

# Dynamic, Reliability-driven Scheduling of Parallel Real-time Jobs in Heterogeneous Systems\*

†Xiao Qin ††Hong Jiang

††Department of Computer Science and Engineering  
University of Nebraska-Lincoln

Lincoln, NE 68588-0115, ([jiang@cse.unl.edu](mailto:jiang@cse.unl.edu))

†School of Computing and Information Technology  
Griffith University, Brisbane, Queensland 4111, Australia

## Abstract

*In this paper, a heuristic dynamic scheduling scheme for parallel real-time jobs in a heterogeneous system is presented. The parallel real-time jobs studied in this paper are modelled by directed acyclic graphs (DAG). We assume a scheduling environment where parallel real-time jobs arrive at a heterogeneous system following a Poisson process. The scheduling algorithms developed in this paper take the reliability measure into account, in order to enhance the reliability of the heterogeneous system without any additional hardware cost. In addition, scheduling time and dispatch time are both incorporated into our scheduling scheme so as to make the scheduling result more realistic and precise. Admission control is in place so that a parallel real-time job whose deadline cannot be guaranteed is rejected by the system. The performance of the proposed scheme is evaluated via extensive simulations. The simulation results show that the heuristic algorithm performs significantly better than two other algorithms that do not consider reliability cost. Furthermore, results suggest that shortening the scheduling time results in a higher guarantee ratio. Hence, if parallel scheduling algorithm is devised and employed to shorten the scheduling time, the performance of the heterogeneous system will be further enhanced.*

## 1. Introduction

Heterogeneous systems have become increasingly widely used for scientific and commercial applications. These systems require a mixture of general-purpose processors, programmable digital processors, and application specific integrated circuits [22]. A heterogeneous system involves multiple heterogeneous modules that interact with one another to solve a problem [3][23]. In a heterogeneous system, applications have subtasks that have diverse execution requirements. The

subtasks must be assigned to machines (processors) and ordered for execution such that the overall application execution time is minimized [14].

The studies in heterogeneous systems reveal a number of challenges that must be addressed. One of them is the dynamic scheduling of parallel real-time jobs in heterogeneous systems. The scheduling of parallel jobs is a key factor in obtaining high performance in heterogeneous systems [11][19][26]. The objective of real-time scheduling is to map tasks onto processors and order their execution so that task precedence requirements are satisfied and a minimum schedule length, when attainable, is given. Many scheduling schemes have been introduced for parallel computing systems [7]. Some of them assume precedence constraints among tasks being scheduled, and use directed acyclic graphs (DAG) to model such constraints [9][14][26].

The purpose of this work is to provide a framework of real-time scheduling by which parallel jobs are scheduled dynamically, as they arrive at a heterogeneous system. In this framework, a designated processor, called the *scheduler*, is responsible for dynamically scheduling real-time jobs as they arrive, and dispatching them to other processors, called the *processing elements*, to execute. The proposed algorithm takes into account the dispatch time, the schedule time and the reliability cost, factors that have been ignored by most scheduling schemes that deal with real-time heterogeneous systems. 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 most relevant to our work is briefly described. The system model and assumptions are presented in Section 3. Section 4 proposes three dynamic scheduling algorithms. Performance analysis 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.

\* This work was partially supported by an NSF grant (EPS-0091900) and a Nebraska University Foundation grant (26-0511-0019)

## 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][16][17][18] and dynamic (on-line) [10][13][12][21][25]. While many algorithms assume that real-time tasks are independent with one other [16][18][25], others schedule tasks with precedence constraints [17][18], which are represented by directed acyclic graphs (DAG). We have recently extended the non-real-time DAG into a real-time DAG to study real-time scheduling of tasks with precedence constraints [17]. Paper [18] presents an algorithm for offline scheduling of communicating tasks with precedence and exclusion constraints in distributed hard real-time systems. However, these algorithms, while do consider 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 tasks are often represented by a directed acyclic graph (DAG) [20][27][2][4][12]. Paper [27] describes a runtime parallel incremental DAG scheduling approach. Paper [4] presents a scheduling algorithm for a parameterized DAG, which first derives symbolic linear clusters and then assigns task clusters to processors. The scheduling algorithms for DAGs in these two papers, however, are also devised for homogeneous systems.

The issue of scheduling on heterogeneous systems has been studied in many papers [9][19][26][20][5][24]. It is suggested that minimizing task's completion time leads to a minimal start time of the task [14][26]. The performance of the low cost static list scheduling algorithm FCF and dynamic list scheduling algorithm FLB is investigated in [10]. Two low-complexity efficient heuristics, the heterogeneous Earliest-Finish-Time (HEFT) algorithm and the Critical-Path-on-a-Processor (CPOP) algorithm for DAGs, are studied in [26]. Paper [9] presents a matching and scheduling framework where multiple applications compete for computational resources on the network. A scalable scheduling scheme called STDS for heterogeneous systems is developed in [20]. A dynamic matching and scheduling algorithm for heterogeneous system is investigated in [14], whereas, static job scheduling schemes that distribute workload in a heterogeneous system are proposed in [27]. In [5], two cost functions that can be incorporated into a matching and scheduling algorithm for tasks with precedence constraints are introduced to consider the reliability of different resources in the system while making decisions. Unfortunately, all these scheduling algorithms assume that tasks in the system are non-real-time. Non-real-time scheduling algorithms are unable to schedule real-time

jobs efficiently, simply because they cannot meet the predictability requirement of real-time jobs.

Some work has been done to combine real-time computing with heterogeneous systems [6][8][21]. In [6], the real-time scheduling issue in heterogeneous systems is addressed. A solution for the dynamic resource management problem in real-time heterogeneous systems is proposed in [8]. These two papers, however, do not consider reliability. Paper [21] introduces a probabilistic model for a clients/server multimedia system, in which the server provides different qualities of service for hard real-time and soft real-time jobs. A new real-time scheduling concept based on the hybrid deterministic/probabilistic analysis is also presented in [21]. This model takes both the real-time and heterogeneous issues into consideration. However, one major difference between this approach and ours is that the model in [21] does not consider reliability. We have proposed a real-time scheduling in heterogeneous systems, which can minimize the reliability cost of the system [17]. While scheduling algorithms developed in [17] are static, algorithms studied in this paper, on the other hand, are dynamic.

To the best of our knowledge, scheduling time and dispatching time are ignored by most of the dynamic non-real-time and real-time scheduling algorithms. To make the real-time scheduling results more precise, both scheduling time and dispatching time should be incorporated in dynamic scheduling algorithms. It is for this reason that our study takes a close look at the impact of scheduling time and dispatching time on scheduling performance.

## 3. System

In this session, the workload model is presented, followed by a description of the scheduler model in the dynamic environment, and of the relevant definitions.

### 3.1 The Workload Model

It is assumed that jobs arrive at the heterogeneous system according to a Poisson Process. The parallel job is model by a directed acyclic graph (DAG). A real-time DAG is defined as  $J = \{V, E\}$ , where  $V = \{v_1, v_2, \dots, v_n\}$  represents the set of real-time tasks, and the set of weighted and directed edges  $E$  represents communication among real-time tasks.  $e_{ij} = (v_i, v_j) \in E$  indicates a message sent from task  $v_i$  to  $v_j$ , and  $|e_{ij}|$  denotes the volume of data sent between these tasks.

The heterogeneous system is modelled by a set  $P = \{p_1, p_2, \dots, p_m\}$ , where  $p_i$  is a processor with local memory. Processors in the heterogeneous system are connected with one other by a high-speed network. A processor communicates with other processors through message passing, and the communication time between two tasks

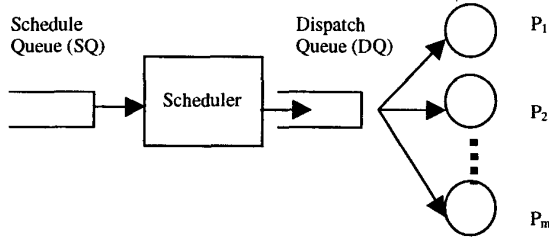
assigned to the same processor is assumed to be zero. A measure of *computational heterogeneity* is modelled as a function,  $C: V \times P \rightarrow R$ , which represents the execution time of each task on each available processor in the heterogeneous system.  $c(v_i)$  denotes the execution time of task  $v_i$  on processor  $p_j$ .

A measure of *communicational heterogeneity* is modelled by a function,  $M: E \times P \times P \rightarrow R$ , in which the communication time for transferring a message  $e = (v_s, v_r)$  from task  $v_s$  on processor  $p_i$  to task  $v_r$  on processor  $p_j$  is determined by  $w_{ij} * |e|$ .  $w_{ij}$ , the weight on the edge between processors  $p_i$  and  $p_j$  represents the time for transmitting a message of unit length between the two processors. Thus,  $w_{ij}$  can be viewed as a measure of *communicational heterogeneity*.

We use the same reliability model as the one defined in [17][23]. In this model, processor failures are assumed to be independent, and follow a Poisson Process with a constant failure rate [23]. Failures of communication links are not considered here. The reliability cost of a task  $v_i$  on a processor  $p_j$  is the product of  $p_j$ 's failure rate  $\lambda_j$  and  $v_i$ 's execution time on  $p_j$ . Thus, the reliability cost of a task schedule is the summation over all tasks' reliability costs based on the given schedule. Given a heterogeneous  $P$ , the reliability cost is defined below [17], where  $pr(v_i) = j$  indicates that task  $v_i$  is allocated to  $p_j$ .

$$RC(P) = \sum_{j=1}^m \sum_{pr(v_i)=j} \lambda_j c^j(v_i)$$

The reliability is given by  $e^{-RC(P)}$ . It is intuitive that scheduling a task with larger execution time to a more reliable processor is a good approach to increase the system's overall reliability.



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

### 3.2 The Scheduler Model

Fig.1 depicts the scheduler model in the real-time heterogeneous distributed environment. We assume that all parallel jobs arrive at a central processor, called the *scheduler*, where they wait in a *schedule queue* (SQ) to be scheduled in the heterogeneous system. The scheduler 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 sent to designated processors, also called processing elements, for execution. A parallel job is considered acceptable if all tasks in this job can be completed before their deadlines; otherwise, the job is rejected.

### 3.3 Scheduling and Dispatching Times

In a dynamic scheduling environment, it takes the 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 of the dynamic real-time scheduling algorithms either assume zero scheduling time or do not take scheduling time into account. In real systems, a scheduling algorithm that does not consider scheduling time may not be practical. Therefore in this study we will incorporate scheduling time into the proposed scheme. In addition, dispatch time, the time it takes the scheduler to send real-time tasks of an accepted job from the DQ to other processors is also taken into consideration in our scheme.

Assume that  $v_i$  is a real-time task in job  $J_k$ , thus,  $v_i \in V(J_k)$ . Let  $p(v_i)$  denote the time overhead of dispatching task  $v_i$  from the scheduler to the processing element. Let  $s(J_j)$  denote the scheduling time for job  $J_j$ . Let  $w^D(v_i)$  denote the queuing delay in DQ experienced by task  $v_i$  and  $w^S(J_j)$  denote the queuing delay in SQ experienced by job  $J_j$ . Let  $a(J_k)$  denote the inter-arrival interval between two consecutive jobs  $J_{k-1}$  and  $J_k$ . Let  $d(v_i)$  denote the time interval between  $J_k$ 's arrival at the scheduler and  $v_i$ 's arrival at the target processing element. Thus,

$$d(v_i) = w^S(J_k) + w^D(v_i) + s(J_k) + p(v_i) \\ = \sum_{J_j \in SQ} s(J_j) + \sum_{v_j \in DQ} p(v_j) + s(J_k) + p(v_i) \quad (1)$$

$$\text{where } w^S(J_k) = \sum_{J_j \in SQ} s(J_j) =$$

$$\begin{cases} w^S(J_{k-1}) + s(J_{k-1}) - a(J_k), & \text{if } w^S(J_{k-1}) + s(J_{k-1}) > a(J_k) \\ & \text{with probability } p^S(J_k) \\ 0, & \text{if } w^S(J_{k-1}) + s(J_{k-1}) \leq a(J_k) \\ & \text{with probability } 1 - p^S(J_k) \end{cases} \quad (2)$$

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

$$w^S(J_k) = P^S(J_k)[w^S(J_{k-1}) + s(J_{k-1}) - a(J_k)] \quad (3)$$

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

$$w^S(J_k) = \sum_{j=1}^k \prod_{i=j}^k \{p^S(J_i)[s(J_{i-1}) - a(J_i)]\} \quad (4)$$

## 4. Scheduling Algorithms

### 4.1 Definitions and Assumptions

To facilitate the presentation of the proposed

algorithm, it is necessary to introduce some additional definitions and assumptions. Each task  $v_i$  in a parallel job has a known execution time  $c^j(v_i)$  on processor  $p_j$ , and deadline  $dt(v_i)$ . Our scheduling algorithms are devised to determine  $v_i$ 's start time  $st(v_i)$ . Let  $ft(v_i)$  be the finish time of  $v_i$ , which is subject to constraints:  $ft(v_i) = st(v_i) + c^j(v_i)$  and  $ft(v_i) \leq dt(v_i)$ , where  $j = pr(v_i)$  is the processor to which  $v_i$  is allocated.

Let  $est^j(v_i)$  be  $v_i$ 's earliest start time on processor  $p_j$ .  $est^j(v_i)$  satisfies the following three conditions:

- (a) It is later than the time when all messages from  $v_i$ 's predecessors arrive at processor  $p_j$ ,
- (b) It is later than the delay time  $d(v_i)$ , and
- (c) Processor  $p_j$  has an idle time slot sufficient to accommodate  $v_i$ .

Before  $est^j(v_i)$  is computed, we assume that tasks  $v_{i1}, v_{i2}, \dots, v_{iq}$  have been allocated to  $p_j$ . Obviously, idle time slots on processor  $p_j$  are  $[0, st(v_{i1})]$ ,  $[ft(v_{i1}), st(v_{i2})]$ ,  $\dots$ ,  $[ft(v_{i(q-1)}), st(v_{iq})]$ ,  $[ft(v_{iq}), \infty)$ . We scan all idle time slots from left to right, then select the first idle time slot  $[ft(v_{ik}), st(v_{i(k+1)})]$  that satisfies the following inequality,

$$st(v_{i(k+1)}) - \text{MAX} \{eat^j(v_i), d(v_i), ft(v_{ik})\} \geq c^j(v_i) \quad (5)$$

Thus, the earliest start time is determined as follows:

$$est^j(v_i) = \text{MAX} \{eat^j(v_i), d(v_i), ft(v_{ik})\} \quad (6)$$

where  $eat^j(v_i)$  is the earliest available time when all messages sent from  $v_i$ 's predecessors arrive at  $p_j$ .

Let  $D(v_i)$  be the 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  $e \in D(v_i)$  represents the *only* precedence constraint. The earliest available time  $eat^j(v_i)$  is computed as,

$$eat^j(v_i) = \text{MAX}_{e \in D(v_i)} \{eat^j(v_i, e)\} \quad (7)$$

$eat^j(v_i, e)$  can be derived from the earliest start time of the message  $e$ ,  $mst(e)$ . Algorithms for calculating  $mst(e)$  and  $eat^j(v_i, e)$  are given in [17] and will not be presented here due to space constraint for this paper.

## 4.2 Dynamic Scheduling Algorithm

In this section we first present two dynamic scheduling algorithms, AEAP (schedule As Early As Possible) and ALAP (schedule As Late As Possible), then develop a dynamic reliability-driven scheduling algorithm, which enhances the system reliability by considering the reliability cost.

### 4.2.1 Scheduling As Early As Possible

The dynamic AEAP algorithm (DAEAP), 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 processor  $p_j$ , then the processor 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 processors in the heterogeneous system.

### The DAEAP 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 processor  $p_j$  in  $P$  **do**
6.     **if** ( $est^j(v_i) < est$ ) **then**  $est \leftarrow est^j(v_i)$ ;  $pr(v_i) \leftarrow j$ ;
7.   **end for**
8.   **if**  $est + c^j(v_i) \leq dt(v_i)$ , where  $j = pr(v_i)$  **then**
9.      $st(v_i) \leftarrow est$ ;  $ft(v_i) \leftarrow est + c^j(v_i)$ ;
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;

### 4.2.2 Scheduling As Late As Possible

The algorithm outlined below is a dynamic ALAP (DALAP). In this algorithm, tasks start as late as possible, subject to the constraint that deadlines of all real-time tasks in a job be guaranteed. Let  $lst^j(v_i)$  be the latest start time of task  $v_i$  on processor  $p_j$ .  $lst^j(v_i)$  must satisfy the following four conditions:

- (a) It is later than the time when all messages from  $v_i$ 's predecessors arrive at processor  $p_j$ ,
- (b) It is later than the delay time  $d(v_i)$ ,
- (c) Processor  $p_j$  has an idle time slot sufficient to accommodate  $v_i$ , and
- (d) It can complete before deadline  $dt(v_i)$ .

Again, we assume that before calculating  $lst^j(v_i)$ , tasks  $v_{i1}, v_{i2}, \dots, v_{iq}$  have been allocated to  $p_j$ . The idle time slots on processor  $p_j$  are  $[0, st(v_{i1})]$ ,  $[ft(v_{i1}), st(v_{i2})]$ ,  $\dots$ ,  $[ft(v_{i(q-1)}), st(v_{iq})]$ ,  $[ft(v_{iq}), \infty)$ . 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  $[ft(v_{i(k+1)}), st(v_{ik})]$  that satisfies the following inequality,

$$\text{MIN} \{st(v_{i(k+1)}), dt(v_i)\} - \text{MAX} \{eat^j(v_i), d(v_i), ft(v_{ik})\} \geq c^j(v_i) \quad (8)$$

Hence, the latest start time is computed as below:

$$lst^j(v_i) = \text{MAX} \{eat^j(v_i), d(v_i), ft(v_{ik})\} \quad (9)$$

The DALAP, shown below, 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 processor in the system, then selects the processor on which  $v_i$  has the latest  $lst^j(v_i)$ .  $v_i$  is moved into  $DQ$ , if such proper processor is available. Otherwise, job  $J$  is not schedulable, and all scheduled tasks belonging to  $J$  are deleted from  $DQ$ .

### 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 processor  $p_j$  in  $P$  **do**
6. **if**  $lst(v_i)$  is available **then**
7.  $schedulable \leftarrow yes$ ;
8. **if**  $lst(v_i) > lst$  **then**  $lst \leftarrow lst(v_i)$ ;  $pr(v_i) \leftarrow j$ ;
9. **end if**
10. **end for**
11. **if** ( $schedulable = yes$ ) **then**
12.  $st(v_i) \leftarrow est$ ;  $ft(v_i) \leftarrow est + c^j(v_i)$ , where  $j = pr(v_i)$ ;
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;

### 4.2.3 Dynamic RCD Scheduling

For a real-time job, if all tasks in it can be completed before their deadlines, then this job is said to have a feasible schedule. In other words, a job  $J$  has a feasible schedule if  $\forall v_i \in J: st(v_i) + c^j(v_i) \leq dt(v_i)$ , where  $j = pr(v_i)$ .

**Theorem 1.** If job  $J$  has a feasible schedule, the following inequality must be satisfied:

$$\forall v_i \in J: dt(v_i) \geq \text{MIN}_{p_j \in P} \{c^j(v_i)\}$$

**Proof.** Assume that the theorem is incorrect, then,  $\exists v_i \in J: dt(v_i) < \text{MIN}_{p_j \in P} \{c^j(v_i)\}$ , we have  $\forall p_j \in P: dt(v_i) <$

$c^j(v_i)$ , in other words,  $\forall p_j \in P: dt(v_i) - c^j(v_i) < 0$ . Since  $st(v_i) > 0$ , we obtain the inequality,  $\forall p_j \in P: st(v_i) > dt(v_i) - c^j(v_i)$ , we get  $\forall p_j \in P: st(v_i) + c^j(v_i) > dt(v_i)$ . Thus,  $\exists v_i \in J \{ \forall p_j \in P: st(v_i) + c^j(v_i) > dt(v_i) \}$ . Based on the definition of feasible schedule,  $J$  has no feasible schedules. A contradiction. Thus, Theorem 1 is proven.

DAEAP and DALAP are two algorithms that do not take reliability cost into account. In what follows, we design a dynamic reliability-cost-driven scheduling algorithm (DRCD). DRCD improves the reliability of the system with no extra hardware cost, by incorporating reliability cost into task scheduling. The main objective of DRCD is to minimize the system reliability cost. Each real-time task is allocated to the processor with the minimum reliability cost. DRCD is described below.

### The DRCD algorithm:

1. Get a job  $J$  from the head of the schedule queue  $SQ$ ;
2. According theorem 1, check the deadlines of all real-time tasks in  $J$ ;
3. **If** one task's deadline could not be guaranteed **then**
4. Goto 1. to schedule the next job;
5. Sort tasks in  $J$  by their deadlines in increasing order;

6. **for** each task  $v_i$  in job  $J$  **do**
7.  $st \leftarrow \infty$ ;  $find \leftarrow no$ ;  $rc \leftarrow \infty$ ;
8. **for** each processor  $p_j$  in  $P$  **do**
9.  $est \leftarrow est(v_i)$ ;  $rc_j \leftarrow \lambda_j c^j(v_i)$ ;
10. **if**  $est + c^j(v_i) \leq dt(v_i)$  **then**
11.  $find \leftarrow yes$ ;
12. **if** ( $rc_j < rc$ ) or ( $rc_j = rc$  and  $est < st$ )
13. **then**  $st \leftarrow est$ ;  $rc \leftarrow rc_j$ ;  $pr(v_i) \leftarrow j$ ;
14. **end if**
15. **end for**
16. **if** ( $find = yes$ ) **then**
17.  $st(v_i) \leftarrow est$ ;  $ft(v_i) \leftarrow est + c^j(v_i)$ ;
18. Move  $v_i$  into the dispatch queue  $DQ$ ;
19. **else** Reject  $v_i$  and deleted the scheduled tasks in  $J$  from the dispatch queue  $DQ$ ;
20. Update information of each message;
21. **end for**
22. Goto 1. to schedule the next job;

The time complexity of algorithm DRCD is given in Theorem 2 as follows.

**Theorem 2.** Let  $n$  be the number of tasks,  $m$  be the number of processors in the heterogeneous system, and  $u$  be the number of messages in the 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  $est$ , thus, the time complexity for calculating  $est$  is  $O(n \times u)$ . Since there are  $m$  processors in the heterogeneous system and  $n$  real-time tasks in the job, the iteration of the *for* loop takes  $O(m \times n)O(n \times u)$ , that is, time complexity of this algorithm is  $O(m \times n^2 \times u)$ .

## 5. Performance Evaluation

In this section we present results from our extensive simulations which assess the performance of the proposed scheme. In our simulation experiments, three kinds of DAGs for real-time tasks are assumed, which are representative of many real-life algorithms. One type of task graphs is binary tree, a second type is lattice and a third type is a random graph. Workload parameters are chosen in such a way that they are either based on those used in the literature or represent reasonably realistic workload and provide some stress tests for our algorithms.

For each point in the performance curves, the number of jobs arriving in the heterogeneous system is 20,000. The parameters used in the simulation studies are given in Table 1. The heterogeneous system for the simulation experiments is described as follows:

- (1) The number of processors in the heterogeneous system reflects system size. It is fixed to be 8.
- (2) Failure rate for each processor is uniformly selected between the range 0.95 to  $1.05 \times 10^{-6}$ /hour ( $10^{-4}$ ) [17][23].

**Table 1 Parameters for simulation experiments**

Parameter	Explanation	Value (Fixed) (Varied)
FR	Failure rate of processors	$-(0.95, 0.96, \dots, 1.05) \times 10^{-6}$
MIN_E	Minimum execution time	(5) - (15, 25, 35)
MAX_E	Maximum execution time	(200) (100, 120, 140, 160, 180, 200) (170, 180, 190)
$\delta$	Range for generating the deadline	([1, 10]) ([1, 100], ..., [1, 300]), ..., [1, 1100])
MIN_W	Minimum communication weight	(0.5) -
MAX_W	Maximum communication weight	(1.5) -
MIN_V	Minimum communication volume	(1) -
MAX_V	Maximum communication volume	(10) -
$m$	Number of processor in the heterogeneous system	(8) (10, 15, 20, ..., 40)
$n$	Number of tasks in a job	(30, 50, 70) for Btree (9, 25, 36, 49) for Lattice
Sched_t	Scheduling time	(0) (10, 30, ..., 110)
$\lambda$	Job arrival rate	$-(5, 10, 15, 20, 25) \times 10^{-4}$
MIN_D	Minimum dispatch time	(1) -
MAX_D	Maximum dispatch time	(10) (5, 10, 15, 20, 25)

The parallel real-time jobs and communications 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  $MAX\_E$ . The scale of this range approximates the level of computational heterogeneity.

(2) Given  $v_i \in V(J)$ , if  $v_i$  is on  $p_s$  and  $v_j$  is on  $p_r$ , then  $v_i$ 's deadline is chosen as follows:  $dt(v_i) = \max\{dt(v_j)\} + 1 + |e| \times w_{ik} + \max\{c^k(v_i)\} + \delta$ , where  $e = (v_p, v_r) \in E(J)$ ,  $k \in [1, m]$ , and  $\delta$  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 2, 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 the following table.

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

n	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

(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.

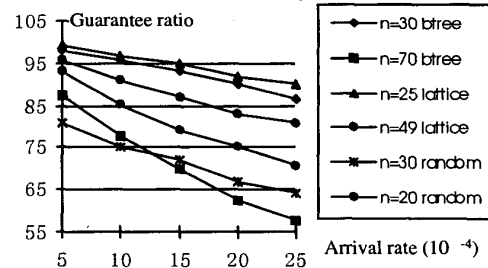
(6) Communication volume between two real-time tasks is uniformly selected between  $MIN\_V$  and  $MAX\_V$ . This range reflects the variance in message size.

The performance measures in our simulation study are reliability cost (RC) and guarantee ratio (GR) (i.e., percentage of jobs guaranteed to meet their deadlines). While the former gives a measure of system reliability as a result of a particular schedule, the latter indicates how

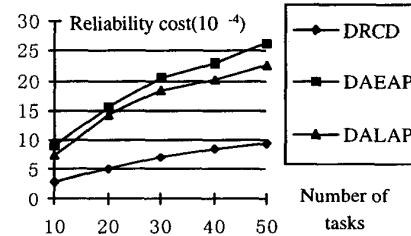
many of the arriving jobs can be scheduled, thus measuring the effectiveness and power of a scheduling algorithm.

### 5.1 Effect of Workload and Job Size

Fig.2 displays guarantee ratios of binary tree, lattices, and random graphs, respectively, as a function of job arrival rate  $\lambda$ . We ran our DRCD algorithm on each task graph.  $n$  is the number of tasks in a job, which is fixed for each curve. From fig.2, it is clear that, for all values of  $n$ , GR decreases as  $\lambda$  increases, as expected. GR also decreases as  $n$  increases. This is because increased job size results in increased scheduling time and dispatching time, giving rise to lowered guarantee ratio.



**Fig. 2 Effect of job load and task load, task graph is binary tree**

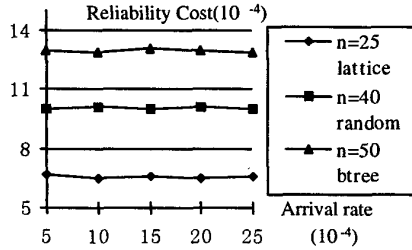


**Fig.3 Effect of task load on RC arrival rate =  $15 \times 10^{-4}$  no./sec., binary tree**

### 5.2 Reliability Cost

Fig.3 and Fig.4 illustrate reliability cost as a function of job size,  $n$ , and job arrival rate  $\lambda$ , respectively; for DRCD, DAEAP and DALAP. Since results for lattices and random graphs have similar patterns as those for binary tree, we only show results of binary tree in Fig. 3.

As can be observed from Fig. 3, DRCD significantly outperforms both DAEAP and DALAP in terms of reliability cost (RC), and the advantage of DRCD over DAEAP and DALAP becomes more pronounced as the job size increases. This is expected because DAEAP and DALAP do not consider RC as one of their scheduling objectives. DRCD, however, tends to assign tasks to processors on which their reliability cost are minimum.



**Fig. 4 Effect of job load on RC**  
Task graph is binary tree

Another interesting result from this experiment is that job arrival rate seems to have no impact on reliability cost performance. Since DRCD, DAEAP and DALAP share the same feature, we only ran DRCD algorithm on three task graphs. The results shown in Fig 4 indicate that the reliability cost performance depends on job size, but not on job arrival rate. This is because the scheduling algorithm employs an admission control strategy, which rejects jobs whose deadlines cannot be guaranteed.

**Table 3 Comparison of dynamic and static scheduling algorithms in terms of reliability cost (Task Graphs are Btrees)**

N	10	20	30	40	50
RCD	2.62	5.24	7.85	10.44	13.01
DRCD	2.61	5.21	7.92	10.49	12.88
AEAP	9.25	19.35	28.38	37.89	46.74
DAEAP	9.88	19.20	29.02	37.73	46.81
ALAP	8.12	14.82	23.47	28.55	35.19
DALAP	8.27	14.92	23.35	28.24	35.24

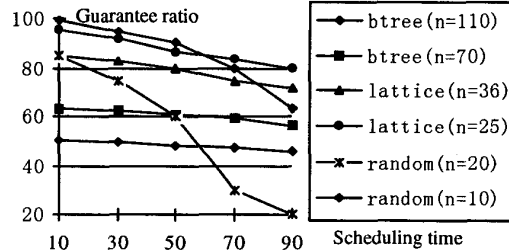
We also compare the RC performances between the static algorithms [17] and our dynamic algorithms. We only examine Btree here, since other two task graphs have similar behaviors. As can be noticed from Table 3 that RC of DRCD is nearly the same as that of static RCD. Similarly, the RC performances of DAEAP and static AEAP are the very close, so are those of DALAP and static ALAP.

### 5.3 Scheduling Time

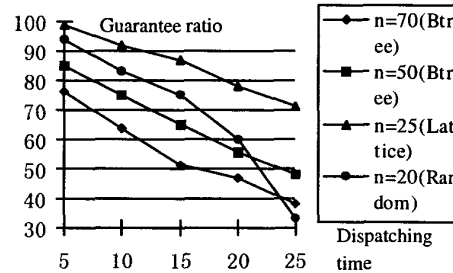
This experiment studies relationship between scheduling time and guarantee ratio. As scheduling time makes the same impact on three algorithms, we only illustrate the results for the DRCD algorithm in Fig 5. Both binary tree-, lattice- and random graph-based jobs are considered. For each curve in Fig. 5, the job size is fixed and the job arrival rate is set to be  $20 \times 10^4$  no./sec. Fig. 5 shows GR as a function of scheduling time.

It reveals the significant impact that scheduling time has on the system performance of a dynamically scheduled real-time heterogeneous system. Without considering scheduling time, 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 GR, thus allowing more jobs to be completed before their given deadlines. Therefore, it is highly desirable to parallelize scheduling algorithms so that their scheduling times can be significantly shortened to ultimately enhance the performance of the heterogeneous system.



**Fig. 5 Effect of scheduling time,**  
Arrival rate is  $20 \times 10^4$  no./sec.



**Fig. 6 Effect of dispatching time of DRCD algorithm on GR.** Arrival rate is  $20 \times 10^4$  no./sec.

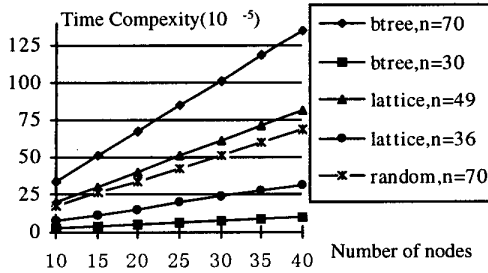
### 5.4 Dispatching Time

Fig. 6 shows the effect of dispatching time of the DRCD algorithm on guarantee ratio for different values of  $n$ . Again, job arrival rate is fixed at  $20 \times 10^4$  no./sec. Dispatching time is increased from 5 to 25 with increments of 5. Job size is set to 50 and 70 for binary tree, 25 for lattice, and 20 for random graph, respectively. Fig. 6 clearly shows that decreasing dispatching time can significantly improve GR of the heterogeneous system. This result strongly suggests that using a high-speed network to speed up the dispatching of scheduled tasks can substantially enhance the system performance.

### 5.5 Heterogeneous System Size

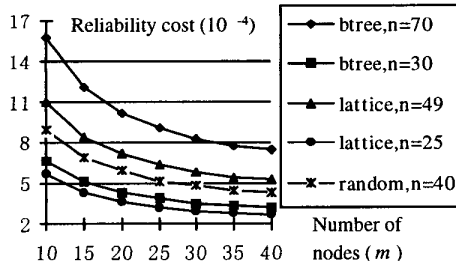
To study the impact of the heterogeneous system size  $m$  (number of processors) on the performance of DRCD algorithm, we fixed job arrival rate at  $10 \times 10^4$  no./sec., and increased  $m$  from 10 to 50 with increments of 5. Fig. 7 shows scheduling time as a function of the heterogeneous system size, indicating a noticeable impact of both the heterogeneous system 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 pronounced as the job size increases. This is because scheduling time is the product of  $m$ ,  $n$  and  $u$  (Theorem 2).



**Fig. 7 Impact of  $m$  on time complexity of DRCD algorithm, Arrival rate is  $10 \cdot 10^{-4}$  no./sec.**

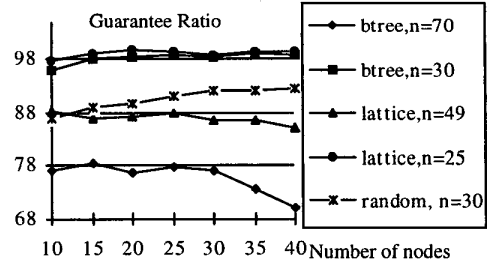
Fig. 8 illustrates the impact of the heterogeneous system size on the reliability cost. It shows that under the same workload, the RC performance improves as the heterogeneous system size increases. The main reason behind this is that for a large heterogeneous system, the DRCD algorithm has more choices for scheduling a real-time task. It is also observed from Fig. 8 that when the system size is more than 30 the improvement in RC performance starts to diminish. This is because a higher value of  $m$  can result in a longer scheduling time (see Fig. 7), 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 processors. An optimal value of  $m$  for a particular workload may be determined by experiments.



**Fig. 8 Effect of the  $m$  on RC of DRCD algorithm, arrival rate is  $10 \cdot 10^{-4}$  no./sec.**

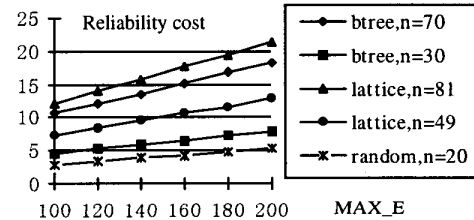
The impact of system size on guarantee ratio is shown in Fig. 9, where GR is plotted as a function of  $m$ . The results indicate that the impact of the system size on the GR performance is mixed. On the negative side, a higher value of  $m$  can lead to a longer scheduling time, as illustrated in Fig. 7. On the positive side, increasing the number of processors 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. 9 Effect of the  $m$  on GR of DRCD algorithm, arrival rate is  $10 \cdot 10^{-4}$  no./sec.**

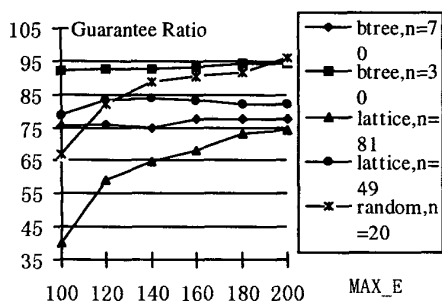
As shown in Fig. 9, three curves illustrate the positive side, and two other curves depict the negative side. We observe that when  $n$  (job size) is comparatively low, the net effect is positive (see the three curves in Fig. 9 with  $n = 25$ ,  $n = 30$ ); whereas, the negative effect emerges as  $n$  becomes relatively high (see two curves in Fig. 9, with  $n = 49$  and  $n = 70$ ). This suggests that the number of processors is a critical parameter for scheduling parallel real-time jobs, which must be determined carefully based on experiments.



**Fig. 10 Effect of the execution time on RC of DRCD algorithm. Arrival rate is  $10 \cdot 10^{-4}$  no./sec.**

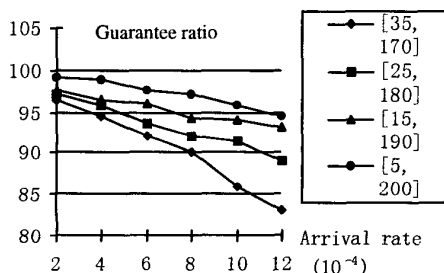
## 5.6 Execution time

Fig. 10 shows the effect of execution time on reliability cost. We only consider the DRCD algorithm, since DAEAP and DALAP have similar properties and are less relevant. In this experiment, the job arrival rate is set at  $10 \cdot 10^{-4}$  no./sec., and  $MAX\_E$  is varied from 100 to 200 with increments of 20. For each value of  $MAX\_E$ , we ran the DRCD algorithm on binary trees, lattices and random graphs. From the simulation results shown in Fig. 10, we observe that the reliability cost increases with the increase in the execution time. This is due to the simple fact that, when execution time in each  $c'(v_i)$  increases the task reliability cost  $\lambda_i c'(v_i)$  also increases. We can conclude from this experiment that, as the execution time increases, the reliability cost of the system also increases.



**Fig.11 Effect of the execution time on GR of DRCD algorithm. Arrival rate is  $10 \cdot 10^{-4}$  no./sec.**

As shown in Fig. 11, execution time also has noticeable impact on guarantee ratio. When the value of  $n$  is low (see the curve with  $n = 30$ ), execution time does not make a significant impact on GR, but when  $n$  is sufficiently large (see the curve with  $n = 81$ ), we observe that GR is affected noticeably by the execution time. Since deadline is assumed to be a function of execution time in our simulation model, when execution time increase, so does the deadline of the system. More real-time tasks can be guaranteed if their deadlines are relaxed.



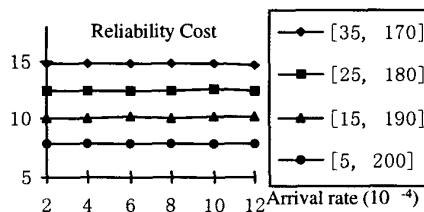
**Fig. 12 Effect of computational heterogeneity on GR, Task graph is btree**

### 5.7 Computational Heterogeneity

Fig. 12 shows GR as a function of job arrival rate, with different variances in task execution time, where job arrival rate increases from  $2 \cdot 10^{-4}$  no./sec. to  $12 \cdot 10^{-4}$  no./sec. with increments of  $2 \cdot 10^{-4}$  no./sec. Again, we only consider the DRCD algorithm and binary tree 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. Fig. 12 and Fig. 13 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 increase guarantee ratio and reduce reliability cost, thus, enhancing the schedulability. This may 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 processors and reduced heterogeneity implies reduced variance in tasks' reliability costs. It is proved by this experiment which shows that DRCD is efficient for heterogeneous jobs, and its performance varies with the heterogeneity of the parallel real-time jobs.



**Fig. 13 Effect of computational heterogeneity on RC, Task graph is btree**

## 6. Conclusion

Most research work in the area of real-time task scheduling in heterogeneous systems either ignored the reliability issues, or only considered homogeneous systems, or assumed independent tasks, or only schedule tasks with precedence constraints offline. In this paper we attempted to address these issues by proposing a dynamic reliability-cost-driven algorithm (DRCD) that schedules real-time tasks with precedence constraints in a heterogeneous system. Furthermore, both scheduling time and dispatching time are incorporated in our scheduling algorithms to make the scheduling more practical and realistic. In the DRCD algorithm, reliability cost is used as one of the objective functions for scheduling tasks in a parallel job. For comparison purposes, this paper also presents two other algorithms, DAEAP and DALAP, that are greedy in nature but do not consider reliability cost. Simulation results show that DRCD is significantly better than DAEAP and DALAP, in terms of reliability cost, a measure of reliability. In addition, our experiments suggested that higher computational heterogeneity is conducive to improving DRCD's schedulability, while computational heterogeneity had no apparent effect on reliability cost. Simulation results also reveal that both scheduling time and dispatching time can significantly impact the effectiveness of a scheduling algorithm. Thus, it is highly desirable to substantially reduce these times by parallelizing the scheduler and by providing high-speed network capability for task dispatching.

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

## References

- [1] T.F. Abdelzaher and K.G. Shin., "Combined Task and Message Scheduling in Distributed Real-Time Systems," *IEEE Transaction on Parallel and Distributed Systems*, Vol. 10, No. 11, Nov. 1999.
- [2] I.D. Baev, W.M. Meleis, A. Eichenberger, "Lower bounds on precedence-constrained scheduling for parallel processors," *In Proc. of the 29th International Conference on Parallel Processing*, 2000, pp. 549-553.
- [3] Z. Chen, K. Maly, P. Mehrotra, V. K. Praveen and M. Zubair, "An Architecture to Support Collaborative Distributed Heterogeneous Computing Applications," *Technical Report, TR97-26*, Department of Computer Science, University of Bristol, UK, 1997.
- [4] M. Cosnard, E. Jeannot and T. Yang, "SLC: Symbolic Scheduling for Executing Parameterized Task Graphs on Multiprocessors," *In Proc. of the 28th International Conference on Parallel Processing*, Aizu-Wakamatsu, Fukushima, Japan, 1999.
- [5] A. Dogan, F. Ozguner, "Reliable matching and scheduling of precedence-constrained tasks in heterogeneous distributed computing," *In Proc. of the 29th International Conference on Parallel Processing*, 2000, pp. 307-314.
- [6] N. Edgar, "Real-Time Behaviour in a Heterogeneous Environment?," *In Proc. Of the 3rd International Workshop on Object-oriented Real-time Dependable Systems*, Newport Beach, California, February 6-7, 1997.
- [7] D.G. Feitelson and L. Rudolph, "Job Scheduling for Parallel Supercomputers," *In Encyclopaedia of Computer Science and Technology*, Vol38, Marcel Dekker, Inc., New York, 1998
- [8] E.N. Huh, L.R. Welch, B.A. Shirazi and C.D. Cavanaugh, "Heterogeneous Resource Management for Dynamic Real-Time Systems," *In Proc. of the 9th Heterogeneous Computing Workshop*, 2000, 287-296.
- [9] M. Iverson and F. Ozguner, "Dynamic, Competitive Scheduling of Multiple DAGs in a Distributed Heterogeneous Environment," *In Proc. of the Seventh Heterogeneous Computing Workshop*, Orlando, Florida, USA, 1998, pp.70-78.
- [10] V. Kalogeraki, P.M. Melliar-Smith, L.E. Moser, "Dynamic scheduling for soft real-time distributed object systems," *In Proc. of Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 2000, pp.114-121.
- [11] D. Kebbal, E.G Talbi, and J.M Geib, "Building and scheduling parallel adaptive applications in heterogeneous environments," *In Proc. of the 1st IEEE Computer Society International Workshop on Cluster Computing*, Melbourne, Australia, 1999, pp.195-201.
- [12] Y.K. Kwok, I. Ahmad, J. Gu, "FAST: A Low-Complexity Algorithm for Efficient Scheduling of DAGs on Parallel Processors," *In Proc. of the 25th International Conference on Parallel Processing*, 1996, pp. 150-157.
- [13] T. Lundqvist and P. Stenstrom, "Timing anomalies in dynamically scheduled microprocessors," *In Proc. of the 20th IEEE Real-Time Systems Symposium*, 1999, pp.12-21.
- [14] M. Maheswaran and H.J. Siegel, "A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems," *In Proc. of the Seventh Heterogeneous Computing Workshop*, Orlando, Florida, USA, 1998, pp.57-69.
- [15] G. Manimaran and C.S.R Murthy, "An Efficient Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 3, 1998, pp. 312-319.
- [16] J.C. Palencia, and H.M. Gonzalez, "Schedulability analysis for tasks with static and dynamic offsets," *In Proc. of the 19th IEEE Real-Time Systems Symposium*, 1998, pp.26-37.
- [17] Xiao Qin, Hong Jiang, C.S. Xie, and Z.F. Han, "Reliability-driven scheduling for real-time tasks with precedence constraints in heterogeneous distributed systems," *In Proc. of the 12th International Conference Parallel and Distributed Computing and Systems* 2000.
- [18] Xiao Qin, Z.F. Han, H. Jin, L.P. Pang and S.L. Li., "Real-time Fault-tolerant Scheduling in Heterogeneous Distributed Systems," *in Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, June 26-29, 2000. Vol. I, pp.421-427.
- [19] A. Radulescu, A.J.C van Gemund, "Fast and effective task scheduling in heterogeneous systems," *In Proc. of the 12th Euromicro Conference on Real-Time Systems*, 2000, pp.229-238.
- [20] S. Ranaweera, D.P. Agrawal, "A scalable task duplication based scheduling algorithm for heterogeneous systems," *In Proc. of the 29th International Conference on Parallel Processing*, 2000, pp. 383-390.
- [21] R.M. Santos, J. Santos, and J. Orozco, "Scheduling heterogeneous multimedia servers: different QoS for hard, soft and non real-time clients," *In Proc. of the 12th Euromicro Conference on Real-Time Systems*, 2000, pp.247-253.
- [22] G. C. Sih and E. A. Lee, "A Compile-Time Scheduling heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Trans. Parallel and Distributed Systems*, 4(2), 1993, pp. 175-187.
- [23] S. Srinivasan, and N.K. Jha, "Safty and Reliability Driven Task Allocation in Distributed Systems," *IEEE Trans. Parallel and Distributed Systems*, 10(3), 1999, pp. 238-251.
- [24] X.Y. Tang, S.T. Chanson, "Optimizing static job scheduling in a network of heterogeneous computers," *In Proc. of the 29th International Conference on Parallel Processing*, 2000, pp. 373-382.
- [25] M.E. Thomadakis and Jyh-Charn Liu, "On the efficient scheduling of non-periodic tasks in hard real-time systems," *In Proc. of the 20th IEEE Real-Time Systems Symposium*, 1999, pp.148-151.
- [26] H. Topcuoglu, S. Hariri and M.Y.Wu, "Task Scheduling Algorithms for Heterogeneous Processors," *In Proc. of the 8th Heterogeneous Computing Workshop*, 1999, pp3-14.
- [27] M.Y. Wu, W. Shu, Y. Chen, "Runtime parallel incremental scheduling of DAGs," *In Proc. of the 29th International Conference on Parallel Processing*, 2000, pp. 541-548.