RELIABILITY-DRIVEN SCHEDULING FOR REAL-TIME TASKS WITH PRECEDENCE CONSTRAINTS IN HETEROGENEOUS SYSTEMS*

XIAO QIN† HONG JIANG† CHANGSHENG XIE‡ and ZONGFEN HAN‡

†*Department of Computer Science and Engineering*
University of Nebraska-Lincoln
Lincoln, NE 68588-0115, (jiang@cse.unl.edu)
‡*School of Computing and Information Technology*
Griffith University, Nathan, Queensland 4111, Australia
‡ *Department of Computer Science and Engineering*
Huazhong University of Science and Technology, Wuhan, P.R.China 430074

ABSTRACT: Some work has been done in the past in scheduling tasks in real-time distributed systems, considering schedulability as the main objective function to be maximized. Since real-time distributed systems are more complex than centralized systems, the complexity of such system could increase the potential for system failures. This is even more pronounced in a heterogeneous system where processors operate at different speeds and communication channels have different bandwidths. Hence, reliability should also be regarded as the objective function to be maximized. In this paper, we describe a two-phase scheme to determine a scheduling of tasks with precedence constraints that employs a reliability measure as one of the objectives in a real-time and heterogeneous distributed system. We devise a new off-line scheduling of communicating tasks, based on the concept of reliability cost, to schedule real-time tasks for maximized reliability. The simulation results show that, for task graphs with precedence constraints in a heterogeneous distributed system, our heuristic performs significantly better than the two heuristics presented that do not consider reliabiliy cost. Furthermore, the results suggested that higher computational heterogeneity is conducive to improving the schedulability of the reliability cost-driven (RCD) algorithm, while the opposite is true for the two non-RCD algorithms.

KEY WORDS: Reliability cost, Real-time, Scheduling, Heterogeneous distributed systems, Performance

1 INTRODUCTION

Distributed systems are increasingly being applied for critical real-time applications, in which each task must be guaranteed a priori to meet its timing constraint. Therefore, efficient scheduling algorithms and schedulability analysis are needed. Many real-time scheduling algorithms, in which schedulability is a main objective function to be maximized, can be found in the literature [1][2][4][5][7]. These real-time scheduling algorithms fall into two categories: static and dynamic scheduling. Most of these real-time scheduling algorithms have an assumption that no error occurs in real-time systems. In order to make real-time scheduling algorithm more practical, task precedence constraints need to be taken into account [9][10]. Paper [11] presents a flexible scheme, which combines off-line analysis with on-line guarantees. This scheme uses off-line analysis to convert task precedence and communication constraints into pseudo deadlines of tasks and messages, then employs an on-line guarantee routine to find a runtime task and message schedule that minimizes the number of tasks missing deadlines. An optimal algorithm for scheduling tasks in a distributed real-time system, which improves the quality of the solution, was presented in [12]. Since all the above algorithms are devised for homogeneous distributed systems, which assume that processors in the system are identical, they will not be directly applicable for heterogeneous distributed system.

In general, distributed heterogeneous computing involves multiple heterogeneous modules that interact with one another to solve a problem [3][8]. In a heterogeneous computing 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 [16]. Extensive research has been done on scheduling algorithms for heterogeneous systems. A generic technique for mapping heterogeneous task graphs onto heterogeneous system graphs is presented in [13]. Paper [14] discussed in depth the

* This work was partially supported by a Nebraska University Foundation grant and NSF of China under the Grant No.: 69873017.

Scheduling Expert Advisor in a heterogeneous environment, which processes a high level description of a computational task provided by a user and converts it into a set of facts and rules. Unfortunately, these scheduling algorithms for heterogeneous systems are non-real-time, which are not suitable for scheduling the tasks with timing constraints. In [15] we have investigated a real-time and fault-tolerant scheduling algorithm for heterogeneous systems, however, it is assumed that the real-time tasks are independent with one another.

For the algorithm to be of more practical value, task precedence constraints need to be taken into account for heterogeneous systems. Therefore, our main objective in this paper is to investigate real-time scheduling algorithms for tasks with precedence constraints in a heterogeneous distributed system. More specifically, we consider the off-line scheduling of tasks with precedence constraints, represented by directed acyclic graphs (DAG), on a distributed heterogeneous system where processors operate at different speeds and inter-processor links have different bandwidths. Further, a reliability measure, called reliability cost, is used as one of the objective functions for task scheduling. Simulation results show that the reliability-driven scheduling algorithm significantly outperforms two previously proposed algorithms that are not reliability driven. The paper is organized as follows. In Section 2, the scheduling model and assumptions are presented. Algorithms of reliability cost driven and real-time scheduling are proposed in Section 3. The performance analysis is presented in Section 4. In Section 5, we summarize the contributions of this paper and suggest future directions of this work.

2 SYSTEM MODEL

A real-time job, input into the distributed system, is composed of inter-communicating real-time tasks described as a real-time directed acyclic graph (DAG). Each task is an atomic component of the real-time job. In other words, real-time tasks are not divisible. Real-time scheduling will find a proper processor for each real-time task and decide the start time for each task. In our work, we explore a two-phase method to schedule real-time tasks and analysis algorithms generated by this method.

A real-time DAG is defined as RG = {T, E}, where T = $\{t_1, t_2,...,t_n\}$ represents real-time task set, and E denotes the set of weighted and directed edges among the tasks. Each element m in E is a message from one task to another. m = (ts, tr, s, f, c), where ts, tr are two tasks that sends/receives m, respectively; s and f are start time and finish time; and c represents the volume of data being sent. Each task t is modeled as a tuple. t = (C, d, s, f, g, ρ) where C, d, s and f represent the execution time vector, deadline, start time and finish time, respectively. g is the in-degree, namely, the number of t's parent tasks in RG, and ρ indicates the processor to which t is allocated. The vector C_i =[c(i, 1),…,c(i, m)], a measure of *computational heterogeneity*, is adopted from [8], where c(i, j) denotes the execution time of t_i on processor j. The heterogeneous distributed system is modeled as a set of processors, Ω = $\{P_1, P_2,..., P_m\}$, $P_i = (\Delta_i, M_i, \lambda_i)$, where task set Δ_i contains tasks allocated to P_i, and λ_i denotes failure rate. M_i is a set for message lists which is defined as, $M_i = \{M_{i1}, \ldots, M_{ij}, \ldots, M_{im}\}$, where j≠i and all messages in M_{ij} are transmitted from P_i to P_j. Messages in M_{ij} are sorted in the increasing order of their start times.

We use the same model of reliability as defined in [6][8]. The processors are modeled as nodes and there is a weighted edge between two nodes if the corresponding processors can communicate with each other. The weight w_{ij} on the edge between processors p_i and p_j represents the delay involved in sending or receiving a message of unit length from one processor to another. Thus, w_{ij} is a measure of *communicational heterogeneity*. In order to have an approximate estimate of this delay, irrespective of the two processors, we use the average of the weights on all the edges in the processor graph. This is called the average unit delay. In this model, processor failures are assumed to be independent, and follow a Poisson Process with the constant failure rate [6][8]. Failures of communication links are not considered here. The reliability cost of a task t_i on a processor P_j is the product of P_j's failure rate and t_i's execution time on P_j. Thus, the reliability cost of a given task schedule is the summation over all tasks' reliability costs based on the given schedule. That is, given the heterogeneous system Ω, the reliability cost is defined below, which is similar as that in paper [6]. The reliability is given by $e^{-RC(\Omega)}$. It is clear that scheduling taks with larger execution time to more reliable processor is a good approach to increase the reliability of the system.

$$RC\ (\Omega) = \sum_{j=1}^{m} \sum_{t_i \in \Delta_j} \lambda_j c(i, j)$$

3. REAL-TIME SCHEDULING

The real-time scheduling approach consists of two steps. The first step is to determine a scheduling order and the second step is to schedule tasks to the distributed system. The input parameter of the algorithms for determining a scheduling order is RG, and its output is an ordered list of tasks OL = $\langle t_{i1}, t_{i2}, \ldots, t_{in} \rangle$. The input parameters of the second step are OL and Ω. The function of the second step is to allocate each task to a processor and decides the start time. If the algorithm could guarantee the completion of all the real-time tasks before deadlines, the result will be generated; otherwise a warning message will be given. Failure of scheduling result from two reasons. The first one is the limited number of processors, which can be solved by adding more processors. Secondly, if RG is not well designed, even increasing the number of processors can not make the set of real-time tasks

schedulable. In our work, we assume that all RG are schedulable, thus case two will not happen. How to design a good and schedulable real-time direct acyclic graph is beyond the scope of this paper, it will not be discussed in this paper.

Dividing the scheduling procedure into two parts enables one to investigate sorting tasks and scheduling tasks separately. We design a mechanism that allows the addition and deletion of algorithms of sorting and scheduling flexibly. The mechanism is formally modeled as MSSP = {O, OL, S}, where O and S are sets of sorting algorithms and scheduling algorithms, respectively, and OL is an ordered list of tasks. Generally speaking, a scheduling algorithm in such mechanism is a combination of O_i and S_j, where $O_i \in O$ and $S_j \in S$.

3.1 SORTING ALGORITHMS

A sorting algorithm generates an accurate scheduling sequence, which has the following property:

Property: $\forall t_j, t_k \in T\ [(t_j, t_k) \in E \rightarrow (\exists t_{ia}, t_{ib} \in OL) \wedge (t_{ia} = t_j, t_{ib} = t_k) \wedge (a < b)]$.

In other words, if task t_j transmits data to task t_k, then t_j will be scheduled before task t_k. The algorithms sort tasks according to their number of parent tasks that remain unsorted. If one parent task of t is put into OL, then t.g will decrease by 1. Only when t.g drops to zero, could the sorting algorithm insert task t into OL. This approach guarantees each task to be scheduled after all its parent tasks have been scheduled. SORT_LIFO and SORT_FIFO algorithms are quite similar. SORT_LIFO algorithm uses a stack to store all the tasks with zero t.g, while SORT_FIFO stores tasks that have zero t.g into a queue. Since they are relatively straightforward, details of these two algorithms are omitted in this paper.

SORT_EDF temperately stores tasks with zero t.g into a list ZDL and, at the time of selecting a task in ZDL, the algorithm chooses the task that has the earliest deadline, in such way, the task with the earliest deadline has the highest priority. The algorithm is described as follows,

SORT_EDF ALGORITHM (Input: RG, Output: OL)
```
OL ← ∅;
ZDL ← Create_A_LIST();
FOR each t_i.g ∈ T DO /* Initialize the list ZDL */
    IF t_i.g = 0 THEN ENLIST(ZDL, t_i);
WHILE list ZDL is not empty DO
    Select t from ZDL, where ∀t_i ∈ ZDL(t.d ≤ t_i.d);
    OL ← Append_OL(t); /*Append task t to OL */
    v ← First_Son(RG, t); /*Get first son of t */
    WHILE v ≠ NULL DO
        v.g ← v.g -1; /*In-degree of v decrease 1*/
        IF v.g = 0 THEN Select t from ZDL,
            where ∀t_i ∈ ZDL(t.d ≤ t_i.d);
        v ← Next_Son(RG, t, v);
    END WHILE
END WHILE
END
```

3.2 SCHEDULING ALGORITHMS

3.2.1 SCHEDULING AS EARLY AS POSSIBLE

After the sorting phase, the task graph has been translated into a sequence. The next stage is to allocate real-time tasks to processors in the system, and decide the start time for each task. In order to determine the start time for the task, we need to compute the earliest start time (EST) of a task on each processor. EST can be determined by the earliest available time (EAT), i.e., the earliest possible start time ignoring any resource constraints, that is derived from the finish times of the task's parent tasks and the message arrival times. In our model, if task t is allocated to the same processor as one of its parent tasks t_p, the communication time between t_p and t is assumed to be zero. If t and t_p are allocated to different processors, the message arrival time will be determined by the finish time of parent task and data volume being transmitted. The following procedure decides the message start time (MST) of m being sent from processor P_i to P_j.

get_MST (Input: $m \in E$, $M_{ij} \in M_i$; Output: mst)
```
t ← m.ts; /* m.ts is the task on P_i that sends
            data/message to m.tr on processor P_j */
c ← m.c × w_ij; /* c is the communication time for
            transmitting the data/message m */
FOR each message m_k in M_ij DO
    mst ← MAX(m_k.f, t.f);
    IF (m_{k+1}.s - mst ≥ c) THEN RETURN(mst);
END FOR
END
```

EAT of a task t_i on processor P_j can be derived from procedure get_MST. Let $MS(t_i)$ be a set of messages being transmitted to task t_i, where $MS(t_i) = \{m \in E \mid m.tr = t_i\}$. The procedure outlined below computes the earliest available time of t_i on P_j. The procedure determines the start time of each message in set $MS(t_i)$, which leads easily to the finish time of the message. EAT of t_i is equal to the latest finish time of messages in $MS(t_i)$

get_EAT(Input: $t_i \in T$, $P_j \in \Omega$; Output: eat)
```
eat ← 0; /* Initialize the earliest available time */
FOR each message m in MS(t_i) DO
    t_k ← m.ts; /* Let t_k be t_i's parent */
    P_s ← t_k.ρ; /* t_k is allocated to P_s */
    IF (P_s ≠ P_j) THEN /* t_i and its parent task t_k are
            allocated to different processors */
        mst ← get_MST(m, M_sj)
        temp_eat ← mst + m.c× w_sj; /* The finish
        time of the message is the start time of t_i */
    ELSE temp_eat ← t_k.f; /* Otherwise, start time
        is just the finish time of its parent task */
    IF eat < temp_eat; THEN eat ← temp_eat;
END FOR
RETURN(eat);
END
```

If the earliest idle time slot of a processor P_j is large enough accommodate t_i and starts later than t's EAT, then the start time of this time slot is the earliest start time (EST) for t. The procedure outlined below computes the EST for a task t_i on a processor P_j.

get_EST (Input: $t_i \in T$, $P_j \in \Omega$,; Output: est)
    FOR each task t_k in $P_j.\Delta$ DO
        est ← MAX(t_k.f, get_EAT(t_i, P_j));
        IF (t_{k+1}.s - est ≥ t_i.c(i,j)) THEN RETURN(est);
    END FOR
    RETURN(est)
END

Since all tasks in set Δ are sorted in the increasing order of the start time, the above procedure checks the idle time slot on processor P_j from the earliest one to the latest one. The procedure terminates checking and returns the EST when a proper idle time slot is found. We propose our first scheduling algorithm based on the three procedures described above. The algorithm is named AEAP, short for schedule as early as possible. For a given task t_i, the algorithm computes EST on each processor, then the processor, on which t_i has the earliest EST, is chosen. The algorithm determines whether the start time can satisfy the specific deadline or not. If the deadline is not guaranteed, a warning message will be returned and the algorithm will halt. AEAP is given below.

AEAP ALGORITHM (Input: OL, Ω; Output: T)
    FOR (each task t_i in OL) DO
        est ← ∞; /* Initialize the earliest start time */
        FOR each processor P_j in Ω DO
            IF (get_EST(t_i, P_j) ≤ est)
            THEN(est ← get_EST(t_i, P_j));
        END FOR
        IF (est+ t_i.c(i,j) > t_i.d) THEN RETURN(FAIL);
        t_i.s ← est; t_i.f ← est + t_i. c(i,j); t_i.ρ ← P_j;
        Insert task t_i into set $P_j.\Delta$;
        Update information of each message;
    END FOR
    RETURN(SUCCESS);
END

3.2.2 SCHEDULING AS LATE AS POSSIBLE

The second algorithm outlined below is called schedule as late as possible (ALAP). Under the constraint that deadlines of all tasks are guaranteed, the real-time tasks start as late as possible. Before giving ALAP, a procedure to compute the latest start time (LST) of t_i on P_j should be described. This procedure checks each idle time slot on processor P_j, by computing the earliest start time and the latest possible finish time (LPFT). An idle time slot is defined to be a proper idle time slot, if time between EST and LPFT is large enough to accommodate t_i. This procedure chooses the latest proper idle time slot and decides the latest start time, the procedure is presented below,

get_LST (Input: $t_i \in T$, $P_j \in \Omega$; Output: lst, find)
    lst ← 0; find ← NO;
    eat ← get_EAT(t_i, P_j)
    FOR each task t_k in $P_j.\Delta$ DO
        est ← MAX(t_k.f, eat); /* Earliest start time */
        lpft ← MIN(t_{k+1}.s, t_i.d);
        IF (lpft - est ≥ t_i. c(i,j)) THEN /* If idle time is large enough to accommodate task t_i */
            find ← YES; st ← lpft - t_i. c(i,j);
        /* Compute the start time in this idle time slot */
            IF (st > lst) THEN lst ← st
        END IF
    END FOR
    RETURN((lst, find));
END

ALAP first decides LST of each task t_i on each processor in the system, then selects the processor on which t_i has the latest LST. If such proper processor can not be found, ALAP will inform that the task set is unschedulable. It is described as follows,

ALAP ALGORITHM (Input: OL, Ω; Output: T)
    FOR (each task t_i in OL) DO
        lst ← 0; /* Initialize the latest start time */
        schedule ← NO; processor_id ← 0;
        FOR each processor P_j in Ω DO
            (st, find) ← get_LST(t_i, P_j,);
            IF (find = YES) THEN
                schedule ← YES;
                IF (st > lst) THEN (lst ← st);
                    processor_id ← j;
                END IF;
            END IF
        END FOR
        IF (schedule = NO) THEN RETURN(FAIL);
        t_i.s ← lst; t_i.f ← lst + t_i. c(i,j); t_i.ρ ← P_j;
        Insert task t_i into set $P_j.\Delta$;
        Update information of each message;
    END FOR
    RETURN(SUCCESS);
END

3.2.3 RELIABILITY COST DRIVEN SCHEDULING

AEAP and ALAP are two algorithms that do not take reliability cost into account. The third algorithm outlined below is named Reliability Cost Driven scheduling (RCD). Compared with AEAP and ALAP, algorithm RCD enhances reliability of the system with no extra hardware cost. The objective of algorithm RCD is twofold: minimize the schedule length and maximize the reliability. Each real-time task is allocated on the processor with the minimum reliability cost, and scheduled as early as possible. RCD is described below.

RCD ALGORITHM (Input: OL, Ω; Output: T)
    RC ← 0;
    FOR each task t_i on OL DO

```
        st ← ∞; find ← NO; rc ← ∞;
        FOR each processor Pj in Ω DO
            est ← get_EST(Ti, Pj); rcj ← λj × c(i, j);
            IF (est + c(i, j) <= ti.d) THEN
                Find ← YES;
                IF ((rcj < rc) OR
                    (rcj = rc and est < st)) THEN
                    st ← est;  p ← pj; rc ← rcj;
                ENDIF
            ENDIF
        ENDFOR
        IF find = NO THEN RETURN(FAIL);
        ti.s ← est;  ti.f ← est + ti. c(i,j); ti.ρ ← Pj;
        Insert task ti into set Pj.Δ;
        Update information of each message;
    ENDFOR
    RETURN(SUCCESS);
END
```

4. PERFORMANCE EVALUATION

4.1 Workload

This section presents simulation results for the purpose of performance evaluation. The simulation program randomly generates two kinds of DAGs for real-time tasks, which are representative of many real-life algorithms. These task graphs are binary tree, and lattices. We define the following workload parameters:

Execution time of a task—a value from a specific range, ET, is randomly chosen for each element of the execution time vector C. The scale of this range approximates the level of computational heterogeneity;

System size, N—number of processors in the system;

Communication weight (w_{ij})—a value from a specific range, CW, is randomly selected for each $w_{ij.}$ The scale of this range approximates the level of communicational heterogeneity;

Communication volume—a value from a specific range, CV, is randomly selected for each message m.c, m∈ E, to reflect the variance in message size;

Processor failure rate—randomly generated from the range 0.95 to 1.05 * 10^{-6}/hour (10^{-4}) for each processor, according to [8].

Task deadline—Given t_i, ∈ T, we suppose t_i on P_k and t_j on P_l, then t_i's deadline is modeled as follows: $t_i.d = MAX(t_j.d) + 1 + m.c \times w_{lk} + MAX[C(i, k)] + \delta$, where m = $(t_j, t_i) \in E$, k∈ [1, n], and δ is randomly computed according to uniform distribution.

The number of tasks in a DAG ranges from 10 to 90. Several performance aspects, such as reliability cost, minimum number of processors, ET vs. reliability cost, and deadline to ET ratio, are examined through simulations in the next few subsections.

4.2 Reliability Cost

Reliability cost results of the simulation running AEAP, ALAP and RCD are presented in this section. The workload parameters in the simulation are as follows,

N = unlimited; CW = [0.5,1.5]; CV = [1,10]; ET = [5,200]; δ = [1,40]

The reliability cost, RC(Ω), of a particular algorithm is computed based on its output (schedule), according to the formula presented on page 2. From tables 4.2.1 and 4.2.2, it is clear that ALAP outperforms AEAP in terms of reliability cost, and the advantage of ALAP over AEAP becomes more pronounced as the number of tasks in the DAG increases. On the other hand, the reliability cost driven RCD outperforms both ALAP and AEAP, which do not consider reliability cost in their scheduling objectives, by an order of magnitude. This is because, by the definition of RC, the RCD algorithm will tend to assign tasks to processors on which their execution times are minimum, since it is the execution time that dominates the RC product. This, coupled with the fact that ET varies between 5 and 200 and ALAP and AEAP ignore RC, results in the discrepancy between RCD and the other two non-RCD algorithms in the reliability cost measure.

N	10	30	50	70	90
AEAP	10.04	29.14	46.58	64.21	79.56
ALAP	8.95	23.17	35.58	45.75	54.58
RCD	0.83	2.50	4.18	5.82	7.44

Table 4.2.1 Btree (System RC)

N	9	25	49	64	81
AEAP	9.42	25.54	49.34	63.18	80.90
ALAP	8.53	22.35	40.80	50.53	63.10
RCD	0.75	2.01	4.08	5.31	6.72

Table 4.2.2 Lattice (System RC)

4.3 Minimum number of processors

After giving a set of real-time tasks and the number of processor k, scheduling algorithms are able to determine whether k processors can complete all tasks before the deadline. An algorithm, called Find Minimum Number of Processors (FMNP), is devised to find the minimum number of processors to (MNP) which all tasks can be scheduled to finish before the specified deadlines. The input parameter Sort_Alg represents the sorting algorithm used to generate OL, and Sch_Alg indicates the algorithm invoked to schedule the task set. The algorithm is described as follows:

```
FMNP (Input: RG, Ω, Sort_Alg, Sch_Alg; Output: k, Ω)
    OL ← Sort_Alg(RG);
    Lower = 1; Upper = n; k = ⌊ (Lower + Upper) / 2 ⌋;
    WHILE (Lower = k) DO
        success ← Sch_Alg(OL, Ω);
        IF (success = SUCCESS) THEN Upper = k;
        ELSE Lower = k;
        k = ⌊ (Lower + Upper) / 2 ⌋;
    END WHILE
    RETURN(k + 1; Ω);
END
```

Workload parameters used to generate simulation

results for MNP are listed below: Failure rate is 1 * 10^{-6}/hour; CW=[0.5,1.5];CV = [1,10]; ET = [5,200];δ= [1,40]

N	10	30	50	70	90
AEAP	3.00	3.64	4.88	6.14	7.25
ALAP	3.03	5.75	7.08	9.15	9.64
RCD	3.00	3.00	3.03	3.32	3.88

Table 4.3.1 Btree (MNP)

N	9	25	49	64	81	100
AEAP	3.00	3.00	3.00	3.01	3.12	3.49
ALAP	3.00	3.25	4.19	4.56	5.09	5.47
RCD	3.00	3.00	3.00	3.00	3.00	3.00

Table 4.3.2 Lattice (MNP)

Minimum number of processors (MNP) is a measure of the schedulability of a scheduling algorithm. In other words, the algorithm with better schedulability needs fewer processors to run a given set of tasks to meet their deadlines. Tables 4.3.1 and 4.3.2 show the impact of the number of tasks on the minimum number of processors. As the number of tasks increases, the minimum numbers of processors for most algorithms increase, except for RCD in the case of lattice DAGs which remains at the value of 3. In general, the more tasks in a B-tree or lattice DAG (thus the higher degree of parallelism), the It is because the more processors are needed in order to guarantee that all the tasks complete before their deadlines. However, in a heterogeneous real-time system MNP depends in a complex way on factors such as level of heterogeneity (computational and communicational), natures of the DAG and scheduling objective functions. Again, since RCD tends to assign tasks to processors that would execute tasks the fastest, or equivalently, fastest processors tend to be picked, thus RCD can potentially give rise to a relatively small MNP, compared to the other two non-RCD algorithms.

4.4 Relationship between execution time and reliability cost for the RCD algorithm

This experiment examines the relationship between execution time and reliability cost. We only consider the RCD algorithm, since AEAP and ALAP share similar property and are less relevant. Simulation parameters in this experiment are also the same as earlier experiments, except that ET has two ranges: (5,100) and (5,200).

N	10	30	50	70	90
ET = (5, 100)	0.64	1.93	3.22	4.49	5.78
ET = (5, 200)	0.83	2.50	4.18	5.82	7.44

Table 4.4.1 Btree (RCD) (System RC)

N	9	25	49	64	81
ET = (5, 100)	0.51	1.42	3.14	4.12	5.21
ET = (5, 200)	0.75	2.01	4.08	5.31	6.72

Table 4.4.2 Lattice (RCD) (System RC)

From the simulation results listed in tables 4.4.1 and 4.4.2, which show the system reliability cost as a function of ET and number of tasks, we observe that the reliability increase with the computational heterogeneity (ET) (from (5,100) to (5,200)). This is due to the fact that, when execution time in each c_{ij} increases the task reliability cost $\lambda_j c_{ij}$ also increase. We can conclude from this experiment that, as the executions time in each vector C_i go up, the reliability cost of the system also increases.

4.5 Deadline to execution time ratio

In this subsection we examine he impact of the deadline to execution time ration by generating through simulations the minimum number of processors (MNP) as a function of ET and MAX_DEADLINE (or MD). Except for ET and MAX_DEADLINE, workload parameters used in the experiment is the same as those in section 4.3.

Number of tasks	10	30	50	70	90	110
ET=(5,200);MD=(1,40)	3.00	3.00	3.03	3.32	3.88	4.1
MD = (1, 10)	3.00	3.00	3.05	3.63	3.98	4.23
MD = (1, 5)	3.00	3.00	3.10	3.67	3.99	4.37
MD = (0,1)	3.00	3.00	3.11	3.71	4.03	4.37
ET=(130,150);MD=(0,1)	3.00	3.19	3.94	4.34	4.87	5.56

Table 4.5.1 Btree (RCD) (MNP)

Number of tasks	10	30	50	70	90
ET = (5,200); MD = (0,1)	3.00	4.17	5.61	6.97	8.3
MD = (1, 5)	3.00	4.08	5.48	6.92	8.14
MD = (1, 10)	3.00	4.01	5.41	6.81	8.05
MD = (1, 40)	3.00	3.64	4.88	6.14	7.25

Table 4.5.2 Btree (AEAP) (MNP)

Number of tasks	10	30	50	70	90
ET= (5,200);MD = (0,1)	3.05	6.26	7.78	10.45	10.78
MD = (1, 5)	3.05	6.12	7.71	10.20	10.73
MD = (1, 10)	3.05	6.15	7.64	10.17	10.59
MD = (1, 40)	3.03	5.75	7.08	9.15	9.64
ET=(185,200);MD=(0,1)	3.00	4.02	4.98	6.46	6.85

Table 4.5.3 Btree (ALAP) (MNP)

Tables 4.5.1-4.5.3 reveal a general trend that the MNP value reduces as the deadline to execution time ratio increases. This is because as the ratio (thus the deadline) increases the constraint on processor allocation eases, reducing the MNP value. What's more interesting is that, for the RCD algorithm, as the ET becomes narrower (see the last row of Table 4.5.1), indicating a lowered computational heterogeneity, the MNP value increases. This suggests that higher computational heterogeneity helps the RCD algorithm in reducing the MNP value, thus enhancing its schedulability. On the other hand, for the ALAP algorithm, it is a lower computational heterogeneity that enhances the algorithm's schedulability (see the last row of Table 4.5.3). This is most likely to be true for AEAP as well since, much like ALAP, it originated from an algorithm for scheduling homogeneous real-time systems and ignores reliability cost. In other words, as the computational heterogeneity reduces the RCD algorithm and non-RCD algorithms start to slowly converge in their MNP values. This may be explained by the fact that the advantage of RCD over the two non-RCD 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.

5. CONCLUSIONS

To the best of our knowledge, most research work in the area of real-time task scheduling in distributed systems either did not consider fault-tolerance and reliability issues, or only considered homogeneous systems, or assumed independent tasks. In this paper we attempted to address all of these issues by proposing a reliability cost driven algorithm that schedules real-time tasks with precedence constraints in a heterogeneous distributed system. In this algorithm, called RCD, reliability cost is used as one of the objective functions for scheduling tasks. Two other algorithms, AEAP and ALAP, that are greedy in nature but do not consider reliability cost, are also presented for comparison purposes. Simulation results showed that RCD is significantly better than either AEAP or ALAP, in terms of system reliability cost and minimum number of processors, a measure of schedulability. Furthermore, the results suggested that higher computational heterogeneity is conducive to improving RCD's schedulability, while the opposite is true for ALAP (and most likely for AEAP as well). The simulations are based on two types of real-time task graphs (DAGs), namely, binary trees and lattice, that are representative of many real application algorithms. Workload parameters are either based on those used in the literature or chosen so as to represent reasonably realistic workload and provide some stress tests for our algorithms.

This work represents our first and preliminary attempt to address a very complicated problem. Future studies in this area could include (1) Present static fault-tolerant algorithms, as opposed to reliability cost driven ones, that schedule real-time tasks with precedence constraints on a heterogeneous system; (2) Study dynamic scheduling algorithms with fault-tolerance for aperiodic real-time tasks. (3) Consider more DAG types to represent more real applications.

REFERENCES

[1] P. Berman and B. DasGupta, Improvements in Throughput Maximization for Real-Time Scheduling, *DIMACS, Technical Report*, TR-99-52, 1999.

[2] G. L Chang., H. Joosun, M. S. Yang et. al, Analysis of Cache-Related Preemption Delay in Fixed-Priority Preemptive Scheduling, *IEEE Trans. on Computers*, 47(6), 1998, 700-713.

[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] H. Liu and M.E. Zarki, Adaptive Source Rate Control for Real-Time Wireless Video Transmission, *Mobile Networks and Application*, 3, 1998, 49-60.

[5] G. Manimaran and C. Siva Ram Murthy, An Efficient Dynamic Scheduling Algorithms for Multiprocessor Real-Time Systems, *IEEE Trans. On Parallel and Distributed Systems*, , 9(3), 1998, 312-319.

[6] S.M. Shatz, J.P. Wang, and M.Goto, Task Allocation for Maximizing Reliability of Distributed Computer Systems, *IEEE Trans. Computers*, 41(9), 1992, 1156-1168.

[7] S. H. Son and C. Chaney, *Supporting the Requirements for Multilevel Secure and Real-time Databases in Distributed Environments, Database Security: Status and Prospects*, (Chapman and Hall Publishing, 73-91, 1998)

[8] S. Srinivasan, and N.K. Jha, Safty and Reliability Driven Task Allocation in Distributed Systems, *IEEE Trans. Parallel and Distributed Systems*, 10(3), 1999, 238-251.

[9] H. Chetto, M. Silly, and T. Bouchentouf, Dynamic Scheduling of Real-Time Tasks under Precedence Constraints, *J. Real-Time Systems*, 2(3), 1990, 181-194.

[10] K. Ramamritham, Allocation and Scheduling of Complex Periodic Tasks, *Proc. Int'l Conf. Distributed Computing Systems*, 1990, 108-115.

[11] M.D. Natale and J.A. Stankovic, Dynamic End-to-End Guarantees in Distributed Real-Time Systems, *Proc. Real-Time Systems Symp*., 1994, 216-227.

[12] T. F. Abdelzaher and K. G. Shin, Combined Task and Message Scheduling in Distributed Real-Time Systems, *IEEE Transactions on Parallel and Distributed Systems*, 10(11), 1999.

[13] Eshaghian, M.M., Wu, Y.C, Mapping heterogeneous task graphs onto heterogeneous system graphs, *IEEE Proceedings 6th Heterogeneous Computing Workshop (HCW '97)*, Geneva, SWITZERLAND April 1, 1997.

[14] Sirbu, M.G.; Marinescu, D.C, A scheduling expert advisor for heterogeneous environments, *IEEE Proceedings 6th Heterogeneous Computing Workshop (HCW '97)*, Geneva, SWITZERLAND April 1, 1997.

[15] Xiao Qin, Z.F. Han, H. Jin, L.P Pang and SL Li, Real-time Fault-tolerant Scheduling in Heterogeneous Distributed Systems, *Proceeding of the 2000 International Workshop on Cluster Computing-Technologies, Environments, and Applications (CC-TEA'2000)*, Las Vegas, Nevada, USA, June, 2000.

[16] M. Maheswaran and H. J. Siegel, A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems, *IEEE Proceedings of the Seventh Heterogeneous Computing Workshop*, Orlando, Florida, 1998