An Efficient Non-Preemptive Algorithm for Soft Real-Time Systems using Domain Cluster–Group EDF

R. Kalpana
Dept. of CSE
Pondicherry Engg. College

S. Keerthika
Dept. of CSE
Pondicherry Engg. College

ABSTRACT
In this paper a new non-preemptive domain clustering scheduling algorithm for soft real time application is proposed. The main aim of this algorithm is to achieve high deadline meeting ratio of the group Earliest Deadline First (gEDF) algorithm by maintaining the excellent performance during normal load. An efficient non-preemptive algorithm called Domain Cluster – group EDF (DC-gEDF) is proposed for real time systems which makes clustering the task according to the domain specification with their deadlines and schedules the tasks within a group. The results are analysed and compared for the metric deadline meeting ratio of gEDF and DC-gEDF under different deadline acceptable values. It shows an improvement in the deadline meeting ratio for the proposed DC-g-EDF algorithm.

General Terms
Real time scheduling

Keywords
Group scheduling, EDF, group EDF, soft real time, non-preemptive

1. INTRODUCTION
Earliest deadline first (EDF)is a dynamic scheduling algorithm [5] used in real-time task scheduler to place the tasks in a priority queue [19]. Whenever a scheduling processes need to perform the queue will be searched for the task closest to its deadline. This task is to be scheduled for next execution [2], [4]. EDF is an *optimal* scheduling algorithm on preemptive scheduling processors, in that sense: if a collection of independent *tasks*, each characterized by an start time, an requirement and a deadline, can be scheduled by any algorithm in a way that ensures all the tasks complete by their deadline, the **EDF** will schedule this collection of tasks so they all complete by their deadline.

In preemptive systems, tasks are pre-empted by higher priority tasks or by some other techniques, while non-preemptive systems will not permit pre-emption. Preemptive scheduling algorithms are easy to develop but non-preemptive scheduling algorithms are more efficient than the preemptive. However, non-preemptive EDF techniques have produced near optimal schedules for periodic and aperiodic tasks, particularly when the system is lightly loaded [3]. When the system is overloaded, it has been shown that the EDF approach leads to very poor performance (i.e., low success rates). In this dissertation, a system load or utilization is used to refer to the ratio of the sum of the execution times of pending tasks and the time available to complete the tasks [9], [12]. The reason for the poor performance of EDF is because of tasks that are scheduled based on their deadlines, sometimes they miss their deadlines due to overload, so that other tasks waiting for their turn for long time so their deadlines are also have the chance to miss also – it is sometime known as the domino effect. To overcome these issues, the derivation of EDF algorithm is used. It is group EDF (gEDF), where the tasks with "similar" deadlines are grouped together (i.e., deadlines that are very close to one another), and the Shortest Task First (SJF) algorithm is used for scheduling tasks within a group. It should be noted that our approach is different from adaptive schemes that switch between different scheduling strategies based on system load. gEDF is used in overloaded as well as under loaded conditions.

However gEDF produces efficient scheduling results, it does not concerns domain specification of tasks. Our approach is to analyze the specification of given tasks and grouping them with similar one using gEDF and with new scheduling algorithm called Domain Cluster – group EDF (DC – gEDF). This algorithm groups the tasks according to the domain and applies gEDF to each group. So that each group can schedule simultaneously, this reduces the overall duration to complete the tasks.

2. RELATED WORK

2.1 Shortest Task First (SJF)

Shortest Task First (SJF) scheduling is probably optimal but requires clairvoyance, profiling, or expected execution time to fully implement. SJF can be implemented either pre-emptively or non-pre-emptively. SJF has low average waiting time [10], [11], [20]. In fact, SJF is optimal with respect to average waiting time. It is very easy to prove this claim by comparing it with other real-time algorithms.

In Soft Real Time systems [13], [14], each task may have one or more than one predecessors. SJF algorithm [7] cannot analyze the predecessors of the tasks. To overcome this issue another real time scheduling algorithm has been proposed, called Earliest Deadline First (EDF).

2.2 Earliest Deadline First (EDF)

Each task in an EDF scheduler is assigned with a deadline. Every time a task is inserted in the system, the scheduling system looks for the other task which is present in the queue and which has the next nearest deadline and select it for execution [7]. In order to ensure that the scheduling application is still able to meet each deadline, a 'scheduler must evaluate if each new incoming task doesn't overload the system and slow down the execution if it will do so.

However, EDF[8] performs worst during overload (i.e. when the total CPU utilization exceeds 100%) condition [4], [15]. This poor performance may create the chance to miss the deadline of the tasks. And this leads that other tasks need to wait for a turn longtime and miss their deadline too.

2.3 Group Earliest Deadline First (gEDF)

In this concept the deadlines are formed as a group according to their range of deadlines, dynamically. Here groups are separated by the concept of traditional EDF but the group may be arranged by different techniques say priority or SJF value, etc. We use SJF to develop EDF, but it is one of the techniques among the scheduling methods. gEDF is mainly suitable for non-preemptive soft-real-time systems. A group in the gEDF algorithm depends on a set of deadline group range that is,these are grouped according to the deadlines named as parameter Gr. A taskτjbelongs to the same group as taskτi if di≤dj≤ (di + Gr*(di – ti)), where ti is the current time, 1 ≤i, j≤N [1]. We group tasks with deadlines which are close to each other. The tasks with close deadlines are in a group.

We assume a uniprocessor system [16]. QgEDF is a queue for gEDF scheduling. The current time is represented by t. |QgEDF| represents the length of the queue QgEDF. τ = (r, e, D, P) is the task at the head of the queue.

- gEDF Group = {τk | τk∈ QgEDF, dik – di1≤D1* Gr, 1 ≤k≤m, where m≤ |QgEDF|}, and D1 is the deadline of the first task in a group

The pseudo code of the functions Enqueue and Dequeue are given below:

Function Enqueue (Q_{gEDF}, τ)

if (τ's deadline $d>ti$) *then*

inserttask τ into $Q_{gEDF}$by Earliest Deadline First, i.e. $d_i \leq di_{i+1} \leq di_{i+2}$, where τ_i, $\tau_{i+1}, \tau_{i+2} \in Q_{gEDF}$, $1 \leq i \leq |Q_{gEDF}| - 2$;

end

- Enqueue is called on eachtask arrivals.

Function Dequeue (Q_{gEDF})

if $Q_{gEDF} \neq \phi$*then*

find a task τ_{min} with e_{min} = min $\{e_k \mid \tau_k \in Q_{gEDF}, di_k - di_1 \leq Gr * D_1, 1 \leq k \leq m$, where $m \leq |Q_{gEDF}|\}$;run it and delete τ_{min} from Q_{gEDF};

end

- Dequeue is called when the execution of all task ends.

3 DOMAIN CLUSTER – GROUP EDF (DC-G-EDF)

A. Definitions

To define our algorithm we use the following notations.

τi - denotes task*i*

Di – absolute deadline of task*i*

di - dynamic deadline of task*i*

Ut – denotes total cpu utilization

Tr – denotes the deadline tolerance of soft real time system

ei – denotes the execution time of task*i*

B. Scheduling Model: Non-Preemptive Real Time

Non-preemptive scheduling [17], [18] is much efficient than preemptive scheduling since preemption requires context switching method overhead which may be significant in well-formed multithreading systems. In this non-preemptive model SJF algorithm is used to schedule the tasks according to the deadlines (i.e. which has small deadlines that are arranges first).

C. The Algorithm

We first group the tasks with their domain specification together based on our novel algorithm. Within the group the tasks were sorted based on another algorithm, where the task at the head of the queue will be executed first followed by the rest of the tasks in the queue sequence. A group member in the Domain cluster group is to be rescheduled according to the predecessors. And then, have to apply the gEDF algorithm to schedule finally.

The clustering is based on the domain using the following algorithm

Function Domain-Cluster (Q_{DCgEDF}, DC)

for i=1 to i<= | Q_{DC} | {

for j=i+1 to j<= | Q_{DC} | {

findDC$_i$which has same domain name and put it in $Q_{DCgEDF;}$ }

}

//gEDF with predecessor constraints

ID2: for i=1 TO i<= | Q_{DCgEDF} | {

$T_{DCgEDF} = Q_{DCgEDF;}$

If (Pr[τi]!==0) then

If (flag[τi]==0) then ID1: if (m!==0) then $C_{DCgEDF} = T_{DCgEDF;}$

For j=1 to j<=Pr[τi] {

Sub[τi]=S[T[τij];

if(flag[Sub[τi]]==0) then $T_{DCgEDF=}$ Sub[τi]; m++; goto ID1;

end }

end

end

printτi from Q_{DCgEDF}; }

4 NUMERICAL RESULTS

The following Table 1 depicts the resources, predecessor associated with each domain along with the time (duration) taken for each activity.

Table 1.Resource allocation

Activity	Domain	Resources	Predecessor	Time(days)
A	D1	3	0	2
B	D2	3	0	3
C	D1	3	A	2
D	D2	2	A,B	4
E	D3	1	C	4
F	D1	4	C	3
G	D1	4	D,E	5
H	D2	2	F,G	2
				25

The following graph in Fig. 1 depicts the dependency graph for the activities tabulated above:

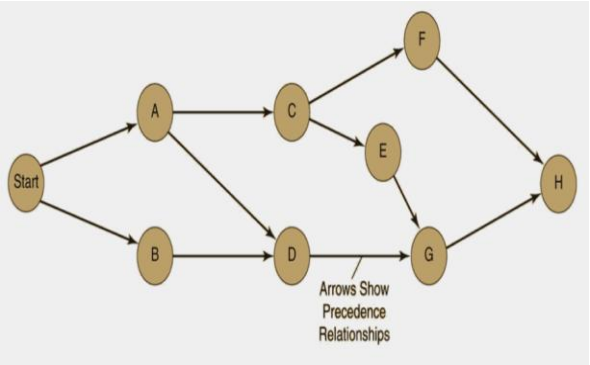

Fig 1: Determining the Project Schedule

1. Some activities can be done simultaneously so project duration should be less than 49 days.
2. Critical path analysis is used to determine project duration.
3. The critical path is the *longest* path through the network.

Domain Clusters

Now here in Fig. 2, this process has been formed into three clusters according to their Domain specification. (i.e.) Domain 1 (D1) consists of processes A, C, F and G. Domain 2 (D2) consists of processes B, D and H. Finally Domain 3 (D3) consists of process E. So have to consider each cluster for Critical Path analysis and have to apply gEDF to each cluster. Using EDF and gEDF algorithm we couldn't do the domain analysis.

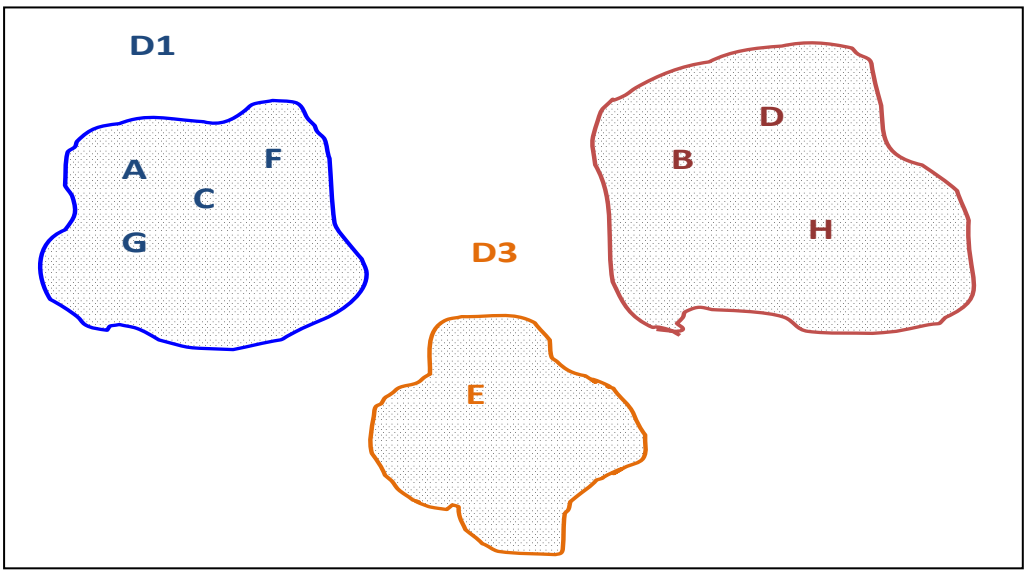

Fig 2: Representation of Domain Clusters

Without Domain analysis we cannot able to schedule the process to the members efficiently. If processes are scheduled to the members randomly, there is a chance of misleading and which results the projects may not be completed within scheduled duration. So this DC-gEDF algorithm possesses domain clustering to schedule the projects to the members according to the specification. Domain clustering analysis took place for assignment of project and not for scheduling.

Critical Path Analysis (CPA)

For CPA analysis, have to find the following values:

1 Earliest Start Time (EST)
2 Earliest Finish Time (EFT)
3 Latest start time (LST)
4 Latest Finish Time (LFT)

EFT Rule states that

EFT = EST + activity time

Node Notation: The node notation in Fig. 3 showing the EST, EFT, LST and LFT is as follows:

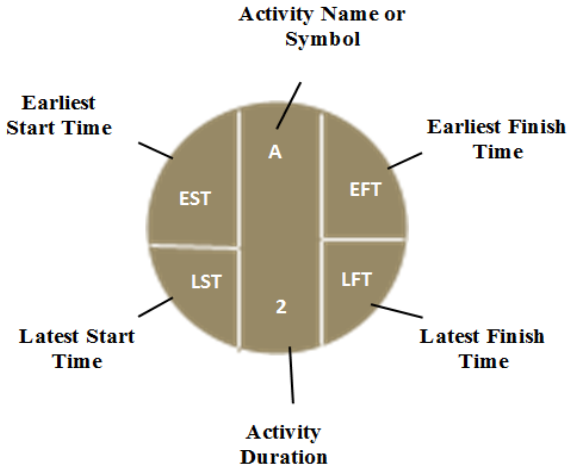

Fig 3: Notation of a Node

Forward Pass

EST Rule: All immediate predecessors must be executed before an activity starts

Identifies *earliest* times (EST and EFT)

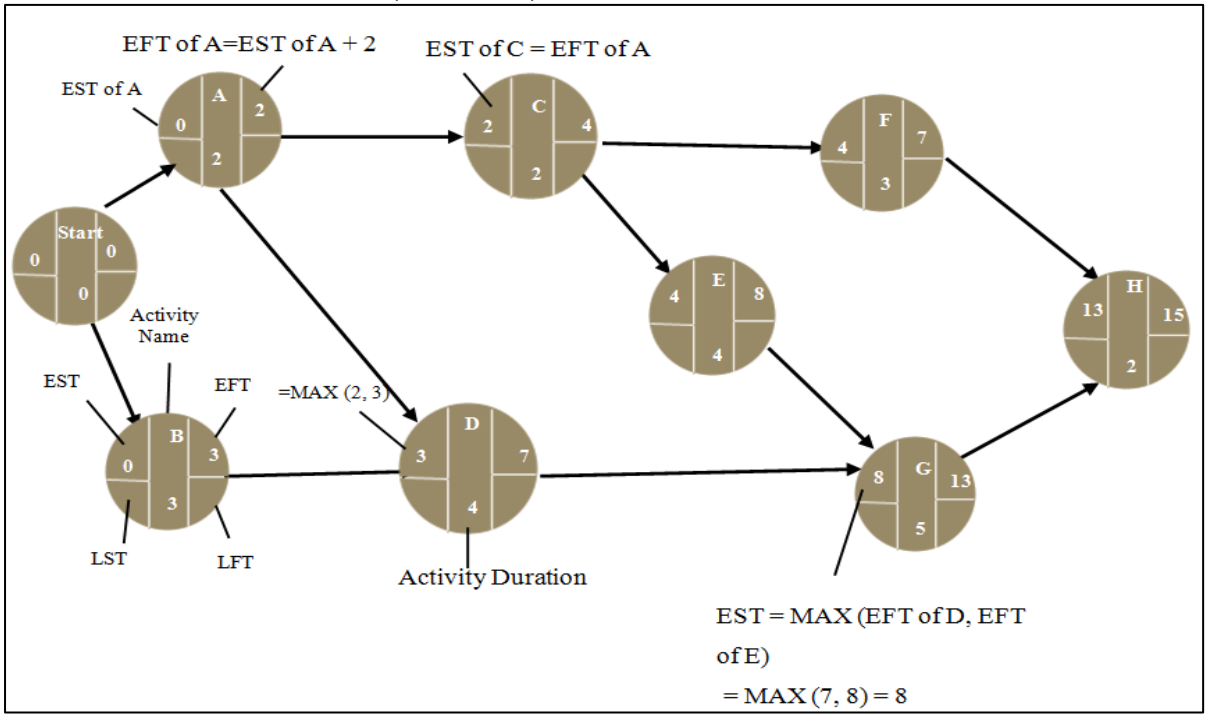

Fig 4: Forward Pass

- If there one immediate predecessor, then
  EST = EFT of predecessor
- If >1 immediate predecessors, then
  EST = MAX{EFT's of all predecessor}

Forward Pass: The forward pass is based on the Earliest Start and Finish Times, which is given in Fig. 4.

Backward Pass

The pseudo code for the backward pass is shown below:

1. Identifies *latest* times (LST an LFT)
2. **LFT Rule:**
   a. If activity is the immediate predecessor to only 1 activity, then
   b. LFT = LST of immediate follower
   c. If activity is the immediate predecessor to multiple activities, then
   d. LFT = Min {LST of all immediate followers}
3. **LST Rule:**
   LST = LFT – activity time

Backward Pass: The backward pass is based on Latest Start and Finish Times which is given in Fig. 5.

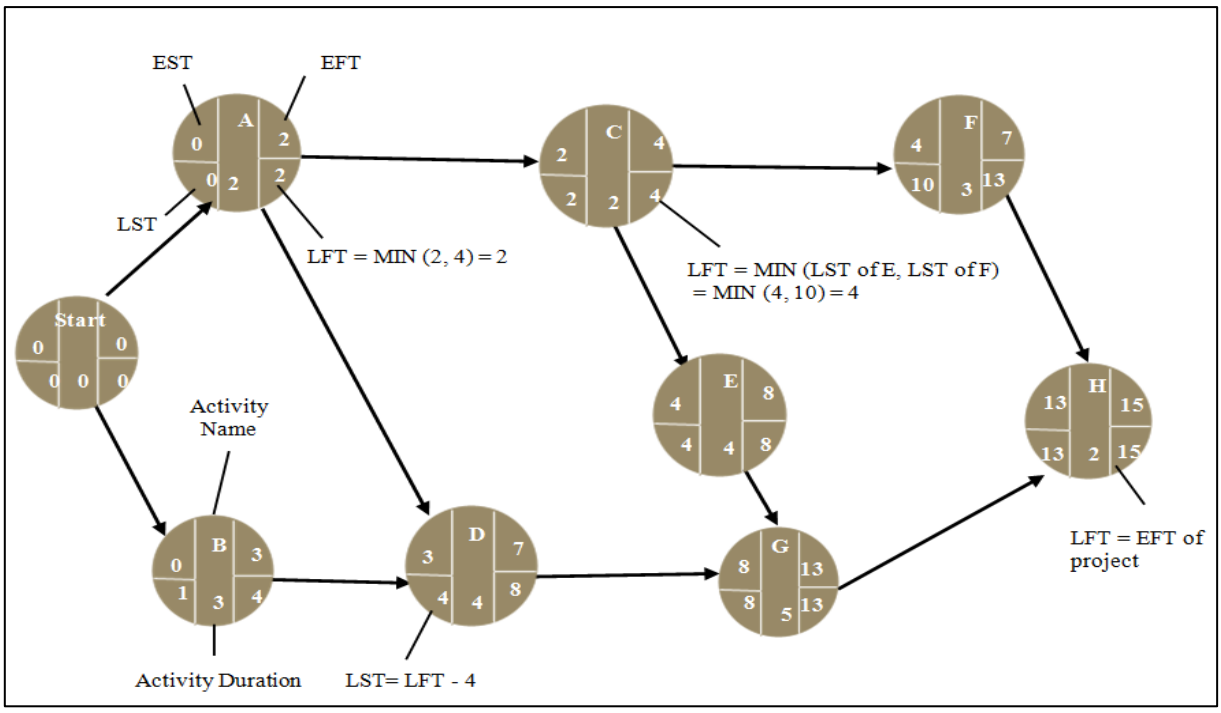

Fig 5: Backward Pass

Slack Time and Critical Path(s)

1. **Slack** is the total duration of an activity which may be delayed without delaying the overall project
   i. Slack = LST – EST
2. Activities which have slack 0 are **CriticalActivities**
3. The **Critical Path** is a continuous path throughout the project from start to endwhich includes only critical activities

Project Schedule and Slack Times

The following Table 2 shows the EST, EFT, LST, LFT, Slack time corresponding to every activity. In addition to it, whether an activity is a part of critical path is also shown in Fig. 6.

Table 2. Project Schedule

ACTIVITY	EST	EFT	LST	LFT	SLACK, LST-EST	ON CRITICAL PATH?
A	0	2	0	2	0	Yes
B	0	3	1	4	1	No
C	2	4	2	4	0	Yes
D	3	7	4	8	1	No
E	4	8	4	8	0	Yes
F	4	7	10	13	6	No
G	8	13	8	13	0	Yes
H	13	15	13	15	0	Yes

Critical Path and Slack Times

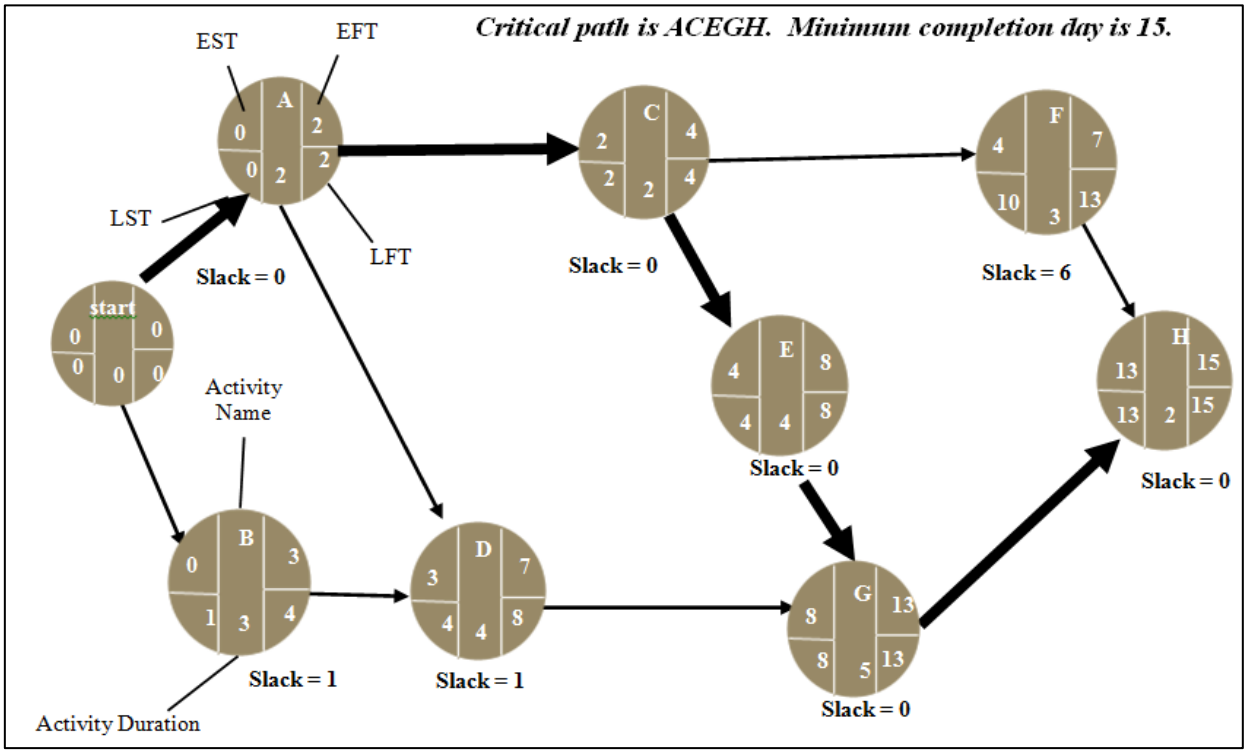

Fig 6. Critical in the schedule

GANTT CHART

The Gantt chart for all the activities involved is depicted below in Fig. 7:

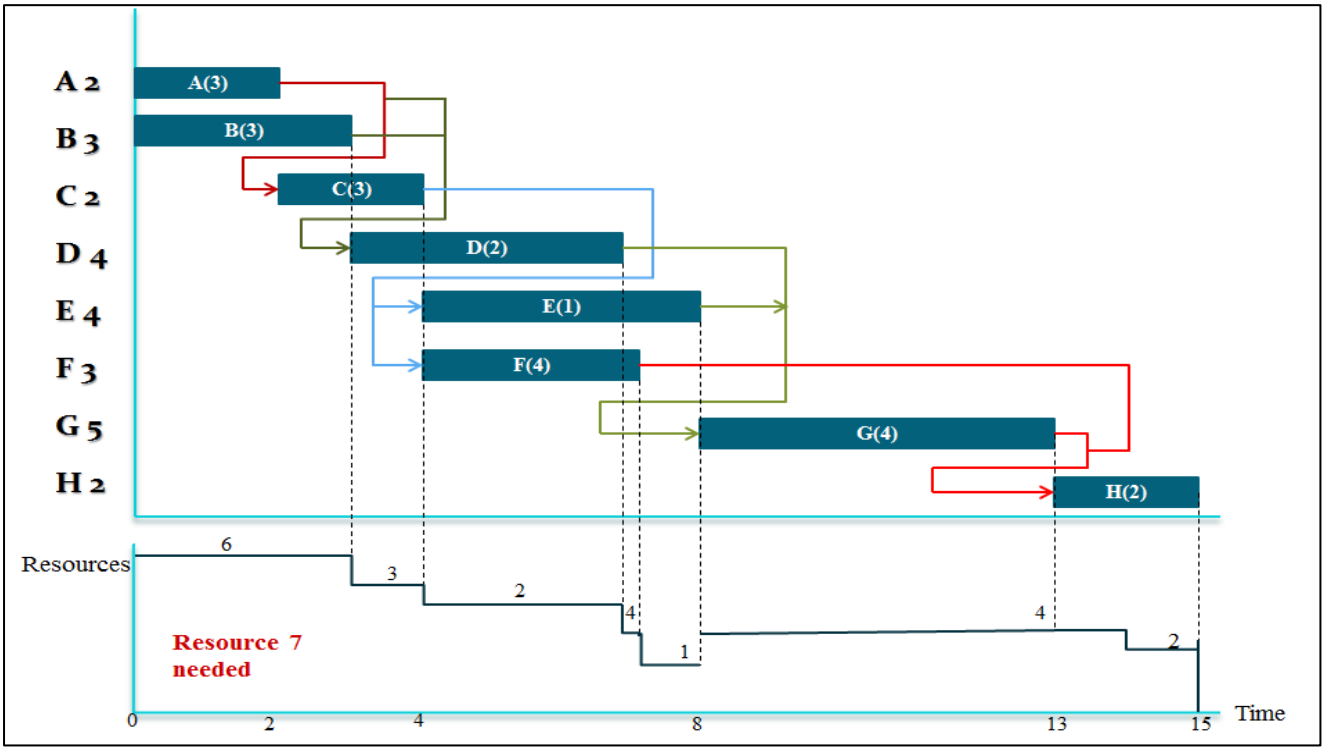

Fig 7. Gantt chart for activities

In this Gantt chart the clear work schedule has been defined. The process A(D1) and B(D2) has no predecessors so they are scheduled first. Then C (D1) has been scheduled followed by D(D2) is scheduled. Now E(D3) and F(D1) are scheduled concurrently because both are dependent on and completed before itself. Then G(D1) and H(D2) are scheduled. So total time taken to complete this project will 15 days and the resource needed for this allocation is 7 Units.

4. Effect of Deadline Tolerance on Deadline Meeting Rate

Comparison between EDF, gEDF and DC – gEDF DMTR

The below figure Fig. 8 shows the deadline meeting rate of EDF, gEDF and DC – gEDF algorithm as Tr in 100%. We observe that DC – gEDF achieve a much better DMTR.

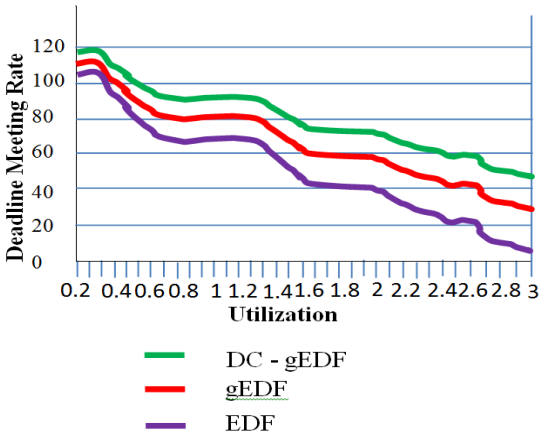

Fig. 8. Effect of Deadline Tolerance

From the above figure, it clearly shows that DC- gEDF has produce much better DMTR during normal load and has shown very significant increment during overload as the deadline tolerance value increases. And it clearly states that duration of a project will be much lower than EDF and gEDF because DC – gEDF has high deadline meeting rate. It never misses the deadline.

5. Comparisonof EDF, g-EDF And DC-g-EDF

The comparison between Earliest Deadline First, Group Earliest Deadline First and Domain Clustering- Group Earliest Deadline first is tabulated in Table 3.

Table 3. Comparison of EDF, g-EDF and DC-g-EDF

EDF	GEDF	DC-GEDF
Pre-emptive scheduling	Non-preemptive scheduling	Domain specified non-preemptive scheduling
No predecessor constraints	No predecessor constraints	Solved with predecessor constraints
No group formed	Group based on small units of deadlines	Group based on both domain and deadlines
It may have chance to miss the deadlines of the tasks when the system is overloaded.	Deadlines are grouped together. So no chance to miss the deadlines.	Efficient then both EDF& gEDF.
Even small process need to wait long for its turn.	Deadline groups are scheduled independently.	Domain groups are scheduled independently.
Scheduling order will not be efficient.	It is efficient. However, there is no proper domain matching process.	It solved both the problem.

6. CONCLUSION

Non-preemptive scheduling is more efficient for soft real time systems where in context switching overhead is totally eliminated. As, the Earliest Deadline First scheduling algorithm is based on preemptive scheduling, it works fine only when the system is slightly loaded. Group Earliest Deadline First algorithm is based on non-preemptive scheduling solves this problem by grouping tasks with the same deadlines to be met. However, Group Earliest Deadline First algorithm doesn't consider the tasks in a domain-specific manner. As proposed in this paper, Domain Clustering Group Earliest Deadline First (DC-gEDF), groups tasks considering their domain along with their deadlines which eliminates the drawback in gEDF. Moreover, gEDF had no predecessor constraints. But DC-gEDF includes the predecessor constraints and is thereby more efficient than EDF and gEDF.

REFERENCES

[1] Wenming Li, B.S., M.S., GROUP EDF – A NEW APPROACH AND AN EFFICIENT NON-PREEMPTIVE ALGORITHM FOR SOFT REAL-TIME SYSTEMS, UNIVERSITY OF NORTH TEXAS, August 2006.

[2] R. Jain, C. J. Hughes, and S. V. Adve, "Soft Real-Time Scheduling on Simultaneous Multithreaded Processors", In Proceedings of the 23rd IEEE International Real-Time Systems Symposium, December 2002.

[3] K. Jeffay and C. U. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks", Proceedings of the 12th IEEE Real-Time Systems Symposium, San Antonio, Texas, December 1991, IEEE Computer Society Press, pp. 129-139.

[4] C. D. Locke, "Best-Effort Decision Making for Real-Time Scheduling", CMU-CS-86-134 (PhD Thesis), Computer Science Department, Carnegie-Mellon University, 1986.

[5] S. Zilberstein, "Using Anytime Algorithms in Intelligent Systems", AI Magazine, fall 1996, pp.71-83.

[6] R. Heckmann, M. Langenbach, S. Thesing, and R. Wilhelm, "The Influence of Processor Architecture on the Design and the Results of WCET Tools", Proceedings of IEEE July 2003, Special Issue on Real-time Systems.

[7] J. Nieh and M. S. Lam, "A SMART Scheduler for Multimedia Applications", ACM Transactions on Computer Systems, Vol. 21, No. 2, May 2003.

[8] S. K. Baruah and J. R. Haritsa, "Scheduling for Overload in Real-Time Systems", IEEE Transactions on Computers, Vol. 46, No. 9, September 1997.

[9] B. D. Doytchinov, J. P. Lehoczky, and S. E. Shreve, "Real-Time Queues in Heavy Traffic with Earliest-Deadline-First Queue Discipline", Annals of Applied Probability, No. 11, 2001.

[10] W. T. Chan, T. W Lam, K. S. Liu, P. W. H. Wong, "Resource augmentation analysis of SRPT and SJF for minimizing total stretch in multiprocessor scheduling", University of Liverpool, UK.

[11] L. Sha, R. Rajkumar, and S. S. Sathaye, "Generalized Rate-Monotonic Scheduling Theory: A Framework for

Developing Real-Time Systems", Proceedings of the IEEE, Jan. 1994.

[12] G. Buttazzo, M. Spuri, F. Sensini, "Value vs. Deadline Scheduling in Overload Conditions", Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS 1995), Pisa, Italy, pp. 90-99, December 5-7, 1995.

[13] S. Baskiyar, N. Meghanathan, "A Survey of Contemporary Real-Time Operating Systems", Informatica 29 (2005) 233-240.

[14] IEEE Information Technology – Portable Operating System Interface (POSIX): IEEE/ANSI Std 1003.1, 1996 Edition.

[15] S. Agrawal, P. Bhatt, K.K Shukla, "Modified MUF and EDF Algorithms for Overload Soft Real Time", WSEAS Conferences on Recent Advances in Systems, Communications and Computers, April 6-8 2008.

[16] J. H. Anderson, V. Bud, U. C. Devi, "An EDF-based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems", 17th Euromicro Conference on Real-Time Systems, 2005. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[17] G Buttazzo, Research Trends in Real Time Computing for Embedded Systems, 2006

[18] F. Balarin, L. Lavagno, P. Murthy, and A. S. Vincentelli, "Scheduling for Embedded Real-Time Systems", IEEE Design & Test of Computer, January-March, 1998.

[19] N. C. Audsley, A. Burns, M. F. Richardson, A. J. Wellings "Hard Real-Time Scheduling: The Deadline-Monotonic Approach (1991)", Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software

[20] P. Brucker, "Scheduling Algorithms", Third Edition, Springer, 2001.