Controlled duplication scheduling of real-time precedence tasks on heterogeneous multiprocessors

Jagpreet Singh[1] and Nitin Auluck[2]

[1] Indian Institute of Information Technology Allahabad, Uttar Pradesh, India
jagpreets{@iiita,@iitrpr}.ac.in
[2] Indian Institute of Technology Ropar, Rupnagar, Punjab, India
{nitin@iitrpr.ac.in}

Abstract. Duplication based heuristics have been widely utilized for scheduling communication intensive, precedence constrained tasks on multiple processors. Duplicating the predecessor of a task on the processor to which the task is assigned can result in the minimization of the communication cost. This helps in reducing the schedule length. However, this reduction comes at the cost of extra computing power required to duplicate the tasks. We have tried to address this trade-off in this paper. We propose "controlled" duplication algorithms for scheduling real-time periodic tasks with end-to-end deadlines on heterogeneous multiprocessors. We observe that whether to duplicate tasks or not is decided by the task deadlines. In the case that the deadline can be met without duplication, more schedule holes are created. These holes can be used to schedule other tasks. Simulations show that the proposed algorithms efficiently utilize the holes and improve the success ratio by $15\% - 50\%$ versus comparable algorithms.

1 Introduction

The requirement of scientific and industrial applications to generate logical as well as time bound results have posed various challenges, namely: exploiting the parallelism offered by the current hardware and completing applications under strict timing constraints. Due to these requirements, heterogeneous systems have gained widespread popularity. These systems allow for the combination of high performance, low cost and different capability hardware with the help of heterogeneous interconnections such as: Network on a chip (NoC) and Network of Workstations (NoWs). The timing constraints are fulfilled by employing an efficient real-time scheduler. The scheduling algorithm allocates and schedules jobs to ensure that all the task instances in the task set meet their deadlines. If a task set meets its deadlines, then it is said to be schedulable.

Definition 1. ***(Task Set).*** *A task set (Fig. 1) models multiple real-time applications where each application (known as a task) is represented as a directed acyclic graph (DAG) with release time, period and hard end-to-end deadline. The*

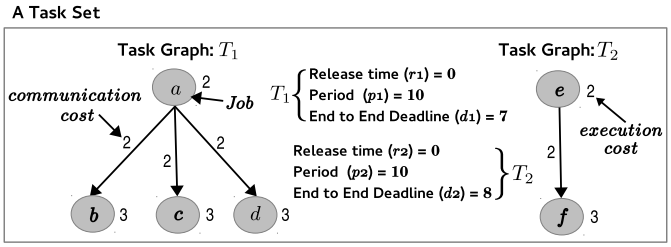


Fig. 1. A Task Set

nodes of each DAG represent subtasks[3] *and the edges represent the precedence constraints, as well as the communication cost between the subtasks.*

In real-time systems, scheduling algorithms can be broadly classified into two categories: static and dynamic. In static algorithms, information about the tasks is known in advance, which is not the case in dynamic algorithms. Heuristics for real-time scheduling on heterogeneous multiprocessors have been proposed for both the static [16, 8, 3] as well as the dynamic [15, 23] environments. This paper falls into the domain of static algorithms .

Scheduling a DAG on multiprocessors in real-time and non real-time systems is a challenging problem [12, 13]. It has become harder with the introduction of heterogeneous processing and networking components. Basically, the problem on these two systems differs because of the properties of the task graph and the objective. The majority of the algorithms in non real-time systems consider a single task graph with an objective of minimizing the maximum schedule length, also known as the *makespan* [12]. On the other hand, in real-time systems, the input to the algorithm is a task set (periodic or non-periodic) consisting of a number of independent or dependent tasks with deadlines. The main objective is to meet the hard deadlines and decrease the tardiness of the soft deadlines, where tardiness is the subtraction of the deadline from the schedule length.

More often that not, the real-time algorithms are inspired from or are an extension of a non-real time scheduling approach [15, 3, 4]. On the basis of the design, these algorithms for scheduling a DAG on multiprocessors (homogeneous & heterogeneous) are broadly classified into: list-based and clustering based, with or without duplication. List-based scheduling [15] assigns priorities to all the ready jobs, stores them in a list and later assigns to processors according to the priorities to minimize a particular cost function. In clustering, the jobs are combined to form clusters on the basis of communication delays, data dependencies etc. After that, the clusters are allocated to processors [3, 9].

Duplication has been widely used to achieve reliability and fault tolerance in real-time and non real-time scheduling [15, 21]. It has also proved to be a vital heuristic for minimizing the makespan [1]. By duplicating the heavily communicating jobs on a single processor, the interprocessor communication cost can be minimized. The jobs are made to start earlier and hence, finish earlier, which

[3] (the terms node, job and subtask have been used interchangeably)

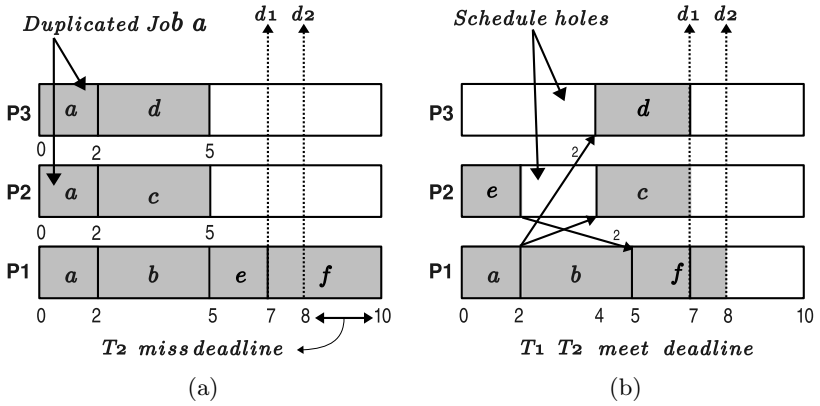


Fig. 2. For task set in Fig. 1, (a) represent schedule with duplication that misses the deadline (b) schedule without duplication that meets the deadlines

reduces the overall makespan of the task graph. Duplication is a well researched heuristic for non real-time scheduling of a single task graph on heterogeneous multiprocessors [4–6, 18].

However, duplication in the context of meeting deadlines is still relatively unexplored. We observe that, duplicating a job utilizes the extra computing power (or a schedule hole) on a processing element (PE). This extra computing power may be used to schedule jobs of other task instances. Therefore, although using duplication can help a task graph instance to meet its deadline, it may cause the other tasks to miss their deadlines because of the unavailability of the appropriate schedule holes. Hence, an interesting tradeoff exists between the number of duplicated jobs and the number of schedule holes available. This article identifies this tradeoff and proposes controlled duplication based heuristics. Simulations have shown that the proposed algorithms improve the success ratio by 15% to 50% vs. the other known non-duplication and duplication based algorithms, even under higher processor utilizations and communication costs.

2 Motivation-W^2H^2

The design of a duplication heuristic has two steps: "*where* to duplicate jobs" and "*how* to perform duplication". Mainly, there are two strategies that are used for the first step: duplicating subtasks in schedule holes [5] or allocating extra space other than the holes [4]. The first approach, also known as an insertion based approach, adds more to the computational complexity of the algorithm to find an appropriate schedule hole for duplication, but is more effective than the second non-insertion based approach. A number of approaches have been used for the second step: 1) duplicate a single immediate predecessor (SIP) [5], 2) duplicate a chain of predecessors till the root node (COP) [4], 3) duplicate the immediate predecessors, and then the ancestors (IPFA) [6].

Duplication in real-time systems adds two more challenges to the above: "*when* to duplicate" and "*how much* duplication" is to be performed. Fig. 2

demonstrates these challenges. It shows schedules of two task instances: $T1$ ($r = 0, p = 10, d = 7$) and $T2$ ($r = 0, p = 10, d = 8$) of Fig. 1, where r, p and d are defined as release time, period and deadline of the tasks respectively. Both tasks are required to be scheduled on 3 processing elements $P1$-$P3$ and have the same execution cost on all PEs. Since task $T1$ has a lesser deadline, it is given the higher priority and is scheduled first. If $T1$ is scheduled with duplication, it finishes before its deadline at 7. However, task $T2$ is unschedulable now (Fig. 2(a)). In the other case, scheduling $T1$ without duplication leaves enough schedule holes on $P2$ and $P3$ which are then used by $T2$ to meet its deadline (Fig. 2(b)).

The first challenge: "*when* to duplicate" exists because of our objective of meeting deadlines. In case, the deadline of a task is higher (as for $T1$), there are enough chances to meet it without duplicating any job, which can create more schedule holes for other tasks to use, hence, increasing the schedulability. Interestingly, finding whether a task graph can be scheduled under a certain deadline is an NP-Complete problem [11]. Here, we make use of *tentative scheduling* which refers to temporarily scheduling the jobs of the task graph without duplication on processors, to evaluate the upper bound on the makespan. If the upper bound meets the deadline, then the temporary schedule for that task becomes the final schedule, otherwise it is removed. This upper bound approach has been proposed in RTCDA-W^2H heuristic to implement the "when to duplicate" challenge. To further enhance the performance, if we decide to perform duplication in the first step, our motive is to control the amount of duplication according to the deadline, which is the next challenge of finding *how much* duplication is required. The proposed RTCDA-W^2H^2 extends RTCDA-W^2H with steps to control the amount of duplication to propose a controlled duplication algorithm. Hence, RTCDA-W^2H^2 addresses all the four proposed challenges with respect to duplication: **w**here, **h**ow, **w**hen and **h**ow much (W^2H^2). Further, we propose RTCDA-*Extended*, which is based on Mixed Integer Programming (MIP), and a search and repair method based extension of RTCDA-W^2H.

Next, we discuss the related work (Section 3) followed by the assumptions and the system model in Section 4. Algorithms RTCDA-W^2H and RTCDA-W^2H^2 are described in Sections 5 and 6 respectively. Time complexities of the algorithms are described in Section 7. Simulation results with a discussion are presented in Section 8. Finally, Section 9 concludes the paper with possible future directions.

3 Related work

Researchers have focused on developing heuristics driven by specific Quality of Service (QoS) parameters such as Reliability [21], Fault-Tolerance [15] and Security [22] [24]. Qin et al. [15] have presented two dynamic list scheduling algorithms: DASAP (Dynamic AS early As Possible) and DALAP (Dynamic As Late As Possible) for scheduling task graphs. Stavrinides et al. [19] demonstrate a dynamic, list-based scheduling algorithm with a bin-packing heuristic. It has

been reported that exploiting schedule holes with bin-packing (First Fit, Best Fit and Worst Fit) significantly improves the success ratio. Dave et al. [9] have used a cluster-based algorithm named COSYN (CO-SYNthesis of Hardware-Software) which is not only able to schedule the tasks, but also find an optimal hardware-software architecture which involves the selection of processors, FPGAs, ASICs and communication links.

S Ranaweera et al. in [16] used duplication for enhancing the schedulability of periodic time critical applications for pipelined execution on heterogeneous systems. Auluck et al. in [3, 2] proposed algorithms which are an extension of the original duplication strategy proposed in [4]. The algorithm in [3], named RT-DBA (real-time duplication based algorithm), is the closest to our work. RT-DBA is a low complexity algorithm with a few shortcomings. This has motivated the research in this paper. Firstly, it performs duplication for all the tasks in the task set, which may not be always required, as our motive is not to minimize the makespan, but to meet deadlines. A late deadline can be met without duplication. The over use of duplication can reduce possible schedule holes (created due to precedence delays). These holes can be utilized by the other tasks in the task set to meet their deadlines. Secondly, RT-DBA uses a very static approach for scheduling and does not consider the current processor scheduling load. Lastly, it does not utilize schedule holes for scheduling or duplication. Doing so can help in achieving a better utilization of the computing power. We introduced the idea of controlled duplication in [17]. The initial results of RTCDA were presented with the upper bound evaluated using sequential scheduling of jobs of a task on a single processor. This work enhances RTCDA [17] with an improved upper bound using tentative scheduling and proposes an enhanced version of the EDF algorithm to propose RTCDA-W^2H. In addition, we present one more enhancement, RTCDA-W^2H^2 that addresses the "how much" challenge.

4 System model

The system consists of a set P of m heterogeneous processors and a task set T of n precedence-constrained task graphs. All the processors $p \in P$ are connected with a fully connected, contention free network. It is assumed that the local memory of a processor is used for data exchange between assigned subtasks. A vector of the form $< G(V_i, E_i, \mu_i, c_i), rt(i), pe(i), dl(i) >$ represents a task $t_i \in T$. The first element of the vector is the directed acyclic graph G. The node set V_i represents the jobs s_{ijk} (k is the instance, V_i remains the same during instances) of t_i and the edges in E_i represent the communication between the jobs. An edge $e_{ij} \in E$ represents the communication from node s_{ijk} to node s_{ilk}. A positive weight $\mu_i(j, p_q)$ is associated with node s_{ijk}. This represents its computation cost on processor $p_q \in P$ and the non-negative weight $c_i(j, l)$ associated with edge $e_{ij} \in E$ represents the communication cost from s_{ijk} to s_{ilk}. The elements μ_i and c_i are matrices of the order $v_i \times m$ and $v_i \times v_i$ (v_i is the number of subtasks in task t_i). We further assume that the DAG has single *entry* and *exit* nodes. If a DAG has multiple entry (exit) nodes, then they are connected to zero-cost

Table 1. Mathematical Notations used for Task Parameters

Notation	Task Parameter
T	Task set of independent periodic task graphs (DAGs)
P	Set of available processors
n	Number of tasks in the task set
m	Number of processors
SQ_t & SQ_{st}	Task and subtask schedule queues respectively
i, j, k	Ids used for task, subtask and task instance respectively
q	Processor id
t_i	i^{th} task of T
V_i & v_i	Set & number of subtasks of task t_i
p_q & p_{ub}	q^{th} processor & the processor which gives the upper bound
s_{ijk}	j^{th} subtask of k^{th} instance of task t_i
$s_{i(entry)k}$ and $s_{i(exit)k}$	Entry and exit subtasks of k^{th} instance of task t_i
rt_i, pe_i, dl_i	Release time, period and deadline of task t_i
$c_i(j, l)$	Communication cost from s_{ijk} to s_{ilk} for all k
$\mu_i(j, q)$	Execution cost of s_{ijk} on p_q for all k
$\overline{\mu_i(j)}$	Average execution cost of s_{ijk} for all k
$bl(s_{ijk})$ & $sl(s_{ijk})$	b-level and s-level of s_{ijk}
$aft(s_{ijk})$	Actual minimum finish time of s_{ijk} after it is scheduled
$pred(s_{ijk})$ & $succ(s_{ijk})$	Predecessors and successors of s_{ijk}
$P_z(s_{ijk})$	Set of processors on which s_{ijk} is scheduled
$H_h^S(p_q)$ & $H_h^F(p_q)$	H_h^S&H_h^F are the start and finish times of a hole between s_h & s_{h+1}, where $s_1, s_2, \cdots, s_h$ are the subtasks already scheduled on p_q

pseudo entry (exit) nodes with zero-cost edges. Performing this operation does not affect the final schedule. Next, $rt(i)$ is the release time of the task t_i and $pe(i)$ represents its period. Hence, each task graph has an instance after every pe time units. The release time, $rt(ik)$ of the k^{th} task instance of t_i is evaluated as $rt(i) + (k-1) * pe(i)$. The deadline $dl(i)$ is the relative end-to-end deadline of the task t_i, i.e., the exit node $s_{i(exit)k}$ of the k^{th} invocation of task t_i should finish by the absolute time $dl(ik) = rt(ik) + dl(i)$, where $dl(ik)$ is the deadline of the k^{th} invocation of the task t_i.

5 RTCDA-W^{2}H: "when to Duplicate"

5.1 RTCDA-W^{2}H Concept

RTCDA-W^2H (Algorithm 1) proposes a solution to the challenge of "when to duplicate". Notations and mathematical equations used in RTCDA-W^2H are described in Tables 1 and 2 respectively. The central idea of the algorithm is to

Table 2. Mathematical Equations for RTCDA and Subtask Parameters

(1)	**Release time of k^{th} instance of t_i** $\quad rt(ik) = rt(i) \ + \ (k-1) \ \times \ pe(i)$
(2)	**b-level and s-level of s_{ijk}** $bl(s_{i(exit)k}) = \overline{\mu_i(exit)}$ $sl(s_{i(entry)k}) = zero$ $bl(s_{ijk}) = \overline{\mu_i(j)} + \max\limits_{s_{ilk} \in succ(s_{ijk})} \big(c_i(j,l) + bl_i(s_{ilk})\big)$ $sl(s_{ijk}) = \max\limits_{s_{ilk} \in pred(s_{ijk})} \big(c_i(l,j) + sl_i(s_{ilk}) + \overline{\mu_i(l)}\big)$
(3)	**Data Arrival Time (DAT) of $s_{i(entry)k}$ and s_{ijk} (from its predecessors) on p_q** $DAT(s_{ijk}, p_q) = \max\limits_{s_{ilk} \in pred(s_{ijk})} \Big(\min\limits_{p \in P_z(s_{ilk})} \Big(\underset{if \ p=p_q}{\text{EFT}}(s_{ilk}, p) \ \Big\| \ \underset{if \ p \neq p_q}{\text{EFT}}(s_{ilk}, p) + c_i(l,j) \Big) \Big)$ $DAT(s_{i(entry)k}, p_q) = rt(ik)$
(4)	**Schedule hole (SH) on p_q, where s_{ijk}** can be scheduled $SH(s_{ijk}, p_q) = H_h^S \ if \ \big[H_h^F - max\big(DAT(s_{ijk}, p_q), H_h^S\big)\big] > \mu_i(j,q)$
(5)	**Earliest Finish Time (EFT) of s_{ijk} on p_q** $EFT(s_{ijk}, p_q) = max\big(DAT(s_{ijk}, p_q), H_h^S\big) + \mu_i(j,q)$, where $H_h^S = SH(s_{ijk}, p_q)$
(6)	**Earliest Finish Time (EFT) of s_{ijk}** $EFT(s_{ijk}) = \min\limits_{p_q \in P} EFT(s_{ijk}, p_q)$
(7)	**Earliest Tentative Start Time (ETST) of s_{ijk} on p_{ub}** $ETST(s_{ijk}, p_{ub}) = max\bigcap\big(DAT(s_{ijk}, p_{ub}), H_h^S\big)$, where $H_h^S = SH(s_{ijk}, p_{ub})$ and $P_z(s_{ijk}) = p_{ub}$ for all $s_{ijk} \in V_i$

evaluate a "without duplication" upper bound (UB) (step 5, algorithm 1) on the makespan of every task instance using tentative scheduling. If the deadline

Algorithm 1: RTCDA-W^2H pseudocode

Data: Task Set T
Result: Return *true* if task set meets deadlines otherwise return $false$.
Schedule of Tasks on processors

```
1  Evaluate hyperperiod (hp);
2  Maintain task schedule queue (SQ_t) of all instances of tasks upto hp in T;
3  while SQ_t is non empty do
4      Fetch the higher priority ready task instance (t_ik);
5      Evaluate Upper_Bound (UB^ik) by calling
       RTCDA-W^2H-Sched(t_ik, false, UB^ik);
6      if dl(ik) ≤ UB^ik then
7          Make tentative schedule from step 5 as the final schedule;
8      else
9          Schedule task graph with duplication, call
           RTCDA-W^2H-Sched(t_ik, true, UB^ik);
10         if dl(ik) > UB^ik then Scheduling task set failed, return false;
11 return true;
```

of that instance is greater than or equal to the UB (step 6), then the tentative schedule obtained while evaluating the upper bound in step 5 becomes the final schedule (step 7), otherwise the task instance is scheduled with duplication (step 8). RTCDA-W^2H is a combination of list-based and duplication scheduling heuristics. It begins by calculating the hyper period (hp) of the task set T. The hp is evaluated as the least common multiple of all tasks periods. The schedule is generated from time unit $zero$ till hp (steps 1-2). The same schedule is repeated after hp. Each task t_i has $hp/pe(i)$ number of instances in the generated schedule. The separate priority schemes for tasks $t_i \in T$ and subtasks $s_{ijk} \in V_i$ are used to generate schedule queues (step 2). These priority schemes direct RTCDA-W^2H to select a task instance among all tasks and a further ordering of subtasks for allocation to the processing elements (section 5.2). A task t_{ik}, is fetched from the head of SQ_t for processing. The next step is to evaluate the UB using tentative scheduling (step 5) and scheduling with duplication if the UB does not meet the deadline (section 5.3). Algorithm 2 is used for both "without duplication" and "duplication" scheduling by setting the *dupl* parameter to *false* and *true* respectively.

5.2 Assigning Priorities

The tasks in the task set are considered for scheduling, one at a time and are prioritized according to a modified version of the *earliest deadline first* (EDF) algorithm [14]. Since the information of all the tasks is available in advance, the task schedule queue SQ_t is generated before the actual scheduling. The k^{th} task instance of a task i is given higher priority than the l^{th} instance of task j if the deadline of the former task $dl(ik)$ is lesser than the latter i.e., $dl(jl)$, irrespective of their release times, which is not the case in the original EDF. In EDF, a task starts its execution after it is released (if a processor is available) and is preempted if another task with a lesser deadline arrives. However, as our algorithm is non-preemptive, we assign a higher priority to a task which is released later but has a lesser deadline. If two task instances have the same deadline, then the ties are broken by assigning a higher priority to the instance with the earlier release time $rt(ik)$ (equation 1, Table 2).

After the selection of a task instance t_{ik} for scheduling, all the subtasks s_{ijk} of t_{ik} are inserted in the subtask schedule queue (SQ_{st}) according to a non-increasing order of their *b-level* ($bl(s_{ijk})$) values. The ties are broken using s-level ($sl(s_{ijk})$) values. The b-level (s-level) stands for the bottom (start) level, which is evaluated recursively in a bottom-up (top-down) fashion, traversing the task graph starting from the exit (entry) node as shown in equation 2 in Table 2 (step 1, algorithm 2).

In the equations above, $bl(s_{i(exit)k}) = \overline{\mu_i(exit)}$ and $sl(s_{i(entry)k}) = zero$, whereas $succ(s_{ijk})$ and $pred(s_{ijk})$ is the list of immediate successors and predecessors of s_{ijk} and $\overline{\mu_i(j)}$ represents the average execution cost of subtask s_{ijk}. The $bl(s_{ijk})$ value is the critical path from the subtask s_{ijk} to $s_{i(exit)k}$. We have used $b-level$ as the primary priority parameter because the critical path based algorithms are known to generate better schedules. Secondly, sl is the distance

of a subtask s_{ijk} from $s_{i(entry)k}$. The subtask with lower $s-level$ is given higher priority, as it is present at a higher level in the task graph. It is worth noting that sorting the nodes according to $b-level$ also performs a topological sort on all the $s_{ijk} \in V_i$, which satisfies the precedence constraints.

5.3 Scheduling a task instance with an upper bound

We define an *upper bound* (UB^{ik}) for every k^{th} invocation of task t_i. The UB^{ik} is the time up to which the task instance t_{ik} can be scheduled without duplicating any of its jobs. This bound is computed at run time considering the release time $rt(ik)$ of t_{ik}. One straightforward way to evaluate the upper bound is to use an already proposed "without duplication" heuristic to schedule a DAG on heterogeneous multiprocessors. The heuristic will tentatively schedule all the jobs of the task graph instance t_{ik} by considering the current load on all the processors. Here, the upper bound is the actual finish time, $aft(s_{i(exit)k})$, of the exit job $s_{i(exit)k}$. A well known "without duplication" low complexity insertion based heuristic is Heterogeneous Earliest Finish Time (HEFT) [20]. HEFT greedily allocates jobs to processors that give the earliest finish time. HEFT is very effective for scheduling applications with low communication costs. However, in our experiments, we observed that as the communication cost among jobs increases, the greedy approach of HEFT tends to generate schedule lengths even greater than the sequential schedules or the trivial upper bound ($TUB(ik)$). The $TUB(ik)$ for a task instance t_{ik} is defined as the minimum schedule length when all the jobs of t_{ik} are scheduled on a single processor. Again, we use tentative scheduling to find $TUB(ik)$. The processor which gives the $TUB(ik)$ is called the upper bound processor p_{ub}. RTCDA-W^2H uses a modified version of HEFT (Algorithm 2) that generates schedules with a worst case length of $TUB(ik)$ i.e., $UB^{ik} \leq TUB(ik)$.

The subtasks $s_{ijk} \in V_i$ are inserted into SQ_{st} (step 2) for processing according to their b-level ($bl(s_{ijk})$) and s-level (sl) values (step 1). Before scheduling, RTCDA-W^2H calculates the $TUB(ik)$ and tentative earliest start time of all $s_{ijk} \in V_i$ (equation 7, Table 2) on processor p_{ub} if they execute according to their order in SQ_{st} on processor p_{ub} (step 3). The p_{ub} represents the processor on which the current task instance t_{ik} has the UB^{ik}. A subtask s_{ijk} is fetched from the head of SQ_{st} till all the jobs are processed (steps 4 and 5). Next, the algorithm finds the earliest finish time of s_{ijk} with or without duplication, as decided by the input parameter *dupl* using algorithm 3 (step 6 of algorithm 2). Algorithm 3 is called to evaluate EFT of s_{ijk}, where boolean parameter *dupl* decides whether to duplicate jobs while calculating EFT or not. The parameter p_z in a call to Algorithm 3 stores the processor that gives the $EFT(s_{ijk})$.

Job s_{ijk} is tentatively scheduled on all the processors and p_z is set to the processor which gives that minimum finish time (steps 2 and 6, algorithm 3). Equation 5 in Table 2 describes the evaluation of $EFT(s_{ijk}, p_q)$ on a particular processor p_q. Since RTCDA-W^2H is an insertion based algorithm, we look for an earliest available schedule hole of minimum size equal to the execution cost of s_{ijk} on p_q i.e., $\mu_i(j,q)$ which can accommodate s_{ijk}. The start time of this hole

Algorithm 2: RTCDA-W^2H-Sched(t_{ik},$dupl$,UB^{ik})

Data: *Task instance : t_{ik}, bool dupl, UB,Processor p_{ub}*
Result: *Schedule of task t_{ik}*

1 *Evaluate $sl(s_{i(exit)k})$ and $bl(s_{i(entry)k})$ of t_{ik};*
2 *Insert the subtasks $s_{ijk} \in V_i$ in SQ_{st} in non-increasing values of $bl(s_{ijk})$, breaking ties in non-decreasing values of $sl(s_{ijk})$;*
3 *Evaluate $TUB(ik)(t_{ik}, p_{ub})$ and $ETST(s_{ijk}, p_{ub})$ for all $s_{ijk} \in V_i$ following their order in SQ_{st};*
4 **while** *there are unscheduled subtasks in SQ_{st}* **do**
5 *fetch the subtask s_{ijk} from the head of SQ_{st};*
6 *Find $EFT_{s_{ijk}} \leftarrow EFT(s_{ijk}, dupl, p_z)$;* `// processor` p_z `gives EFT`;
7 **if** $p_z \neq p_{ub}$ **then**
8 $shift = true$;
9 **foreach** $s_{ilk} \in succ(s_{ijk})$ **do**
10 **if** $EFT_{s_{ijk}} + c_i(j,l) > ETST(s_{ijk}, p_{ub})$ **then** *shift* $\leftarrow$ *false;*
11 **if** *shift* **then**
12 *Schedule($s_{ijk}, p_z, dupl$);*
13 **else** *Schedule($s_{ijk}, p_{ub}, dupl$);*
14 $UB^{ik} = aft(s_{i(exit)k})$;

Algorithm 3: EFT(s_{ijk},$dupl$,p_z)

Data: *Subtask : s_{ijk}, duplication (true/false): dupl, Processor: p_z*
Result: *Returns Earliest Finish Time of s_{ijk}, p_z store the processor on which s_{ijk} has EFT*

1 *EFT* $\leftarrow$ *INFTY;*
2 **foreach** $p_q \in P$ **do**
3 *Temp* $\leftarrow$ *EFT(s_{ijk},p_q);*
4 **if** *dupl* **then**
5 *Perform duplication (in schedule holes) of immediate predecessors in the order that they delay s_{ilk} the most, if it improve Temp;*
6 **if** *Temp* $<$ *EFT* **then** $p_z \leftarrow p_q$; *EFT* $\leftarrow$ *Temp;*
7 *return EFT;*

H_h^S should be greater than the data arrival time of s_{ijk} from its predecessors on p_q (equations $3-5$ in Table 2).

During the evaluation of $EFT(s_{ijk})$ (algorithm 3), we look for the possibility of duplicating predecessors of s_{ijk} if it improves the $EFT(s_{ijk})$ on p_z (steps 4-6). RTCDA-W^2H is flexible in performing duplication. Here, we allow duplication of immediate predecessors only and the predecessors are selected for duplication according to a non-increasing order of the time that they delay s_{ijk}. Duplication of a job is only performed if it improves the $EFT(s_{ijk})$.

In case the processor on which s_{ijk} has the earliest finish time is the same as that of p_{ub}, s_{ijk} is scheduled on p_{ub} (step 13, algorithm 2). In the other scenario,

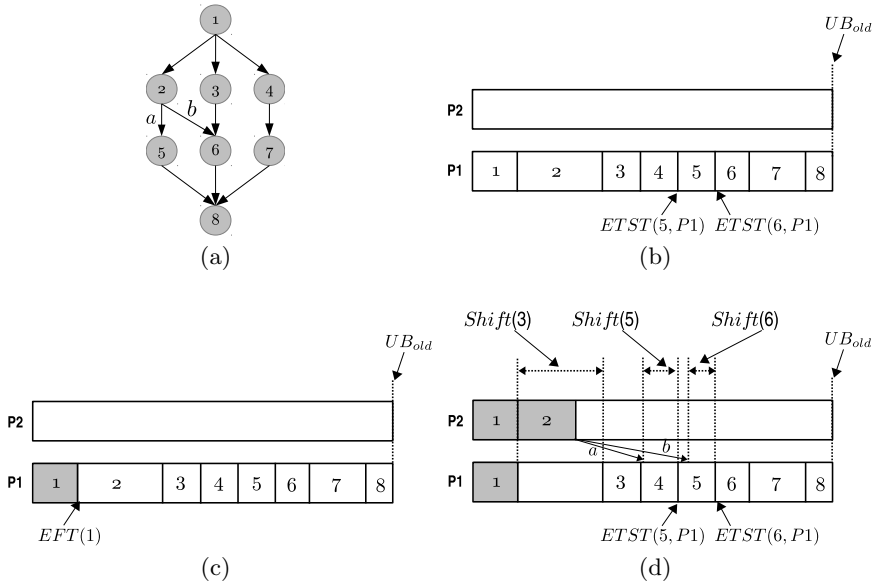


Fig. 3. (a) Example DAG showing precedence among jobs (b) Tentative schedule of example DAG on upper bound processor P1 (c) Job 1 scheduled on P1 (d) Job 2 has EFT on P2. Duplication of Job 1 leads to updation in the tentative schedule of remaining jobs and hence of UB_{old}

RTCDA-W^2H makes sure that scheduling s_{ijk} on any processor p_z other than p_{ub} does not increase the worst case schedule length i.e., $TUB(ik)$ by satisfying the following condition for all $s_{ilk} \in succ(s_{ijk})$ (steps 8-13, algorithm 2).

$$\textbf{Condition-1: } EFT_{s_{ijk}} + c_i(j,l) \leq ETST(s_{ilk}, p_{ub})$$

If a subtask s_{ijk} satisfies the above equation, then it is scheduled on p_z, other wise on p_{ub}. Subroutine *Schedule* (steps 12 and 13, algorithm 2) has a similar pseudo code as algorithm 2, except that it schedules the subtask after finding the EFT. Condition-1 is the primary difference with the original HEFT algorithm. Removing this condition will convert RTCDA-W^2H into HEFT. We call this condition "selective duplication", since it does not allow schedule lengths to be greater than the trivial upper bound, $TUB(ik)$. The results show that selective duplication improves the performance of RTCDA-W^2H over the original HEFT algorithm.

6 RTCDA-W^2H^2: How much to Duplicate

RTCDA-W^2H^2 proposes an extension to RTCDA-W^2H (when duplicating) by inculcating an approach to dynamically improve the upper bound UB^{ik} after a subtask is scheduled on a processor other than the upper bound processor p_{ub}. After every update in UB^{ik}, we again determine if $dl_{ik} \geq UB^{ik}$, if it's true,

then the remaining subtasks are scheduled without duplication. Thus, even after deciding that a task instance will be scheduled with duplication, RTCDA-W^2H^2 controls the amount of duplication and hence solves the "how much to duplicate" problem.

The concept of RTCDA-W^2H^2 is elaborated with the scheduling of jobs with precedence relations on two processing elements as depicted in an example DAG shown in Fig. 3(a). All eight jobs of the task graph are assumed to be processed in the order of their values from $1-8$. Let's say processor $P1$ gives the trivial upper bound (UB_{old}) of the task instance as shown in Fig. 3(b). The actual scheduling starts with job 1 and $EFT(job1)$ is evaluated. Let's say Job 1 has an EFT on the upper bound processor $P1$ and is scheduled on $P1$ (refer to 3(c)). The unfilled boxes on P1 represent the tentative schedule where as the grey filled boxes refer to the actual scheduling of jobs. The next job in the schedule queue is job 2. Fig. 3(d) shows a state when job 2 has an EFT on processor $P2$ with the help of a replicated copy of job 1. Job 2 can be scheduled on $P2$, which is not an upper bound processor, only if both of it's successor jobs 5 and 6 satisfy the selective duplication Condition-1, i.e.,

$$EFT(2) + c(2,5) \leq ETST(5, P1)$$
$$EFT(2) + c(2,6) \leq ETST(6, P1)$$

If the above conditions are satisfied, job 2 is scheduled on $P2$. According to the above procedure, RTCDA-W^2H continues with the scheduling of the remaining jobs. However, in RTCDA-W^2H^2, an additional step is performed to update the value of the upper bound. It is observed that with the actual scheduling of job 2 on $P2$, which is not the upper bound processor, three of the unscheduled jobs have been affected in the tentative trivial upper bound schedule on $P1$. Two of these jobs are the successors of job 2 i.e., job 5 and job 6 and the third is the next job in the schedule queue, job 3. We define a set AF of all these affected jobs as follows:

Definition 2. ***Set of Shift Affected Jobs (AF).*** *If a job s_{ijk} is scheduled on a processor other than the upper bound processor, then the set of $succ(s_{ijk})$ and the next job in the schedule queue is defined as the set AF.*

For all the jobs $s_{ilk} \in AF$, we evaluate a parameter $shift(s_{ilk})$ as shown in equation 1.

$$shift(s_{ilk}) = ETST(s_{ilk}, p_{ub}) - DAT(s_{ilk}, p_{ub}) \tag{1}$$

The parameter $shift$ describes the maximum improvement in the $ETST$ of the affected jobs on p_{ub} in the tentative upper bound schedule considering available schedule holes. The overall improvement in the upper bound is evaluated as:

$$min\_shift = \min_{s_{ilk} \in AF} \big(shift(s_{ilk})\big) \tag{2}$$

$$UB^{ik}_{new} = UB^{ik}_{old} - min\_shift \tag{3}$$

RTCDA-W^2H^2 keeps improving the UB^{ik} whenever a job is scheduled on a processor other than p_{ub}. As soon as $dl_{ik} \leq UB^{ik}_{new}$, the remaining jobs are scheduled without duplication, hence, controlling the amount of duplication. The above discussed steps to update UB^{ik} should be added to the *if*-condition at step 11 of Algorithm 2 to implement the required functionality of RTCDA-W^2H^2. Hence, RTCDA-W^2H^2 includes schemes to handle all W^2H^2 challenges.

7 Time Complexity

The time complexity of RTCDA-W^2H^2 and RTCDA-W^2H has been found to be $O(n^2 I^2_{max} v^2_{max} m d_{max})$, where I_{max}, v_{max} and d_{max} are defined as the maximum number of instances of a task in a task set, maximum subtasks in a task and maximum in-degree of a task in a task set respectively. This time complexity is higher than that of RTDBA [3] $O(nI_{max}v^2_{max})$ because both the proposed algorithms are insertion based and take $O(nI_{max}v_{max})$ time in finding a particular hole for scheduling jobs. However, the increase in the time complexity is reflected in the performance of RTCDA-W^2H^2 as described by better results. Since this is a static variation of the scheduling problem, the increase in complexity has been compensated with increasing performance of the heuristics.

For an instance of a task t_{ik}, the algorithm evaluates $bl(s_{ijk})$, $sl(s_{ijk})$ for all the jobs. These two parameters can be found by a breadth-first search on the DAG. It visits every vertex exactly once, so the time taken is $O(v_i)$. RTCDA-W^2H^2 spends a significant amount of time in searching for a valid schedule hole for a subtask on a processor. Therefore, the time required to find a valid hole is proportional to the number of holes present on a processor, which is further equal to the number of jobs already scheduled on it. In the worst case, before scheduling t_n, all of the other tasks and their instances may have finished. Hence, the total number of the scheduled jobs are as given by the equation:

$$\begin{aligned} nholes &\leq \Big(\frac{hp}{p_1} \times v_1 + \frac{hp}{p_2} \times v_2 + \cdots + \frac{hp}{p_{n-1}} \times v_{n-1}\Big) \\ &\leq v_{max}\Big(\frac{hp}{p_1} + \frac{hp}{p_2} + \cdots + \frac{hp}{p_{n-1}}\Big) \\ &\leq v_{max}\Big(I_1 + I_2 + \cdots + I_{n-1}\Big) \\ &\leq nI_{max}v_{max} \end{aligned} \tag{4}$$

In evaluating the upper bound UB of a task, we schedule all jobs on all the processors with schedule holes, which gives $O(nI_{max}v^2_{max}m)$. While scheduling a job, we find the EFT of the job on all the processors. Therefore, scheduling all the jobs takes $O(nI_{max}v^2_{max}m)$ time. Each task duplication also considers the schedule holes between all previously scheduled tasks for the processor. Therefore, each task duplication has a time complexity of $O(nI_{max}v_{max})$. The maximum number of duplications would be d_{max} which gives the complexity $O(nI_{max}v^2_{max}md_{max})$. The controlled duplication step can be done on $O(v_i)$

Table 3. Simulation Parameters

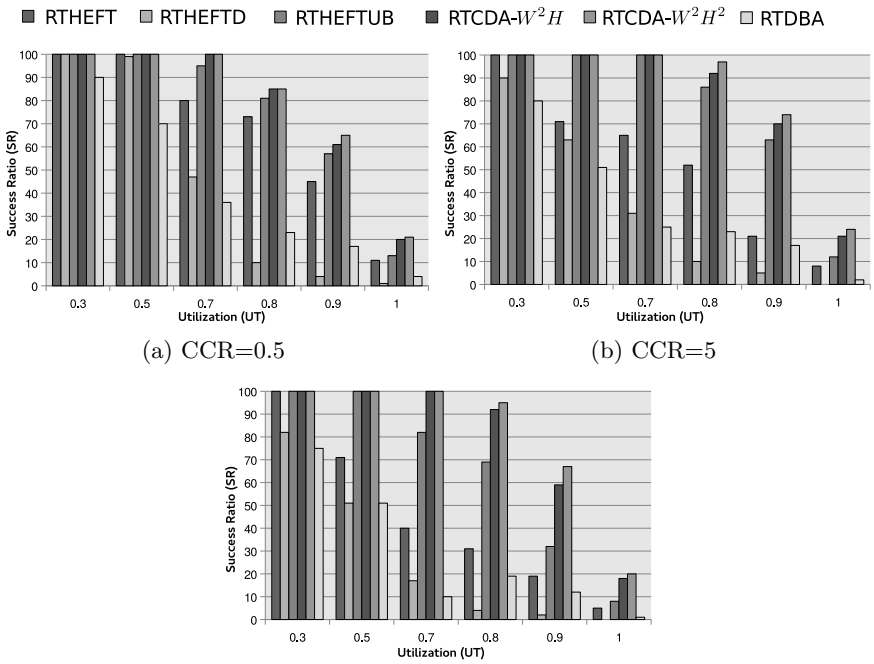
Parameter	**Range**	**Parameter**	**Range**
Number of tasks in a task set	$2-100$	Number of subtasks in a task	$10-2000$
Subtask execution cost	$1-100$	Communication to Computation Ratio (CCR)	$0.5, 1, 5, 10$
Utilization (UT)	$0.3-1.0$	Heterogeneity Factor (HF)	$5-40$

time. Therefore, the total time becomes $O(n^2 I_{max}^2 v_{max}^2 m d_{max})$ for scheduling a maximum of nI_{max} instances.

8 Simulation Results

The proposed algorithms RTCDA-W^2H and RTCDA-W^2H^2 have been compared with RTDBA [3] and three real-time variants of the HEFT scheduling algorithm viz. RTHEFT, RTHEFTD and RTHEFTUB. RTHEFT is the real-time version of HEFT proposed in [20]. This algorithm schedules jobs of every task instance with the earliest finish time heuristic without duplicating any job. RTHEFTD, a "duplication" version of HEFT has been proposed by [5]. This algorithm always uses duplication for scheduling task instances. RTHEFTUB is essentially our RTCDA-W^2H, without doing the selective duplication step proposed in Condition-1 i.e., RTHEFTUB can generate schedules more than the trivial upper bound. The parameters used for the simulation are summarized in Table 3. The number of processors are varied by keeping the ratio $\left(\frac{average\_subtasks\_in\_taskset}{number\_of\_processors}\right)$ as constant [7]. Total utilization UT of a task set is the summation of the utilization of all the tasks in the task set. For a single task, UT is defined as $UT = \frac{average\_computation}{m * period}$, whereas $average\_computation$ is the summation of the averages of jobs execution costs in the task. Parameters CCR, HF are averaged over all the tasks in the task set. CCR of a task is defined as $\frac{total\_communication}{average\_computation}$. HF of a task corresponds to the average standard deviation of the job execution costs. For a job s_{ijk} of a task t_i, it is evaluated as $\sqrt{\sum_{p_q \in P} (\overline{\mu_i(j)} - \mu_i(j, p_q))^2}$. For every combination of CCR and UT, 1000 task sets were generated by uniformly selecting the remaining parameters using a well known real-time benchmark, Task Graphs For Free (TGFF) [10]. The task deadlines have been set equal to their periods. All the algorithms have been implemented in C++. Schedulability or Success Ratio (SR) is used as the primary performance metric.

Definition 3. ***(Success Ratio).*** *It is defined as the ratio of the number of task sets that meet their deadlines to the total number of task sets considered [3] i.e* $SR = \frac{number\_of\_tasksets\_meeting\_deadlines}{total\_number\_of\_tasksets}$.



(a) CCR=0.5

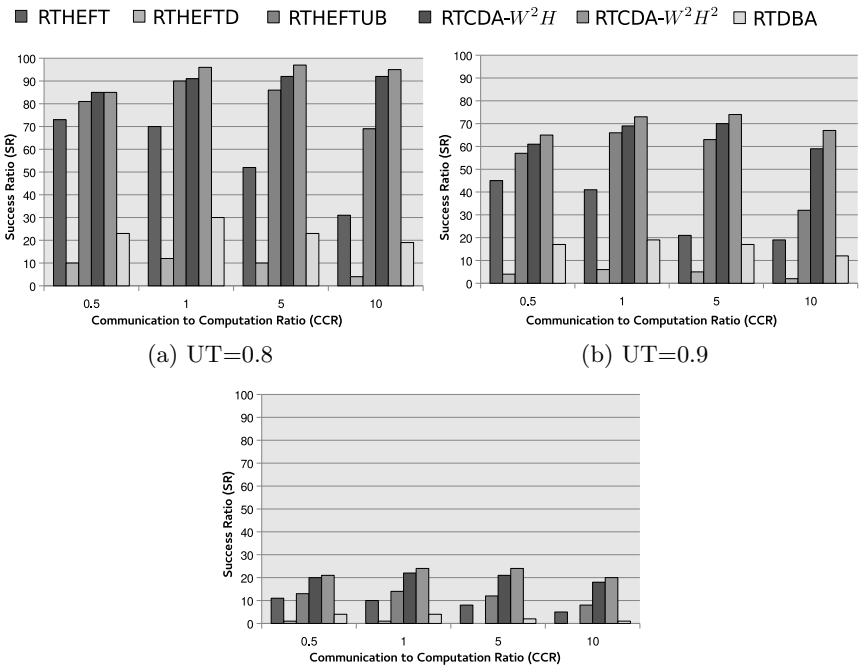
(b) CCR=5

(c) CCR=10

Fig. 4. Effect of UT with fixed CCR

8.1 Effects of CCR and UT

Figs. 4 and 5 show the effect of varying CCR and UT on the algorithms. The results show that RTCDA-W^2H^2 and RTCDA-W^2H improve the SR more than the others in every combination of CCR and UT. Among all algorithms, RTHEFTD and RTDBA achieved the lowest SR values, even lesser than those of RTHEFT, which is a "without duplication" algorithm. The primary reason for this is that they "always" perform duplication of the jobs. Also, RTDBA is not an insertion based algorithm. This reflects in its SR being lower than RTHEFTD. All three proposed upper bound algorithms managed to improve the SR by $> 15\%$ for UT$>= 0.7$ across all CCR values (Figs. 4 and 5). Hence, these algorithms make an efficient use of schedule holes by switching between "without duplication" and "duplication" scheduling algorithms at run time. In Fig. 4(a), for a low CCR of 0.5, "without duplication" RTHEFT scheduled all task sets with UT≤ 0.5. However, for higher utilizations and higher CCR values, upper bound based algorithms improved the SR by performing duplication for the tasks which can not be scheduled without duplication. The gap in SR values of RTHEFT and proposed duplication algorithms increases with increase in CCR (Fig. 5). Importantly, duplications are less effective at a low CCR value $= 0.5$, however, upper bound algorithms first tentatively schedule tasks without duplication and then try duplication, only when the "without duplication" approach fails. This helps in increasing the SR by $10 - 20\%$.



(a) UT=0.8

(b) UT=0.9

(c) UT=1.0

Fig. 5. Effect of CCR with fixed UT

Generally, the SR for all the algorithms decreases with an increase in UT at all CCRs. This scenario is a combined effect of increasing the demand of computing power with an increase in UT and the delay caused by communication costs. However, duplications helps in achieving an SR close to 70% for UT=0.9. At maximum UT=1.0, all the algorithms perform poorly. However, RTCDA based algorithms are able to schedule 20% of the task sets. The UT=1.0 describes a case when the CPU is 100% utilized. However, heterogeneity in the computation costs help scheduling 20% of the task sets (refer to Section 8.2 for details). RTCDA-W^2H and RTCDA-W^2H^2 have performed slightly better than RTHEFTUB by making use of selective duplication that bounds their schedule lengths to the trivial upper bound at higher CCR values. The major difference in their SR can be seen at CCR value = 10 across different utilizations (Fig. 5), due to RTHEFTUB generating schedules larger than the trivial upper bound. RTCDA-W^2H^2 is able to schedule $5-10\%$ more task sets than RTCDA-W^2H due to a control in the amount of duplication while scheduling.

8.2 Effects of Heterogeneity

To study the effect of heterogeneity, task sets are generated by keeping CCR and UT fixed to 1.0 and 0.8 respectively and by varying the execution costs in a range of $1-50$. The parameter HF is varied from 1 to 20 as shown in Fig.

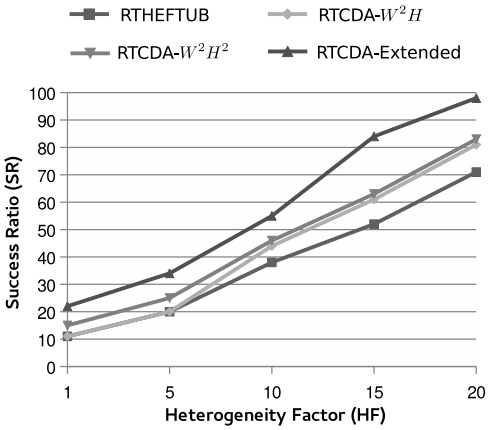


Fig. 6. Effect of Heterogeneity

6. A low value of HF=1, depicts less variation in the execution costs of jobs on multiprocessors. Hence, for a UT=0.8, a task set will require approximately 80% of the total CPU computing power to meet all deadlines, because each job will execute almost for its average execution cost. The remaining 20% of the computing power is only utilized by duplicated copies of jobs to reduce the delay caused by the higher communication cost. Hence, more jobs are delayed. This reduces the success ratio. With the increase in HF, the SR has been found to increase, because more variation in execution costs causes jobs to schedule on processors with execution cost less than their averages hence, providing more computing power for duplication. RTCDA-Extended is able to improve the SR more than the other variations in this case as well.

9 Conclusion

We have observed that duplication is not always required for the scheduling of real-time static tasks. Whether to duplicate or not depends on the task deadlines. In addition, the controlled duplication strategy has addressed the W^2H^2 duplication challenges. Increasing the simulation time using RTMIP and local search techniques further improves the success ratio by $> 20\%$ for a maximum utilization of 1.0. In the future work, we will look to decrease the time complexity of the algorithms. Also, energy consumption of computation and communication resources can be optimized for the cases where 100% SR is achieved.

References

1. Ahmad, I., Kwok, Y.K.: On exploiting task duplication in parallel program scheduling. IEEE Trans. Parallel Distrib. Syst. 9, 872–892 (September 1998)
2. Auluck, N.: An integrated scheduling algorithm for precedence constrained hard and soft Real-Time tasks on heterogeneous multiprocessors. In: Lecture totes in Computer Science. vol. Volume 3207/2004, pp. 199–207 (2004)

3. Auluck, N., Agrawal, D.: Enhancing the schedulability of Real-Time heterogeneous networks of workstations (NOWs). Parallel and Distributed Systems, IEEE Transactions on 20(11), 1586–1599 (2009)
4. Bajaj, R., Agrawal, D.P.: Improving scheduling of tasks in a heterogeneous environment. IEEE Trans. Parallel Distrib. Syst. 15, 107–118 (February 2004)
5. Bansal, S., Kumar, P., Singh, K.: Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs. J. Parallel Distrib. Comput. 65, 479–491 (April 2005)
6. Baskiyar, S., Dickinson, C.: Scheduling directed a-cyclic task graphs on a bounded set of heterogeneous processors using task duplication. J. Parallel Distrib. Comput. 65, 911–921 (August 2005)
7. Davare, A., Chong, J., Zhu, Q., Densmore, D.M., Sangiovanni-Vincentelli, A.L.: Classification, customization, and characterization: Using MILP for task allocation and scheduling. Tech. Rep. UCB/EECS-2006-166, EECS Department, University of California, Berkeley (Dec 2006)
8. Dave, B., Jha, N.: COFTA: hardware-software co-synthesis of heterogeneous distributed embedded systems for low overhead fault tolerance. Computers, IEEE Transactions on 48(4), 417–441 (1999)
9. Dave, B., Lakshminarayana, G., Jha, N.: COSYN: hardware-software co-synthesis of heterogeneous distributed embedded systems. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 7(1), 92–104 (1999)
10. Dick, R.P., Rhodes, D.L., Wolf, W.: Tgff: Task graphs for free (1998)
11. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences). W. H. Freeman, first edn. (Jan 1979)
12. Kwok, Y.K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. 31, 406–471 (December 1999)
13. Liu, C., Anderson, J.: Supporting graph-based real-time applications in distributed systems. In: Proceedings - 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2011. vol. 1, pp. 143–152 (2011)
14. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a Hard-Real-Time environment. Journal of the ACM (JACM) 20(1 (January 1973)), 46 – 61 (1973)
15. Qin, X., Jiang, H.: A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. Journal of Parallel and Distributed Computing 65(8), 885–900 (Aug 2005)
16. Ranaweera, S., Agrawal, D.P.: Scheduling of periodic time critical applications for pipelined execution on heterogeneous systems. In: Parallel Processing, International Conference on. p. 0131. Los Alamitos, CA, USA (2001)
17. Singh, J., Auluck, N.: Controlled duplication for scheduling real-time precedence tasks on heterogeneous multiprocessors. In: High Performance Computing (HiPC11) Student Research Symposium. Bangalore, India (Dec 2011)
18. Singh, J., Betha, S., Mangipudi, B., Auluck, N.: Contention aware energy efficient scheduling on heterogeneous multiprocessors. IEEE Transactions on Parallel and Distributed Systems Early Access Online (2014), 00000
19. Stavrinides, G.L., Karatza, H.D.: Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques. Simulation Modelling Practice and Theory 19(1), 540–552 (Jan 2011)

20. Topcuouglu, H., Hariri, S., Wu, M.y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. 13, 260–274 (March 2002)
21. Tosun, S.: Energy and reliability-aware task scheduling onto heterogeneous MPSoC architectures. The Journal of Supercomputing pp. 1–25 (Nov 2011)
22. Xie, T., Qin, X.: Security-Aware resource allocation for Real-Time parallel jobs on homogeneous and heterogeneous clusters. Parallel and Distributed Systems, IEEE Transactions on 19(5), 682–697 (2008)
23. YuHai, Y., Shengsheng, Y., XueLian, B.: A new dynamic scheduling algorithm for Real-Time heterogeneous multiprocessor systems. In: Workshop on Intelligent Information Technology Application (IITA 2007). pp. 112–115. Zhang Jiajie, China (2007)
24. Zhu, X., Lu, P.: Multi-Dimensional scheduling for Real-Time tasks on heterogeneous clusters. Journal of Computer Science and Technology 24(3), 434–446 (2009)