Dynamic Real-Time Task Scheduling on Hypercubes[1]

Prasant Mohapatra
Department of Electrical and Computer Engineering
201 Coover Hall
Iowa State University
Ames, IA 50011
E.mail: *prasant@iastate.edu*

Abstract

Multiprocesssor systems have emerged as an important computing means for real-time applications and have received increasing attention than before. However, until now, little research has been done on the problem of on-line scheduling of parallel tasks with deadlines in partitionable multiprocessor systems. Meshes and hypercubes belong to this class of multiprocessors. In this paper, we propose a new on-line scheduling algorithm, called Deferred Earliest Deadline First (DEDF) for hypercube systems. The main idea of the DEDF algorithm is to defer the scheduling as late as possible, so that a set of jobs is scheduled at a time instead of one at a time. For processor allocation using DEDF, we have introduced a new concept - Available Time Window (ATW). By using ATW, the system utilization can be improved and thereby the deadlines of more number of tasks can be met. Simulation results indicate that the DEDF algorithm performs significantly better than the earlier proposed Buddy/RT and Stacking algorithms for a wide range of workloads.

Keywords: Available Time Window, Deferred Earliest Deadline First, Hypercube, Processor Allocation, Real-Time Scheduling.

[1]This research was supported in part by the National Science Foundation under the grants MIP-9628801 and CCR-9634547. A preliminary version of this paper was presented as the Symposium of Parallel and Distributed Processing, 1996.

1 Introduction

A real-time system is a system in which correctness of the computation depends not only on the logical correctness of the execution, but also on the timeliness of producing results. To function correctly, it must produce a correct result within a specified time or before its deadline. The system is considered incapable of producing the result if the time constraints are not met.

Scheduling of aperiodic tasks on real-time systems is one of the most difficult problems in the design of real-time systems. Due to their unpredictable nature in terms of arrival time and service demand, it is very difficult to design real-time systems that have a high guaranteed rate of servicing aperiodic tasks successfully. The characteristics of the aperiodic tasks are not known a priori and the scheduling decisions are made upon their arrival. Thus arises the need for on-line or dynamic scheduling of tasks. The functionality of the dynamic scheduler can be described as follows. When a new job arrives, its service demand (execution time and subcube size) and deadline are made known to the dynamic scheduler. The scheduler then decides whether the new job along with all the old jobs can finish their execution before their respective deadlines. If it is not possible then the scheduler tries to accommodate as many jobs as possible based on the specified processor allocation algorithms and task priorities. The jobs that cannot be serviced before its deadline are not accepted by the system. The scheduler must make decision dynamically and quickly so as to have a high ratio of job acceptance and low scheduling overhead.

Multiprocessor system are emerging as potential candidates for use in real-time applications [17]. Hypercubes and mesh-connected multiprocessors are well-suited for use as real-time control units due to their high scalability, reliability and throughput. An additional advantage of these systems is their ability to be partitioned into subsystems each of which can be allocated to a different job. An efficient scheduling algorithm is required for allocating multiple jobs on a partitionable real-time multiprocessors. Dynamic scheduling of real-time tasks in partitionable multiprocessor systems is an ongoing and active research area. In this paper, we propose a new dynamic scheduling scheme for hypercube-based real-time multiprocessor system. This technique involves job scheduling and processor allocation on multiprocessor systems. Determination of an optimal scheduling or allocation scheme is known to be an NP-hard problem [2] [11] [17].

The dynamic scheduling scheme proposed in this paper is based on heuristic is called Deferred Earliest Deadline First (DEDF) algorithm. The main idea of DEDF is to defer the scheduling as late as possible so that the scheduler schedules a set of jobs instead of one at a time. Scheduling a set of jobs reduces fragmentation and preemption of tasks as compared to scheduling one job after another. However, it must be ensured that the tasks meet their deadlines. The DEDF scheme works as follows. Whenever tasks arrive at the system, they are enqueued in a job arrival queue. A few triggering conditions are predetermined that initiate the scheduling. When triggered, the scheduler begins scheduling a set of newly arrived jobs that are waiting in the arrival queue. If the scheduling is successful then the set of newly arrived jobs are allocated to the system and are guaranteed for completion, otherwise optionally the scheduler does preemptive scheduling for all the jobs in the system. The process is repeated whenever the triggering conditions are met. We have also introduced a new concept of the Available Time Window (ATW) for processor allocation using the DEDF algorithm. ATW of a processor is defined as

the available time interval for which that processor is available for use. The system utilization is improved by using the ATW for processor allocation. Both of the new ideas, deferred scheduling and ATW, help in improving the performance of real-time scheduling in hypercube systems. We have evaluated the performance of the proposed algorithm by means of simulation. Job miss ratio and work miss ratio are two parameters that are used as performance measures for the real-time scheduling algorithms. The results indicate that the proposed DEDF algorithm has a significantly lower job/work miss ratio when compared with the previously proposed Buddy/RT and Stacking algorithms [1].

The remainder of this paper is organized as follows. In Section 2, we describe the related works about on-line scheduling algorithms for real-time tasks on partionable multiprocessors. The proposed DEDF algorithm is discussed in Section 3 along with its implementation details. The simulation platform and the results are presented in Section 4, followed by the concluding remarks in Section 5.

2 Preliminaries

The two major aspects of processor management in multiprocessor systems are processor allocation and job scheduling. In a real time system, processor allocation is concerned with the selection of processors on which the tasks executes and the time when the allocated jobs start their execution. The job scheduler decides the order in which jobs are considered for processor allocation. Several schemes for processor allocation for non-real-time parallel tasks have been proposed such as buddy [12], single and multiple gray code [3], free list [6], MSS [2], tree collapsing [4] PC-graph[18], and the partitioned-based scheme [19]. These allocation algorithms vary in terms of their subcube recognition ability and time complexity. The buddy allocation scheme is simple and has low time complexity. In spite of its low subcube recognition ability, it has been shown that the buddy strategy performs well in hypercube systems when compared with the perfect recognition allocation schemes [9]. The scheduling schemes proposed for non-real time hypercubes include scan [9], and lazy [10]. Krishnamurti and Narahari have proposed a preemptive scheduling scheme for non-real-time, partitionable parallel architectures [8]. Hong and Leung have proposed an on-line scheduling algorithm for real-time tasks with one common deadline [5]. Sahni has shown that when the common deadline is more than 1, no optimal nearly on-line scheduler (and hence, on-line scheduler) can exist for task systems with several distinct deadlines [14]. However, the work reported on on-line scheduling of real-time tasks on partitionable multiprocessors have been limited.

2.1 Dynamic Task Scheduling

A job J_i is characterized by its time of arrival A_i; its execution time E_i; its deadline D_i; and Dim_i, the dimension of the subcube required by the job. We assume that all these parameters are made known to the scheduler when the job arrives. A job set K consists of k independent jobs to be scheduled. Job set K is feasible on an n-dimensional hypercube if there is a preemptive schedule for the jobs in K such that each job J_i is executed during its execution interval $[A_i,$

D_i]. Such a schedule is called a feasible schedule. Job scheduling algorithms based on simple heuristics such as Earliest Deadline First (EDF) and Minimum Laxity First (MLF) are often used in uniprocessor systems [11]. Under EDF, jobs are scheduled in ascending order according to their deadlines. The job with the earliest deadline gets the highest priority. On the other hand, a scheduler based on MLF schedules jobs in ascending order according to their laxities, where laxity of a job is the time difference between the current time and the Latest Start Time (LST) of the job. The LST of a job is the latest time by which the job must be started in order to be finished before its deadline. It has been shown that EDF and MLF are optimal for uniprocessor systems with independent preemptable tasks [11]. While these heuristics are suitable for uniprocessor systems, they are not feasible for multiprocessor systems due to the large search space of the feasible schedules.

2.2 Related Scheduling Algorithms for Hypercubes

Efficient algorithms for scheduling a set of tasks on real-time multiprocessors are proposed by Ramamritham et al. [13]. These algorithms include contention of resources and data structures. They have not considered partitionable multiprocessors where the processor allocation schemes also play a significant role in real-time scheduling. Next, we describe the two on-line scheduling algorithms proposed so far for real-time partitionable multiprocessors. Both of these schemes are based on heuristics and are applicable to hypercube systems.

The Buddy/RT processor allocation algorithm for real-time systems proposed by Babbar and Krueger [1] is an extension of the buddy scheme proposed earlier for memory systems [12] and processor allocations [3]. The basic idea of the scheme is to include an additional restriction - *time*, while doing allocation. The Earliest Available Time (EAT) of a processor is defined as the earliest time at which that processor will become available for allocation. Instead of an array of bits (as used in the conventional buddy system), an array of EATs is used in the Buddy/RT scheme. The Buddy/RT algorithm has two phases where the second phase is optional. All the jobs are tried for scheduling on their arrival based on the FCFS discipline. For a job J_i, requesting a subcube of dimension k within a hypercube of dimension n, the Buddy/RT algorithm tries to find the smallest integer j, $0 \leq j \leq 2^{n-k} - 1$, such that all the processors in the subcube $\{j2^k,\ (j+1)2^k - 1\}$ have EATs earlier than the LST of job J_i, and allocates these processors to job J_i. If no such j exists, no subcube can complete job J_i before its deadline. In this case, the scheduler fails in the first phase. The decision to go to the second phase is based on the amount of time available before the LST of all the jobs. If the available time is more than the worst-case execution time of the second phase, the second phase is initiated, otherwise the job is rejected. In the second phase, the scheduler tries to reschedule all the jobs - both the currently allocated ones and the newly arrived one.

Coalesces and splits while using Buddy/RT algorithm increase the fragmentation and thereby limit the performance. Coalesces lead to *holes* in the schedule and splits hurt the ability to schedule future jobs which need larger subcubes. The Stacking algorithm tries to minimize fragmentation by avoiding unnecessary coalesces and splits of subcubes [1]. The approach used by the Stacking algorithm is to stack equal-sized jobs. Instead of choosing the first available subcube that allows for the job deadline to be met, the Stacking algorithm chooses the earliest available

subcube that allows the deadline to be met from among those that require the least number of splits or coalesces. The Stacking algorithm also has two phases as that of the Buddy/RT algorithm. In addition to the EAT entry of every processor, the Stacking algorithm maintains the EATs at the subcube level. The dimension of the job last allocated is also recorded for all the nodes which enables in stacking of equal-sized jobs. In [1], the authors have shown that stacking performs better than that of the Buddy/RT algorithm.

3 Deferred Earliest Deadline First (DEDF) Scheduling

Although Buddy/RT and Stacking algorithms have good subcube allocation ability, they have several shortcomings. First, since these algorithm schedule the jobs as soon as they arrive, the scheduling overhead is high. It is even worse when they get into the second phase. Second, in the allocation scheme, whenever they find a free subcube or a subcube which has the EAT earlier than the LST of the newly arrived job, they scheduled the new job on that subcube. This kind of greedy scheme could create holes and splits that affect the later allocations. Third, the use of EAT may leave a subcube idling while jobs of the corresponding size are being rejected. The idling time may be enough for the completion of the rejected jobs. This problem is discussed in the form of an example in the following subsection. The proposed DEDF algorithm avoids all these weaknesses and thus exhibits better performance in terms of lower job/work miss ratio.

3.1 DEDF Algorithm

The main idea of the DEDF algorithm is the deferment of the scheduling as late as possible, so that the scheduler schedules a set of jobs as opposed to a single job as is done while using the Buddy/RT and Stacking algorithms. It should be ensured that the deferment does not becomes the cause of missing the deadline for any task. Deferred scheduling may accumulate more than one job for allocation and scheduling at the same time. By providing more number of jobs to be scheduled, we can have an optimal allocation for a specific set within the available system constraints. This idea has been motivated by the fact that, for the feasible job sets, the static (or off-line) real-time scheduling can be optimal, in which the characteristics of all the tasks are known a priori. Thus, the static real-time scheduling can be performed on a given set of jobs to improve the performance of on-line scheduling. In addition to the processors being allocated for immediate use, subcubes are also allowed to be reserved for future use.

In DEDF, the scheduler will not start the scheduling until some triggering conditions are met. Determination of the triggering conditions is very important and must be selected carefully. There are two important factors that are taken into consideration while deciding the triggering conditions. These are Latest Start Time(LST) of jobs to be scheduled and the length of the queue for the arriving jobs. We assume that the system has a queue called Arrival Queue (AQ) for all arriving real-time jobs. The scheduler defers the scheduling as long as every job in the AQ has enough slack time to its LST. It may not be wise to wait exactly until the minimum LST of all jobs to trigger the scheduling process as there may not be any free subcube available at that time. If the minimum LST of all jobs is L_t (=min{LST}), and the average subcube hold

time (service time) of the jobs is S_t, then the triggering time t_{start} is equal to the maximum of {current time, $L_t - S_t$}. The heuristic behind this selection of time is that if there are no available subcubes, then a subcube is most likely to be available after the average hold time. It should be noted that this is a pessimistic assumption. At an average, a subcube should be available after one-half of the average hold time. Simulation experiments are conducted to validate the heuristics and have shown good results for a variety of workload conditions.

The scheduling is also triggered when the AQ is full. In a practical implementation, the AQ can accommodate finite number of jobs. So we need to schedule as many jobs as we can (if not all) as soon as the AQ is full. If one of these two conditions (i.e., t_{start} or length of AQ) is met, the scheduler starts to schedule the jobs waiting in the AQ.

For processor allocation with DEDF scheduling algorithm, we have introduced a new concept of the Available Time Window (ATW). ATW of a subcube is defined as the available time interval at which that subcube is available for use. An available time interval is represented by the bottom of the ATW and the top of the ATW. The bottom of a ATW is the time at which the subcube starts to idle, and the top of a ATW is the time at which the subcube is expected to start executing the next reserved job. If there is(are) no reserved job(s) for a subcube, the top of the ATW is said to be open. A subcube could have multiple intervals when it is available for use resulting in multiple ATWs. In such cases, the multiple ATWs are maintained as a linked list and all of them are checked for possible allocations. We have shown in Section 4 that the ability of available subcube recognition using ATW is better than that of Buddy/RT and Stacking that use EAT for processor allocation. Furthermore, the use of ATW guarantees that a subcube will not idle when jobs of the same size are being rejected and could have been completed during the idling period.

The scheduling of new jobs are deferred until one of the triggering conditions is met. In the first phase, the scheduler tries to schedule all new jobs in the AQ in the order of their deadline, i.e., Earliest Deadline First (EDF). Once a new job is ready to be scheduled, the scheduler performs processor allocation for that job based on the ATW. If it finds an appropriate subcube, it is either allocated or reserved for the job. It should be noted that several jobs can reserve the same busy subcube as long as their deadlines can be met. This scenario is different from the non-real-time parallel task allocation. If the scheduler does not find the required subcube, it is said to have failed in the first phase. If enough time for the second phase is left, it enters the second phase. This time is computed based on the amount of time left before the LST of all the jobs (L_t). Since the second phase involves preemption, the overhead due to preemption is also taken into account while deciding whether or not to initiate the second phase. The allocation procedure of the second phase is almost the same as that of the first phase except that the job set to be scheduled includes all the jobs in the system, i.e., all currently running jobs, all jobs that have reserved a subcube, and jobs in the AQ. If a feasible schedule is possible then all the jobs currently executing are preempted and the new schedule is adopted.

The effectiveness of the DEDF algorithm can be illustrated by considering an example of a sample job set as shown in Table 1. The system size is assumed to be 4-cube. Jobs J_1 and J_2 arrive at time t. Job J_3 arrives at time $t+1$, and J_4 at $t+2$. Their subcube requirement, execution time, and deadlines are indicated in Table 1. We will first analyze the scheduling performance using Buddy/RT and Stacking. Both Buddy/RT and Stacking use EAT to find an

Table 1: A sample job set.

Job	Arrival Time	Subcube Size	Execution Time	Deadline	LST
J_1	t	2	4	$t+4$	t
J_2	t	3	7	$t+7$	t
J_3	$t+1$	3	2	$t+8$	$t+6$
J_4	$t+2$	2	2	$t+4$	$t+2$

available subcube for allocation. The order of allocation is done through EDF. At time t, the AQ will have the jobs J_1 and J_2, in order. Using Buddy/RT or Stacking, the jobs J_1 and J_2 are first allocated. The EATs of the subcubes {1-4}, {5-8}, and {9-16} are updated to $t+4$, t, and $t+7$, respectively. J_3 arrives at time $t+1$ with a deadline of $t+8$. To schedule J_3 that requires a 3-cube, we need to check the EATs of the two 3-cubes existing in the system. The EAT of subcube {1-8} and {9-16} are $t+4$ and $t+7$, respectively. Thus J_3 reserves the subcube {1-8}. The EAT of all the processors in the subcube {1-8} are updated to $t+6$. The allocations and reservations at time $t+1$ are as shown in Figure 1. At time $t+2$, J_4 arrives requesting a 2-cube with a deadline of $t+4$. As the EATs of all the 2-subcubes are either $t+6$ or $t+7$, J_4 cannot be allocated in phase 1. The jobs J_1 and J_2 need to be preempted to enter phase 2 in order to accommodate J_4, if possible. In this specific case, incidentally, the second phase of Buddy/RT and Stacking will not be initiated and J_4 will be rejected. Because there is no laxity for J_4, the preemption overhead will prohibit the initiation of the second phase.

When we schedule the jobs by using the DEDF algorithm, the concept of ATW is used instead of EAT for processor allocation or reservation. We will analyze the same example with a few variations to reflect different scenarios. Consider the four jobs with their characteristics as depicted in Table 1. At time t, the DEDF scheduler is triggered as the LST of J_1 is t. The AQ has jobs J_1 and J_2. J_1 is scheduled on subcube {1-4}, J_2 in subcube {9-16}. J_3 arrives at time $t+1$. The LST of J_3 is $t+6$, so the scheduler is not triggered. J_4 arrives at time $t+2$ with a LST of $t+2$. So the scheduler is triggered at $t+2$. When the scheduler starts to schedule, it has two jobs in the AQ (J_3 and J_4). Since the DEDF scheduling algorithm schedules the jobs with earlier deadline first, it will schedule J_4 before J_3, even though J_3 arrived earlier. In this case, the scheduler can schedule both J_3 and J_4 successfully to meet their deadlines as shown in Figure 2. We expect this kind of situation to be more prevalent when the workload has jobs with large laxities. Using Buddy/RT or Stacking, J_4 may be scheduled if we use the optional preemptive scheduling and there is enough laxity. However, by deferring scheduling, DEDF scheduling algorithm can avoid the unnecessary preemptive scheduling overhead.

To illustrate the effectiveness of the ATW concept, let us assume that the scheduler is triggered at $t+1$ after the arrival of job J_3 because of the AQ becoming full. Note that the length of AQ is usually long enough to hold more than one job but just to illustrate the utility of the ATW concept we assume here that the AQ becomes full at $t+1$, and the scheduler is triggered. As J_3 requires a 3-cube, the ATW of all the 3-subcubes are checked. They are [$t+4$ $\rightarrow$ open] for subcube {1-8}, and [$t+7 \rightarrow$ open] for the subcube {9-16}. Thus J_3 reserves the

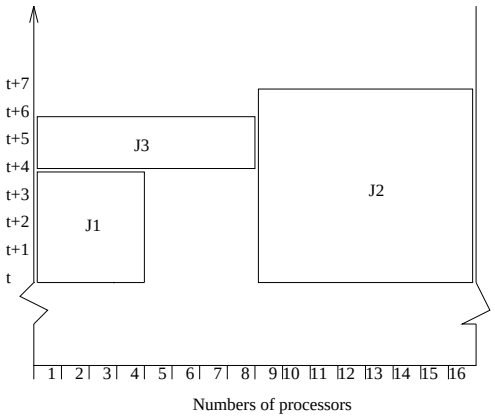

Figure 1: Scheduling using Buddy/RT or Stacking.

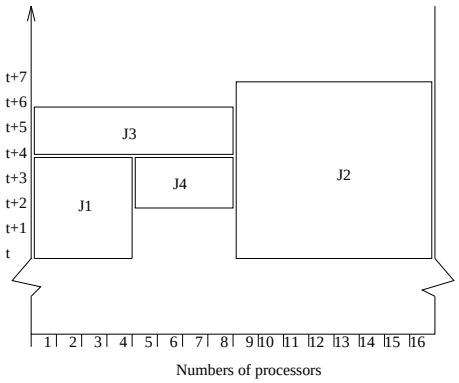

Figure 2: Scheduling using DEDF.

subcube {1-8}. The ATW of all the subcubes are then updated. When J_4 arrives at time $t+2$, the scheduler is again triggered as the LST of J_4 is $t+2$. J_4 requires a 2-cube. The ATW of all the 2-subcubes in the system are $[t+6 \rightarrow$ open] for subcube {1-4}, $[t+2 \rightarrow\ t+4]$ and $[t+6 \rightarrow$ open] for subcube {5-8}, $[t+7 \rightarrow$ open] for subcube {9-12}, and $[t+7 \rightarrow$ open] for subcube {13-16}. Thus, J_4 is allocated to the subcube {5-8}. With the concept of EAT, the subcube {5-8} is not used for the interval $(t+2,\ t+4)$ and J_4 would have been rejected. But using the concept of ATW, we can use the subcube {5-8} and can execute J_4 successfully before its deadline.

3.2 Data Structure and Complexity

For efficient scheduling using the DEDF algorithm, a data structure in the form of a binary tree can be maintained. For an n-cube, the root of the binary tree represents the entire hypercube and is called as the level n. At the $(n-1)$th level, there are two subcubes of dimension $n-1$. The leaves of the binary tree represent level 0 that consist of individual processors (0-cubes). A node at any level of the binary tree is represented as xy, where x denotes the level, and y represents the subcube number within the level. The level also represents the subcube size. Thus node 23 in the binary tree represents the third subcube of dimension 2. Two children combine to form their parent node. Each node of the binary tree maintains information about the status of the subcube it represents. To clarify the concepts further, we have illustrated the data structure of the DEDF schedule of the example considered in the previous subsection in Figure 3. For the sake of clarity, we have shown only one-half of the binary tree. Each node maintains the ATW as indicated in the figure. It also maintains the subcube size of the job last scheduled (this is not shown in the figure).

The time complexity of the DEDF scheduling can be derived as follows. In the first phase, there are two steps: finding an appropriate subcube, and updating the binary tree. The complexity of finding a subcube is $O(2^{n-i})$, where i is the dimension of the subcube required by the job, and the system size is n-cube. The binary tree updating needs a constant time since updating an entry takes constant time and the number of levels is fixed for a given hypercube.

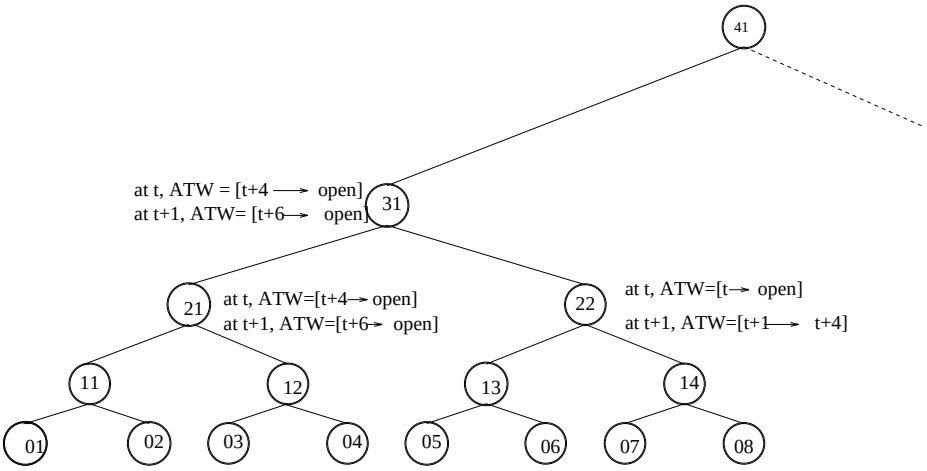

Figure 3: Data Structure of the DEDF Schedule.

Thus, the complexity of the first phase is $O(2^{n-i})$. In the worst case of the second phase, all the jobs in the job set require a 0-subcube. The complexity of the allocation of the whole job set then becomes $O(2^n)$. These time complexities are better than that of Buddy/RT and equal to that of the Stacking algorithm.

4 Performance Measurements

We evaluate the performance of the DEDF scheduling algorithm through simulations. For real-time systems, two useful metrics for performance evaluation are the Job Miss Ratio (JMR), which is the ratio of jobs refused to the total number of jobs that arrive at the system, and the Work Miss Ratio (WMR), which is the ratio of refused work to the offered workload. For WMR, the work requested by a job is computed as its subcube hold time multiplied by the number of processors constituting the subcube.

4.1 Workload Characterization

In our study, a Poisson job arrival process is assumed. The Poisson process has been found to model natural physical and organic processes realistically, and is commonly used to model random, independent arrivals of jobs to computer systems from an external population [7]. We assume that the subcube hold times are normally distributed with a standard deviation equal to the mean, and with the distribution truncated at 0 and at one standard deviation greater than the mean. Without loss of generality, we assume a mean subcube hold time of 3 time units. Laxities are also assumed to be normally distributed, again truncated at 0 and one standard deviation above the mean. Subcube hold time is independent of the subcube size. We assume that jobs request complete subcubes, and three different distributions for the subcube dimensions are considered. They are: modified geometric distribution, the discrete uniform distribution, and a reverse geometric distribution, which is simply a mirror image of the geometric distribution. All these distribution are truncated at 0 and n-1, where n is the

dimension of the hypercube. The probability mass function for the truncated modified geometric distribution is $p(k) = \frac{p(1-p)^k}{\sum_{k=0}^{n-1} p(1-p)^k}$, where $p = \frac{1}{mean+1}$. For the discrete uniform distribution, $p(k) = \frac{1}{n}$, and for reverse geometric distribution, $p(k) = \frac{p(1-p)^{n-1-k}}{\sum_{k=0}^{n-1} p(1-p)^k}$. While all subcube sizes are equally likely in the uniform distribution, the geometric distribution has a high proportion of small jobs and the reverse geometric distribution has a high proportion of large jobs. All the above mentioned assumptions and workload characteristics are also considered in previous studies [1, 15, 16].

4.2 Simulation Environment

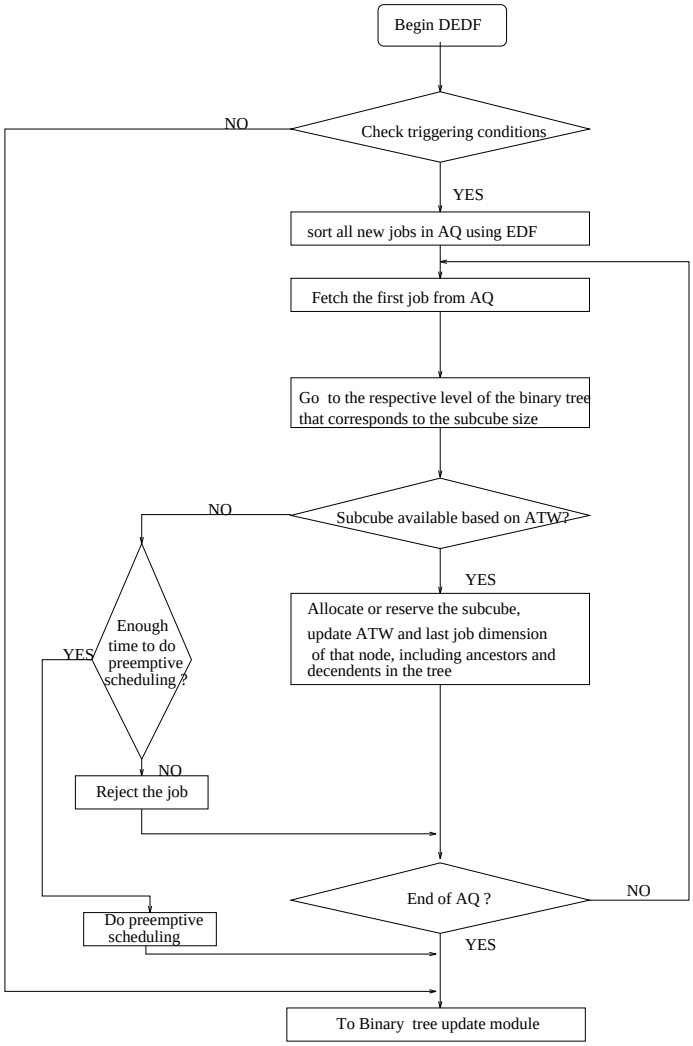

Figure 4: Flowchart of the DEDF scheduling scheme.

We have simulated three different scheduling algorithms: Buddy/RT, Stacking, and DEDF with various workload conditions. The input parameters are the scheduling algorithm to be used, dimension of hypercube, observation interval, offered system load, job arrival rate, subcube

size distribution, mean subcube size, standard deviation of subcube size, mean laxity, standard deviation of laxity, mean subcube holding time, and the standard deviation of holding time. The simulation model outputs are the JMR and WMR, which are plotted with respect to the offered system load or the mean laxity. If λ is the job arrival rate, X is the mean subcube hold time, P is the mean number of processors in a subcube request, and N is the number of processors in the hypercube, then the offered system load is $\frac{\lambda XP}{N}$. The mean laxity is evaluated as a percentage of the mean subcube hold time.

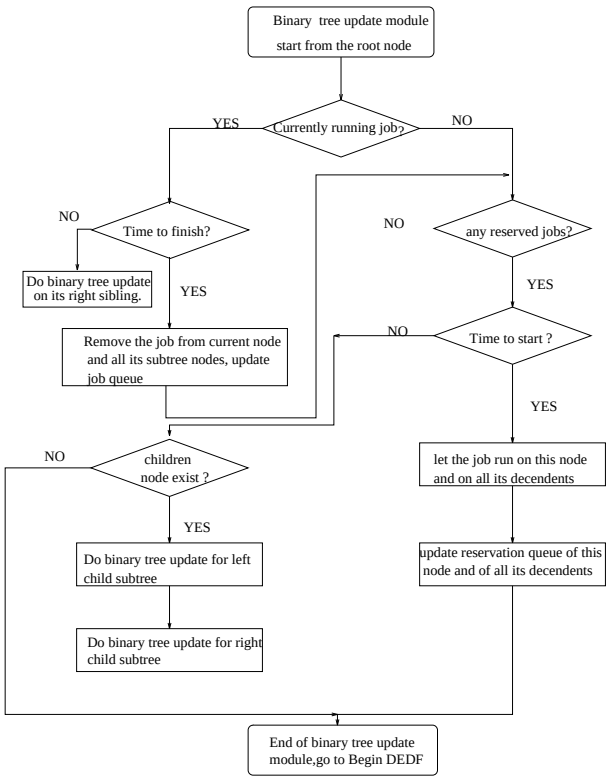

Figure 5: Flow chart of the binary tree update module.

In our simulation, the hypercube system is represented as a binary tree. Each node in the binary tree represents a possible subcube in the hypercube and maintains its status. The complete structure of the DEDF scheduling simulator is shown as a flowchart in Figure 4. The binary tree update module is shown in Figure 5. The block in the flow chart that represents the updating process of the right sibling is terminated after reaching the rightmost sibling. The triggering conditions are mentioned earlier in Section 3.1. Triggering conditions are not required to be checked in the second phase. A predetermined preemption overhead is included to capture the time taken in making the rescheduling in the second phase, if necessary. We have experimented with several values of the preemption overhead and have observed similar trend. As the emphasis of the paper is to demonstrate the relative performance improvement, we have only shown the performance results with a preemption overhead of 2 time units. A temporary binary tree is used for the scheduling of all the jobs in the system including the jobs currently executing, the jobs that have reserved subcubes, and all jobs in the AQ. All jobs are sorted by

EDF. If the scheduler cannot find an available subcube for a job in the second phase, the job is rejected by the system. In addition, once the scheduler starts preemptive scheduling, it does not schedule new jobs arriving after it has begun arranging the jobs based on EDF.

4.3 Results and Inferences

We have computed both JMR and WMR for the system and workloads presented in this section. The trends of both these plots are observed to be the same. For the sake of brevity, we have reported only the plots of the JMR. The JMR is expressed in percentage and is plotted against the offered system load or the mean laxity.

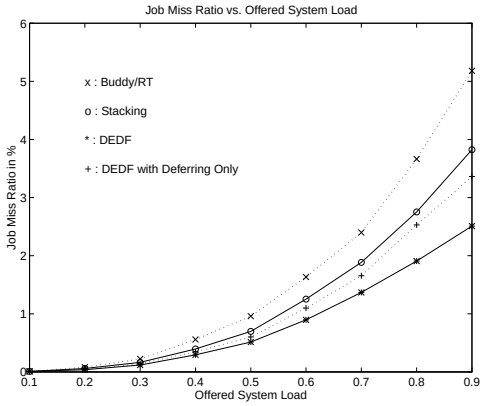

Figure 6: Job miss ratio versus offered system load.
(mean subcube size = 9.36, geometric job size distribution)

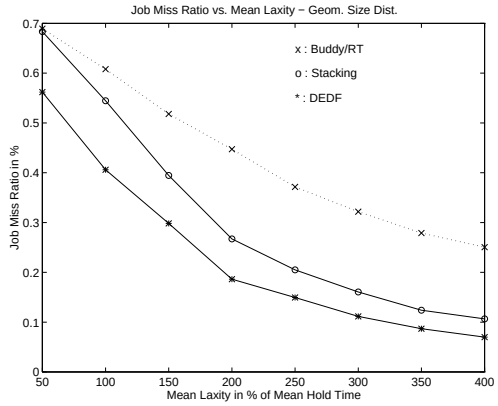

Figure 7: Job miss ratio versus mean laxity.
(mean subcube size = 9.36, geometric job size distribution)

Figure 6 shows the plot of job miss ratio with respect to the offered system load for an 8-dimensional hypercube (256 nodes). The job size distribution is considered geometric with a mean laxity of 150% of the mean service demand. It can be observed from the plot that the JMR of the DEDF is considerably less than that of Buddy/RT or stacking algorithms. We have also plotted a curve that represents the DEDF algorithm that does not uses the concept of ATW. It just defers the allocation and uses EAT for allocation. The curves show that performance is also gained by only deferring the scheduling. It can be also inferred that both deferment of scheduling, and allocation using ATW have impact on reducing the miss ratio. Similar trends are also observed when the job size distribution is uniform or reverse geometric.

The job miss ratio with respect to the mean laxity (of the subcube hold time) for an 8-dimensional hypercube is plotted in Figures 7, 8, and 9. The offered system load is assumed to be 0.4. The job size distribution in Figures 7, 8, and 9 are geometric, uniform, and reverse geometric, respectively. It is observed that the JMR using the DEDF scheduling in considerably lower than that of Buddy/RT and Stacking for all the distributions. For low laxities, there is not much difference between the Buddy/RT and Stacking algorithms. But the DEDF algorithm demonstrate a substantial improvement in performance from low to intermediate mean laxities.

At high mean laxities, all the three algorithms have less JMR as almost all the jobs have a very high probability of being accepted due to the large laxities. It is also observed that the JMR for any given laxity is lower in case of geometric distribution as compared to the other two distributions. This is because of the more number of small-sized jobs in geometric distribution.

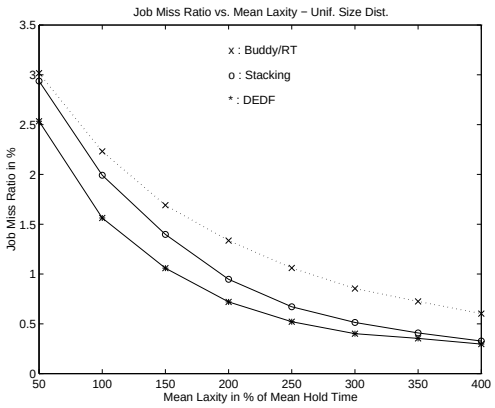

Figure 8: Job miss ratio versus mean laxity.
(mean subcube size = 32, uniform job size distribution)

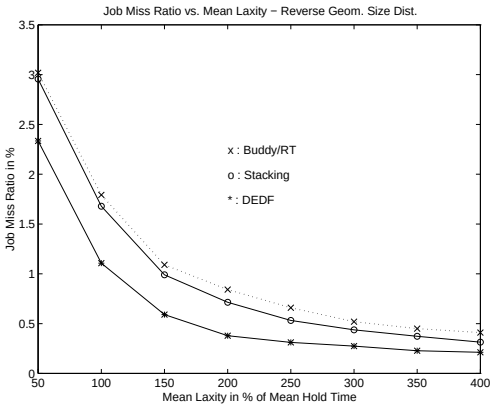

Figure 9: Job miss ratio versus mean laxity.
(mean subcube size = 66.67, reverse geometric job size distribution)

The performance improvement using the DEDF scheduling is between 25% to 150% compared to Buddy/RT, and in between 10% to 75% compared to the Stacking algorithm for the workload studied in this paper.

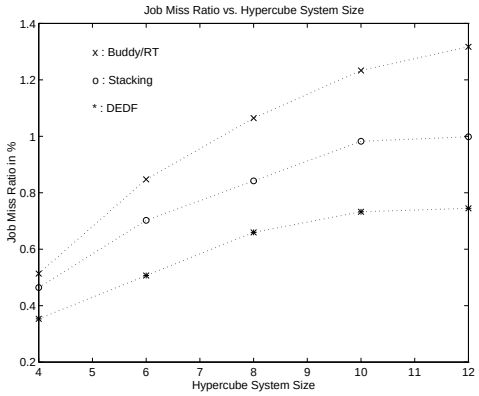

Figure 10: Job miss ratio versus the system size.
(uniform job size distribution, offered load = 0.4, mean laxity = 150%)

The performance of various scheduling schemes with the increase in the hypercube size is plotted in Figure 10. The curves for the job miss ratio (in %) is plotted against the hypercube dimension for uniform job size distribution. The offered system load and the mean laxity are assumed 0.4 and 150%, respectively. It is observed that the JMR of the DEDF scheduling is

always lower than that of Buddy/RT and Stacking. Furthermore, the increase in JMR with the increase in the system size is slower in case of DEDF as compared to the other algorithms. Similar trends are also observed when the job size distribution is geometric or reverse geometric.

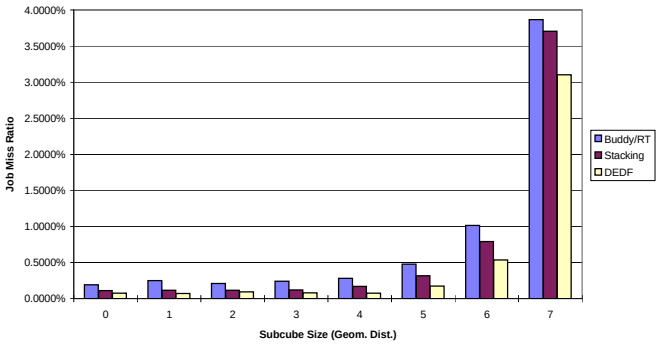

Figure 11: Job miss ratio with respect to various subcube sizes in an 8-cube. (geometric job size distribution, offered load = 0.4, mean laxity = 150%)

Finally, we compare the miss ratios of the jobs of different sizes for the three scheduling algorithms - Buddy/RT, Stacking, and DEDF. Figures 11, 12, and 13 reflect the relative job miss ratios of different size subcubes in an 8-cube system for the three scheduling algorithms. The job size distributions in the figures are geometric, uniform, and reverse geometric, respectively. The plots can be also read as the probability of rejection of a particular size job for each of the algorithms. For all the three distributions, it is observed that the miss ratio of the large size jobs are high, as expected. While comparing the three algorithms, it is seen that the miss ratio of the DEDF algorithm is lower than the other two algorithms for all job sizes. This observation is true for all the three job size distributions.

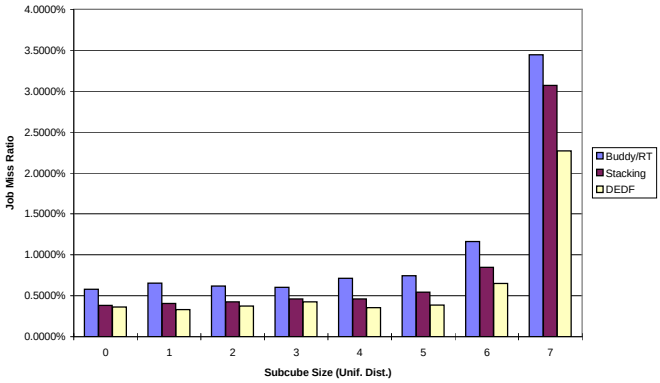

Figure 12: Job miss ratio with respect to various subcube sizes in an 8-cube. (uniform job size distribution, offered load = 0.4, mean laxity = 150%)

It can be argued that by deferring the scheduling, one may delay informing the rejection of a job to the user. The delay may not be acceptable for certain applications. The DEDF

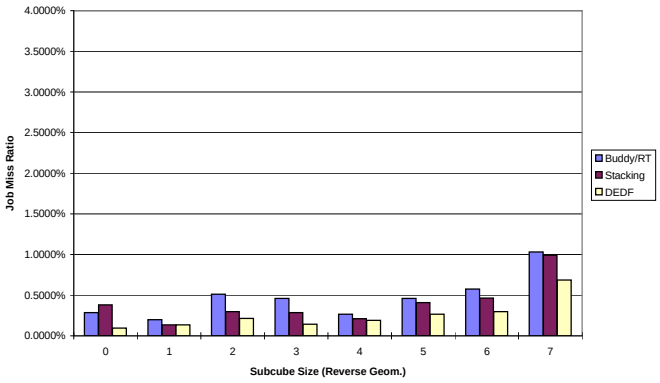

Figure 13: Job miss ratio with respect to various subcube sizes in an 8-cube. (reverse geometric job size distribution, offered load = 0.4, mean laxity = 150%)

algorithm is thus not suitable for such applications. However, in any case, the concept of ATW can be still employed to enhance the performance of real-time scheduling. Furthermore, the concept of deferred scheduling can be employed in conjunction with Buddy/RT or Stacking algorithms to extract enhanced performance for application that are insensitive to the dealy in acceptance/rejection decisions.

5 Conclusion

A new dynamic real-time scheduling algorithm called DEDF is proposed in this paper for hypercube systems. The DEDF scheduling scheme is based on heuristics and two new ideas are introduced in the algorithm. First, the scheduling of the jobs are not necessarily done as soon as they arrive. The scheduling starts when certain triggering conditions are met. Delayed scheduling allows us to schedule a set of tasks instead of a single one. A set of tasks is usually scheduled more efficiently compared to one-at-a-time. Second, the concept of ATW is used instead of EAT which was used in the earlier proposed scheduling algorithms. By using ATW, the utilization of the system is improved as it enables the allocation of more number of jobs. The improvement in utilization, in turn, improves the performance of the DEDF scheduling algorithm. The complexity of the DEDF algorithm is less than that of Buddy/RT and is equal to that of the Stacking algorithm. Although we have discussed the applicability of the DEDF algorithm for hypercube computers, it is equally well-suited for any partitionable multiprocessor.

References

[1] D. Babbar and P. Krueger, "On-Line Hard Real-time Scheduling of Parallel Tasks on Partitionable Multiprocessors," International Conference on Parallel Processing, vol. II, pp. 29 - 38, 1994.

[2] S. Dutt and J. P. Hayes, "Subcube Allocation in Hypercube Computers," IEEE Trans. on Computers, pp. 341 - 352, March 1991.

[3] M. S. Chen and K. G. Shin, "Processor Allocation in an N-Cube Multiprocessor Using Gray Codes," IEEE Trans. on Computers, pp. 1396-1407, Dec. 1987.

[4] P. J. Chuang and N. F. Tzeng, "Dynamic Processor Allocation in Hypercube Computers," Int. Symp. on Computer Architecture, pp. 40-49, May 1990.

[5] K. S. Hong and J. Y. T. Leung, "On-Line Scheduling of Real-Time Tasks," IEEE Trans. on Computers, pp. 1326-1331, Oct. 1992.

[6] J. Kim, C. R. Das, and W. Lin, "A Top-down Processor Allocation Scheme for Hypercube Computers," IEEE Trans. on Parallel and Distributed Systems, pp. 20 - 30, Jan. 1991.

[7] L. Kleinrock, *Queueing Systems: Volume 1, Theory,* John Wiley and Sons, New York, 1976.

[8] R. Krishnamurti and B. Narahari, "Preemptive Scheduling of Independent Jobs on Partitionable Parallel Architectures," Int. Conf. on Parallel Processing, vol. I, pp. 268-275, 1992.

[9] P. Krueger, T. H. Lai, and V. A. Radiya, "Job Scheduling is More Important than Processor Allocation for Hypercube Computers," IEEE Trans. on Parallel and Distributed Systems, May 1994.

[10] P. Mohapatra, C. Yu, and C. R. Das, "A Lazy Scheduling Scheme for Hypercube Computers," Journal of Parallel and Distributed Computing, 27, pp. 26-37, May 1995.

[11] A. K. Mok and M. L. Dertouzos, "Multiprocessor Scheduling in a Hard Real-time Environment," Proc. Seventh Texas Conf. Computer System, Nov. 1978.

[12] J. L. Peterson and T. A. Norman, "Buddy Systems," Communications of the ACM, pp. 421-431, June 1977.

[13] K. Ramamritham, J. Stankovic, and P. Shiah, "Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems," IEEE Trans. on Parallel and Distributed Systems, pp. 184-194, April 1990.

[14] S. Sahni, "Preemptive Scheduling with Due Dates," Oper. Res. Vol. 27, pp. 925-934, 1979.

[15] N. G. Shivaratri and M. Singhal, "A Transfer Policy for Global Scheduling Algorithms to Schedule Tasks With Deadlines," Proc. of 11th. Int. Conf. on Distributed Computing Systems, pp. 248 - 255, May 1991.

[16] N. G. Shivaratri and M. Singhal, "A Load Index and a Transfer Policy for Global Scheduling of Tasks With Deadlines," Concurrency: Practice and Experience, pp. 671-688, Oct. 1996.

[17] J. A. Stankovic, M. Spuri, M. D. Natale, and G. C. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems," IEEE Computer, pp. 16-25, June 1995.

[18] H. Wang and Q. Yang, "Prime Cube Graph Approach for Processor Allocation in Hypercube Multiprocessors," Int. Conference on Parallel Processing, pp. 25 - 32, Aug. 1991.

[19] P. Mohapatra, "Processor Allocation Using Partitioning in Mesh-Connected Parallel Computers," Journal of Parallel and Distributed Computing, pp. 181-190, vol. 39, December 1996.