Design of an Efficient Ready Queue for Earliest-Deadline-First (EDF) Scheduler

Risat Mahmud Pathan
Chalmers University of Technology, Sweden
Email: risat@chalmers.se

***Abstract*—Although dynamic-priority-based EDF algorithm is known to be theoretically optimal for scheduling sporadic real-time tasks on uniprocessor, fixed-priority (FP) scheduling is mostly used in practice. One of the main reasons for FP scheduling being popular in the industry is its efficient implementation: operations on the ready queue can be done in constant time. On the other hand, ready queue of EDF scheduler is generally implemented as a priority queue, for example, using a binary min-heap data structure in which (insertion/deletion) operation *cannot* be done in constant time.**

This paper proposes a new design of ready queue for EDF scheduler: a simple data structure for the ready queue and efficient operations to insert and remove task control blocks (TCBs) to and from the ready queue are proposed. Insertion of a TCB of a newly released job (that cannot preempt the currently-executing job) is done in non-constant time. However, insertion of a TCB of a preempted job or the removal of the TCB of job having the highest EDF priority from the ready queue can be done in constant time. Simulation using randomly generated task sets shows that the overhead of managing jobs in our proposed ready queue for EDF scheduler is significantly lower than that of other approaches. We believe that theoretically optimal EDF algorithm implemented based on our proposed ready-queue data structure will make EDF popular in industry.

I. Introduction

Meeting hard deadlines while utilizing the CPU capacity as much as possible is a major challenge in real-time tasks scheduling problem. In this paper, preemptive EDF scheduling of a collection of sporadic real-time tasks on uniprocessor is considered. In addition to the execution time of the tasks, *scheduling overhead* must be taken into account during the design of the system so that no deadline is missed due to such overhead at run-time. One such source of scheduling overhead that we consider in this paper is management of tasks in the ready queue.

An EDF scheduler at each time instant executes the job with the earliest deadline. When the processor is busy executing a job and another job with relatively shorter deadline (i.e., higher EDF priority) is released, the currently-executing job is preempted by the new job. Active jobs that cannot be executed due to having relatively lower EDF priorities wait in a *ready queue*. A ready-queue manager inserts and removes the task control blocks (TCBs) of such active jobs to and from the ready queue. Unfortunately, such management of jobs in the ready queue of EDF scheduler is more complex and has higher overhead in comparison to that of FP scheduler (please see the discussion by Buttazzo [2]). Although EDF being optimal [3] can (theoretically) better utilize the CPU, many practical systems implement FP scheduling [5] due to its efficient (i.e., low overhead) run-time support in managing jobs in the ready queue.

To minimize overhead of managing jobs in EDF scheduler, Short [9] proposed different ready-queue data structure considering small systems and may have higher overhead for systems that require larger time representation. Pathan [8] proposed a mechanism to efficiently manage jobs in the ready queue of priority-promotion-based scheduling in which jobs are essentially executed in EDF priority order. This work is extended in [7] by proposing new technique to manage jobs in the ready queue. However, the techniques in [8], [7] require promotion of the priority of each job and each such promotion causes the TCB of that job (if it is in ready queue) to be remapped to a new position in the ready queue. Such remapping incurs overhead. The main endeavor of this paper is to reduce such overhead to exploit full schedulability power of EDF.

Building upon the data structure proposed in [8], efficient (insertion/removal) operations to manage ready jobs for EDF scheduler are proposed. Tasks are scheduled based on EDF priorities and do not require any priority promotion. The ready-queue management using the proposed scheme has the following features: insertion of a TCB of a *newly* released job cannot be done in constant time but the insertion of a TCB of a *preempted* job to the ready queue and removal of the TCB of the *highest EDF priority* job from the ready queue can be done in constant time.

To measure the effectiveness of the proposed ready-queue management scheme, the execution of randomly generated task sets is simulated. The ready queue of EDF is simulated using three alternatives: (i) our proposed scheme (presented in this paper), (ii) the approach proposed in [8] for priority-promotion-based scheduling, and (iii) a priority queue implemented as a binary min-heap. The simulation result shows that ready-queue management of our proposed scheme suffers significantly less overhead in comparison to that of the other two alternatives. Although this paper considers uniprocessor scheduling, the proposed scheme also applies to global EDF scheduling [1] for multiprocessors.

II. System Model and Background

The EDF scheduling of a collection of n sporadic tasks $\tau_1, \ldots \tau_n$ is considered. Each task τ_i has three parameters (C_i, D_i, T_i), where C_i is the worst-case execution time (WCET) of the task, D_i is the relative deadline, and T_i is the minimum inter-arrival time of the instances (called, jobs) of task τ_i. After a job of τ_i is released at time t, it requires at

978-3-9815370-6-2/DATE16/ ©2016 EDAA

most C_i units of execution time before its *absolute deadline*, which is at time $(t + D_i)$. Tasks are indexed in *deadline-monotonic* order: if $k < \ell$ for any two tasks τ_k and τ_ℓ, then $D_k \leq D_\ell$. A job is called *active* at time t if it is released but has not completed its execution. An active job may be in execution or awaiting execution in the ready queue.

Scheduling Events. In EDF scheduling, the active job with earliest absolute deadline has the highest priority and is always the one in execution. The ready-queue manager updates the ready queue based on one or more of the following scheduling events that may occur at time t:

- If the processor is busy and a new job (denoted by J_{new}) with shorter absolute deadline than that of the currently-executing job (denoted by J_{exe}) is released, then J_{new} starts execution by preempting J_{exe}. The TCB of the preempted job J_{exe} is inserted in the ready queue. This insertion event managed by the ready-queue manager is called the "rel_prmt" event.
- If the processor is busy and a new job J_{new} with absolute deadline higher than or equal to that of the currently-executing job J_{exe} is released at time t, then J_{new} does not preempt J_{exe} and the TCB of J_{new} is inserted in the ready queue. This insertion event is called "rel_no_prmt" event.
- If the processor is idle while the ready queue is non-empty, then the TCB of the job with earliest absolute deadline (i.e., highest EDF priority) from the ready queue is removed and that job starts execution. This removal event is called "idle_remv" event.

The main contribution of this paper is to propose efficient operations (presented in next section) to manage each of the rel_prmt, rel_no_prmt and idle_remv events.

III. Efficient Ready Queue Management

This section presents the data structure of the ready queue and operations to handle the events rel_prmt, rel_no_prmt and idle_remv for EDF scheduler.

We use the same data structure that is proposed in [8]: an array of n linked lists is used to store the TCBs of the ready jobs of EDF scheduler (see Figure 1). Each linked list has a head and a tail pointer that respectively points the first and last TCB in that particular linked list. The head and tail pointers of the kth linked list are denoted by head[k] and tail[k] for $k = 1, 2, \ldots n$. In order to specify which linked lists are non-empty, we use a bitmap $B[n \ldots 1]$. We set $B[k] = 1$, whenever the kth linked list is non-empty; otherwise, $B[k] = 0$.

Function NEXT_LIST(k,B). Given index k of some linked list of the ready queue, the index of its immediately next (i.e., higher-indexed) *non-empty* linked list is computed using functions NEXT_LIST(k,B). For example, consider a bitmap of length $n = 8$ such that $B[8 \ldots 1] = 11010110$ and we want to find NEXT_LIST(k,B) for $k = 5$. The next non-empty higher-indexed linked list relative to 5th linked-list has index NEXT_LIST(5,11010110)=7. Function NEXT_LIST(k,B) will be used later and can be implemented in constant time (please see [6] for details).

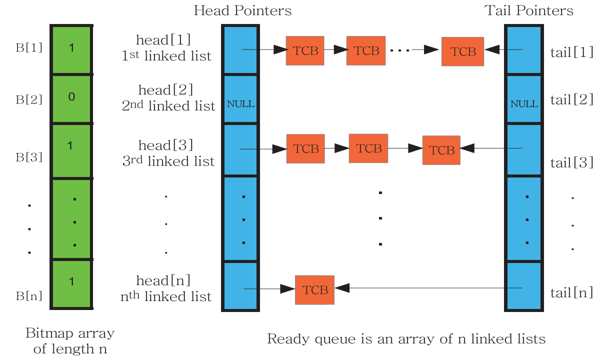

Figure 1. Data structure of EDF ready queue proposed in [8].

A. Operations on the Ready Queue

Operations on the ready queue are performed so that the following properties P1–P3 always hold:

- **P1**: If $k < \ell$, then all jobs stored in the (lower-indexed) k^{th} linked list of the ready queue have higher EDF priorities than any other job stored in the (higher-indexed) ℓ^{th} linked list of the ready queue.
- **P2**: All jobs in the ℓth linked list are stored in order of non-increasing EDF priority, i.e., the head[ℓ] and tail[ℓ] respectively points the highest and lowest EDF priority job in the ℓth linked list for $\ell = 1, \ldots n$.
- **P3**: All the jobs that are stored in the ℓth linked list at time t have their absolute deadlines no larger than D_ℓ relative to time t.

Assume that all these three properties hold at some time t_0. Consider that some event (rel_prmt, rel_no_prmt, or idle_remv) occurs at time t such that there is no other event after t_0 and before t. We will show how properties P1–P3 continue to hold after ready queue is updated to handle the event that occurs at time t (note that multiple events occurring at the same time can be processed in any order).

Event rel_prmt. This event occurs if a newly released job J_{new} starts executing by preempting the currently-executing job J_{exe}. The TCB of J_{exe} has to be inserted in the ready queue. Without loss of generality assume that J_{exe} is an instance of task τ_k. Therefore, the absolute deadline of this job is not larger than D_k relative to time t since tasks are indexed in deadline-monotonic order and J_{exe} was in execution just before time t.

To insert the TCB of job J_{exe} in the ready queue, we determine in constant time the least set bit of the bitmap $B[n \ldots 1]$. The least set bit of B can be determined in constant time using machine instruction if supported; otherwise, using deBruijn sequence [4]. Let the position of the least set bit of bitmap B is ℓ. There must be at least one non-empty linked list so that ℓ is a *valid* index, i.e., $1 \leq \ell \leq n$. If ℓ is not a valid index, then we set $\ell = (n + 1)$. We define $\rho = min\{\ell, k\}$.

The TCB of job J_{exe} is **inserted at the front** of the ρth linked list of the ready queue in constant time using the head[ρ] pointer. Because J_{exe} was in execution just before t and EDF always executes the highest EDF priority job, the EDF priority of J_{exe} is larger than or equal to that of any

other job in the ready queue. Notice that property P1 and P2 hold before TCB of J_{exe} is inserted in the ready queue and all the $(\rho - 1)$ lower-indexed (i.e., $1^{st}, 2^{nd}, \ldots (\rho-1)^{th}$) linked lists are empty. Since the EDF priority of J_{exe} is higher than or equal to any other job in the ready queue, inserting the TCB of job J_{exe} at the front of ρth linked list ensures that property P1 and P2 continue to hold at time t.

Property P3 holds for the ρth linked list before job J_{exe} is inserted. This implies that the absolute deadline relative to time t of each job (if any) in the ρth linked list is not larger than D_ρ. Because the EDF priority of job J_{exe} is higher than or equal to any other job in the ready queue, the absolute deadline of job J_{exe} is not larger D_ρ relative to time t regardless whether $\rho = \ell$ or $\rho = k$. Therefore, property P3 continues to hold after the TCB of J_{exe} is inserted in the ρth linked list. If $B[\rho] = 0$, we also set $B[\rho] = 1$ to specify that there is a TCB awaiting execution in the ρth linked list. In summary, the TCB of the preempted job is inserted in the ready queue in constant time and properties P1–P3 hold. ■

Event `rel_no_prmt`. This event occurs if a newly released job J_{new} cannot preempt the currently-executing job J_{exe}. In such case, the TCB of J_{new} is inserted in the ready queue. Assume without loss of generality that J_{new} is an instance of τ_k. Therefore, the deadline of J_{new} is *exactly* equal to D_k relative to time t since J_{new} is released at t.

Because the deadline of J_{new} is exactly D_k relative to time t, the EDF priorities of the jobs in the k lower-indexed (i.e., $1^{st}, \ldots k^{th}$) linked lists are not lower than that of job J_{new} (follows from property P3). In order to maintain properties P1–P3, the TCB of job J_{new} (which is a job of task τ_k) can be inserted at the end of the kth linked list.

However, inserting the TCB of J_{new} at the end of the kth linked list is *not suff cient* to maintain properties P1–P3. This is because some jobs in the $(n - k)$ higher-indexed linked lists (i.e., ν^{th} linked list where $\nu > k$) may have higher EDF priorities than that of the newly released job J_{new}. The TCBs of such jobs having higher EDF priorities than that of J_{new} have to be removed from the $(n - k)$ higher-index linked lists and inserted at the end of the kth linked list before the TCB of J_{new} is inserted at the end of the kth linked list. To insert the TCB of J_{new} at the end of kth list, we execute the following steps (Step 1–Step 4).

Step 1: We find $h =$ `NEXT_LIST(k,B)`, which is the position of the immediately next higher-indexed *non-empty* linked list of kth linked list. If h is an invalid index (i.e., all the higher-indexed list are empty), then go to Step 4; otherwise (i.e., h^{th} list is non-empty), we follow next steps.

Step 2: If the last element of the hth linked list (pointed by `tail[`h`]`) has absolute deadline smaller than that of J_{new}, then all the jobs in h^{th} linked list have higher EDF priorities than that of job J_{new} due to property P2. In such case, the entire h^{th} linked list is inserted (i.e., appended) at the end of the kth linked list in constant time using the `head[`h`]` and `tail[`h`]` pointers. We set `head[`h`]` $=$ `NULL`, `tail[`h`]` $=$ `NULL` and $B[h] = 0$ to specify that the hth list is now empty. We repeat the process (for next higher-indexed list) by jumping to Step 1.

Step 3: If the last element of the hth linked list has absolute deadline larger than or equal to that of J_{new}, then the EDF priorities of all the jobs in the hth linked list are *not* higher than that of job J_{new}. In such case, we only remove the TCBs of those jobs from h^{th} list that have their EDF priorities higher than that of J_{new} and insert these TCBs at the *end* of the kth list. These removal and insertion operations are done by testing one-by-one element of hth list starting from the *fi st* element, which ensures that jobs from h^{th} list are inserted at the end of the kth list in non-increasing EDF priority order. As soon as a TCB of a job that has absolute deadline larger than or equal to that of J_{new} is found in h^{th} linked list, the removal and insertion process stops. At this stage there is no job with higher EDF priority than that of J_{new} in any of the $(n-k)$ higher-indexed linked lists (follows from property P1 and P2). We go to step 4.

Step 4: The TCB of job J_{new} is inserted at the end of the kth linked list. If $B[k] = 0$, then we set $B[k] = 1$ to specify that the kth linked list is now not empty.

The jobs in the higher-indexed $(k+1)^{th}, (k+2)^{th}, \ldots n^{th}$ non-empty linked lists satisfy properties P1–P3 before the TCB of job J_{new} is inserted. Notice that TCBs of the jobs from these higher-indexed linked lists are removed and inserted in the kth linked list in non-increasing EDF priority order. Because only jobs that have higher EDF priorities than J_{new} are removed from these higher-indexed lists and inserted in the kth linked list, it is guaranteed that properties P1–P3 hold after inserting J_{new} at the end of kth linked list.

In contrast to the approach of this paper, the work in [7] handles the `rel_no_prmt` event by linearly searching each element in the higher-indexed linked list even if all the jobs in that list have higher EDF priorities than that of J_{new}. Therefore, the proposed technique of this paper is more efficient. Although a linear linked-list implementation of the ready queue does not need any remapping of other TCBs but it requires linear search to find the appropriate position of the new job. The number of comparisons for such linear search is larger than the number of remapping required in the proposed approach. ■

Event `idle_remv`. This event occurs when the processor becomes idle and the ready-queue is not empty. In such case, the TCB of the highest EDF priority job from the ready queue is removed and that job is dispatched for execution. The TCB of the highest EDF priority job is the first element in the *lowest-indexed non-empty* linked list since P1 and P2 hold at time t. The position of lowest-indexed non-empty linked list is found in constant time using bitmap B. Let the ℓth linked list is the lowest-indexed non-empty linked list.

The job from the **front** of the ℓth linked list is **removed** and dispatched for execution. The removal from the front is

done in constant time using `head[ℓ]` pointer. If `head[ℓ]` becomes `NULL`, then we set the $B[\ell] = 0$. Since removal of a job from any linked list of the ready queue cannot violate P1–P3, these properties continue to hold. ■

IV. EVALUATION

In this section, the results demonstrating the effectiveness of the proposed ready-queue management mechanism using randomly generated task sets are presented. Task sets are randomly generated using the same approach as in [8]. At each utilization level (starting from 0.5 with a step size of 0.025), total 1000 task sets are generated. The randomly-generated task sets that are exclusively schedulable using `EDF` and *not* schedulable using `FP` are considered. Results for task set with utilization larger than 0.5 at which task sets are difficult to schedule using `FP` scheduling are presented.

For each task set, the execution is simulated using `EDF` scheduling. The ready jobs that need to await execution are stored in the ready queue and remapped to new position when necessary. Three alternatives for `EDF` ready queue are considered: (i) the approach proposed in this paper, (ii) the approach proposed for priority-promotion-based scheduling in [8], and (iii) binary min-heap based priority queue. We denote Our_{avrg}, $Prom_{avrg}$, and $Heap_{avrg}$ as the average number of remapping of the TCBs in the ready queue that is implemented respectively using our proposed scheme, the approach proposed in [8], and using binary min-heap.

The binary min-heap-based ready queue is considered as the baseline. The improvement at each utilization level of the proposed approach (denoted as `Our_vs_Heap`) and the priority-promotion-based approach proposed in [8] (denoted as `Prom_vs_Heap`) over the min-heap-based ready queue management approach for `EDF` scheduler is computed as follows:

$$\texttt{Our_vs_Heap} = \frac{Our_{avrg} - Heap_{avrg}}{max\{Our_{avrg}, Heap_{avrg}\}} \times 100\%$$

$$\texttt{Prom_vs_Heap} = \frac{Prom_{avrg} - Heap_{avrg}}{max\{Prom_{avrg}, Heap_{avrg}\}} \times 100\%$$

The results of simulation using $n = 10, 20$ tasks for constrained- and implicit-deadline task sets are presented in Fig. 2 (please see [6] for additional results and details of simulation setup). The x-axis is the utilization level and the y-axis represents *Improvement* (i.e., value of `Our_vs_Heap` and `Prom_vs_Heap`) over the base case. It is evident that both approaches perform better than the base case since most of the values are positive. The value of `Our_vs_Heap` is significantly larger than that of `Prom_vs_Heap`. The value of `Our_vs_Heap` is around 90% at most utilization levels, which implies that about 90% of all remapping of the TCBs in the min-heap-based ready queue can be reduced using our proposed scheme. In other words, the proposed ready-queue management approach outperforms the priority-promotion-based approach proposed in [8].

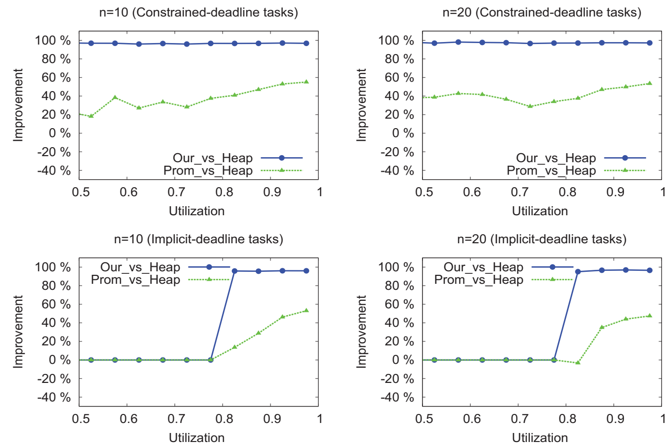

Figure 2. Values of `Our_vs_Heap` and `Prom_vs_Heap`.

V. CONCLUSION

Building upon an array of linked-lists-based ready queue data structure, this paper presents efficient management of jobs in the ready queue of `EDF` scheduler. It is shown that insertion and removal operations of jobs to and from our proposed ready queue is very efficient. In particular, almost 90% remapping of the TCBs required in binary min-heap-based ready queue can be eliminated using our proposed scheme. I expect that such low overhead in managing jobs in ready queue — combined with optimal schedulability power — will make `EDF` popular in industry.

VI. ACKNOWLEDGMENT

This research has been funded by the MECCA project under the ERC grant ERC-2013-AdG 340328-MECCA and by the ARTEMIS Joint Undertaking under grant agreement no. 621429 for the EMC^2 project.

REFERENCES

[1] M. Bertogna and S. Baruah. Tests for global edf schedulability analysis. *J. Syst. Archit.*, 57(5):487–497, May 2011.

[2] G. C. Buttazzo. Rate monotonic vs. edf: Judgment day. *Real-Time Syst.*, 29(1):5–26, 2005.

[3] M. L. Dertouzos. Control robotics: The procedural control of physical processes. In *IFIP Congress*, pages 807–813, 1974.

[4] C. Leiserson, H. Prokop, and K. H. Randall. using de bruijn sequences to index a 1 in a computer word. *MIT Technical Report*, 1998. http://supertech.csail.mit.edu/papers/debruijn.pdf.

[5] J. Y. T. Leung and J. Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. *Performance Evaluation*, 2:237–250, 1982.

[6] R. M. Pathan. Design of an efficient ready queue for earliest deadline first scheduler (extended version). 2015. http://www.cse.chalmers.se/~risat/papers/DATE2016Ext.pdf.

[7] R. M. Pathan. "real-time scheduling on uni- and multiprocessors based on priority promotions". *In submission to Leibniz Transactions on Embedded Systems*, 2015. http://www.cse.chalmers.se/~risat/papers/FPPextension.pdf.

[8] R. M. Pathan. Unifying fixed- and dynamic-priority scheduling based on priority promotion and an improved ready queue management technique. In *Proc. of RTAS*, 2015.

[9] M. Short. Improved task management techniques for enforcing edf scheduling on recurring tasks. In *Proc. of RTAS*, 2010.