessor that generates e are allocated to different machines, otherwise $eat^j(v_i, e)$ is fixed to be the finish time of its predecessor.

$$eat^j(v_i, e) = \begin{cases} mst(e) + |e| \times w_{sj}, & \text{if } j \neq s, \\ \quad \text{where } x_{ij} = 1 \text{ and } x_{ks} = 1 & \\ ft(v_k) & \text{otherwise.} \end{cases} \tag{12}$$

4.2. Non-reliability-cost-driven scheduling algorithms

In this section we present two variations of the list-scheduling family of algorithms, DASAP (schedule *A*s *S*oon *A*s *P*ossible) and DALAP (schedule *A*s *L*ate *A*s *P*ossible), in which system reliability is not considered. The DASAP algorithm is an extended version of ASAP, a well-known static scheduling algorithm presented in [23,39].

The DASAP algorithm, shown formally below, picks a job J at the head of SQ, if it is not empty, to schedule. The real-time tasks in J are sorted in the increasing order of their deadlines. Thus, the task with the earliest deadline is scheduled first. For each task v_i, the algorithm computes its earliest start time $est^j(v_i)$ on each p_j, then the machine on which v_i has the earliest start time is chosen. If the deadline is not guaranteed, all scheduled tasks that belong to J are rejected and deleted from DQ, otherwise v_i is moved to

the dispatch queue. Only when all the tasks in J have been moved into the dispatch queue, can these tasks be dispatched to designated machines in the heterogeneous cluster.

The DASAP algorithm:

1. Get a job J from the head of the schedule queue SQ;
2. Sort tasks in J by their deadlines in increasing order;
3. **for** each task v_i in J **do**
4. $est \leftarrow \infty$;
5. **for** each p_j in P **do**
6. **if** $(est^j(v_i) < est)$ **then** $est \leftarrow est^j(v_i)$; $x_{ij} \leftarrow 1$; $x_{ik} \leftarrow 0$ where $k \neq j$;
7. **end for**
8. **if** $est + c_{ij} \leqslant dt(v_i)$, where $x_{ij} = 1$ **then**
9. $st(v_i) \leftarrow est$; $ft(v_i) \leftarrow est + c_{ij}$;
10. Move v_i into the dispatch queue DQ;
11. **else** Reject v_i and deleted the scheduled tasks in J from the dispatch queue DQ;
12. Update information of each message;
13. **end for**
14. Goto 1. to schedule the next job;

The algorithm outlined below is a DALAP. In this algorithm, tasks start as late as possible, subject to the constraint that deadlines of all real-time tasks in a job are guaranteed. Let $lst^j(v_r)$ be the latest start time of task v_r on p_j. $lst^j(v_r)$ is subject to four conditions, of which three are identical to conditions 4.1a–4.1c presented in Section 4.1, and the fourth condition is described below. (4.2a) Task v_r has to be finished before deadline $dt(v_r)$.

Again, before calculating $lst^j(v_r)$, we assume that, without loss of generality, tasks $v_{i1}, v_{i2}, \ldots, v_{iq}$ have been allocated to machine p_j. The idle time slots on machine p_j are $[0, st(v_{i1})], [ft(v_{i1}), st(v_{i2})], \ldots, [ft(v_{iq}), \infty)$. To find the latest idle time slot that satisfies above four conditions, we scan idle time slots from right to left to select the first idle time slot $[ft(v_{ik}), st(v_{ik+1})]$ that satisfies the following inequality:

$$MIN\{st(v_{ik+1}), dt(v_i)\} - MAX\{eat^j(v_i), t^{\text{task}}_{\text{interval}}(v_i), ft(v_{ik})\} \geqslant c^j(v_i). \tag{13}$$

Hence, the latest start time is computed as below:

$$lst^j(v_i) = MAX\{eat^j(v_i), t^{\text{task}}_{\text{interval}} d(v_i), ft(v_{ik})\}. \tag{14}$$

The DALAP algorithm is a modified version of the ALAP algorithm (As Late As Possible), a static scheduling algorithm described by Marwedel [20]. Since ALAP belongs to the static scheduling category, its applications are limited to offline scheduling only. However, DALAP is an online scheduling algorithm in the sense that can dynamically schedule tasks with precedence constraints. The DALAP picks a job J at the head of SQ, if it is not empty, computes $lst^j(v_i)$ of each task v_i in J on each machine in the cluster, then selects the machine on which v_i has the latest $lst^j(v_i)$. v_i is moved into DQ, if such proper machine is available. Otherwise, job J is not schedulable, and all scheduled tasks belonging to J are deleted from DQ. DALAP is shown below.

The DALAP algorithm:

1. Get a job J from the head of the schedule queue SQ;
2. Sort tasks in J by their deadlines in increasing order;
3. **for** each task v_i in job J **do**
4. $lst \leftarrow 0$; $schedulable \leftarrow no$;
5. **for** each machine p_j in P **do**
6. **if** $lst^j(v_i)$ is available **then**
7. $schedulable \leftarrow yes$;
8. **if** $lst^j(v_i) > lst$ **then** $lst \leftarrow lst^j(v_i)$; $x_{ij} \leftarrow 1$; $x_{ik} \leftarrow 0$ where $k \neq j$;
9. **end if**
10. **end for**
11. **if** $(schedule = yes)$ **then**
12. $st(v_i) \leftarrow est$; $ft(v_i) \leftarrow est + c_{ij}$, where $x_{ij} = 1$;
13. Move v_i into the dispatch queue DQ;
14. **else** Reject v_i and deleted the scheduled tasks in J from the dispatch queue DQ;
15. Update information of each message;
16. **end for**
17. Goto 1. to schedule the next job;

4.3. A dynamic reliability-cost-driven scheduling algorithm

To improve the performance of the above algorithms, we make use of the following necessary condition to identify jobs that are not feasible for any scheduling algorithm.

Necessary Condition 1: Let v_i be a task of job J running on a cluster with m machines, then the deadline of v_i must be greater than or equal to the minimum execution time of v_i. This argument is formalized in the following expression. If job J has a feasible schedule, then:

$$\forall v_i \in J : dt(v_i) \geqslant MIN_{i=1}^{m}\{c_{ij}\}.$$

Due to the fact that DASAP and DALAP do not take reliability cost into account, we design in what follows a dynamic reliability-cost-driven (DRCD) scheduling algorithm. The DRCD algorithm improves the reliability of the system with no extra hardware cost, by incorporating reliability cost into task scheduling and reducing the overall reliability cost. The main objective of DRCD is to minimize the system reliability cost, thereby increasing the reliability that is inversely proportional to the reliability cost. Each real- time task is allocated in such a way that results in a minimal reliability cost. DRCD is described below.

The DRCD algorithm:

1. Get a job J from the head of the schedule queue SQ;
2. **If** the job is not feasible for any scheduling algorithms (based on **Necessary Condition 1**) **then**
3. **Goto** 1. to schedule the next job;
4. Sort tasks in J by their deadlines in increasing order;
5. **for** each task v_i in job J **do**
6. $st \leftarrow \infty$; $find \leftarrow no$; $rc \leftarrow \infty$; /*Initialization*/
7. **for** each machine p_j in P **do**
8. $est \leftarrow est^j(v_i)$; /* Calculate the earliest start time of v_i on p_j */
9. **if** $est + c_{ij} \leqslant dt(v_i)$ **then** /* Check whether the deadline of v_i can be guaranteed */
10. $find \leftarrow yes$;
11. $x'_{ik} \leftarrow x_{ik}$, where $1 \leqslant k \leqslant m$; $x_{ij} \leftarrow 1$; $x_{ik} \leftarrow 0$ where $k \neq j$; /* Backup the previous schedule */
12. $rc_{\mathrm{PN}} \leftarrow \lambda_j c_{ij}$; $rc_{\mathrm{LINK}} \leftarrow \sum_{v_k \in D(v_i)} \sum_{a=1}^{m} \sum_{b=1}^{m} \{-\mu_{ab} x_{ja} x_{ib} (w_{ab} e_{ji})\}$; /* Calculate the reliability cost */
13. **if** $(rc_{\mathrm{PN}} + rc_{\mathrm{LINK}} < rc)$ **or** $(rc_{\mathrm{PN}} + rc_{\mathrm{LINK}} = rc$ **and** $est < st)$ **then**
14. $st \leftarrow est$; $rc \leftarrow rc_{\mathrm{PN}} + rc_{\mathrm{LINK}}$; /* Update the schedule, minimizing the reliability cost */
15. **else** $x_{ik} \leftarrow x'_{ik}$, where $1 \leqslant k \leqslant m$; /* Rollback to the previous schedule */
16. **end if**
17. **end for**
18. **if** $(find = yes)$ **then**
19. $st(v_i) \leftarrow est$; $ft(v_i) \leftarrow est + c_{ij}$;
20. Move v_i into the dispatch queue DQ;
21. **else** Reject v_i and deleted the scheduled tasks in J from the dispatch queue DQ;
22. Update information of each message;
23. **end for**
24. **Goto** 1. to schedule the next job;

The time complexity of DRCD is given in Theorem 1 as follows:

Theorem 1. *Let n be the number of tasks, m be the number of machines in a heterogeneous cluster, and u be the number of messages in a job. The time complexity of the DRCD algorithm is* $O(m \times n^2 \times u)$.

Proof. It takes DRCD $O(\log(n))$ time to sort the real-time tasks according to their deadlines. It takes $O(u)$ time to compute the *eat*, thus, the time complexity for calculating *est* is $O(n \times u)$. Since there are m machines in the heterogeneous cluster and n real-time tasks in the job, the *for* loop takes $O(m \times n)O(n \times u)$. Therefore, time complexity of this algorithm is $O(m \times n^2 \times u)$. □

5. Performance evaluation

To evaluate performance of the proposed scheduling approach, we present in this section several sets of experimental results obtained from extensive simulations. In Section 5.1, we describe the simulator, the workload parameters, and the performance metrics of interest. Performance comparisons between our reliability-driven algorithm (DRCD) and two existing scheduling algorithms (DASAP and DALAP) are provided in Section 5.2. Section 5.3 presents results showing how job arrival rates affect guarantee ratios. Section 5.4 presents simulation results illustrating the impact of scheduling times on guarantee ratio performance. A study of the performance impact of dispatching times is presented in Section 5.5. In Section 5.6, we show how cluster sizes affect the performance of DRCD. The effect of execution time on reliability cost is illustrated in Section 5.7. Section 5.8 reports experimental results that show impacts of computational heterogeneity on the guarantee ratio. Finally, to validate the results generated from synthesized benchmarks, to study the scalability and effectiveness of DRCD on real-world applications, we applied DRCD to a benchmark representing digital signal processing (DSP) applications in Section 5.9.

5.1. The experimental platform

In our simulation experiments, it is assumed that jobs arrive at the heterogeneous cluster according to a Poisson Process [3,35]. In addition to a real-world real-time application, DSP [40], chosen as our benchmark for the experimental study, we also conducted simulation studies on three different types of real-time task graphs that are representative of many real-life parallel applications, namely, binary trees [26,34], lattices [26,34] and random graphs [1,2,8]. Workload parameters are chosen in such a way that they are either based on those used in the literature (see, for example, [1,2,8,19,29,34]) or represent reasonably realistic workload and provide some stress tests for our algorithm. For each point in the performance curves, the number of jobs arriving in the heterogeneous cluster is 20,000. The parameters used

Table 3
Parameters for simulation experiments

Parameter	Explanation	Value (Fixed) (Varied)
FR_{PN}	Failure rate of machines	$-(0.95, 0.96, \ldots, 1.05) \times 10 - 6$ /h
FR_{LINK}	Failure rate of links	Chosen uniformly from the range 7.5×10^{-6} to 12.5×10^{-6} /h
MIN_E	Minimum execution time	(5)–(15, 25, 35 s)
MAX_E	Maximum execution time	(200)–(100, 120, 140, 160, 180, 200) (170,180,190 s)
δ	Range for generating deadlines	([1, 10])–([1, 100), [1, 300],…, [1, 1100 s])
MIN_W, MAX_W	Communication weights	(0.5, 1.5)–
MIN_V, MAX_V	Communication volumes	(1, 10)–
M	Number of machines in clusters	(8)–(10,15,20,…,40)
N	Number of tasks in a job	(30, 50, 70) for Btree, (9, 25, 36, 49) for Lattice
Γ	Job arrival rate	$-(5,\ 10,\ 15,\ 20,\ 25) \times 10^{-4}$ No./s
MIN_D, MAX_D	Minimum and maximum dispatch	(1, 10)–(5, 10, 15, 20, 25 s)

Table 4
Scheduling time as a product of $m = 8$, n^2 and u

	10	30	50	70	90
Btree	0.07	2.09	9.80	27.05	57.67
Random	0.04	1.08	5	13.72	29.16
N	9	25	49	64	81
Lattice	0.08	2.00	16.14	36.70	75.59

in the simulation studies are given in Table 3. The heterogeneous cluster for the simulation experiments is described as follows:

(1) The number of machines reflects the system size of a cluster [4,18]. Its default value is 8.
(2) The Failure rate for each machine is uniformly distributed [8] in the range from 0.95×10^{-6}/h to 1.05×10^{-6}/h [26,34].
(3) The link failure rates are uniformly distributed in the range from 0.75×10^{-7} to 1.25×10^{-7}/h [34].

The computation and communication workloads for the simulation are generated as follows:

(1) For each real-time task, the worst-case computation time in the execution time vector is randomly chosen, uniformly distributed between MIN_E and MAX_E [18,19]. The scale of this range approximates the level of computational heterogeneity.
(2) Given $v_i \in V(J)$, if v_i is on p_k and v_j is on p_l, then v_i's deadline is chosen as follows: $dt(v_i) = max\{dt(v_j)\} + 1 + |e_{ji}| \times w_{lk} + max\{c_{ik}\} + \delta$, where $e_{ji} \in E(J)$, $k \in [1, m]$, and δ is randomly computed according to a uniform distribution.
(3) The dispatch time of each task is chosen uniformly between MIN_D and MAX_D. This range reflects the variance in job's size and parallelism.
(4) Since the time complexity of the scheduling algorithm is $O(m \times n^2 \times u)$, given in Theorem 1, we model the scheduling time of a job as a function of m, n, and u, namely, $10^{-5} \times (m \times n^2 \times u)$. For random graphs, we assume that $u = n/2$. The scheduling time is given in Table 4.
(5) Communication weight (w_{ij}) is chosen uniformly between MIN_W and MAX_W. The scale of this range approximates the level of communicational heterogeneity.
(6) Communication volume between two real-time tasks is uniformly selected [18] between MIN_V and MAX_V. This range reflects the variance in message size.

The performance measures in our simulation study are reliability cost (RC) that is defined in expression (4) and guarantee ratio (GR) defined as follows [19,26].

$$GR = \frac{\text{Total number of jobs guaranteed to meet their deadlines}}{\text{Total number of jobs arrived}} \times 100\%. \quad (15)$$

While reliability cost gives a measure of system reliability as a result of a particular schedule, guarantee ratio indicates how many of the arriving jobs can be scheduled, thus measuring the effectiveness and power of a scheduling algorithm.

5.2. Reliability cost

To validate the DRCD algorithm and compare its performance against two existing approaches, we have tested the reliability cost performance of DRCD, DASAP, and DALAP. The benchmark task graphs used for the evaluation include random graphs, binary trees, and lattices. We have chosen this collection of task graphs as a set of benchmarks because they are representative of various applications modeled as directed acyclic graphs. In Figs. 2 and 3 we plot reliability cost with increasing job size and job arrival rate. Since results for lattices and random graphs have similar patterns as those for binary trees, we only show results of binary trees in Fig. 2.

Fig. 2 shows that compared with the existing scheduling approaches, DRCD reduces the reliability cost of DASAP and DALAP by up to 71.4% and 66.8% (with average of 63.7% and 61.3%), respectively. The advantage of DRCD over DASAP and DALAP becomes more pronounced as the job size increases. This is expected because DASAP and DALAP do not consider reliability cost as one of their

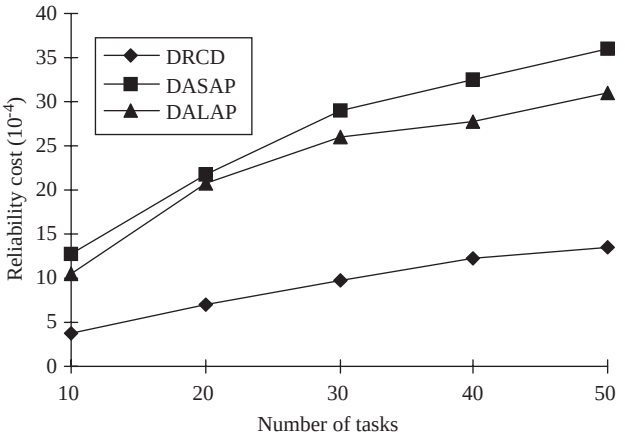

Fig. 2. Impact of task load on RC, job arrival rate $= 15 \times 10^{-4}$ No./s, binary trees and used.

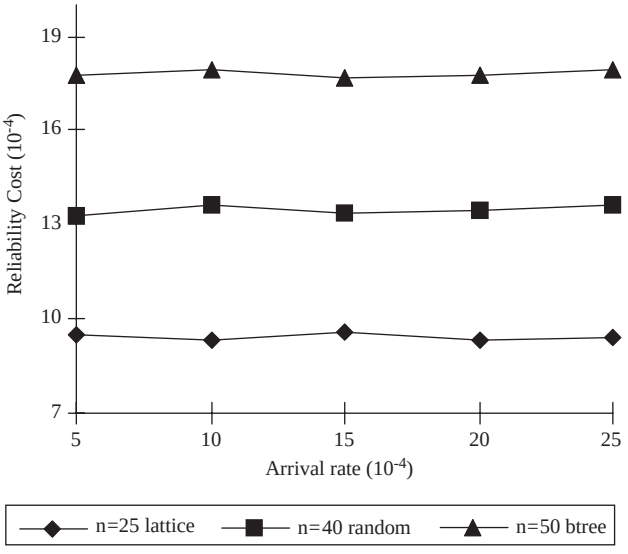

Fig. 3. Impact of job arrival rate on reliability cost, binary tree, lattice, and random graphs are used.

scheduling objectives. DRCD, however, tends to assign tasks to machines on which their reliability cost are minimum.

Another interesting result from this experiment is that job arrival rate seems to have no impact on reliability cost performance. Since DRCD, DASAP and DALAP share the same feature, we only ran DRCD algorithm on three benchmark task graphs. The results shown in Fig. 3 indicate that the reliability cost performance depends on job size rather than job arrival rate. This can be attributed to the fact that the scheduling algorithms employ an admission control strategy, which rejects jobs whose deadlines cannot be guaranteed.

5.3. Guarantee ratio

In this section we present some experimental results with respect to guarantee ratios. We present two different groups of experimental results based on a set of synthetic

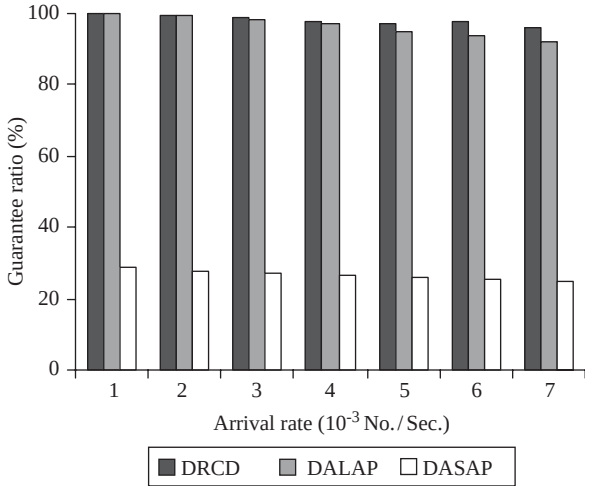

Fig. 4. Guarantee ratios of DRCD, DASAP, and DALAP, random graphs are tested, $n = 0$.

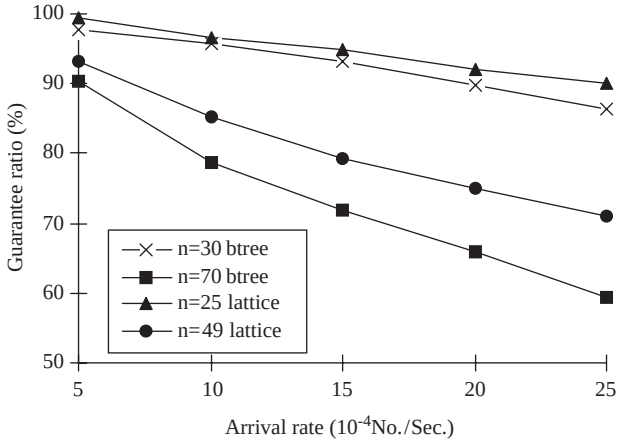

Fig. 5. Impact of job load and task load on guarantee ratio, binary trees are tested.

benchmarks. First we present performance comparison of the DRCD, DASAP, and DALAP algorithms on a heterogeneous cluster. Second, we illustrate the impact of workload and job size on guarantee ratios.

Fig. 4 shows the results of the experiment where random graphs are used as benchmarks. Results for binary trees and lattices are omitted because they are expected to be similar to those of random graphs. We set the job arrival rate from 1×10^{-3} to 7×10^{-3} No./s in increments of 1×10^{-3} No./s. Fig. 4 shows that the guarantee ratio of DRCD is slightly higher than that of the DALAP algorithm, and DRCD significantly outperforms DASAP in terms of guarantee ratio. This is mainly because the resource utilization of DRCD is less than those of DASAP and DALAP. In general, this result can be attributed to the fact that, in an effort to minimize reliability cost, the DRCD approach constantly strives to shorten execution times of each task in a job.

Fig. 5 shows the results of the second experiment on two types of benchmark task graphs: binary trees and lattices.

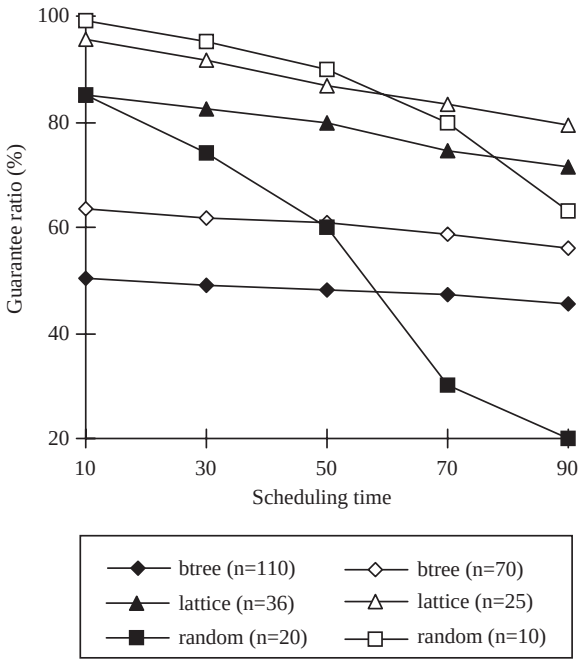

Fig. 6. Impact of scheduling time on guarantee ratio, arrival rate is 2×10^{-3} No./s.

For each curve in Fig. 5, the number of tasks in each job is fixed, whereas the job arrival rate is changed from 5×10^{-4} to 25×10^{-4} No./s. Figs. 4 and 5 show the drop in guarantee ratio with increasing values of job arrival rate. Additionally, Fig. 5 illustrates that guarantee ratio decreases as the number of tasks increases. This is because increasing job arrival rate and size results in increased scheduling and dispatching times, which in turn give rise to lowered guarantee ratios.

5.4. Scheduling time

To study the impact of scheduling time on guarantee ratios, we present in this section two sets of experimental results. First, we illustrate the results for the DRCD algorithm in Fig. 6. Second, we compare DRCD against DASAP and DALAP with respect to the impact of the scheduling time on their guarantee ratios (See Fig. 7). Although the scheduling time of each job can be estimated as a function of m, n, and u (see Section 5.1 item 4), the scheduling time in this experiment varies from 10 to 90. This simplification deflates any correlations between scheduling times and other workload parameters, but the goal of this simulation is to examine the impact of the scheduling time on system performance by controlling the scheduling time as a parameter.

Both binary tree-, lattice- and random graph-based jobs are considered. For each curve in Fig. 6, the job size is fixed and the job arrival rate is set to be 2×10^{-3} No./s. Fig. 6 shows guarantee ratio as a function of scheduling time. It reveals that the scheduling time makes significant impact on the performance of a dynamically scheduled real-time heterogeneous cluster. Without considering scheduling time,

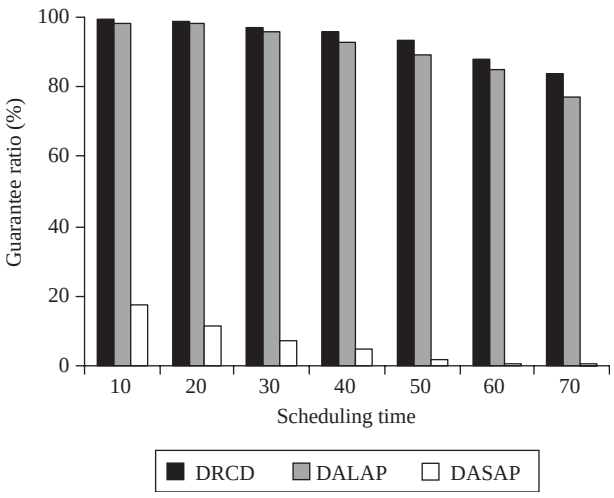

Fig. 7. Guarantee ratios of the three heuristics when scheduling time is varied, random graphs are tested, $N = 10$, arrival rate is 2×10^{-3} No./s.

the predictions on which scheduling is based cannot be accurate, thus lowering GR. This impact is more pronounced as job arrival rate increases. The result also suggests that, under the same workload, shortening the scheduling time can improve guarantee ratios, thus allowing more jobs to be completed before their given deadlines. Ahmad and Kwok have developed a parallel algorithm (referred to as PBSA) that could perform scheduling using multiple processors [2]. Therefore, it is highly desirable to apply the parallel technique reported in [2] to our algorithm, thereby shortening scheduling time to ultimately enhance the performance of the heterogeneous cluster.

Fig. 7 compares the guarantee ratios of the three heuristics when the scheduling time is varied in this experiment. We find that DRCD can outperform the other alternatives in terms of guarantee ratio, and this finding is consistent with the results shown in Fig. 4. We also observe from Fig. 7 that the guarantee ratios of the DASAP and DALAP algorithms are more sensitive to changes in the scheduling time than DRCD. The result reveals that the improvement in guarantee ratio offered by DRCD becomes more pronounced when the scheduling time is relatively large.

5.5. Dispatching time

Fig. 8 shows the impact of dispatching time of the DRCD algorithm on guarantee ratio for different values of n. Again, the job arrival rate is fixed at 2×10^{-3} No./s. Dispatching time is increased from 5 to 25 with increments of 5 s. Job size is set to 50 and 70 for binary trees, 25 for lattices, and 20 for random graphs, respectively. Fig. 8 clearly shows that decreasing dispatching time can significantly improve guarantee ratios of the heterogeneous cluster. This result strongly suggests that using a high-speed network to speed up the dispatching of scheduled tasks can substantially enhance the system performance.

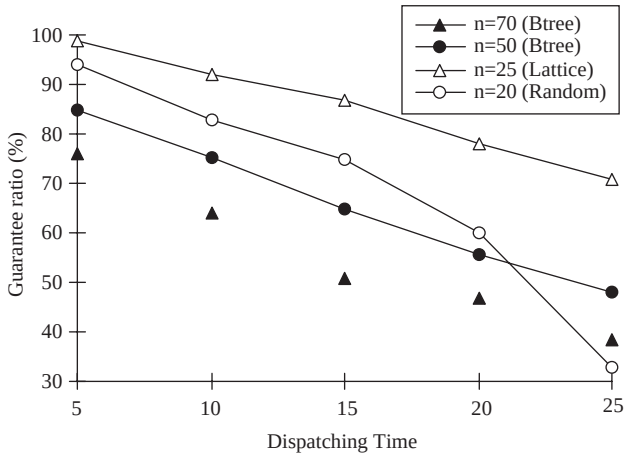

Fig. 8. Impact of dispatching time of DRCD algorithm on GR, arrival rate is 2×10^{-3} No./s.

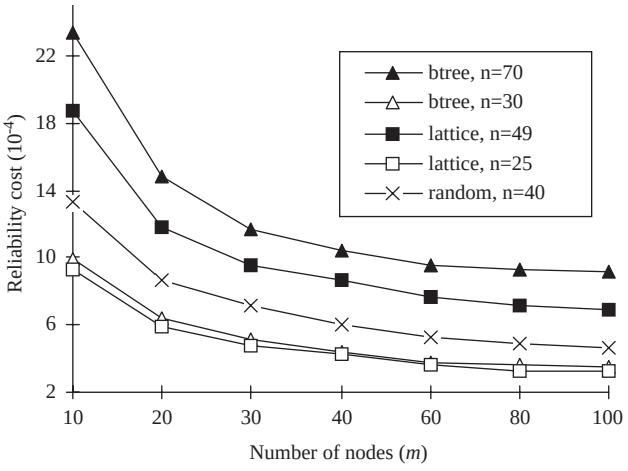

Fig. 10. Impact of the number of nodes on reliability cost of DRCD, arrival rate is 1×10^{-3} No./s.

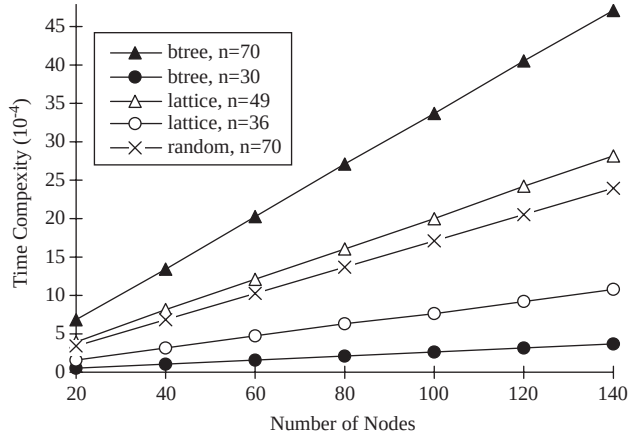

Fig. 9. Impact of the number of nodes on time complexity of the DRCD algorithm.

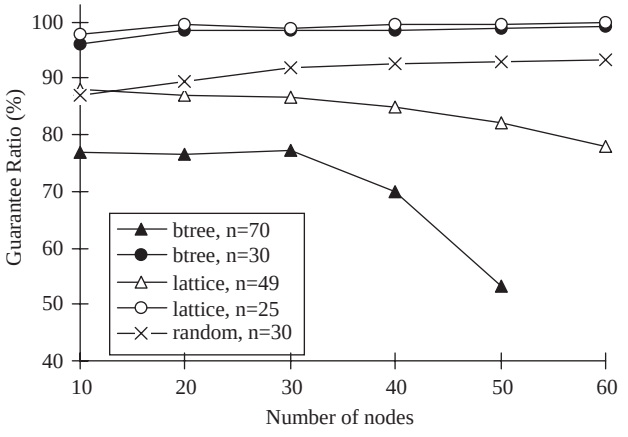

Fig. 11. Impact of the number of nodes on guarantee ratio of the DRCD algorithm, arrival rate is 1×10^{-3} No./s.

5.6. Heterogeneous cluster size

To study the impact of the heterogeneous cluster size m (number of nodes) on the performance of DRCD, we fixed the job arrival rate at 10×10^{-4} No./s and increased m from 10 up to 140. Fig. 9 shows scheduling time as a function of the heterogeneous cluster size, indicating a noticeable impact of both the heterogeneous cluster size (m) and job size (n) on scheduling time. When the job size is small, the impact of m on scheduling time is not very significant. But this impact becomes increasingly noticeable as the job size increases. This is because scheduling time is the product of m, n and u (Theorem 1).

Fig. 10 illustrates the impact of the heterogeneous cluster size on the reliability cost. It shows that under the same workload, the performance with respect to reliability cost improves as the heterogeneous cluster size increases. The main reason behind this is that for a large heterogeneous cluster, the DRCD algorithm has more choices for scheduling a real-time task. It is also observed from Fig. 10 that when the cluster size is more than 30 the improvement in reliability cost starts to diminish. This is because a higher value of m can result in a longer scheduling time (see Fig. 9), especially when the value of n is also high. This result suggests that under the workload in this experiment, it may not be cost-effective for the system to grow beyond 30 machines. An optimal value of m for a particular workload may be determined by experiments.

The impact of cluster size on guarantee ratio is shown in Fig. 11, where the guarantee ratio is plotted as a function of the number of nodes. The results indicate that the impact of the cluster size on guarantee ratios is mixed. On the negative side, a higher value of m can lead to a longer scheduling time, as illustrated in Fig. 9. On the positive side, increasing the number of machines enhances the computational capability of the system, which may in turn guarantee more jobs to be completed before their deadlines. The final result depends on which side makes more significant impact.

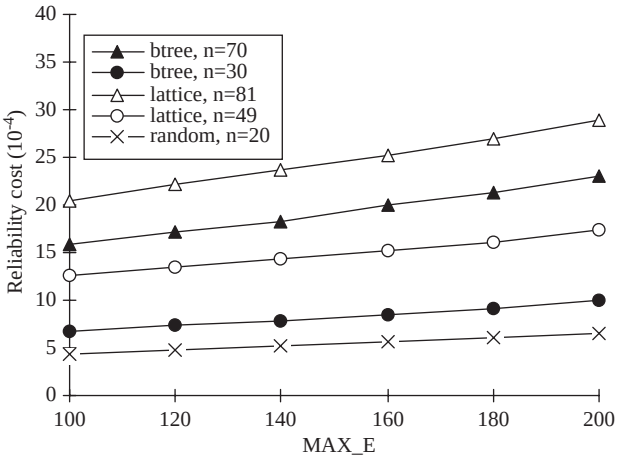

Fig. 12. Effect of the execution time on reliability cost of DRCD, arrival rate is 1×10^{-3} No./s.

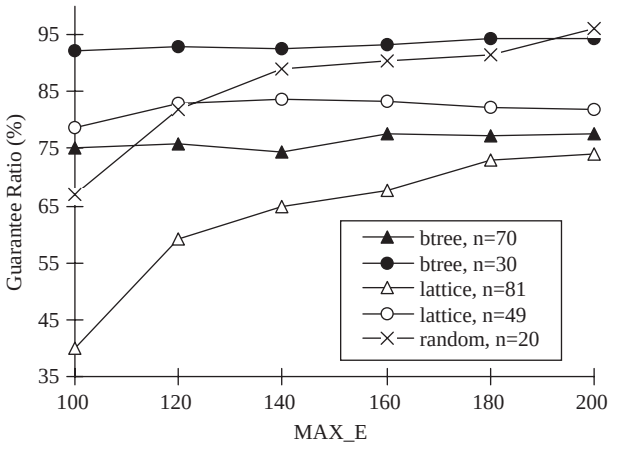

Fig. 13. Effect of the execution time on GR of DRCD, arrival rate is 1×10^{-3} No./s.

As shown in Fig. 11, three curves illustrate the positive side, and two other curves depict the negative side. We observe that when n (job size) is comparatively low, the net effect is positive (see the three curves in Fig. 11 with $n =$ 25, 30); whereas, the negative effect emerges as n becomes relatively high (see two curves in Fig. 10, with $n = 49$ and 70). This suggests that the number of machines is a critical parameter for scheduling parallel real-time jobs, which must be determined carefully based on experiments.

5.7. Execution time

Fig. 12 shows the impact of execution time on reliability cost. We only consider the DRCD algorithm, since DASAP and DALAP have similar properties and are less relevant. In this experiment, the job arrival rate is set at 10×10^{-4} No./s, and *MAX_E* is varied from 100 to 200 s, with increments of 20 s. For each value of *MAX_E*, we ran the DRDC algorithm on binary trees, lattices and random graphs. From the simulation results shown in Fig. 12, we observe that the reliability cost increases with the increase in the execution time. This is due to the simple fact that when the execution time in each c_{ij} increases the task reliability cost of machines, RC_{PN}, also increases. We can conclude from this experiment that as the execution time increases, the reliability cost of the cluster also increases.

As shown in Fig. 13, execution time also has a noticeable impact on guarantee ratios. When the value of n is low (see the curve with $n = 30$), execution time does not make a significant impact on guarantee ratio, but when n becomes large (see the curve with $n = 81$), we observe that guarantee ratios are affected noticeably by the execution time. Since the deadline is assumed to be a function of the execution time in our simulation model, the deadlines of tasks increase accordingly when execution times increase. More real-time tasks can be guaranteed if their deadlines are relaxed.

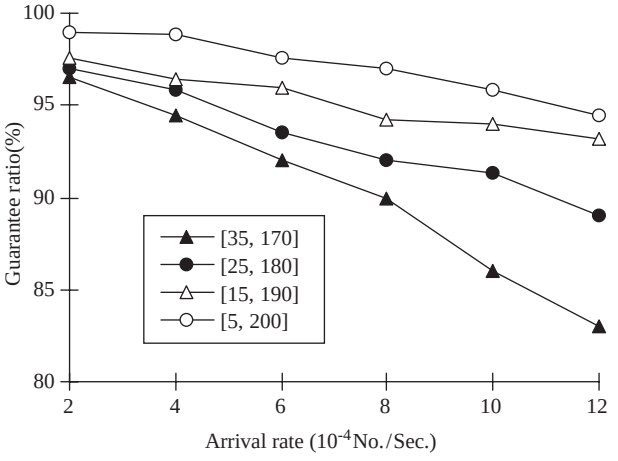

Fig. 14. Impact of computational heterogeneity on guarantee ratios, task graphs are btrees.

5.8. Computational heterogeneity

Fig. 14 shows the guarantee ratio as a function of job arrival rates, with different variances in task execution time, where job arrival rate increases from 2×10^{-4} to 12×10^{-4} No./s with increments of 2×10^{-4} No./s. Again, we only consider the DRCD algorithm and binary trees based jobs in this experiment, since the other two types of jobs behave similarly.

Computational heterogeneity is reflected by the variance in execution times. In the experiment four sets of execution times, all with the same average value, are selected uniformly from the four ranges, [5, 200], [15, 190], [25, 180] and [35, 170], respectively. These four ranges correspond to four different levels of heterogeneity, with [5, 200] being the highest. Figs. 14 and 15 indicate that the DRCD scheduling algorithm has better performance for jobs with higher computational heterogeneity. This result suggests that high computational heterogeneity helps the DRCD algorithm

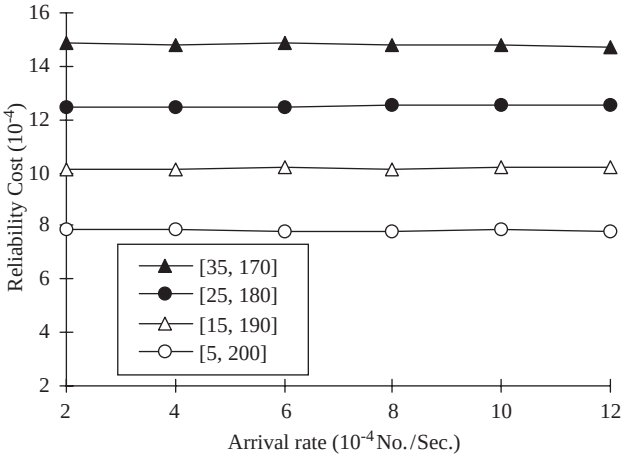

Fig. 15. Impact of computational heterogeneity on reliability cost, task graphs are btrees.

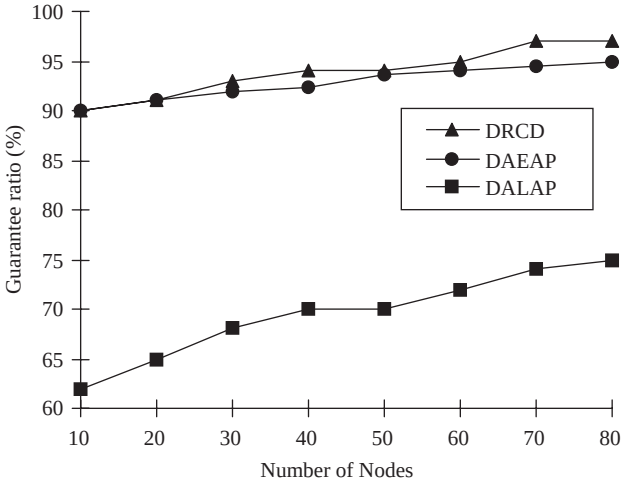

Fig. 16. Impact of the number of nodes on guarantee ratio for the DSP example.

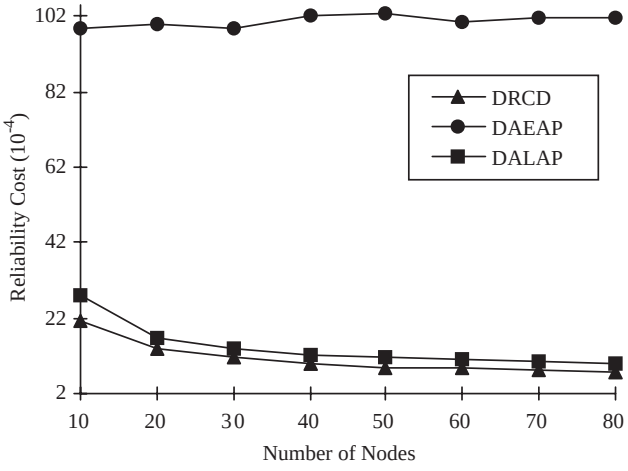

Fig. 17. Impact of the number of nodes on reliability cost for the DSP example.

increase guarantee ratios and reduce reliability costs, thereby enhancing the schedulability. This can be explained by the fact that the advantage of DRCD over the two non-reliability driven algorithms in schedulability mainly comes from the variance in tasks' reliability costs among different machines and reduced heterogeneity implies reduced variance in tasks' reliability costs. It is proved by this experiment that DRCD is efficient in terms of scheduling heterogeneous jobs, and its performance varies with the heterogeneity of the parallel real-time jobs.

5.9. Performance on real applications

The goal of this experiment is two-fold: (1) to validate the results from the synthetic application cases and (2) to test the scalability of the proposed algorithm. We chose a real-life application, a digital signal processing (DSP) system with 119 tasks in the task graph [40], as a case study to quantitatively evaluate the improvements in guarantee ratio and reliability cost as we increase the number of nodes in the cluster.

We conducted experiments with eight cluster sizes (the number of nodes is varied from 10 to 80). The guarantee ratio and reliability cost were obtained for each heterogeneous cluster, where machine failure rates were randomly chosen between 9.5×10^{-7} and 10.5×10^{-7}/h and link failure rates between 7.5×10^{-6} and 12.5×10^{-6}/h. Job arrival rates were kept constant at 1.0×10^{-4} No./ms, and ranges for generating deadlines were fixed to 500 ms. Fig. 16 shows the guarantee ratios of the DRCD, DASAP, and DALAP algorithms running on eight heterogeneous clusters. Comparing DRCD with two other algorithms, we find that the DRCD algorithm performances better than the other alternatives, and DRCD improves guarantee ratios over DASAP and DALAP by up to 3% and 45%, respectively. Fig. 17 compares reliability cost for the DRCD, DASAP, and DALAP algorithms. We observe that with respect to reliability cost, DRCD is constantly better than the other two algorithms. Specifically, DRCD can reduce the reliability cost of DASAP and DALAP by up to 92% and 25% (with average of 89% and 21%), respectively. From these results, we conclude that the proposed DRCD algorithm can achieve the most reliable allocations for both small- and large-scale applications by leveraging the reliability-cost driven technique while improving resource utilization.

6. Conclusion

Most research work in the area of real-time task scheduling in heterogeneous systems either ignored reliability issues, or only considered homogeneous clusters, or assumed independent tasks, or only schedule tasks with precedence constraints offline. In this paper, we have addressed these

issues by proposing a reliability aware algorithm (DRCD) that dynamically schedules real-time tasks with precedence constraints in heterogeneous cluster environments. To make the scheduling more practical and realistic, both scheduling time and dispatching time are incorporated in our scheduling algorithm. In the DRCD algorithm, reliability cost is used as one of the objective functions for scheduling tasks in parallel jobs. For comparison purposes, we have introduced two variations of the list-scheduling family of algorithms, DASAP and DALAP, which are greedy in nature but make no effort to maximize reliability. Simulation results show that DRCD outperforms DASAP and DALAP with respect to reliability and schedulability in both synthetic workloads and real life applications. In addition, our experiments suggested that higher computational heterogeneity is conducive to improving the schedulability of DRCD, while computational heterogeneity had no apparent effect on reliability. Simulation results also reveal that both scheduling times and dispatching times can significantly impact the effectiveness of a scheduling algorithm. Thus, it is highly desirable to substantially reduce both scheduling and dispatching times by parallelizing schedulers and providing high-speed network capabilities.

This work represents our first and preliminary attempt to study a very complicated problem. Future studies in this area are twofold. First, based on the DRCD algorithm, a dynamic fault-tolerant scheduling algorithm will be investigated, and primary/backup versions will be our approach. Second, we plan to study a more complex version of DRCD algorithm, in which power conservative issues are taken into account.

Acknowledgments

This is a substantially revised and improved version of a preliminary paper [26] that appeared in the *Proc. of* 2001 *Int'l Conference on Parallel Processing* (*ICPP*2001), pp. 113–122, September 2001. The revisions include a detailed reliability cost model, an improved scheduling algorithm that considers the reliability of links connecting machines in a cluster, consideration of more workload parameters, and performance evaluation with a real application. This work was partially supported by an NSF Grant (EPS-0091900), a start-up research fund (103295) from the research and economic development office of the New Mexico Institute of Mining and Technology, a Nebraska University Foundation Grant (26-0511-0019), a UNL Academic Program Priorities Grant, and a Chinese NSF 973 project grant (2004cb318201). We are grateful to the anonymous referees for their insightful suggestions and comments.

References

[1] T.F. Abdelzaher, K.G. Shin, Combined task and message scheduling in distributed real-time systems, IEEE Trans. Parallel Distrib. Systems 10 (11) (November 1999).

[2] I. Ahmad, Y.K. Kwok, Parallelizing the multiprocessor scheduling problem, IEEE Trans. Parallel Distrib. Systems 10 (4) (April 1999) 414–432.

[3] M. Arora, S.K. Das, R. Biswas, A de-centralized scheduling and load balancing algorithm for heterogeneous grid environments, in: Proceedings of the Workshop Scheduling and Resource Management for Cluster Computing, Vancouver, Canada, August 2002, pp. 499–505.

[4] I.D. Baev, W.M. Meleis, A. Eichenberger, Lower bounds on precedence-constrained scheduling for parallel machines, in: Proceedings of the 29th International Conference on Parallel Processing, 2000, pp. 549–553.

[5] O. Beaumont, V. Boudet, Y. Robert, A realistic model and an efficient heuristic for scheduling with heterogeneous processors, in: Proceedings of the 11th Heterogeneous Computing Workshop, 2002.

[6] O. Beaumont, A. Legrand, Y. Robert, ENS Lyon, L. Carter, J. Ferrante, Bandwidth-centric allocation of independent tasks on heterogeneous platforms, in: Proceedings of the International Parallel and Distributed Processing Symposium, 2002.

[7] M. Cosnard, E. Jeannot, T. Yang, SLC: symbolic scheduling for executing parameterized task graphs on multimachines, in: Proceedings of the 28th International Conference on Parallel Processing, Fukushima, Japan, 1999.

[8] A. Doğan, F. Özgüner, Reliable matching and scheduling of precedence-constrained tasks in heterogeneous distributed computing, in: Proceedings of the International Conference on Parallel Processing, 2000, pp. 307–314.

[9] D.G. Feitelson, L. Rudolph, Job scheduling for parallel supercomputers, Encyclopedia of Computer Science and Technology, vol. 38, Marcel Dekker, Inc., New York, 1998.

[10] E.N. Huh, L.R. Welch, B.A. Shirazi, C.D. Cavanaugh, Heterogeneous resource management for dynamic real-time systems, in: Proceedings of the 9th Heterogeneous Computing Workshop, 2000, pp. 287–296.

[11] M. Iverson, F. Özgüner, Dynamic, competitive scheduling of multiple DAGs in a distributed heterogeneous environment, in: Proceedings of the 7th Heterogeneous Computing Workshop, 1998, pp. 70–78.

[12] X. Jia, W. Zhao, J. Li, An integrated routing and admission control mechanism for real-time multicast connections in ATM networks, IEEE Trans. Commun. 49 (9) (September 2001) 1515–1519.

[13] V. Kalogeraki, P.M. Melliar-Smith, L.E. Moser, Dynamic scheduling for soft real-time distributed object systems, in: Proceedings of the IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2000, pp. 114–121.

[14] D. Kebbal, E.G. Talbi, J.M. Geib, Building and scheduling parallel adaptive applications in heterogeneous environments, in: Proceedings of the IEEE International Workshop Cluster Computing, 1999, pp. 195–201.

[15] Y.K. Kwok, I. Ahmad, FASTEST: a practical low-complexity algorithm for compile-time assignment of parallel programs to multiprocessors, IEEE Trans. Parallel Distrib. Systems 10 (2) (February 1999) 147–159.

[16] Y.K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surveys 31 (4) (December 1999) 406–471.

[17] T. Lundqvist, P. Stenstrom, Timing anomalies in dynamically scheduled micromachines, in: Proceedings of the IEEE Real-Time Systems Symposium, 1999, pp. 12–21.

[18] M. Maheswaran, H.J. Siegel, A dynamic matching and scheduling algorithm for heterogeneous computing systems, in: Proceedings of the 7th Heterogeneous Computing Workshop, 1998, pp. 57–69.

[19] G. Manimaran, C.S.R. Murthy, An efficient dynamic scheduling algorithm for multimachine real-time systems, IEEE Trans. Parallel Distrib. System 9 (3) (1998) 312–319.

[20] P. Marwedel, A new synthesis algorithm for the MIMOLA software system, Proceedings of the 23rd Design Automatic Conference (Las Vegas, NV), July 1986, pp. 271–277.

[21] J.C. Palencia, H.M. Gonzalez, Schedulability analysis for tasks with static and dynamic offsets, in: Proceedings of the IEEE Real-Time Systems Symposium, 1998, pp. 26–37.
[22] M.A. Palis, Online real-time job scheduling with rate of progress guarantees, in: Proceedings of the sixth International Symposium on Parallel Architectures, Algorithms, and Networks, Manila, Philippines, 2002, pp. 65–70.
[23] P.G. Paulin, J.P. Knight, Force directed scheduling for the behavioral synthesis of ASIC's, IEEE Trans. Comput. Aided Design 8 (June 1989) 661–679.
[24] X. Qin, H. Jiang, C.S. Xie, Z.F. Han, Reliability-driven scheduling for real-time tasks with precedence constraints in heterogeneous distributed systems, in: Proceedings of the International Conference on Parallel and Distributed Computing Systems, 2000, pp. 617–623.
[25] X. Qin, H. Jiang, Y. Zhu, D. Swanson, Dynamic load balancing for I/O-intensive tasks on heterogeneous clusters, in: Proceedings of the International Conference on High Performance Computing, 2003, pp. 300–309.
[26] X. Qin, H. Jiang, Dynamic, reliability-driven scheduling of parallel real-time jobs in heterogeneous systems, in: Proceedings of the International Conference on Parallel Processing, Valencia, Spain, 2001, pp. 113–122.
[27] X. Qin, H. Jiang, D.R. Swanson, An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems, in: Proceedings of the International Conference on Parallel Processing, Vancouver, Canada, 2002, pp. 360–368.
[28] A. Radulescu, A.J.C. van Gemund, Fast and effective task scheduling in heterogeneous systems, in: Proceedings of the Euromicro Conference on Real-Time Systems, 2000, pp. 229–238.
[29] S. Ranaweera, D.P. Agrawal, Scheduling of periodic time critical applications for pipelined execution on heterogeneous systems, in: Proceedings of the International Conference on Parallel Processing, 2001, pp. 131–138.
[30] S. Sahni, G. Vairaktarakis, Scheduling for distributed computing, in: Proceedings of the IEEE Workshop Future Trends of Distributed Computing Systems, 1997, pp. 284–289.
[31] R.M. Santos, J. Santos, J. Orozco, Scheduling heterogeneous multimedia servers: different QoS for hard, soft and non real-time clients, in: Proceedings of the Euromicro Conference on Real-Time Systems, 2000, pp. 247–253.
[32] B. Shirazi, H.Y. Youn, D. Lorts, Evaluation of static scheduling heuristics for real-time multiprocessing, Parallel Process. Lett. 5 (4) (1995) 599–610.
[33] G.C. Sih, E.A. Lee, A compile-time scheduling heuristic for interconnection-constrained heterogeneous machine architectures, IEEE Trans. Parallel Distrib. Systems 4 (2) (1993) 175–187.
[34] S. Srinivasan, N.K. Jha, Safety and reliability driven task allocation in distributed systems, IEEE Trans. Parallel Distrib. Systems 10 (3) (1999) 238–251.
[35] X.Y. Tang, S.T. Chanson, Optimizing static job scheduling in a network of heterogeneous computers, in: Proceedings of the International Conference on Parallel Processing, 2000, pp. 373–382.
[36] M.E. Thomadakis, Jyh-Charn Liu, On the efficient scheduling of non-periodic tasks in hard real-time systems, in: Proceedings of the IEEE Real-Time Systems Symposium, 1999, pp. 148–151.
[37] H. Topcuoglu, S. Hariri, M.Y. Wu, Task scheduling algorithms for heterogeneous machines, in: Proceedings of the Heterogeneous Computing Workshop, 1999, pp. 3–14.
[38] D.B. Tracy, Noemix, H.J. Siegel, A. Maciejewski, Static mapping heuristics for tasks with dependencies, priorities, deadlines, and multiple versions in heterogeneous environments, in: Proceedings of the International Parallel and Distributed Processing Symposium, 2002.
[39] C. Tseng, D.P. Siewoirek, Automated synthesis of data paths in digital systems, IEEE Trans. Comput. Aided Design CAD-5 (July 1986) 379–395.
[40] C.M. Woodside, G.G. Monforton, Fast allocation of processes in distributed and parallel systems, IEEE Trans. Parallel Distrib. Systems 4 (2) (February 1993) 164–174.
[41] M.Y. Wu, W. Shu, Y. Chen, Runtime parallel incremental scheduling of DAGs, in: Proceedings of the International Conference on Parallel Processing, 2000, pp. 541–548.
[42] Y. Zhang, H. Kameda, K. Shimizu, Adaptive bidding load balancing algorithms in heterogeneous distributed systems, in: Proceedings of the International Symposium on Modeling, Analysis, and Simulation on Computer and Telecommunication Systems, 1994, pp. 250–254.
[43] Y. Zhang, A. Sivasubramaniam, Scheduling best-effort and real-time pipeline application on time-shared clusters, in: Proceedings of the International Symposium Parallel Architecture and Algorithm, 2001.

Xiao Qin received the B.S. and M.S. degrees in computer science from Huazhong University of Science and Technology in 1992 and 1999, respectively. He received the Ph.D. degree in computer science from the University of Nebraska-Lincoln in 2004. Currently, he is an Assistant Professor in the department of computer science at the New Mexico Institute of Mining and Technology. He had served as a subject area editor of IEEE Distributed System Online (2000–2001). His research interests are in parallel and distributed systems, storage systems, real-time computing, performance evaluation, and fault-tolerance. He is a member of the IEEE.

Hong Jiang received the B.Sc. degree in Computer Engineering in 1982 from Huazhong University of Science and Technology, Wuhan, China; the M.A.Sc. degree in Computer Engineering in 1987 from the University of Toronto, Toronto, Canada; and the Ph.D. degree in Computer Science in 1991 from the Texas A&M University, College Station, Texas, USA. Since August 1991 he has been at the University of Nebraska-Lincoln, Lincoln, Nebraska, USA, where he is Associate Professor and Vice Chair in the Department of Computer Science and Engineering. His present research interests are computer architecture, parallel/distributed computing, computer storage systems and parallel I/O, performance evaluation, middleware, networking, and computational engineering. He has over 90 publications in major journals and international conferences in these areas and his research has been supported by NSF, DOD and the State of Nebraska. Dr. Jiang is a Member of ACM, the IEEE Computer Society, and the ACM SIGARCH and ACM SIGCOMM.