An Efficient Scheduling Strategy for Overloaded Real Time System

Vijayshree Shinde
PG Scholar, Department of Computer Engineering
Terna Engineering College, Navi Mumbai, India

Seema C. Biday, PhD
Professor, Department of Electronics Engineering
Terna Engineering College, Navi Mumbai, India

ABSTRACT
Scheduling is a technique which makes an arrangement of performing certain tasks at specified period. The intervals between each function have been clearly defined by the algorithm to avoid any overlapping. The bound in which real time applications are needed to respond to the stimuli is known as deadline. In order to achieve optimized results in real time operations the various scheduling techniques are developed. Earliest Deadline First algorithm is optimal scheduling algorithm for single processor real time systems when the systems are preemptive and underloaded. The limitation of this algorithm is, its performance decreases exponentially when system becomes slightly overloaded.

The objective of work is to achieve optimum performance in underloaded condition and achieve high performance in overloaded condition. Proposed algorithm is design for scheduling periodic task on uniprocessor platform. With this algorithm we group jobs with nearly identical deadline and execute the jobs of a group by determining both slack time and deadline of job is another approach. The performance of the proposed algorithm is measured in terms of miss count, average response time, average waiting time and number of preemption count with existing Earliest Deadline First and Group Priority Earliest Deadline first scheduling algorithm. Results are presented by comparing proposed algorithm with other real-time algorithms including, EDF and GPEDF. The Proposed algorithm improves the success count and decrease miss count more than 10% compared with GPEDF and more than 30% compared with EDF.

Keywords
Real-time scheduling algorithms, deadline, overload condition, EDF, GPEDF.

1. INTRODUCTION
A real-time system has to give quick response to its requested services and complete its work within a specific period of time. The most important attribute of real-time systems is that the correctness of such systems depends not only on the computed results but also on the time at which results are produced. Since the timing constraints are the most important characteristic of real-time systems, they are classified as hard or soft according to the usefulness of the computed results produced after the timing deadlines. In hard real-time systems, all jobs must complete execution prior to their deadlines; a missed deadline constitutes a system failure. Such systems are used where the consequences of missing a deadline may be serious or even disastrous. In a soft real-time system, the behavior after missing a timing deadline reduces the quality of service but the operational results could still be useful, or at least do not compromise the integrity of the system, such as in a multimedia system where an occasional delay in the display of a frame decreases the video's quality but will not result in a disaster. This paper focuses on soft real-time systems which do not have stringent timing requirements that must be guaranteed.

Real-time systems are distinguished based on their implementation. In preemptive systems, tasks may be preempted by higher priority tasks, while non-preemptive systems do not permit preemption. The simpler policy like non-preemptive scheduling is easier to implement but might cause blocking time to safety critical tasks. This is due to the large schedulability overhead imposed by non-preemptive scheduling, it also leads to the higher priority task misses its deadline. It is our contention, however that preemptive scheduling is more efficient than the non-preemptive approach since the preemptive model always allows the higher priority tasks to preempt currently running lower priority task.

However, preemptive EDF techniques have produced near optimal schedules for periodic and aperiodic tasks, particularly when the system is lightly loaded. When the system is overloaded, it has been shown that the EDF approach leads to very poor performance (i.e., low success rates) [1]. The poor performance of EDF is due to the fact that, as tasks that are scheduled based on their deadlines, miss their deadlines, other tasks waiting for their turn are likely to miss their deadlines also – an outcome sometimes known as the domino effect. In this paper a proposed methodology is used to improve the success rate of EDF algorithm. A new approach for scheduling soft real-time systems is based on task grouping. Grouping approach is related to GEDF algorithm [2], where jobs with identical near deadlines are queued together and later execute them using another novel approaches i.e SJF, Least Laxity First algorithm. We are in the opinion that, dynamically grouping the jobs by means of incorporating as many as dynamic parameters into the grouping algorithm would boost the performance of soft real time system.

The rest of paper is organized as follows: Section 2 explains the literature studied. In section 3, described real-time task model. Section 4 focuses on introducing real-time task scheduling methodology. Section 5 focuses on results and discussions. Finally, conclusion and future work is provided in section 6.

2. LITURATURE SURVEY
Many researchers have extensively worked on real-time scheduling algorithms. The Earliest Deadline First (EDF) algorithm is a priority driven algorithm in which higher priority is assigned to the request that has earlier deadline, and a higher priority request always preempts a lower priority one [3]. EDF is widely studied as a dynamic priority-driven scheduling scheme. EDF is more efficient than many other scheduling algorithms, including static Rate-Monotonic (RM) scheduling algorithm [4]. For preemptive tasks, when the system is lightly loaded, EDF is able to reach the maximum

possible processor utilization. However, when the processor is over-loaded (i.e., the combined requirements of pending tasks exceed the capabilities of the system) EDF performs poorly [5]. C. D. Locke proposed A Best-Effort algorithm [6] is based on the assumption that the arrival probability of a high value-density task is low. The value-density is defined by V/C, where V is the value of a task and C is its worst-case execution time. Given a set of tasks with defined values if completed successfully, it can be shown that a sequence of tasks in decreasing order by value-density will produce the maximum value when compared to any other scheduling technique. The Best-Effort algorithm admits tasks based on their value-densities and schedules them using the EDF policy. When higher value tasks are admitted, some lower value tasks may be deleted from the schedule or delayed until no other tasks with higher value exist. One key consideration in implementing such a policy is the estimation of current workload, which is either very difficult or very inaccurate in most practical systems that utilize Worst Case Execution Time (WCET) estimations. Best-Effort has been used as an overload control strategy for EDF i.e., EDF is used when a system is underloaded but Best-Effort is used when the overload condition is detected.

Other approaches for detecting overload and rejecting tasks were reported in [7, 8]. G. Buttazzo, M. Spuri, and F. Sensini proposed Guarantee scheme in that, the load on the processor is controlled by acceptance tests on new tasks entering the system. If the new task is found schedulable under worst-case assumptions, it is accepted otherwise, the arriving task is rejected. In the Robust scheme [8] S. K. Baruah and J. R. Haritsa, proposed the system in that, if the system is underloaded, the acceptance test is based on EDF; if the system is overloaded, one or more tasks may be rejected based on their importance. Because the Guarantee and Robust algorithms also rely on computing the schedules of tasks that are based on worst-case estimates, they usually lead to underutilization of resources. Thus Best-Effort, Guarantee, or Robust scheduling algorithms are not good for soft real-time systems or applications that are generally referred to as “anytime” or “approximate” algorithms [9]. In 2004 Peng Li and Binoy Ravindran developed new best effort scheduling algorithms [10] called MDASA (Modified Dependent Activity Scheduling Algorithm) and MLBESA (Modified Locke’s Best Effort Scheduling Algorithm), are novel in the way that they heuristically, yet accurately, mimic the behavior of the Dependent Activity Scheduling Algorithm (DASA) [11] and Locke’s Best Effort Scheduling Algorithm (LBESA)[6] algorithms, but are faster with O(n) and O(n log(n)) worst-case complexities, respectively. MDASA and MLBESA perform almost as well as DASA and LBESA, respectively. However under highly bursty and heavily overloaded situations, DASA and LBESA may perform MDASA and MLBESA. Furthermore, the process response time under MDASA and MLBESA are also found to be very close to the values under their counterpart scheduling algorithms. While MDASA has better performance than MLBESA and has better worst case complexity, MLBESA guarantees the optimal schedule during underload condition. G. C. Buttazzo [12] state best effort algorithm is efficient for soft real time applications. A best effort scheduling algorithm tries to do its best to meet deadlines, but there is no guarantee of finding a feasible schedule. In 2008 S. Agrawal, P. Bhatt, and K.K Shukla [13] developed Modified EDF they Combining Random Dropping (RD) with EDF during overloaded for soft real time application. Basically, a job is segmented into a mandatory and optional part, where the optional part is subjected to abortion at the benefit of meeting the deadline. Ketan Kotecha and Apurva Shah [5] applied the Ant Colony Optimization algorithm on real time operating system in 2010. ACO are computational models inspired by the collective foraging behavior of ant. ACO are the most appropriate for scheduling of task in soft real time system. During simulation result it has been observed that the ACO algorithm is equally optimal during underloaded condition and it performs better during overloaded condition. Many researchers worked on overload condition to improve the system performance, Devendra Thakor and A. Shaha [14] proposed D_EDF scheduling algorithm which combines Earliest Deadline First with Deadline Monotonic. Switch between the algorithms by recording the deadline miss count and deadline meet count. If two jobs miss the deadline continuously occur then switch Earliest Deadline First to Deadline Monotonic. If ten jobs achieve the deadline then switch back to EDF. In this way, advantages of two algorithms combine to speed up overall performance. In 2008 Ketan Kotecha and Apurva Shah [15] proposed the Adaptive algorithm which is the combination of two scheduling algorithms: Earliest Deadline First algorithm (EDF) and Ant Colony Optimization (ACO) Scheduling algorithm. During under loaded condition, the algorithm uses EDF algorithm i.e. priority of the job will be decided dynamically depending on its deadline. During overloaded condition, it uses ACO based scheduling algorithm i.e. priority of the jobs will be decided depending on the pheromone value laid on each schedulable task and heuristic function. Initially the proposed algorithm uses EDF algorithm considering that the condition is not overloaded. But when a job has missed the deadline, it will be identified as overloaded condition and the algorithm will switch to ACO based scheduling algorithm. There are various efficient methods to overcome the problem of overloaded condition of the real time system but these algorithms also have some drawback. Some algorithms required detection of system condition and switching time between algorithms. It increases the overhead of system.

3. REAL-TIME TASK MODEL

Let T= $\{T_1, T_2, T_i, \ldots. T_n\}$ be a set of N periodic task in a uniprocessor system. The tasks are mutually independent and the processor time is the only resource that needs to be scheduled. Each task T_i is defined as $T_i = (C_i, P_i, D_i)$, where C_i is its execution time, P_i is its period and D_i is its deadline, $C_i \leq D_i$.

4. REAL-TIME TASK SCHEDULING ALGORITHMS

A real-time system will usually have to meet many demands within a bound time. The significance of the demands may vary with their nature (e.g. a safety-related demand may be more important than a simple data-logging demand) or with the time available for a response. So the allocation of the system resources needs to be planned so that all demands are met by the time of their respective deadlines. The scheduling is done using a scheduler which implements a scheduling policy that defines how the resources of the system are allocated to the program. Scheduling policies are revealed mathematically so the accuracy of the formal specification and program development stages can be complemented by a mathematical timing analysis of the program properties.

4.1 Earliest Deadline First scheduling algorithm

In 1973 Liu and Layland, suggested the most popular real time scheduling algorithms Earliest Deadline First (EDF) [3].

EDF is a dynamic priority algorithm in which task with the earliest deadline has the highest priority. EDF is an optimal uniprocessor scheduling algorithm. The optimal scheduling algorithm gives 100% CPU utilization. EDF algorithm gives best performance and minimize miss ratio, when systems operating under low or moderate levels of resource and data contention. However, the performance of Earliest Deadline First algorithm is suddenly degraded in an overloaded system. This is because, under heavy loading, tasks gain high priority only when they are close to their deadlines.

Algorithm

- Enter number of $_{\text{tasks}}$ n.
- For n tasks enter different times as periodic time, execution time and deadline time.
- Check the schedulability condition
$$U=\sum_{i=1}^{n} \frac{Ci}{Pi} \leq 1$$
- If U <=1 then EDF is schedulable else not schedulable.
- Arrange different tasks in ascending order of deadline.
- Then schedule the task according to its deadline and release time.
- If execution time of current task comes in between the release time of next task then, check deadline of current task and next task.
- If deadline of current task is less than deadline of next task then, current task execution continues else next task is executed.
- This loop repeats till i<n condition satisfies.
- End Loop.

Consider Table 1, which represents a sample tasks set that will be used as common example throughout this paper to better understand the differences among real time task scheduling approaches. This task set is schedule using fully pre-emptive Earliest Deadline First Scheduling algorithm show in figure 1.

Table 1. The repetition period, computation time and deadline of the tasks T1, T2, T3, T4

Tasks	C_i	D_i	P_i
T1	3	4	4
T2	3	5	5
T3	3	6	6
T4	3	7	7

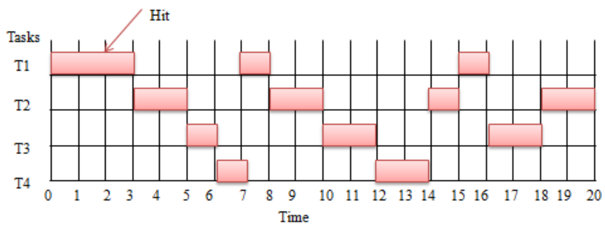

Fig 1: Schedule generated by the EDF algorithm

Schedule generated by the EDF algorithm is shown in Figure 1. T1 with minimum deadline is executed first followed by task T2. At t=4 T1 is activated but still T2 is executed because deadline of T2 is less than T1. At t=5 task T2 is activated, here again deadline of T3 is less than T1 and T2 so that service is given to task T3. At t=6 Task T3 is activated but T4 having the next least deadline than T1, T2, and T3. So priority is given to task T4. From this figure 4.14 we say that if we apply EDF on such a task set then there is lot of chances of miss count. As we see up to 20 times unit only 1 task is executed successfully that is task T1. This task set will be executed based on their deadline until the total time elapses.

4.2 A Group Priority Earliest Deadline First scheduling algorithm

GPEDF perform schedulability test prior to grouping a particular job. Following method is used to solve the problem of how to group jobs together [19].

$$j=1,2,\ldots,N \sum_{i=1}^{j} \frac{C_i}{D_i} + \frac{C_{sum}+C_{ex}}{D_j} \leq 1 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2)$$

All the jobs in job set J_t behind first job in a set can be executed before first job and the system will still be schedulable, where C_{sum} is the sum of the execution times of the jobs in job set except first job in job set, and C_{ex} is the sum of execution time of the jobs that would be ready later than time t and have absolute deadline shorter than last job in job set. If there is a job set J_t which satisfies eq. (2) at time t, the order of the jobs in job set J_t can be changed randomly. This means that the jobs can be form in job set Jt into a group, in which the jobs can be reordered as required without reducing the schedulability. GPEDF scheduling algorithm can be described in three parts as follows;

First part is enqueue, when a new job arrives, enqueue sort new job into Jt. The second dequeue and third exqueue methods invoked every time unit. dequeue deletes the jobs which have absolute deadlines shorter than current time t. exqueue creates group of jobs and execute shortest job first algorithm within groups. When there is no group in the system, the jobs are put into group one by one according to the order they appeared in job set Jt and will be stopped until one job cannot satisfy Eq. (2). If a job cannot form a group with other jobs, then it forms the group with itself. If the system is overloaded and when there is only one job in the group at that time a job may not be completed successfully because the remaining time may not be enough for the job to execute.

In the GPEDF scheduling algorithm, jobs with short execution time can be executed first in the group, which leaves more time for other jobs to execute. This allows more jobs to be completed, the number of switches is decreased and the response is reduced.

Following figure shows the detail working of GPEDF algorithm for task shown in table 1. In figure 2, T1, T2, T3 and T4 arrives at t=0. T1 form a group by itself and run up to three time unit. T2 also form a group by itself and run t=3 to 4, during the execution of T2 task T1 is activated but current group contains T2 with highest priority value so that T2 is continue up to 5 time unit. At t =5, tasks T2 is activated , but latest group contains T3 with highest priority hence, T3 run up to 1 time unit and miss its deadline. Again at t =6, tasks T3 is activated , but latest group contains T4 with highest priority hence, T4 run up to 1 time unit and miss its deadline. Likewise task set shown in table 1 are scheduled by GPEDF algorithm.

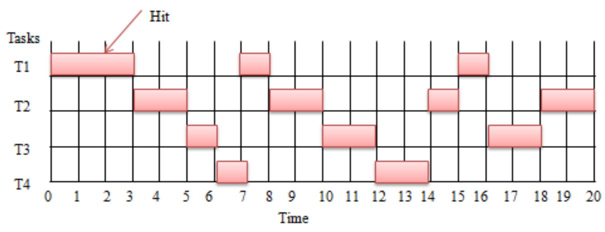

Fig 2: Schedule generated by the GPEDF algorithm

From this figure we can see that GPEDF performed same as EDF for given task set.

4.3 Proposed System

Proposed system is model on uniprocessor system with preemptive periodic task, which compose of job activation, a scheduler, dispatcher, task execution and last one is task termination. In job activation, jobs are activated based on the specified input parameters. Parameters compose of the common four tupples of a real time task namely release time, execution time, relative deadline and period basically we consider release time as 0.

gEDF scheduling algorithm [2] use Group range parameter (Gr). So, we believe that, instead of using Gr, which is a static parameter, incorporating parameters that changes dynamically throughout the scheduling process into the algorithm would produce much better result. Historically, dynamic algorithm has been shown to be more efficient in terms of Deadline Meeting Task Rate than static algorithm [18]. It follows, we then seek to determine all the dynamic parameters which affect the system scheduability and we detected two parameters [17].

- One is the left over time available after the first job executes denoted by:
  (Di – Ci)
- Second, is another dynamic value, which is amount of load on the system competing to fit in the left over time denoted by U.

From these parameters, by means of intuition we design

our algorithm into the following logic. Assuming Ti is the job at the head of group, and then Tj is a member of this group if and only if it satisfied the equation 3.

$D_i <= D_j <= (D_i +(Ut *(D_i – C_i))$…………………………..(3)

In the above equation we have set up lower and upper limit of the dynamic deadline (D_j) of a job T_j which gets into a group of jobs. The lower limit of T_j says that, dynamic deadline must be greater than or equal to the dynamic deadline of the head job. Apart from that, the upper limit of T_j must not be greater than the current value of the left over time upon the execution of the head job multiply by the dynamic current load. This is what we meant by incorporating dynamically changing parameters as determinant of which jobs gets into a group [17].

In proposed system we first group the jobs with near deadlines together based on the above equation (3). Within the group the jobs were sorted based on execution value, where jobs with smaller execution value will be executed first it means SJF is used to improve the performance of the system. In case execution times of tasks are same then we need to apply LLF. Following figure shows the detail working of proposed algorithm for task set shown in table 1.

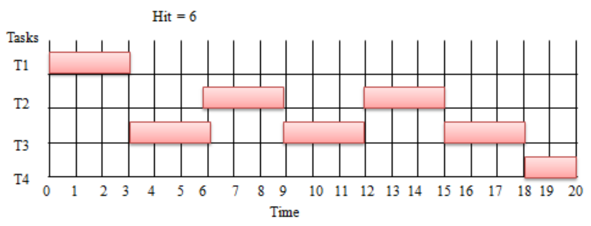

Fig 3: Schedule generated by the proposed algorithm

In figure 3, T1, T2, T3 and T4 arrives at t=0. Task T1,T2 and T3 form a group, all these tasks has same execution time so, tasks are schedule according to laxity. T1 executed first as laxity of T1 is less than T2 and T3. At t=3, laxity of task T2 is negative so that this task is deleted from a queue and priority is given to task T3. At t=4 task T1 is activated but priority is given to task T3 as remaining execution time of task T3 is less than T1. Likewise task set shown in table 4.3 are scheduled by proposed algorithm. The above schedule shows that how and when we used LLF algorithm in proposed system. Here up to 20 time unit 6 task executed successfully only because of LLF logic.

5. RESULTS AND DISCUSSIONS

To evaluate the effectiveness of the proposed algorithm, we performed a series of experiments with different kinds of task sets. We have implemented our algorithms using java programming in NetBeans environment and run simulations to accumulate empirical data. The result of the proposed algorithms is compared with each other in the same environment. Then we report the results; analyze the sensitivity of proposed algorithm along with various parameters used in the experiments, the effects of the percentage of CPU utilization on the miss count, response time, waiting time and how well proposed algorithm performs when compared to EDF and GPEDF algorithms. In this experiment we use the periodic preemptive task model. In which, load of the system is defined as summation of ratio of executable time and period of each task. In our simulation experiments, we assume that:

- All tasks are periodic with deadline parameter of each task equals to its request period.
- All tasks start simultaneously at time zero.

5.1 Performance Evaluation Parameter

A reasonable way to measure the performance of a scheduling algorithm during an overload is the amount of work the scheduler can feasibly schedule according to the algorithm, therefore number of miss task (miss count), average response time, average waiting time and context switching are considered as our main performance measuring criteria and defined as;

- Tasks which are not completed their deadline are called as miss tasks.
- The response time is the time interval between the task execution request (equal to the ready time) and the end of task execution.
- Average Waiting Time is the time spends waiting for the ready queue. It is the sum of the periods spent waiting in ready queue.

5.2 Experimental Results and Analysis

Main goal of proposed algorithm is to determine how well the algorithm performs with respect to their counterpart EDF and GPEDF scheduling algorithms. We conduct simulation studies to determine above performance metrics using different workloads.

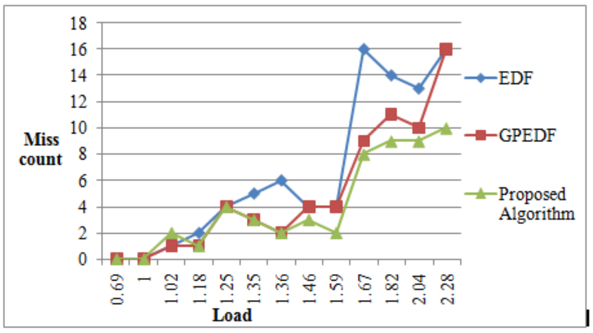

Fig 4: Miss count comparison between three algorithms when Load <=1 to >1

The Our observation from figure 4 reveals that during light load the miss count of EDF, GPEDF and Proposed algorithm are almost identical, where these algorithms manage to successfully schedule almost 100% of the jobs. No doubt, that there are some differences which are very tiny, which we consider negligible. However, when system load is increases, the miss count of EDF and GPEDF starts to increase more abruptly as compared to proposed algorithm. Proposed algorithm minimizes more than 10% miss count of GPEDF algorithm, where as when compared with EDF this value is more than 30%. The SJF used in both proposed algorithm and GPEDF means that more jobs with short execution time are completed successfully than in EDF. The grouping method used in proposed algorithm ensures that every job in a group can be successfully completed, which gives it a higher success count and less miss count than EDF.

Here another parameter of interest is analyzed, which average response time is shown in figure 5. Logically, executing jobs with shorter execution time would result in better average response time. We only presented the average response time of those jobs that manages to meet their deadlines. Hence, during overload we see drop in average response time due to less percentage of jobs manages to meet their deadline. Also if jobs with relatively shorter Execution Time would be the one which successfully execute and finishes before their deadlines leads to drop in average response time.

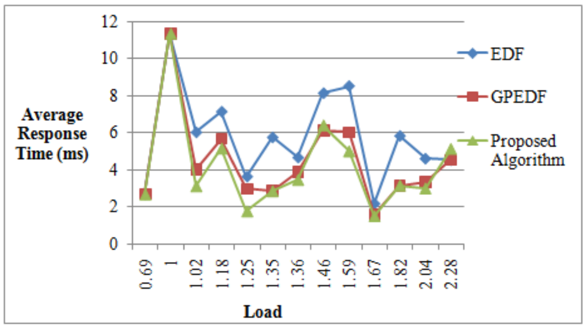

Fig 5: Average response time comparison between three algorithms when Load <=1 to >1

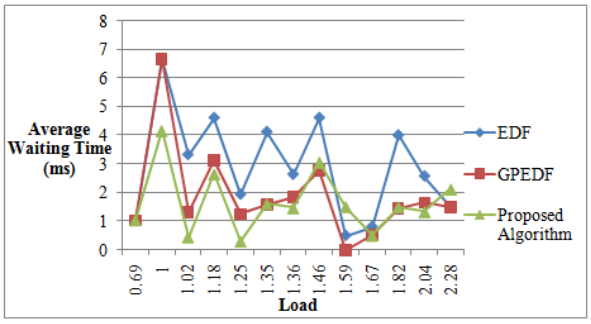

Fig 6: Average waiting time comparison between three algorithms when Load <=1 to >1

Figure 6 shows that waiting time of proposed algorithm is less than EDF and little bit less than GPEDF. Proposed algorithm outperforms EDF in terms of the waiting time, when the system is loaded. As we seen from the above results, if response time of algorithm is less obviously waiting time of that algorithm is also less.

6. CONCLUSION

We developed a new real-time scheduling algorithm that combines Shortest Job First scheduling, Earliest Deadline First scheduling and Least Laxity First Scheduling algorithms. Tasks were grouped together with deadlines that are very close to each other, and scheduled jobs within a group based on using SJF scheduling. In case execution ties occur between the tasks then LLF scheduling algorithm is used. Based on the experimental results, proposed algorithm has lower miss count as well as faster response times with less waiting time.

EDF produces practically acceptable performance even for preemptive systems when the system is underloaded, EDF performs get reduced when the system is heavily loaded. Proposed algorithm performs is same as EDF in terms of miss rate when a system is underloaded. In addition, proposed algorithm consistently outperforms EDF in overloaded systems. We also compared proposed algorithm with Group Priority Earliest Deadline First scheduling algorithms in underloded as well as overloaded condition. In general, proposed algorithm used under all system loads that show performance similar or better than EDF and GPEDF. Our test results show that proposed algorithm can be used effectively in real world systems. The performance of the proposed algorithm is better than EDF and little bit better than GPEDF with respective miss count, average response time and average waiting Time.

Future Work

In proposed algorithm number of preemption is more because of least laxity first. So, in future a new algorithm should be developed that will minimize context switching under overloaded as well as underload system condition.

7. REFERENCES

[1] Apurva Shah and Ketan Kotecha, “Scheduling Algorithm for Real –Time Operating Systems using ACO”, International Conference on Computational Intelligence and Communication Networks, 2010, pp. 617 – 621, DOI 10.1109/CICN.2010,122.

[2] Wenming Li, Krishna Kavi and Robert Akl, “A non-preemptive scheduling algorithm for soft real-time systems”, ELSEVIER Article in Press Computers and Electrical Engineering, 2006.

[3] Liu and J.W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment”, Journal of the ACM, 1973,20(1):46-61.

[4] T. Abdelzaher, V. Sharma, and C. Lu, “A Utilization Bound for Aperiodic Tasks and Priority Driven Scheduling”, IEEE Transaction on Computers, Vol. 53 No. 3, March 2004.

[5] Apurva Shah and Ketan Kotecha, “Scheduling Algorithm for Real –Time Operating Systems using ACO”, IEEE International Conference on Computational Intelligence and Communication Networks, pp. 617 - 621 DOI 10.1109/CICN.2010,122.

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

[7] G. Buttazzo, M. Spuri, and F. Sensini, Scuola Normale Superiore, Pisa, Italy, “Value vs. Deadline Scheduling in Overload Conditions”, Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS’95) Pisa, Italy, pp. 90-99, December 1995.

[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] S. Zilberstein, “Using Anytime Algorithms in Intelligent Systems”, AI Magazine, 1996, pp.71-83.

[10] Peng Li and Binoy Ravindran, “Fast, Best-Effort Real-Time Scheduling Algorithms”, IEEE Transaction on Computers, Vol. 53, No. 9, September 2004.

[11] R. K, Clark, “Scheduling Dependent Real-Time Activities,” PhD Dissertation, Carnegie Mellon University, 1990.

[12] G.C. Bhttazzo, “Hard Real Time Computing System: Predictable Scheduling algorithms and applications”, IEEE Transactions On Computers, Vol. 53, No. 5, May 2001.

[13] S. Agrawal, P. Bhatt, and 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, pp. 99-104.

[14] D. Thakor and A. Shah, “D_EDF, An efficient scheduling algorithm for real-time multiprocessor system”, 2011 World Congress on Information and Communication Technologies (WICT), pp. 1044-1049.

[15] Ketan Kotecha and Apurva Shah, “Adaptive Scheduling Algorithm for Real-Time Operating System”, IEEE World Congress on Computational Intelligence, pp. 2109 - 2112, DOI: 10.1109/CEC.2008.4631078.

[16] http://c2.com./cgi/wiki?RealTime

[17] Zahereel Ishwar Abdul Khalib, Badlishah R Ahmad, and Ong Bi Lynn Ong , “High Deadline Meeting Rate of Non-Preemptive Dynamic Soft Real Time Scheduling Algorithms”, IEEE International Conference on Control System, Computing and Engineering, Penang, Malaysia, 2012, pp. 296 – 301.

[18] G. C. Buttazo, “Rate Monotonic vs EDF: Judgment Day”, Real Time Systems, Vol.29 No.1, pp.5-26, January 2005.

[19] Qi LI and Wei BA, “A group priority earliest deadline first scheduling algorithms”, Research article of frontiers of Computer Science, 2012, 6950: 560-567 DOI 10.1007/s 11704-012-1104-4.