Enhancing the Schedulability of Real-Time Heterogeneous Networks of Workstations (NOWs)

Nitin Auluck, *Member*, *IEEE Computer Society*, and Dharma P. Agrawal, *Fellow*, *IEEE*

Abstract—This paper proposes a Real-Time Duplication-Based Algorithm (RT-DBA) for scheduling precedence-related periodic tasks with hard deadlines on networks of workstations (NOWs). We have utilized selective subtask duplication that enables some tasks to have earlier start times, which enables additional tasks (and, hence, task sets) to finish before their deadlines, thereby increasing the schedulability of a real-time application. We strongly believe that duplication can be used as a tool for obtaining a better quality of service (QoS) from the real-time heterogeneous system, and this is our major contribution. We have taken both the computation and the communication heterogeneities into account while modeling such a system. Both data and control dependencies between the tasks have also been considered. Our algorithm exhibits scalability, fully exploits the underlying parallelism, and is capable of scheduling an application, even if the available number of processors is less than the required number of processors. Based on extensive simulation studies, we observe that RT-DBA offers an enhanced success ratio as compared to other scheduling schemes when communication is a dominant factor.

Index Terms—Favorite predecessor, heterogeneous network of workstations, periodic tasks, precedence constraints, real-time scheduling, subtask compaction, subtask duplication.

1 Introduction

In addition to their correctness [1], a real-time system is defined as one which makes results available in a timely manner, due to decreasing costs of processors, real-time applications are being implemented and run on networks of workstations (NOWs). Real-time requirements may necessitate intensive communication [2], [3] in these NOWs. In order to provide a cost-effective solution while satisfying desired performance requirements, such multicomputer systems may very often be heterogeneous. This means that they usually consist of both programmable processors and dedicated hardware systems for enhanced performance, and heterogeneity can be present within these two domains. For example, programmable processors can be general-purpose microprocessors, microcontrollers, or specialized digital signal processors (DSPs). Therefore, each task could have potentially a different execution time if it were to be executed on a microprocessor, a microcontroller, or a DSP system, and it is imperative to devise efficient scheduling algorithms for extracting a better performance on a heterogeneous multiprocessor platform [4], [5]. Real-time scheduling algorithms can be broadly categorized into two types—static algorithms [6], [7] and dynamic algorithms [8], [9]. Real-time scheduling on multiprocessors has been proven to be NP-complete, even for a very simplified task model such as two processors and unit task execution time [10]. This has encouraged various researchers to propose heuristics for solving the scheduling problem [2], [3], [4], and we also adopt a similar approach in this paper.

2 Motivation

As a major contribution, we are proposing a way to select communicating subtasks for duplication so that there is no need for interprocessor communication, thereby eliminating such precedence delays. Hence, a larger number of tasks and, consequently, a larger task set are able to meet the deadlines. We strongly believe that our proposed algorithm can increase the schedulability of real-life applications such as audio/video on demand wherein the server relays audio or video information to the user's computer by transmitting a certain number of frames per second. By employing our proposed subtask duplication approach, many tasks can be processed relatively faster and therefore enable a larger number of frames to be transmitted within the same time duration, and the server could possibly provide a better QoS. By employing task duplication, a larger number of subtasks can be scheduled within the deadline window, and hence, the utilization factor of the hard real-time system can be improved. This means that a larger number of task sets can meet their deadlines. Therefore, our proposed scheme can be very useful for traditional hard real-time military and command and control applications.

The aim of this research is twofold: 1) to propose a scheduling algorithm (Real-Time Duplication-Based Algorithm (RT-DBA)) that employs duplication in order to improve the schedulability of a hard real-time system on NOWs and 2) to ensure that a real-time schedule is generated, even if an adequate number of processors are not available, and, in doing so, demonstrate downward scalability. The proposed scheme works in the following

- *N. Auluck is with the Department of Computer Science, Quincy University, Quincy, IL 62301. E-mail: aulucni@quincy.edu.*
- *D.P. Agrawal is with the Department of Computer Science, University of Cincinnati, Cincinnati, OH 45221-0030. E-mail: dpa@cs.uc.edu.*

Manuscript received 20 Mar. 2008; revised 2 Oct. 2008; accepted 7 Nov. 2008; published online 20 Nov. 2008.
Recommended for acceptance by P. Srimani.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-03-0112.
Digital Object Identifier no. 10.1109/TPDS.2008.247.

stages: 1) The tasks are scheduled based on the rate-monotonic algorithm [11], i.e., the tasks with lower periods are assigned higher priority and are scheduled first. The proposed scheme computes a set of mathematical parameters for each subtask and then generates clusters of subtasks based on these quantities. 2) It assigns each cluster to an appropriate heterogeneous processor and schedules the messages on appropriate links. Based on the initial number of clusters, the critical number of processors (or CNP) is obtained that leads to reduced computation time. We assume that $\mathrm{CNP} = \text{initial number of clusters}$. Another quantity of interest is the number of processors that are available to schedule the real-time application. This quantity is called the available processors (or AP). 3) AP is compared with CNP and one of the following three actions is taken:

1. If $\mathrm{AP} < \mathrm{CNP}$, the proposed scheme demonstrates downward scalability and invokes the cluster compaction procedure to schedule all generated clusters on to the fewer number of processors.
2. If $\mathrm{AP} = \mathrm{CNP}$, the proposed scheme schedules the clusters to their appropriate processors.
3. If $\mathrm{AP} > \mathrm{CNP}$, the proposed scheme exploits the underlying parallelism and employs subtask duplication to reduce the final finish time of the application, thereby increasing its success ratio (SR).

The rest of this paper is organized as follows: In Section 3, we discuss the related work. Section 4 describes the task model along with some terminology. The proposed algorithm with pseudocode is elaborated in Section 5. The simulation results are described in Section 6. Finally, Section 7 concludes the paper. A running trace of the proposed algorithm, as well as the theoretical analysis, is shown in the Appendix.

3 Related Work

Real-time scheduling on multiple processors has been addressed using various approaches, basically extending the single-processor model, e.g., [4]. In another approach, real-time tasks are grouped into clusters based on communication delays, each cluster is assigned to a processor, e.g., [12], and several heuristics have been proposed to schedule complex periodic tasks on homogeneous multiprocessors. Hence, the algorithms in [12] cannot be directly adopted for heterogeneous systems. Moreover, unlike the schemes discussed in [12], our scheme is scalable. A replication-based scheme that improves the schedulability of a real-time application has been discussed in [13]. However, the scheme makes the naïve assumption that there is only one communication channel in the network. Essentially, the scheme is based on dynamic programming and can potentially have a very large search space. In contrast, our scheme considers multiple channels and is a heuristic-based approach.

Scheduling of real-time tasks on a uniprocessor has been a well-researched problem [11], [14]. Reward-based scheduling has been discussed in [15]. Extensive research has also been carried out for the homogeneous multiprocessor model [8], [9], [16], [17], [18]. Real-Time Scheduling on Heterogeneous Multiprocessors has also been a topic of extensive research [4], [5], [6], [11], [20], [21], [22], [23], [24], [25]. Recent papers [4], [5] extend the popular EDF algorithm for heterogeneous multiprocessors. One of the limitations is that they assume independence between various subtasks.

A reliability-driven scheduling scheme for tasks with precedence constraints on heterogeneous multiprocessor systems has been discussed in [21]. In our previous work [22], [23], we have proposed several scheduling algorithms for heterogeneous NOWs. Omari et al. [26] present an adaptive scheme that schedules fault-tolerant soft tasks on multiprocessors. A replication-based technique for multiprocessor scheduling has been discussed in [27]. A feedback control algorithm called EUCON has been proposed in [28]. In [29], the authors propose a solution to the problem of scheduling a set of precedence-related tasks on variable-speed processors. The roots of our scheme can be found in [30], [31], and [32], which do not consider real-time tasks. In [33] and [34], the authors presented a hardware-software cosynthesis system that partitions and schedules an embedded system consisting of multiple periodic tasks. In [35] and [36], the authors proposed algorithms for scheduling periodic tasks on identical multiprocessors. In this research, we assume processors of varying speeds and configurations. Task duplication has been suggested for I/O-intensive tasks in a cluster-based system [37] so that frequently copying of huge amounts of data can be avoided. Task duplication has also been used for improving the schedule length of the parallel application [38], [39], albeit for non-real-time systems.

4 Problem Specification

Various mathematical notations used for the real-time task parameters are shown in Table 1. The input to the algorithm is the real-time application (T), which consists of a set of independent periodic tasks. Each periodic task $(t_i \in T)$ can be further subdivided into a set of precedence-related subtasks [24]. The real-time system also consists of the set of available heterogeneous processors, $AP = \{p_1, p_2, p_3, \ldots, p_n\}$. A periodic task can be represented by a Directed Acyclic Graph (DAG) [16], [24], with a subtask $s(i,j,k)$ of a task t_i being represented as a node in the DAG, where i denotes the periodic task ID, j denotes the subtask ID within a periodic task, and k stands for the invocation ID of a subtask. The worst-case execution time of a subtask is given by $\mu(s(i,j,k), p_z)$, where p_z is an element of an array P $(p_z \in P)$. The communication cost between two subtasks $s(i,j,k)$ and $s(i,l,k)$ of a task $t_i \in T$ is given by $c((s(i,j,k), s(i,l,k)))$. The length of the longest path from the subtask to the exit node is given by $p\_e(s(i,j,k))$. The set of clusters is given by $CL = \{cl_1, cl_2, cl_3 \ldots\}$. The edges between the subtasks represent the data dependency between them in that a subtask cannot start till it has received the results from all of its predecessor tasks. The control dependency between tasks can be modeled based on the system functionality if certain conditions exist in the DAG. These conditions are represented on the edges of the DAG such that the transmission on such an edge will take place only if the associated condition is satisfied.

An example in Fig. 1 has two independent real-time tasks t_1 and t_2. The edges between the tasks represent the data flow (solid edges) and the control flow (dotted edges). The number along each edge represents the volume of data that needs to be transmitted if the two tasks are allocated to

TABLE 1
Mathematical Notations Used for Task Parameters

Notation	Task Parameter
T	The real-time application (set of independent periodic tasks)
AP	The set of available processors
C	Set of conditions for modeling control dependencies between the tasks
Q	Data structure queue used to select subtask clusters
PL	List of periodic tasks in increasing order of periods
i	Task id
j	Subtask id
k	Invocation id
t_i	Periodic task (such that $t_i \in T$) corresponding to a DAG
pd_i	Period of task t_i
d_i	Deadline of task t_i
$S(t_i)$	Set of subtasks belonging to task t_i
s(i,j,k)	k^{th} invocation of j^{th} subtask of task t_i
p_z	Processor to which current subtask is assigned
$\mu(s(i,j,k),p_z)$	Execution time of subtask s(i,j,k) provided it will execute on processor p_z
c(s(i,j,k), s(i,l,k))	Cost of communication between subtasks s(i,j,k) and s(i,l,k)
p_e(s(i,j,k))	Length of longest path from subtask s(i,j,k) to exit node
pred (s(i,j,k))	Set of immediate predecessors of s(i,j,k)
succ (s(i,j,k))	Set of immediate successors of s(i,j,k)
est(s(i,j,k))	Earliest start time of subtask s(i,j,k) provided it will execute on processor p_z
ect(s(i,j,k))	Earliest completion time of subtask s(i,j,k) provided it will execute on p_z
$fproc_m$ (s(i,j,k))	m^{th} favorite processor for subtask s(i,j,k)
fpred (s(i,j,k))	Favorite predecessor of subtask s(i,j,k)
bst(s(i,j,k))	Best start time of subtask s(i,j,k)
bct(s(i,j,k))	Best completion time of subtask s(i,j,k)
CL	Set of real-time subtask clusters

different processors. Two conditions—namely, C_1 and C_2 are shown. A condition $C_x \in C$ has two possible outcomes—true (C_x) or false $(-C_x)$. Based on the outcomes of various conditions for a task set, a subset of the tasks is executed. We assume that the same conditions exist for all invocations of the real-time tasks. It has been assumed that in each invocation (k) during pd_i, the period of a task (t_i), a task needs to be executed for all subtasks $(s(i,j,k))$. The deadline of a real-time task is given by d_i, and it can be $<=$ the task period. The schedule is generated for a length equal to the *hyperperiod (or hp)*, which is defined as the least common multiple (*LCM*) of all the task periods. Such a scheme may prove expensive if the periods for the real-time tasks are prime with respect to each other (as it will result in a very large hp). Some real-time literature states that "*in many realistic cases, this worst case may not be encountered*" [40] and "*in industry practice, task periods are typically assigned to be integer multiples of each other*" [41]. However, if the tasks in the application have coprime periods, then one can modify the periods of some selective tasks (increase or decrease) by a small amount [33]. Since the periods are modified by a small amount (1-5 percent), the feasibility of the real-time schedule is not affected drastically.

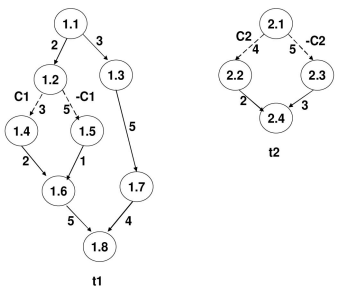

Fig. 1. Representing the data and control flow between two real-time tasks t_1 and t_2.

We assume that a set of processors are connected by a point-to-point network. A deadline-driven protocol called the earliest-due-date deadline (EDD-D) is used to schedule messages between the real-time subtasks [42]. When a subtask called the sender s wishes to send a message to a subtask called the destination d, EDD-D sets up a channel $s \rightarrow d$. The set of messages in the system is represented by $M = \{m_1, m_2, m_3, \ldots, m_n\}$, and each message is characterized by its deadline, given by $d(m_i)$, given $m_i \in M$. Messages are scheduled to be transmitted on links between

TABLE 2
Mathematical Relations for Subtask Parameters

(1)	**Earliest start time of entry node (*est*)** *est* (entry node) = 0
(2)	**Favorite processors of a subtask in the DAG (*fproc*)** *fproc*(s(i,j,k)) = p_z that gives: min (*est*(s(i,j,k)) + μ(s(i,j,k),p_z)), where $p_z \in P$.
(3)	**Earliest start time of a subtask in the DAG (*est*)** *est*(s(i,j,k)) = max (*ect*(s(i,a,k))|| *ect*(s(i,a,k)) + c(s(i,a,k),s(i,j,k))) s(i,a,k)∈ *Pred*(s(i,j,k)) s(i,a,k)∈ *Pred*(s(i,j,k)) *fproc*(s(i,a,k))=p_z *fproc*(s(i,a,k))≠p_z
(4)	**Earliest completion time of a subtask in the DAG (*ect*)** *ect*(s(i,j,k)) = *est*(s(i,j,k)) + μ(s(i,j,k),*fproc*(s(i,j,k)))
(5)	**Favorite predecessor of a subtask in the DAG (*fpred*)** *fpred*(s(i,j,k)) = s(i,b,k)∈ *Pred*(s(i,j,k))| (*ect*(s(i,b,k))+c(s(i,b,k),s(i,j,k)) > *ect*(s(i,c,k))+c(s(i,c,k),s(i,j,k))), where s(i,c,k)∈ *Pred*(s(i,j,k)) and s(i,c,k)≠ s(i,b,k)
(6)	**Best completion time of exit node (*bct*)** *bct* (exit node) = *ect* (exit node)
(7)	**Best completion time of a subtask in the DAG (*bct*)** *bct*(s(i,j,k)) = min(min(*bst*(s(i,d,k)) − c(s(i,j,k),s(i,d,k))), min(*bst*(s(i,d,k)))) s(i,d,k)∈ *Succ*(s(i,j,k)) s(i,d,k)∈ *Succ*(s(i,j,k)) s(i,j,k)≠*fpred*(s(i,d,k)) s(i,j,k)=*fpred*(s(i,d,k))
(8)	**Best start time of a subtask in the DAG(*bst*)** *bst*(s(i,j,k)) = *bct*(s(i,j,k)) − μ(s(i,j,k),*fproc*(s(i,j,k)))
(9)	**Execution cost of a subtask on a fictitious processor (μ_f)** μ_f (s(i,j,k), p_f) = average (μ (s(i,j,k), p_z)), for all $p_z \in P$.

pair of processors. Each link can be represented as $l_d \in L$, where L is the set of links.

Our scheduling algorithm assumes that subtasks are nonpreemptive. An edge between two processors represents the delay involved in sending or receiving messages from one processor to another. The processors are assumed to have local memory. It is also assumed that an I/O coprocessor is available so that computation and communication can be performed concurrently. If two communicating subtasks are assigned to the same processor, they incur zero communication delay. This is because once a subtask completes execution, its results are stored in the local memory of the processor, and the following subtask can access the result from the local memory [34]. It is assumed that when the subtasks are duplicated, they are consistent with the original subtasks. It has also been assumed that there is adequate amount of buffer space to store all the duplicated subtasks. The set of processors has been assumed to be dedicated. In other words, they are *free* unless a subtask is executing on them.

5 RT-DBA

Our proposed scheduling scheme is based on calculating a set of mathematical quantities (given in Table 1) for each real-time subtask. These quantities can be calculated using equations given in Table 2. The pseudocode in Fig. 2 outlines the steps involved in RT-DBA as follows:

1. **Stage 1 (create PL list).** In stage 1, a PL list consisting of all tasks arranged in increasing order of their periods is created using the rate-monotonic algorithm [11] in which the priority of a task is inversely proportional to its period.
2. **Stage 2 (calculate mathematical quantities for all subtasks of all tasks).** In stage 2, various mathematical quantities are calculated for every subtask as follows (please see the equations given in Table 2):

   a. The DAG is traversed from top to bottom, and the earliest start time (est) is calculated for each subtask. The est of a subtask is the earliest time that a particular subtask can start execution.
   b. Next, the favorite processors ($fproc_1$ to $fproc_m$) of all subtasks are calculated. $fproc_1$ is the processor that has the shortest completion time (starting from the root) if that subtask is executed on that particular processor. Similarly, $fproc_2$ is the processor on which the subtask has the next shortest completion time (see (2) in Table 2).
   c. Next, the earliest completion time (ect) is calculated for each subtask in a top-down fashion. The ect of a subtask is defined as the earliest time that a subtask can finish its execution. To calculate the est of a subtask, the ect values of all its predecessors need to be known. The est of that node is the maximum of the ect values of all its predecessors.
   d. Now, the favorite predecessor $(fpred)$ is calculated for each subtask in the graph. Among all the predecessors of a subtask, it is the $fpred$ that makes it wait the longest. Hence, the aim is to assign a task and its $fpred$ to the same cluster. If the

Input:
task set T = (t_1, t_2, …., t_n), subtask set $S(t_i)$ for all tasks in T such that $s(i,j,k) \in S(t_i)$
processor set AP
pred(s(i,j,k)): set of predecessors/parents of subtask s(i,j,k)
succ(s(i,j,k)): set of successors/children of subtask s(i,j,k)
Output:
real-time schedule
Begin:

1. obtain the pd_i values of all $t_i \in T$ and calculate the hp.
2. **if** (hp is large)
3. modify selected pd_i values by a small amount so that hp is reduced.
4. calculate *est*, $fproc_m$, *ect*, *fpred* and *p_e* for all subtasks $\in S(t_i)$ of all tasks $\in T$;
5. calculate *bst* and *bct* for all subtasks $\in S(t_i)$ of all tasks $\in T$;
6. perform cluster generation;
7. schedule the tasks on to the appropriate processors;
8. perform task duplication or cluster compaction;
9. perform message scheduling;

End

Fig. 2. Steps for RT-DBA.

$fpred$ has already been assigned previously, then the remaining set of predecessors are examined; the predecessor that has the minimum execution time on the current processor is chosen. This set will also exclude the predecessors that cannot be executed because of control dependencies.

e. Next, the best start time (bst) and the best completion time (bct) are calculated for each subtask in a bottom-up fashion. These quantities are used to find out if two subtasks are *critical*. Two subtasks $s(i,y,z)$ and $s(i,j,z)$ are critical with respect to each other if $bst(s(i,y,z)) - bct(s(i,j,z)) < c(s(i,j,z), s(i,y,z))$. The significance of subtask criticality implies that if two subtasks are not critical to each other, they can be assigned to different processors. For calculating the bct value of a subtask node, the bst values of all of its successors need to be calculated. If a subtask node has more than one successor, the minimum bst value is chosen as the bct value. In addition, if the subtask node is the $fpred$ of its successor, the communication cost between them is zero. Otherwise, the communication cost is subtracted from the bst value (see (7) in Table 2).

f. Next, the execution cost of a subtask on a fictitious processor is calculated (see (9) in Table 2). This quantity is the average execution cost of a subtask on all available processors. This quantity is used for the execution cost of a subtask in case $AP < CNP$ and the cluster compaction procedure needs to be carried out (please see stage 4).

3. **Stage 3 (perform cluster generation).** In stage 3, the cluster generation procedure is carried out (see Fig. 3). This stage involves the generation of a *queue* that contains all the subtasks in the graph arranged in increasing order of their p_e values. When calculating the p_e values, only the computation costs are taken into consideration (the communication costs are ignored). The clusters are formed by following the $fpred$ chain from the bottom to the top of the DAG. The generation of clusters is accomplished by performing a depth-first search, starting from the exit node. The search progresses by tracing the path from the initial subtask (the first *queue* element) to the entry subtask by following the $fpred$ value. The first cluster completes when the entry subtask is reached for the first time. The next cluster then starts from the next unassigned subtask in the *queue*. If the $fpred$ of a subtask has already been assigned and if it is not critical, then an unassigned subtask (among the current subtask's immediate predecessors), which has the lowest execution time on the current processor, is selected. The cluster generation process continues recursively, till all the subtasks have been assigned. Each cluster is then allocated to the first available $fproc_m$ of the header subtask of that cluster.

The process outlined above assumes that the deadline of a task is equal to its period. If the deadlines of the real-time tasks are smaller than their periods, then the quantity $(d_i - pd_i)$ is calculated for all the tasks. Then, the clusters are formed for the tasks based on a decreasing value of $(d_i - pd_i)$. The main idea behind this approach is that the tasks with higher $(d_i - p_i)$ values need to finish execution earlier. Hence, they deserve to be assigned to faster processors.

It may also be possible that the deadline of a task is greater than its period. Hence, at any time instant, there may be multiple invocations of a real-time task. The proposed scheme can be easily modified to handle such a case. The first invocation of the real-time task is assigned as before. The subsequent invocations will be assigned to the extra processors. After this assignment, the duplication phase can be carried out to enhance the SR.

4. **Stage 4 (compare AP and CNP and perform duplication or compaction).**

i If $AP > CNP$, subtask duplication is carried out to increase the SR. A check is done to see if each subtask in a cluster is preceded by its $fpred$

```
Input: DAG, set of predecessor tasks, set of successor tasks, and queue.
Output: Set of task clusters.
Begin:

1.    s(i,y,k) = first element of queue;
2.    current processor = fproc_1 of s(i,y,k);
3.    assign s(i,y,k) to current processor;
4.    while (NOT all tasks assigned) {
5.      s(i,z,k)= fpred(s(i,y,k));
6.      if(s(i,y,k) has more than one immediate predecessors) {
7.        if((bst(s(i,y,k))-bct(s(i,z,k))<c(s(i,z,k),s(i,y,k))) /* if they are critical to each other*/
8.          assign s(i,y,k) and s(i,z,k) to the same processor;
9.        else if(s(i,z,k) has already been assigned OR cannot be executed because of control dependency)
10.         s(i,z,k) = predecessor of s(i,y,k) that has the lowest run time on this processor;
11.     }
12.     assign s(i,z,k) to current processor;
13.     s(i,y,k) = s(i,z,k);
14.     if(s(i,y,k) is the entry node) {
15.         assign s(i,y,k) to current processor;
16.         s(i,y,k) = next unassigned element in queue;
17.         current processor = next available fproc_m of s(i,y,k);
18.       }
19.   }
End
```

Fig. 3. Cluster generation phase of RT-DBA.

subtask. If this is not the case, then the tail of that particular cluster is replaced with its $fpred$ chain, till the entry node. The cutoff tail is assigned to the next available $fproc_m$ of the last node. Duplication is carried out till all subtasks are preceded by their favorite predecessors or till there are no more remaining processors.

ii If $\mathrm{AP} < \mathrm{CNP}$, then the excess clusters are accommodated in fictitious processors. The procedure for cluster compaction is given in Fig. 4. The clusters are then arranged in decreasing order of $c\_c(cl_x, p_z)$, which is defined as the sum of execution times of all the subtasks assigned to $p_z \in P$. The value of $\mathrm{c\_c(cl_x, p_z)}$ for each cluster $\mathrm{cl_x} \in \mathbf{CL}$ is given by $c\_c(\mathrm{cl_x}, \mathrm{p_z}) = \sum \mu(s(i,j,k), p_z)$, where $s(i,j,k) \in t_i$, and $t_i \in T$. The *least populated* cluster of a system processor is merged with the *most populated* cluster of a fictitious processor. This process is carried out till the number of clusters becomes equal to the number of processors. The p_e value of a subtask is used to find out the place where the subtask is to be inserted. The subtasks are inserted so that the increasing p_e value is maintained as we go from a larger numbered subtask to a smaller numbered subtask.

Our algorithm can also handle the case when AP is variable. Based on the initial AP, RT-DBA performs one of the following actions: 1) compaction if $\mathrm{AP} < \mathrm{CNP}\ (\mathrm{mode_1})$, 2) mapping clusters on to the processors if $\mathrm{AP} = \mathrm{CNP}\ (\mathrm{mode_2})$, and 3) duplication if $\mathrm{AP} > \mathrm{CNP}\ (\mathrm{mode_3})$. The initial value of AP can either increase or decrease. If AP increases, the action taken by RT-DBA will depend on the mode that it is in. If it is in $\mathrm{mode_1}$, then the compaction procedure is called with this new AP value. If the value of AP increases further, it could go to $\mathrm{mode_2}$ or $\mathrm{mode_3}$. If it is in $\mathrm{mode_2}$, then the duplication procedure can be called. If it is in $\mathrm{mode_3}$, then further duplication can be carried out. If AP

```
1.    while(AP<CNP) {
2.    calculate c_c for each cluster cl_x ∈ CL;
3.    sort available processors in ascending order of c_c;
4.    temp=CNP/2;
5.     for(j=0;j<temp;j++)
6.        merge task lists of processors j and (temp*2-j-1); /*use p_e values to determine relative positions */
7.        decrement CNP;
8.    } //end while
End
```

Fig. 4. Pseudocode for cluster compaction.

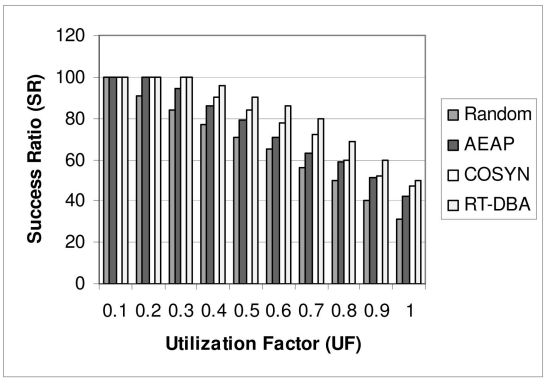

Fig. 5. SR versus UF for $\mathrm{CCR} = 0.1$.

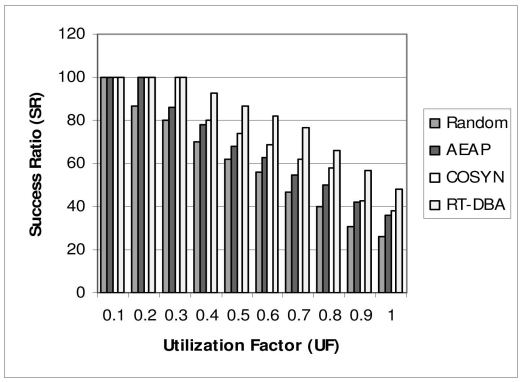

Fig. 6. SR versus UF for $\mathrm{CCR} = 1.0$.

decreases, the action taken by RT-DBA depends on the mode that it is in. If it is in mode_1, then more compaction can be carried. If it is in mode_2, compaction can be carried out by calling the compaction procedure. If it is in mode_3, the amount of duplication can be reduced by calling the duplication procedure with the new AP value. If the AP value decreases further, then it could go to mode_2 or mode_1.

6 Simulation Results

In this section, we compare the performance of RT-DBA with relevant algorithms in the literature, namely, AEAP [21] and COSYN [33] (the performance of MOGAC is similar to COSYN [34]). Both AEAP and COSYN are heuristic-based algorithms that schedule precedence-related periodic tasks on heterogeneous multiprocessors. We have used both randomly generated task graphs and graphs that represent real-life applications [43] such as sparse matrix solver and SPEC fppp. Randomly generated graphs have been used extensively in the past by researchers [12], [21], [22], [23]. Input DAGs have also been generated using the popular Task Graphs for Free (TGFF) suite [44], as well as the benchmark Standard Task Graph Set [43]. The processor architectures have been generated in a random fashion. We have coded a software scheduler module that accepts a set of precedence-related periodic tasks and generates a real-time schedule. Four performance metrics have been used. The first metric is the SR and is defined as the ratio of the number of task sets found to be schedulable (all tasks meet deadlines) by the algorithm to the total number of task sets considered. The second metric is called the Compaction Factor (CF), and it is defined as *AP/CNP*. The third metric is called the Heterogeneity Factor (HF), and it is defined as the standard deviation of the subtask execution costs. To calculate the standard deviation of a set of execution costs, we

1. calculate the mean value of all execution costs,
2. calculate the deviation for each execution cost $(\mathrm{cost} - \mathrm{mean})$,
3. calculate the squares of all the deviations obtained in step 2,
4. calculate the mean of all squared deviations, and
5. calculate the square root of the mean obtained from step 4.

The fourth metric is called the total Communication-to-Computation Ratio (CCR), and it is defined as *average edge cost/average node cost*. The execution cost for a node is the cost on its chosen processor. A low value of CCR would imply low levels of communication (versus computation) and vice versa.

The task and system parameters are generated in a way similar to [12]. The CCR values between subtasks in a periodic task are given by a uniform distribution and have been selected from the range (0, 10). The execution cost of each subtask is given by a uniform distribution and has been selected from the range (1, 100). The period of the first task in the task set has been set as follows: $pd_i = C_{av} * task\_size$ [12]. Here, C_{av} is the average execution time of all the subtasks in the periodic task, and $task\_size$ is the number of subtasks. The utilization factor is given by a uniform distribution, and its value is selected from the range (0, 1). The results shown here are for task sets with five periodic tasks $\{T = \mathrm{t}_1, \mathrm{t}_2, t_3, t_4, t_5\}$. Tasks t_1 to t_5 have {2, 4, 6, 8, 10} subtasks, respectively. The total number of edges has been selected from the range (20, 60). The number of predecessors and successors has been varied from 1 to 10. The periods of the tasks are pd_1, pd_2, pd_3, pd_4, and pd_5. If the period of the first task is pd, then the periods of the subsequent tasks will be $2pd$, $3pd$, $4pd$, and $5pd$. The length of the schedule generated is the LCM of the periods, i.e., it is $60 * pd$. Enough simulation runs are performed to provide 95 percent confidence interval. Each point in the following curves is the result of 1,000 runs of the algorithms. For each point in the following graphs, we generated 1,000 task sets. All the simulations were run on a Pentium 4 2.4-GHz PC running Red Hat Linux.

6.1 Effect of Communication to Computation Ratio

Figs. 5 and 6 indicate the effect of CCR on the proposed schemes. The number of processors ranged from 5 to 25. The utilization factor of the heterogeneous system is given by the equation $UF_{hetero} = \sum \mu(s(i,j,k))/(\mathrm{AP} * hp)$ for all $t_i \in T$. The subtask execution times have been generated randomly from the range {2, 200}. The experiments have been run for CCR values of 0.1, 1.0, and 10.0, representing varying levels of communication. All of the four algorithms have been assigned an equal number of processors. In Figs. 5 and 6, we observe that as the system utilization increases, the SR decreases. This can be explained as follows: as the utilization increases, so do the subtask execution times (for a fixed period). Hence, fewer subtasks (and, consequently, task sets) succeed in meeting their deadlines, and the SR value reduces. We also observe that RT-DBA offers a higher

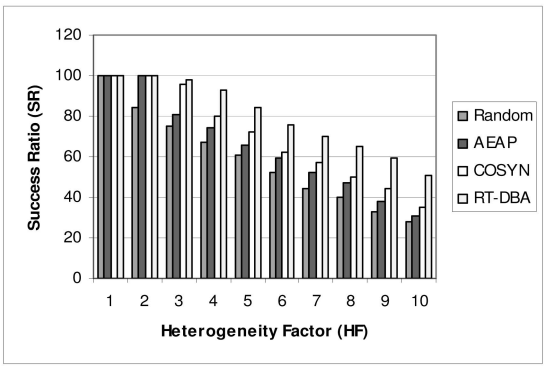

Fig. 7. Effect of heterogeneity.

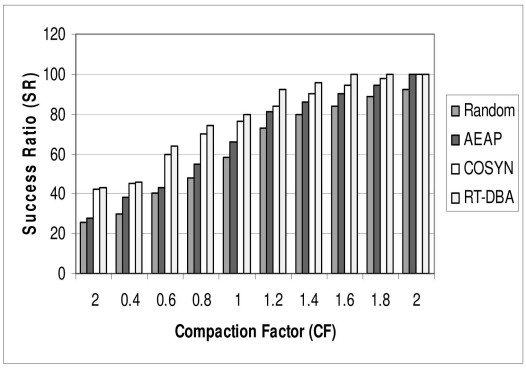

Fig. 8. Effect of compaction.

SR as compared to the other schemes, as selective subtask duplication results in some subtasks finding earlier start (and, hence, finish) times. This results in an increase in the number of task sets that can meet their deadlines. Hence, the SR value is higher as compared to AEAP and COSYN that do not employ duplication. It can also be seen that the improvements in the SR are more pronounced for the case where the CCR value is 1.0 than that when $\mathrm{CCR} = 0.1$. This is due to the fact that the aim of subtask duplication is to reduce the interprocessor communication between subtasks assigned to different processors. Hence, the gains (in terms of SR values) are greater when there is more communication in the system.

6.2 Effect of Heterogeneity

We have captured the effect of subtask heterogeneity (in terms of the execution time) on the performance of the real-time heterogeneous system in this simulation. The number of processors varied from 5 to 27. The min_e value has been kept fixed at 10, and the max_e value has been incremented by a constant factor of 4 time units. The results of this study are shown in Fig. 7. We observe that as the HF value of the system increases, its SR experiences a decrement. This is because with an increase in the HF value, there is more variation in the subtask execution times.

Also, a task set with larger variation in execution times will have a larger average subtask execution time. Hence, it will be assigned a larger period as compared to the same task set with lesser variation in execution times. But there is still a larger disparity between the average subtask execution time and the actual subtask execution times for the task set with a higher HF value. Hence, as the HF value is increased, the actual completion time of the subtasks increases. Therefore, a lesser number of subtasks succeed in meeting their deadlines. This leads to a decrease in the SR of the application. A CCR value of 1.0 has been assumed in this study.

6.3 Effect of Cluster Compaction

This study demonstrates the scalability aspect of RT-DBA. The results of this study are illustrated in Fig. 8. A low value of CF indicates that $\mathrm{AP} << \mathrm{CNP}$. On the other hand, a high value of CF indicates that $\mathrm{AP} >> \mathrm{CNP}$. The number of processors ranged from 3 to 30. We observe that as the CF increases, the SR value also increases. This is due to the fact that as the CF value increases, processors are added to the heterogeneous system, which enables the system to exploit the parallelism offered. Hence, a larger number of subtasks (and, hence, task sets) meet their deadlines. If $\mathrm{CF} < 1$, compaction is carried out; else, duplication is performed. Hence, more task sets are able to meet their deadlines, and the SR increases versus the compared schemes. Although AEAP and COSYN do not support compaction, we included this feature for the sake of fairness in comparison. A CCR value of 1.0 has been assumed in this experiment.

6.4 Effect of Message Load on Success Ratio

Fig. 9 shows the effect of the message load on the SR of the application. A CCR of 1.0 and $\mathrm{N} = 30$ have been assumed for this simulation. The number of messages has been varied from 6 to 60 in increments of six. The message $\mathrm{load} = \mathrm{number\ of\ messages/number\ of\ tasks}$ in the application. It is observed that as the message load increases, the SR value decreases. That is because with increased message load, there are more messages in the system that need to be scheduled (for the same number of tasks and processors). Hence, there is a greater load on the processors, and consequently, more tasks (and, hence, task sets) miss their deadlines.

6.5 Results for Real-Life Application Graphs

We have tested and compared various schemes for the task graphs that represent real-life applications such as sparse matrix solver and SPEC fppp [43]. The results of these simulations are given in Figs. 10 and 11. We observe that our proposed scheme offers a higher utilization value (the utilization value at which task sets begin to miss deadlines) as compared to COSYN, AEAP, and the Random scheme. If there is no communication in the application (i.e., $\mathrm{CCR} = 0$), the performance of COSYN and RT-DBA is similar. However, RT-DBA outperforms COSYN (and the other schemes) when there is significant communication in the system (i.e., $\mathrm{CCR} = 1.0$). This is because, the aim of duplication is to

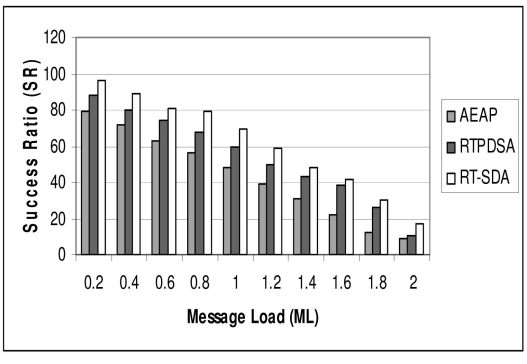

Fig. 9. Effect of message load on SR.

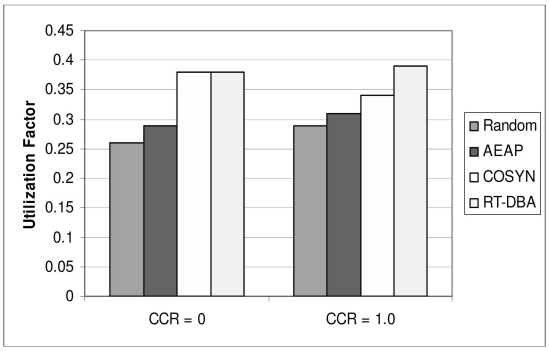

Fig. 10. U_b for the sparse matrix task graph.

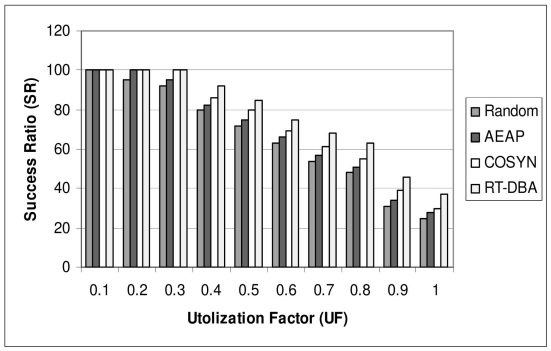

Fig. 12. SR versus UF for TGFF $(\mathrm{CCR} = 1.0)$.

reduce/minimize the IPC time between communicating subtasks assigned to different processors. This ensures that selected subtasks start (and, hence, finish) earlier. Hence, more subtasks (and task sets) are able to meet their deadline requirements. Therefore, it follows logically that the gain in SR will be more when there is more communication in the system. The sparse matrix solver graph has 96 tasks and 67 edges. The number of predecessors has been randomly chosen from {0, 6}. The execution costs have been randomly chosen from {0, 35}. The SPEC fppp graph has 334 tasks and 1,145 edges. The number of predecessors has been randomly chosen from {0, 80}. The execution costs have been randomly chosen from {0, 404}.

In addition, we have also tested all the algorithms using the TGFF suite [44]. The results are shown in Fig. 12. A CCR value of 1.0 has been used. We observe that as the utilization factor increases, the SR decreases. That is because the execution costs of the subtasks increases, and there is more tightness in the schedule. We also see that RT-DBA offers a better SR as compared to other existing schemes, owing to subtask duplication. For this experiment, we considered a task set $\mathrm{T} = \{\mathrm{t}_1, \mathrm{t}_2, \mathrm{t}_3\}$. Tasks t_1, t_2, and t_3 have 4, 8, and 12 subtasks, respectively. The task periods are pd_1, pd_2, and pd_3. If the period of the first task is pd, then the periods of the second and the third tasks will be 2pd and 3pd, respectively. Hence, $\mathrm{hp} = 6\mathrm{pd}$. The execution times have been chosen uniformly from the range {2, 200}. For each subtask, the number of predecessor tasks and successor tasks were chosen uniformly from the range {1, 10}.

6.6 Effect of Data and Control Dependency

Fig. 13 shows the effect of control dependency on the system performance. This simulation has been conducted for a single periodic task consisting of 30 subtasks. The number of conditions have been chosen as four for this task set. This resulted in the formation of eight subgraphs. In the figure, we observe that the addition of control dependency results in a higher SR for the real-time application. This can be explained as follows: with the addition of conditions between tasks, only a subset of them executes (as compared to the scenario in which there is no condition dependency between tasks and, hence, all tasks are executed). This leads to a reduction of load on the NOWs as multiprocessors, and this translates to a higher SR value.

6.7 The Case When Deadline Is Less than Period

In this section, we discuss the case when the deadline is smaller than the period. The results of our simulation are shown in Fig. 14. The same model that is described at the beginning of Section 5 has been used in this study. Looking at the figure, it can be seen that for the largest $(d - pd)$, the first scheme offers a better SR value than the plain RT-SDA. This can be explained as follows: The tasks with larger $(d - pd)$ values have less time to complete execution than the tasks with smaller $(d - pd)$ values and therefore should be executed first. This ensures that such tasks get their "favorite processors," and hence, the SR of the application could improve as compared to the plain RT-SDA.

7 Conclusion

The major contribution of this research is in proposing a selected duplication-based algorithm (RT-DBA) for enhancing the schedulability of precedence-related periodic tasks on real-time heterogeneous NOWs. When sufficient processors are available, subtask duplication cuts down

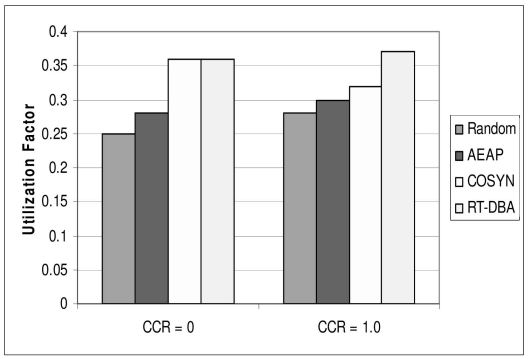

Fig. 11. U_b for SPEC fppp.

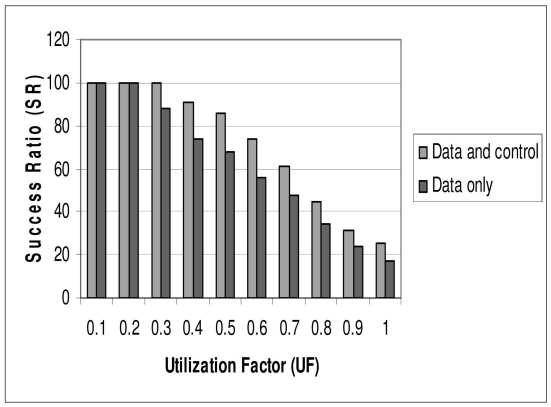

Fig. 13. Effect of control dependency.

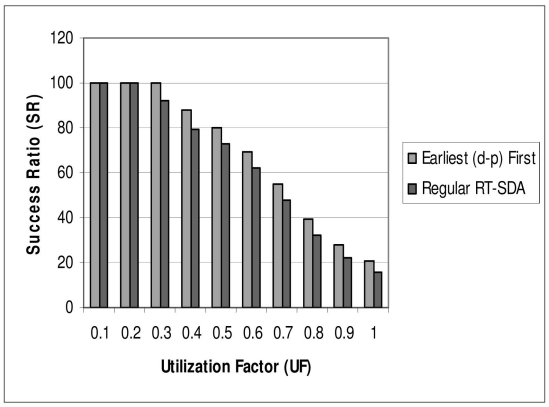

Fig. 14. Case when $\mathrm{d_i} < \mathrm{pd_i}$.

the interprocessor communication between communicating subtasks assigned to different processors. This enables some subtasks to find earlier start times. Hence, more subtasks (and task sets) are able to meet their deadline requirements. This phenomenon enables RT-DBA to offer a higher SR as compared to the other schemes. We strongly believe that the duplication can be used as a valuable tool to improve the schedulability of real-time applications, especially when communication is a dominant factor in the system. In addition, our scheduling scheme supports scalability for the heterogeneous NOWs in that the application is schedulable even if the required number of processors is not available, although this would result in a reduction in the schedulability of the application. Therefore, we conclude that a trade-off exists between scalability and schedulability—increased scalability results in a decrease in schedulability and vice versa.

APPENDIX A

Now, we will illustrate the working of RT-DBA with the help of the sample DAG $(\mathrm{t_1})$ in Fig. 15. The period of $\mathrm{t_1} = 50$. The execution times of all the subtasks of $\mathrm{t_1}$ are shown in Table 3. The algorithm works as follows:

1. First, we obtain the est for all subtasks. Let us consider subtask 10, which has two immediate predecessors: 6 and 7. The $fproc_1$ of 6, 7, and 10 is $\mathrm{p_1}$. According to its equation, $est(10) = \max.\ \{ect(7) \| ect(6) + c(6,10)\} = \max.\ \{11 \| (10+2)\} = \max.\ \{11 \| 12\} = 12$. The est values of all the subtasks in the DAG are shown in Table 4.
2. Next, we obtain the $fproc_m$ for all subtasks. Let us consider subtask 1.10. The sum of its est plus its execution time on all the processors ($\mathrm{p_1}$ to $\mathrm{p_8}$) is calculated as $\{(12+1), (12+5), (12+6), (12+7), (12+4), (12+8), (12+5), (12+6)\} = \{13, 17, 18, 19, 16, 20, 17, 18\}$. On arranging these values in increasing order, we get {13, 16, 17, 17, 18, 18, 19, and 20}. These values correspond to processors $\mathrm{p_1}$, $\mathrm{p_5}$, $\mathrm{p_2}$, $\mathrm{p_7}$, $\mathrm{p_3}$, $\mathrm{p_8}$, $\mathrm{p_4}$, and $\mathrm{p_6}$, respectively, and they are the favorite processors for subtask 10 (in increasing order).
3. Now, we obtain ect for all nodes in the application. For a subtask 10, $ect(10) = est(10) + \mu(10, fproc_1(10)) = 12 + 1 = 13$. Hence, the ect value for 1.10 is 13. ect values for all subtasks are shown in Table 4.

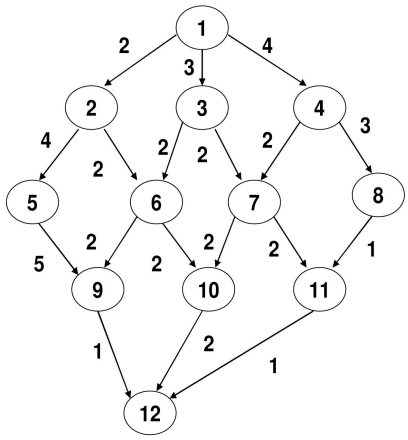

Fig. 15. Sample application $(\mathrm{t_1})$.

4. Now, we obtain the $fpred$ for each node in the application. Let us consider subtask 12. $fpred(12) = \{\max.\ (ect(9)+c(9,12)) \| (ect(10)+c(10,12)) \| (ect(11) + c(11,12))\} = \max.\ \{(19+1),\ (13+2),\ (15+1)\} = \max.\ \{20, 15, 16\} = 20$. Hence, $fpred\ (12) = 9$. The $fpred$ values of all subtasks are shown in Table 4.
5. Next, we obtain bst and bct for all subtasks. Let us consider subtask 3. It has two successor subtasks—6 and 7. Now, $bst(6) = 11$, and $bst(7) = 12$. According to the equation in Table 4, $bct(3) = \min.\ \{(bst(6) - c(3,6)), bst(7)\}$. Hence, $bct(3) = \min.\ \{(11-2), 12\} = \min.\ \{9, 12\} = 9$. Therefore, $bst(3) = bct(3) - \mu(3, fproc_1(3)) = bct(3) - \mu(3, p_1) = 9 - 2 = 7$.
6. Now, we arrange all nodes in the *queue* and start cluster formation. Initially, the *queue* formation takes place. For task $\mathrm{t_1}$, the *queue* is {12, 11, 10, 9, 6, 7, 8, 5, 3, 4, 2, and 1}. The generation of the first cluster starts from 12. It is preceded by $fpred(12) = 9$. Subtask 9 is preceded by its $fpred$, which is 5. Continuing in a similar fashion, the next subtasks will be 2 and 1. Hence, the first cluster is 12-9-5-2-1 and is assigned to its $fproc_1 = p_5$. Similarly, the second, third, and fourth clusters are 11-7-4-1, 10-6-3-1, and 8-4-1, and they are assigned to p_1, p_2, and p_7, respectively. The complete schedule is shown in Fig. 16. It has a schedule length of 34 time units. Since three free processors are available, duplication can be carried out. Subtask 7 in the second cluster is not preceded by its $fpred$. Hence, the subcluster 4-1 is cut off and is replaced with the $fpred$ chain 3-1. The second cluster is modified to 11-7-3-1, and the cutoff subcluster 4-1 is allocated to $\mathrm{p_6}$, which is the best available processor remaining. The schedule length now has reduced to 30 (see Fig. 17). Hence, duplication has resulted in a gain of 4 time units.

A.1 RT-DBA Time and Space Complexity Analysis

Let m_j be the total number of subtasks in DAG $d_j \in D$. Let p be the number of heterogeneous processors and e be the total number of communication edges in the DAG. The first step involves calculating the est, ect, $fpred$, $fproc_m$, p_e, and *queue*. While calculating the est, for each node, all the predecessors need to be examined, and all edges need to be traversed. Hence, the number of $\mathrm{predecessors} = m_i$. Therefore, for m_i nodes, the complexity for calculating est is

TABLE 3
Task Execution Times

Tasks	$\mu(p_1)$	$\mu(p_2)$	$\mu(p_3)$	$\mu(p_4)$	$\mu(p_5)$	$\mu(p_6)$	$\mu(p_7)$	$\mu(p_8)$
1	4	6	7	8	4	6	5	6
2	2	5	9	5	5	6	5	6
3	2	9	4	3	2	6	5	6
4	1	6	9	8	5	4	5	6
5	4	9	5	4	3	3	5	6
6	2	3	4	6	2	5	5	6
7	4	5	6	7	4	6	5	6
8	6	7	8	9	5	7	5	6
9	8	5	6	4	6	7	5	6
10	1	5	6	7	4	8	5	6
11	3	7	6	5	5	5	5	6
12	5	7	8	9	4	6	5	6

TABLE 4
Mathematical Quantities for the Tasks Shown in Fig. 15

Subtask	*fproc*$_1$	*fproc*$_2$	*fproc*$_3$	*fproc*$_4$	*fproc*$_5$	*fproc*$_6$	*fproc*$_7$	*fproc*$_8$	*est*	*ect*	*fpred*	*bst*	*bct*	*p_e*
1	P1	P5	P7	P2	P6	P8	P3	P4	0	4	-	3	7	43
2	P1	P2	P4	P5	P7	P6	P8	P3	4	6	1	9	11	35
3	P1	P4	P5	P3	P7	P6	P8	P2	4	6	1	7	9	33
4	P1	P6	P5	P7	P2	P8	P4	P2	4	5	1	8	9	34
5	P1	P5	P6	P4	P3	P7	P8	P2	6	9	2	12	16	26
6	P2	P1	P5	P3	P6	P7	P4	P8	7	9	2	11	14	23
7	P1	P5	P2	P7	P3	P6	P8	P4	7	11	4	12	16	24
8	P1	P5	P7	P8	P2	P6	P3	P4	5	11	4	9	15	25
9	P4	P1	P2	P7	P3	P5	P8	P6	15	19	5	16	20	17
10	P1	P5	P2	P7	P3	P8	P4	P6	12	13	7	17	18	17
11	P1	P4	P5	P6	P7	P3	P8	P2	12	15	7	16	19	16
12	P5	P1	P7	P6	P8	P2	P3	P4	20	24	9	20	24	9

$O(m_i^2 + e)$. While calculating the $fproc$, for each node, all p processors need to be examined. Therefore, for m_i nodes, the complexity is $O(m_i * p)$. While calculating the ect, all nodes need to be examined. Therefore, the complexity for this stage is $O(m_i)$.While calculating the value of p_e, for each node, m_i nodes need to be examined, and so, the complexity is $O(m_i^2)$. The formation of the *queue* involves sorting nodes based on p_e values (using Quick Sort). Therefore, the complexity is $O(m_i * log_2 * m_i)$. While calculating the $fpred$, for each node, roughly m_i predecessors need to be examined, so the complexity is $O(m_i^2)$. On adding all the above, the complexity becomes $O(m_i^2 + e + m_i + m_i * p + m_i^2 + e + m_i * log_2 m_i)$. For most practical DAGs and dense graphs, $m_i >> p$, and e *is of the order of* m_i^2. Hence, the complexity $= O(m_i^2)$. The second step deals with the calculation of the quantities bst and bct. This involves a bottom-up traversal of the DAG. The complexity of this step is $O(e)$. The next step involves the formation of the clusters. This is similar to a depth-first search of the DAG, and the complexity of this step is $O(m_i + e)$. For a

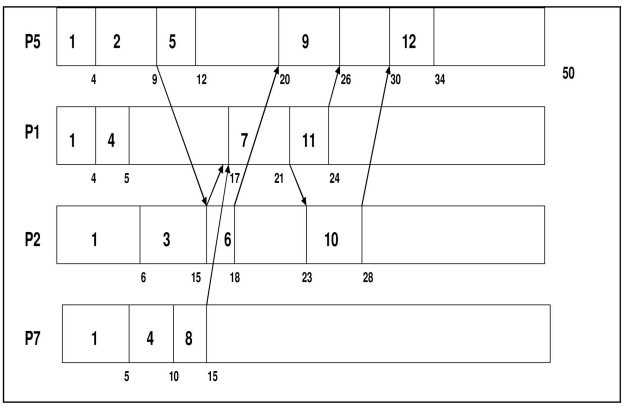

Fig. 16. Initial schedule.

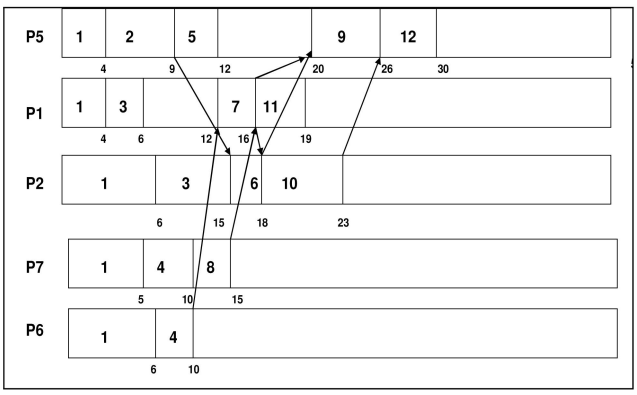

Fig. 17. Schedule after first duplication.

connected graph, this is equivalent to $O(e)$. The complexity of the duplication stage is given by $O(t * m_i)$, where t is the number of subtasks not preceded by their $fpred$ subtasks. In the worst case, all the subtasks are duplicated. In other words, $t = m_i$. Hence, the complexity becomes $O(m_i^2)$. The next stage is the cluster compaction stage, and this step involves sorting of the processors based on the empty slots. Therefore, each pass through the compaction code is of the order of $O(m_i + p * log_2 p)$. In the worst case, $p = m_i$. The number of required passes depends on the difference between AP and CNP and is given by the expression $\mathrm{Q} = log_2(m_i)$. Therefore, the complexity of compaction is $O(m_i^2)$. The calculation of the utilization factor values involves the traversal of all the invocations of all tasks. Therefore, the complexity of this step is $O(m_i)$. Hence, the overall complexity is $O(m_i^2 + e + e + m_i^2 + m_i^2 + m_i)$. For dense graphs, $e = m_i^2$. Hence, the time complexity for the specific DAG with m_i is $O(m_i^2)$. Now, DAG_1 will run hp/pd_1 times. Similarly, DAG_2 will run hp/pd_2 times, and so on. Hence, the time complexity for all n DAGs is given by the following equation:

$$m_1^2 * (hp/pd_1) + m_2^2 * (hp/pd_2) + \cdots + m_n^2 * (hp/pd_n) = hp\left(m_1^2/pd_1 + m_2^2/pd_2 + \cdots + m_n^2/pd_n\right).$$

Each real-time subtask can be represented as a floating point and can therefore be stored in one memory cell. The total number of real-time tasks is $hp/pd_1 + hp/pd_2 + hp/pd_3 + \cdots + hp/pd_n = hp * (1/pd_1 + 1/pd_2 + 1/pd_3 + \cdots + 1/pd_n)$. For the sake of brevity, let us assume that this is equal to x. Since each subtask needs one memory cell, x subtasks need x memory cells. For all DAGs, the total number of edges can be given by E. For very dense graphs, $E = x^2$. Hence, all edges need x^2 memory locations. The execution cost array for a subtask depends on the number of processors. Each subtask needs AP elements. Therefore, all x subtasks will need $x * AP$ elements. In the worst case, there will be one subtask and one processor, so $\mathrm{AP} = x$. Hence, all x subtasks will need x^2 memory locations. The number of subtasks that need to be $\mathrm{duplicated} = x^2$. On adding all the above, we get $x + x^2 + x^2 + x^2$. Hence, the space complexity of RT-DBA is of the order of x^2.

Schedulability test condition S-1. A task $t_i \in T$ is schedulable with RT-DBA if the following condition is satisfied: $ect(s(i,j,k)) \leq hp$ such that $|succ(s(i,j,k))| = 0$.

The exit node will have zero successors. Because of the precedence constraints, the exit node will always be the last node to execute. A task set $T = \{\mathrm{t}_1, \mathrm{t}_2, t_3, \ldots\ldots, t_{ap}\}$ will be schedulable with RT-DBA if the above test condition holds for all $t_i \in T$.

Schedulability test condition S-2. If UF_{hetero} corresponds to the task load of the heterogeneous system, Ω_{hetero} corresponds to the communication load, and Φ_{hetero} corresponds to the idle load, then a task set is schedulable with RT-DBA if $UF_{hetero} \leq AP - (\Omega_{hetero} + \Phi_{hetero})$.

CL_{hetero} refers to the communication load of the real-time application. The communication cost is the delay associated with data transmission and is given by $c(s(i,j,k), s(i,l,k))$, where $s(i,j,k)$ and $s(i,l,k)$ are the two communicating subtasks. Note that based on our model, subtasks that are part of the same task can communicate with each other (hence, the i term in both subtasks is the same). Also, data transmission is done between the same invocations of two communicating subtasks (hence, the k term in both subtasks is the same). Let us define Δ to be the set of all precedence-related subtask pairs that are not assigned to the same processor:

$$\Delta = \{s(i,j,k), s(i,l,k)\} \text{ for all } t_i \in T \text{ where } i, j, k \text{ and } l \text{ are all integers.} \quad (1)$$

The sum total of communication edge costs for all such pairs would give us the total communication cost. Let us say that this $\mathrm{cost} = \Omega_{hetero}$. Hence, we get

$$\Omega_{hetero} = \sum c(s(i,j,k), s(i,l,k)) \text{ for all } t_i \in T \text{ and all } ST(t_i). \quad (2)$$

Since RT-DBA is nonpreemptive, there is a certain amount of time that a processor can be idle. For a specific cluster $cl_\tau \in CL$, this time can be given as $hp - (lact(head_{cl\tau}))$, where $head_{cl\tau}$ corresponds to the head subtask of cluster cl_τ. For all clusters, this time can be represented by (if the total number of clusters in an application is α)

$$\Phi_{hetero} = (hp - lact(head_{cl1})) + (hp - lact(head_{cl2}) + \cdots + (hp - lact(head_{cla})). \quad (3)$$

A real-time application will be schedulable if the sum of the task load, communication delay, and idle time is less than the total processing capacity of the workstations. The total processing $\mathrm{capacity} = \mathrm{AP}$. Hence, we get

$$UF_{hetero} + \Omega_{hetero} + \Phi_{hetero} \leq AP \Rightarrow UF_{hetero} \leq AP - (\Omega_{hetero} + \Phi_{hetero}). \quad (4)$$

ACKNOWLEDGMENTS

This research has been supported by the Ohio Board of Regents PhD Enhancement Funds.

REFERENCES

[1] J. Liu, *Real-Time Systems.* Prentice Hall, 2000, ISBN: 0130996513.
[2] P. Pop, "Analysis and Synthesis of Communication Intensive Heterogeneous Real-Time Systems," PhD thesis, Dept. of Computer and Information Science, Linkoping Univ., 2003.
[3] X. Qin and H. Jiang, "Improving the Performance of Communication Intensive Parallel Applications Executing on Clusters," *Proc. IEEE Int'l Conf. Cluster Computing (CLUSTER '04),* p. 493, Sept. 2004.
[4] S. Baruah, "Scheduling Periodic Tasks on Uniform Multiprocessors," *Proc. 12th Euromicro Conf. Real-Time Systems (Euromicro RTS '00),* pp. 7-14, June 2000.
[5] S. Baruah and J. Goossens, "Rate-Monotonic Scheduling on Uniform Multiprocessors," Technical Report 472, ULB, 2002.
[6] D.T. Peng and K.G. Shin, "Static Allocation of Periodic Tasks with Precedence Constraints in Distributed Real-Time Systems," *Proc. Ninth Int'l Conf. Distributed Computing Systems (ICDCS '89),* pp. 190-195, June 1989.
[7] B. Andersson, S. Baruah, and J. Jonsson, "Static Priority Scheduling on Multiprocessors," *Proc. 22nd IEEE Real-Time Systems Symp. (RTSS '01),* pp. 193-202, Dec. 2001.
[8] G. Manimaran and C. Siva Ram Murthy, "An Efficient Dynamic Scheduling Algorithm for Multi-Processor Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 3, pp. 312-319, Mar. 1998.
[9] H. Chetto, M. Silly, and T. Bouchentouf, "Dynamic Scheduling of Real-time Tasks under Precedence Constraints," *J. Real-Time Systems,* vol. 2, no. 3, pp. 181-194, 1990.

[10] J. Stankovic, M. Spuri, M. Natale, and G. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems," *Computer,* vol. 28, no. 6, pp. 16-25, June 1995.

[11] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM,* vol. 20, no. 1, pp. 46-61, Jan. 1973.

[12] K. Ramamritham, "Allocation and Scheduling of Precedence-Related Periodic Tasks," *IEEE Trans. Parallel and Distributed Systems,* vol. 6, no. 4, pp. 412-420, Apr. 1995.

[13] S.T. Cheng, S.I. Hwang, and A.K. Agrawala, "Schedulability-Oriented Replication of Periodic Tasks in Distributed Real-Time Systems," *Proc. 15th Int'l Conf. Distributed Computing Systems (ICDCS '95),* pp. 196-203, 1995.

[14] K. Jeffay, D. Stanat, and C. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks," *Proc. 12th IEEE Real-Time Systems Symp. (RTSS '91),* pp. 129-139, Dec. 1991.

[15] J.Y. Chung, J.W.S. Liu, and K.J. Lin, "Scheduling Periodic Jobs That Allow Imprecise Results," *IEEE Trans. Computers,* vol. 39, no. 9, pp. 1156-1174, Sept. 1990.

[16] T. Abdelzaher and K. Shin, "Combined Task and Message Scheduling in Distributed Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 11, pp. 1179-1191, Nov. 1999.

[17] H. Kaneko, J. Stankovic, S. Sen, and K. Ramamritham, "Integrated Scheduling of Multimedia and Hard Real-Time Tasks," *Proc. 17th IEEE Real-Time Systems Symp. (RTSS '96),* pp. 206-217, Dec. 1996.

[18] M. Natale and J. Stankovic, "Dynamic End-to-End Guarantees in Distributed Real-Time Systems," *Proc. 15th IEEE Real-Time Systems Symp. (RTSS '94),* pp. 216-227, 1994.

[19] G. Fohler and K. Ramamritham, "Static Scheduling of Pipelined Periodic Tasks in Distributed Real-Time Systems," *Proc. Ninth Euromicro Workshop Real-Time Systems (Euromicro RTS '97),* pp. 128-135, June 1997.

[20] T.F. Abdelzaher and K.G. Shin, "Period-Based Partitioning and Assignment for Large Real-Time Applications," *IEEE Trans. Computers,* vol. 49, no. 1, pp. 81-87, Jan. 2000.

[21] X. Qin, H. Jiang, C. Xie, and Z. Han, "Reliability Driven Scheduling for Real-time Tasks with Precedence Constraints in Heterogeneous Distributed Systems," *Proc. Int'l Conf. Parallel and Distributed Computing and Systems (PDCS '00),* pp. 617-623, Nov. 2000.

[22] N. Auluck and D.P. Agrawal, "Reliability Driven, Non-Preemptive Real-Time Scheduling of Periodic Tasks on Heterogeneous Systems," *Proc. IASTED Int'l Conf. Parallel and Distributed Computing and Systems (PDCS '02),* pp. 803-809, Nov. 2002.

[23] N. Auluck and D.P. Agrawal, "A Scalable Task Duplication Based Algorithm for Improving the Schedulability of Real-Time Heterogeneous Multiprocessor Systems," *Proc. ICPP Second Int'l Workshop on Compile/Run Time Techniques for Parallel Computing (CRTPC '03),* pp. 89-96, Oct. 2003.

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

[25] E. Huh, L. Welch, B. Shirazi, and C. Cavanaugh, "Heterogeneous Resource Management for Dynamic Real-Time Systems," *Proc. Ninth Heterogeneous Computing Workshop (HCW '00),* pp. 287-296, 2000.

[26] R.A. Omari, A.K. Somani, and G. Manimaran, "An Adaptive Scheme for Fault-Tolerant Scheduling of Soft Real-Time Tasks in Multiprocessor Systems," *J. Parallel and Distributed Computing,* vol. 65, no. 5, pp. 595-608, May 2005.

[27] C.H. Papadimitrou and M. Yannakakis, "Towards an Architecture-Independent Analysis of Parallel Algorithms," *SIAM J. Computing,* vol. 19, no. 2, pp. 322-328, Apr. 1990.

[28] C. Lu, X. Wang, and X. Koutsoukos, "Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 6, pp. 550-561, June 2005.

[29] J.W.S. Liu and C.L. Liu, "Bounds on Scheduling Algorithms for Heterogeneous Computing Systems," *Information Processing 74,* pp. 349-353, 1974.

[30] S. Ranaweera and D.P. Agrawal, "A Scalable Task Duplication Based Scheduling Algorithm for Heterogeneous Systems," *Proc. Int'l Conf. Parallel Processing (ICPP '00),* pp. 383-390, Aug. 2000.

[31] S. Darbha and D.P. Agrawal, "Optimal Scheduling Algorithm for Distributed Memory Machines," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 1, pp. 87-95, Jan. 1998.

[32] R. Bajaj and D.P. Agrawal, "Improving Scheduling of Tasks in a Heterogeneous Environment," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 2, pp. 107-118, Feb. 2004.

[33] B.P. Dave, G. Lakshminarayana, and N. Jha, "COSYN: Hardware/Software Co-Synthesis of Embedded Systems," *Proc. 34th Ann. Conf. Design Automation (DAC '97),* pp. 703-708, 1997.

[34] R.P. Dick and N.K. Jha, "MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Co-Synthesis of Hierarchical Heterogeneous Distributed Embedded Systems," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 17, no. 10, pp. 920-935, Oct. 1998.

[35] S. Baruah and J. Goossens, "The Static-Priority Scheduling of Periodic Task Systems upon Identical Multiprocessor Platforms," *Proc. IASTED Int'l Conf. Parallel and Distributed Computing and Systems (PDCS '03),* pp. 427-432, Nov. 2003.

[36] S. Baruah and J. Anderson, "Energy Efficient Synthesis of Periodic Task Systems upon Identical Multiprocessor Systems," *Proc. 24th Int'l Conf. Distributed Computing Systems (ICDCS '04),* pp. 428-435, Mar. 2004.

[37] X. Qin, "Improving Network Performance through Task Duplication for Parallel Application on Clusters," *Proc. 24th IEEE Int'l Performance Computing and Comm. Conf. (IPCCC '05),* pp. 35-42, Apr. 2005.

[38] I. Ahmad and Y.K. Kwok, "On Exploiting Task Duplication in Parallel Program Scheduling," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 9, pp. 872-892, Sept. 1998.

[39] Y.C. Chung and S. Ranka, "Application and Performance Analysis of a Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed Memory Multiprocessors," *Proc. ACM/IEEE Conf. Supercomputing,* pp. 512-521, 1992.

[40] N. Guan, Z. Gu, Q. Deng, S. Gao, and G. Yu, "Exact Schedulability Analysis for Static-Priority Global Multiprocessor Scheduling Using Model Checking," *Proc. IFIP Workshop Software Technologies for Future Embedded and Ubiquitous Systems (SEUS),* 2007.

[41] D. Rhodes and W. Wolf, "Allocation and Data Arrival Design of Hard Real-Time Systems," *Proc. Int'l Conf. Computer Design (ICCD '97),* pp. 393-399, 1997.

[42] C.M. Krishna and K.G. Shin, *Real-Time Systems.* McGraw-Hill, 1997.

[43] *The Standard Task Graph Set,* http://www.kasahara.elec.waseda.ac.jp/schedule, 2008.

[44] R. Dick, D. Rhodes, and W. Wolf, "TGFF: Task Graphs for Free," *Proc. Sixth Int'l Workshop Hardware/Software Codesign (CODES/CASHE '98),* pp. 97-101, Mar. 1998.

Nitin Auluck received the BE degree in electrical and electronics engineering from Gulbarga University, Gulbarga, India, in 1998 and the PhD degree from the University of Cincinnati, Cincinnati, Ohio, in 2005. He has been an assistant professor in the Department of Computer Science, Quincy University, Quincy, Illinois, since August 2004. His research interests include real-time systems, parallel and distributed systems, embedded systems, and high-assurance systems. He is a member of the IEEE Computer Society.

Dharma P. Agrawal is the Ohio Board of Regents distinguished professor of computer science and the founding director of the Center for Distributed and Mobile Computing in the Department of Computer Science, University of Cincinnati, Ohio. He was a visiting professor of ECE at Carnegie Mellon University, on sabbatical leave during the Autumn 2006 and Winter 2007 Quarters. He was a faculty member at North Carolina State University, Raleigh, from 1982 to 1998 and at Wayne State University, Detroit, from 1977 to 1982. His recent research interests include resource allocation and security in mesh networks, efficient query processing and security in sensor networks, and heterogeneous wireless networks. He coauthored the introductory textbook on wireless and mobile computing, which has been widely accepted throughout the world and has been has been reprinted in both China and India and translated in Korean and Chinese languages. His second coauthored book on ad hoc and sensor networks, published in Spring of 2006, has been named as a bestseller by the publisher. A new coedited book entitled *Encyclopedia on Ad Hoc and Ubiquitous Computing* is being published by World Scientific. He has given tutorials and extensive training courses in various conferences in the US and numerous institutions in Taiwan, Korea, Jordan, Malaysia, and India in the areas of ad hoc and sensor networks and mesh networks. He is an editor for the *Journal of Parallel and Distributed Systems*, the *International Journal on Distributed Sensor Networks*, the *International Journal of Ad Hoc and Ubiquitous Computing*, and the *International Journal of Ad Hoc & Sensor Wireless Networks*. He has served as an editor of the *Computer* magazine, the *IEEE Transactions on Computers*, and the *International Journal of High Speed Computing*. He has been the program chair and general chair for many international conferences and meetings. He has received numerous certificates and awards from the IEEE Computer Society and has been elected as a core member. He received a Third Millennium Medal from the IEEE for his outstanding contributions. He has also delivered the keynote speech for 10 international meetings. He also has five patents in the wireless networking area. He has also been named as an ISI Highly Cited Researcher in Computer Science. He is a fellow of the IEEE, the ACM, the American Association for the Advancement of Science (AAAS), and the World Innovation Foundation (WIF).