Reducing Preemptions in EDF Scheduling in Uniprocessor Environment

Dhroov Pandey

I. INTRODUCTION

Real time systems require enforcement of deadline constraints to ensure functionality. The system is composed of tasks that need to be scheduled effectively to meet these deadlines. There are various scheduling policies utilized to optimize schedulability of a task set. These include fixed priority policies such as Rate Monotonic, Deadline Monotonic, Shortest Execution Time First, table based and dynamic priority policies such as Shortest Remaining Execution Time First and Earliest Deadline First (EDF). Table based policies require apriori knowledge of the entire expected task set which can limit runtime behavior; on the other hand fixed priority policies cannot guarantee schedulability for task sets that have high CPU utilization which is suboptimal for performance. Only EDF guarantees schedulability for task sets with 100% CPU utilization. This theoretical utilization upper bound makes EDF desirable, however, there are two important requirements to achieve this: (i) The scheduler must be fully preemptive, i.e. a high priority job must be able to immediately preempt a lower priority job, (ii) The utilization bound assumes that there is no overhead for preemption. These requirements create a problem for us since (ii) is not satisfied by real systems; they always have overhead for context switches. This implies that even if we satisfy (i), we will not have the theoretical upper bound in practice. Furthermore, we find that satisfying (i) in the absence of (ii) can leave our system vulnerable to pathological schedules which can severely deteriorate performance and throughput.

Our problem mainly reduces to minimizing preemption overhead. Since reducing the overhead for context switch is a hardware problem, in this work we focus on lowering preemption overhead by lowering the number of preemptions. In particular, we state our problem as follows:

Problem Statement: Minimize the number of preemptions in an EDF scheduling framework while maintaining a high utilization upper bound.

In addition to schedulability for heavy traffic workloads, lowering the preemption overhead enhances system performance by improving response times for background tasks.

There are various alternatives to fully preemptive EDF (FP-EDF) that seek to minimize preemption overhead. Non preemptive EDF (NP-EDF) uses the earliest deadline scheduling but disables preemptions; meaning that when no job is running, the job with the earliest deadline starts executing, but if a high prioirty job becomes available while a low priority job is executing, it must wait for the low priorit job to complete. This eliminates preemption overhead but loses the schedulability guaratees. Limited preemption EDF [1] (LP-EDF) seeks to minimize preemptions by using a preemption function to mark non-preemptive execution zones. Controlled Preemption EDF [2] (cp-EDF) controls preemptions based on the preempting job instead of the preempted job. Fixed Point Preemption EDF [3] (fpp-EDF) assigns fixed preemption points where preemptions are possible, else the system executes non-preemptively. Preemption threshold EDF [4] (pt-EDF) assigns threshold priorities to tasks to change their effective priorities when being preempted to bias the system towards fewer preemptions. All these frameworks lower preemptions by compromising either the runtime dynamicity of the system (i.e. need comprehensive tast set information apriori) or by severely hindering the utilization bound. We propose Backlogged Density EDF (BD-EDF) which operates on the principle of queuing arriving higher priority tasks and tracking the backlogged density in the system to decide when to release the backlogged queue, effectively creating zones of non-preemptive execution based on runtime information. We show that BD-EDF will only cause deadline misses if FP-EDF would have also caused a deadline miss (i.e. for unschedulable task sets).

II. RELATED WORK

In this section we explore existing policies to reduce preemption overhead and highlight their limitations.

A. Limited Preemption EDF

t	$Q(t)$
$[0, 8)$	∞
$[8, 10)$	6
$[10, 60)$	4
$[60, \infty)$	2

Fig. 1: An example of non-preemption function Q for LP-EDF

LP-EDF utilizes a non-preemption function $Q : R \rightarrow R$ which accepts as input the time to deadline for the executing job and returns the duration for which the job can continue executing non-preemptively. For example, if a job with 7 time units to deadline is running and a job with 6 time units to deadline arrives, the scheduler checks $Q(7) = \infty$ in Fig. 1 and lets the lower priority job continue till it finishes execution. LP-EDF ensures that there are no deadline misses as it generates Q based on the demand bound analysis of all tasks

in the task set. While this limits preemptions in the system, the policy relies on having apriori knowledge of task behaviors; this limits the runtime dynamic behaviors for task sets and is effectively a table driven scheduling policy.

B. Controlled Preemption EDF

cp-EDF assigns a boolean preemption parameter X to every task in the task set. When a job from that task set arrives as a high priority job and wants to preempt a running low priority job, the scheduler checks the value of X to determine whether preemption occurs or not. This strategy allocates the deciding factors for preemption to the higher priority (preempting) job instead of the low priority (preempted) job. However, assigning the optimal X values to tasks requires traversing an exponential search space (2^n configurations for n tasks) which is not feasible. Therefore, cp-EDF uses various heuristics to manage the search space which provide near optimal but not optimal configurations. Furthermore, we have the same problem as LP-EDF of requiring apriori knowledge of runtime behavior, as well as a high initial overhead of configuring near-optimally.

C. Fixed Point Preemption EDF

fpp-EDF introduces fixed preemption points in the schedule where preemption is possible. This allows the system to execute non-preemptively except at predefined points. Preemption points can be defined globally or locally. Global preemption points allow any high priority task to preempt a low priority task whereas local preemption points are defined internally in a task (by a programmer for instance) and a low priority task can only be preempted at its local preemption points. There are various sub-policies within fpp-EDF that optimize system preemptions by defining other criteria that can influence whether preemption should occur at a fixed point or wait till the next fixed point. Eager preemption policies bias the system towards more aggressive preempting whereas lazy preemption policies bias the system towards lower preemptions but can cause deadline misses. Global preemption points are hard to define as they require complex system analysis at high traffic critical points and local preemption points rely on being accurately inserted by the programmer and leaves the system vulnerable to deadline misses due to internal errors of a task.

D. Priority Threshold EDF

pt-EDF assigns threshold priorities to tasks which act as their effective priorities when they are executing. For instance, if a task with priority 3 and threshold priority 2 is running, then a task with priority >5 can preempt it; however, it can still only preempt tasks with effective priority <3. This biases the system towards fewer preemptions. The critical problem with this scheme is defining optimal threshold values. The search space for a comprehensive search is too large to traverse and hence, heuristics are employed to find near-optimal thresholds. This also requires task set attribute foreknowledge as well as adds massive initial overhead.

III. Our Policy: BD-EDF

Since the existing methodologies either compromise dynamic runtime behavior or add massive scheduling overhead to limit preemptions in EDF scheduling, we introduce Backlogged Density EDF scheduling which is a lightweight scheduling algorithm to reduce preemptions in a system based solely on information available to it at runtime.

The main idea behind the algorithm is to model a slack based approach where we backlog as many incoming higher priority tasks as possible without causing deadline misses in the future. We define two quantities *backlogged density* and *actual density* as follows:

$$backloggedDensity = \sum \frac{e_i}{d_i - r}$$

$$actualDensity = \sum \frac{r}{d} + \frac{e_i}{d_i}$$

where r is the remaining execution of current executing job, d is the time to deadline, e_i is the execution time of all jobs in backlogged queue and d_i is the respective deadlines. Whenever a new job is available in the ready, if its priority is higher than the current executing job, we calculate backlogged density and actual density values for the system, accounting for the new job. If the backlogged density is more than the actual density then we preempt the running job and send it back to the ready queue. Then, we merge the backlogged queue into the ready queue and follow standard EDF scheduling. Algorithm 1 shows the working of BD-EDF.

Algorithm 1 BD-EDF Scheduling

```
1:  bd ← backloggedDensityQueue
2:  curr ← currentlyRunningTask
3:  rq ← readyQueue
4:  h ← newlyArrivedTask
5:  if h.priority ≤ curr.priority then
6:      rq.add(h) return
7:  end if
8:  bd.add(h)
9:  if actualDensity < backloggedDensity then
10:     rq ← merge(rq, bd)
11:     preempt(curr)
12:     execute(rq.top) return
13: end if
```

A. Schedulability Analysis

In order to show that BD-EDF matches FP-EDF in schedulability we have two main claims:

(I) *If a task set is schedulable under FP-EDF, then it is schedulable by BD-EDF*

(II) *Actual density and Backlogged density have a theoretical convergence point guaranteeing termination conditions*

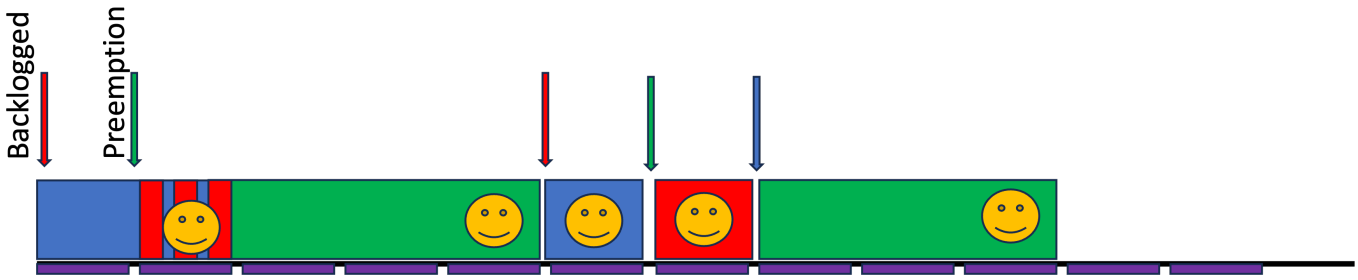


Fig. 2: BD-EDF Scheduling Illustration

Proof(I): Let the current running task have time r remaining for execution and d units till deadline. Without loss of generality let the current time be 0. If a high priority task is added to the backlogged queue then for a deadline miss to occur, new utilization density l_{new} should be added in the period (r, d) such that $l_{new} + l_{backlogged} > 1$ or $l_{new} + \sum \frac{e_i}{d_i - r} > 1$ but this would mean $l_{new} + l_{actual} > 1$ since BD-EDF non-preemption condition requires $l_{backlogged} \leq l_{actual}$. $l_{new} + l_{actual} = l_{fp}$ is the density for FP-EDF; since $l_{fp} > 1$, the task set is not schedulable under FP-EDF. Therefore, we prove that if a deadline miss occurs in BD-EDF, then a deadline miss will occur in FP-EDF.

Proof(II): We show that the values converge by demonstrating that the rate of increase of actual density is higher than the rate of increase of backlogged density. Whenever a new job is added to the backlogged queue, actual density goes up by $\frac{e_i}{d_i}$ whereas backlogged density increases by $\frac{e_i}{d_i - r}$. Since $\frac{e_i}{d_i} \geq \frac{e_i}{d_i - r}$, we have proved that the values for actual density and backlogged density converge and the terminating condition is guaranteed to be fulfilled.

B. An Example

To better understand the working of the BD-EDF algorithm, we walk through an example. Consider Fig. 2, we use three tasks in our task set: $Red(5, 1)$, $Green(6, 3)$, and $Blue(9, 3)$ where (p, e) represent (period, execution time), and we assume the period to be equal to the deadline. At time $t = 0$ we have $Blue$ as the current executing task with 2 units of execution remaining and Red is made available in the ready queue. According to Algorithm 1 we compute the actual and backlogged density: Actual density = $\frac{1}{5} + \frac{2}{8} = 0.45$, and Backlogged Density = $\frac{1}{5-2} = 0.33$. Since the actual density is greater than the backlogged density, we send Red to the backlogged queue and continue executing $Blue$ non-preemptively. At time $t = 1$, $Green$ is made available in the ready queue and has higher priority than blue. We again calculate backlogged and actual density: Actual Density = $\frac{1}{5} + \frac{1}{7} + \frac{3}{6} = 0.84$ and Backlogged Density = $\frac{1}{5-1} + \frac{3}{6-1} = 0.85$. Since the backlogged density is greater than the actual density, we preempt $Blue$ and bring it back to the ready queue. It is important to note that we do not simple replace $Blue$ with $Green$ since there is a backlogged queue, and higher priority tasks may be available there. So, we merge the backlogged queue with the ready queue and continue EDF scheduling which releases Red.

C. For Soft Deadline Systems

We observe that in our scheduling, we constrain our preemption reduction by necessitating meeting deadlines. However, we can lower preemption rates even more aggressively if we relax that constraint. We do so by introducing a threshold parameter that modifies the preemption condition as

$$ActualDensity < BackloggedDensity + Threshold$$

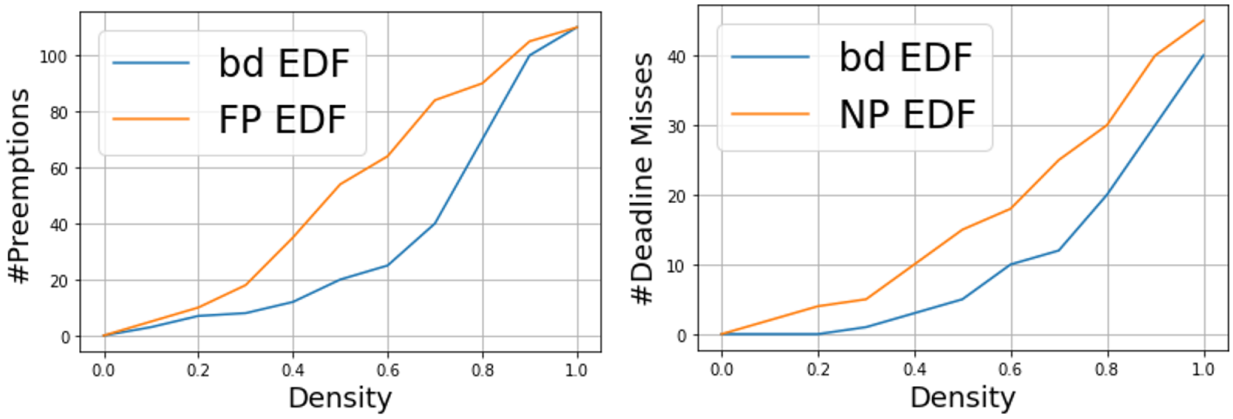
By adding this relaxation, we can no longer guarantee meeting all the deadlines but the *threshold* can be tuned for each task set to minimize deadline misses while heavily lowering preemptions. This can be especially useful for systems that have bounded tolerance for deadline misses. In fact, this modified equation is a super-set of BD-EDF, FP-EDF and NP-EDF, as if we set $threshold = 0$, we get standard BD-EDF, for FP-EDF we set $threshold = \inf$ and for NP-EDF we set $threshold = -\inf$.

IV. Evaluation

We evaluate BD-EDF against FP-EDF and NP-EDF as shown in Fig. 3. We select these policies since they represent the edge cases of EDF scheduling and we observe the efficiency of BD-EDF under the deadline constraints against FP-EDF in terms of the number of preemptions, and its optimality in preserving task hierarchies against NP-EDF in terms of the number of deadline misses.

A. Experimental Setup

To gather our experimental data, we simulate a uniprocessor environment and execute randomly generated task set (with total density < 1) for 40 hyperperiods of the task set. Since every compute timer is discrete fundamentally (at the level of CPU cycles), for simplicity we only consider task sets with integer periods, deadlines, and worst case execution. We make our experimental setup available at https://github.com/BLACKADDER-VIII/EDF-Sched-Policy-Sim.



(a) FP-EDF v BD-EDF

(b) NP-EDF v BD-EDF

Fig. 3: Evaluating BD-EDF against FP-EDF and NP-EDF

B. Results

We observe in Fig. 3 (a) that BD-EDF is able to offer significant compute savings by lowering preemptions for task sets with densities in range $(0.4, 0.8)$ which would characterize the majority of workloads at most of the time. Therefore, we demonstrate the effectiveness of BD-EDF in enhancing a preemptive EDF policy which can reduce preemptions without compromising deadline objectives.

In Fig. 3 (b) we observe that BD-EDF is effective at reducing deadline misses substantively for a low preemption system compared to a non-preemptive policy which struggles against timing constraints. By tuning the $threshold$ hyperparameter effectively, we can ensure low number of preemptions while simultaneously keeping a tight bound around deadline misses.

V. Discussion

While BD-EDF is effective at substantively lowering preemptions for most task densities, we notice that the improvements are marginal at very low traffic and very high traffic workloads. This is due to the fact that at very low traffic loads, the tasks have sufficient room in the hyperperiod to not interfere with each other except at pathological releases, whereas at high traffic workloads, the backlogged density is always going to be very high and trigger the release condition almost immediately. In contrast we notice that the improvement in deadline bounds against NP-EDF is uniform across the density range. This is due to the fact that even at low traffics, unlucky releases can cause priority inversion easily and cause high priority tasks to miss their deadlines. Whereas with BD-EDF and a tuned $threshold$, these scenarios can be avoided. We leave the task of finding an optimal tuning for $threshold$ in relaxed BD-EDF to future work.

VI. Conclusion

In conclusion, we have devised and implemented an EDF scheduling policy that enhances a fully preemptive scheme by eliminating unnecessary preemptions. We have demonstrated that our policy will not compromise meeting deadlines and have showcased its efficacy against two EDF policies.

References

[1] M. Bertogna and S. Baruah, "Limited Preemption EDF Scheduling of Sporadic Task Systems," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 579–591, Nov. 2010. [Online]. Available: https://ieeexplore.ieee.org/document/5466180/

[2] J. Lee and K. G. Shin, "Preempt a Job or Not in EDF Scheduling of Uniprocessor Systems," *IEEE Transactions on Computers*, vol. 63, no. 5, pp. 1197–1206, May 2014. [Online]. Available: https://ieeexplore.ieee.org/document/6361382/

[3] P. Wu, Z. Li, Z. Zhang, T. Yan, and L. Chen, "Three Preemption Approaches towards EDF Scheduling for Homogeneous Multiprocessors," in *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, Dec. 2023, pp. 186–193, iSSN: 2690-5965. [Online]. Available: https://ieeexplore.ieee.org/document/10475967/

[4] G. C. Buttazzo, M. Bertogna, and G. Yao, "Limited Preemptive Scheduling for Real-Time Systems. A Survey," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 3–15, Feb. 2013. [Online]. Available: https://ieeexplore.ieee.org/document/6164261/