

A Soft-Real-Time Optimal Semi-Clustered Scheduler with a Constant Tardiness Bound*

Shareef Ahmed and James H. Anderson

University of North Carolina at Chapel Hill

Abstract—Different global and semi-partitioned schedulers have been proposed that are soft-real-time (SRT) optimal for sporadic task systems, meaning they can guarantee bounded deadline tardiness. However, under known analyses, tardiness bounds increase with respect to the number of processors, which reduces the applicability of these schedulers in systems with a large number of processors. In this paper, a semi-clustered scheduler, SC-EDF, is presented that has a constant tardiness bound. SC-EDF partitions tasks into clusters, each of which may include one fractional processor. Each cluster is scheduled by G-EDF, and the fractional processors are realized using Pfair scheduling techniques.

I. INTRODUCTION

Unlike hard-real-time (HRT) systems, missing deadlines by a bounded amount, i.e., bounded tardiness, is acceptable [9] in a soft-real-time (SRT) system. The optimality of a scheduling algorithm in an SRT system is determined by the ability of the algorithm to ensure bounded tardiness for any task system that does not over-utilize all processors or contain any single task that over-utilizes a single processor [9]. Many global and semi-partitioned algorithms are known to be SRT-optimal [9, 12]. In global scheduling, a job can be scheduled on any available processor, while in semi-partitioned scheduling, most tasks execute only on a fixed processor and the remaining tasks are allowed to migrate among processors.

Unfortunately, existing analyses of global and semi-partitioned algorithms provide tardiness bounds that increase with respect to the number of processors [2, 3, 9–11, 13, 14]. Additionally, existing analyses are seemingly not tight for systems with large processor counts. Thus, the practicality of these scheduling algorithms is questionable for systems with a large number of processors. Moreover, the applicability of relatively tighter analyses [10, 14] is problematic because those bounds are not in closed form and require complex algorithms, even with exponential time complexity [14], to compute a tardiness bound. Although HRT-optimal schedulers ensure a minimum tardiness of zero, the high overheads of such schedulers may be undesirable in practice [4–6].

In this paper, we develop a scheduling approach that can enable constant tardiness bounds without resorting to the costly techniques of HRT-optimal algorithms. Our approach is motivated by the fact that the known closed-form analysis of the global earliest-deadline-first (G-EDF) scheduler provides a relatively tight tardiness bound in closed form [9] for a

small number of processors. This inspires us to propose a new scheduling algorithm that partitions tasks into small-sized clusters where the size of a cluster is determined by the sum of the utilizations of the tasks in that cluster, and schedule each cluster by G-EDF on the required number of processors. However, partitioning tasks into integer-sized clusters is the same as the bin-packing problem, which is NP-hard in the strong sense and incurs utilization loss. Therefore, we allow the size of a cluster to be a rational number and create a periodic server task to schedule the fractional part. The server tasks are scheduled on the required number of processors using Pfair [6] scheduling techniques to provide proportional shares to the server tasks. However, our usage of such techniques avoids the high cost usually seen in Pfair scheduling by using a relatively large allocation quantum. Figs. 2 and 3, which are discussed in detail later, illustrate our approach.

Prior work. Since the original work on tardiness under G-EDF by Devi and Anderson [9], considerable work has been done regarding tardiness under different schedulers. The currently known tightest analysis of G-EDF has been given by Valente [14]. Window-constrained schedulers, a generalization of the G-EDF scheduler, also provide bounded tardiness [13]. The G-FL scheduler is known to be the best G-EDF-like scheduler in terms of minimizing tardiness under a certain type of analysis [10]. Apart from global schedulers, several semi-partitioned schedulers have also been proposed that can ensure bounded tardiness [2, 3, 11].

Contribution. In this paper, we present SC-EDF (semi-clustered earliest-deadline-first), which is the first SRT-optimal scheduler known to us that ensures constant tardiness without excessive preemptions and migrations. Semi-clustered schedulers generalize semi-partitioned ones by partitioning tasks into clusters and allowing different clusters to receive a small fraction of their allocation from common processors. To the best of our knowledge, SC-EDF is the first proposed SRT-optimal semi-clustered algorithm. SC-EDF has a tardiness bound of $c \cdot C_{max}$, where C_{max} is the maximum execution cost among all tasks and c is a constant. SC-EDF does not require particular partitioning techniques, but works for any partitioning strategy provided that the clusters satisfy certain conditions. To assess the efficacy of SC-EDF, we present the results of an experimental study that compares tardiness and the number of preemptions under it to those under G-EDF. These experiments show that SC-EDF is particularly effective in comparison to G-EDF on large multiprocessor platforms where many high-utilization tasks are common.

Work was supported by NSF grants CNS 1563845, CNS 1717589, and CPS 1837337, ARO grant W911NF-17-1-0294, ONR grant N00014-20-1-2698, and funding from General Motors.

Organization. In the rest of this paper, we give necessary background information including the considered system model (Sec. II), present SC-EDF in detail (Sec. III) and tardiness analysis under it (Sec. IV), discuss our experimental results (Sec. V), and conclude (Sec. VI).

II. PRELIMINARIES

We consider a task system τ consisting of n implicit-deadline sporadic tasks $\tau_1, \tau_2, \dots, \tau_n$ to be scheduled on a multiprocessor platform consisting of $m \geq 2$ identical processors. Each task τ_i releases a potentially infinite sequence of jobs $\tau_{i,1}, \tau_{i,2}, \dots$. Each task τ_i is specified by the parameters (C_i, T_i) , where C_i denotes τ_i 's (worst-case) execution cost, and T_i denotes its period, which is the minimum separation time between two consecutive job releases of τ_i . If the separation time between consecutive jobs of each task τ_i is exactly T_i , then the task system is called *periodic*. The *relative deadline* of τ_i is denoted by $D_i = T_i$. The *utilization* of τ_i , denoted by u_i , is the ratio C_i/T_i . The *utilization* of the task system τ is $U = \sum_{i=1}^n u_i$. We require $u_i \leq 1.0$ and $U \leq m$ to hold, which are necessary conditions for SRT schedulability [9]. The maximum and minimum execution cost among all the tasks in τ are denoted by C_{max} and C_{min} , respectively. The ceiling of the utilization of the task system $\lceil U \rceil$ is denoted by U^+ . We assume all task parameters to be rational and time to be continuous. For any time $t > 0$, the notation t^- is used to denote an instant $t - \epsilon$ where $\epsilon \rightarrow 0^+$.

The *release time*, *absolute deadline*, *completion time*, and *execution cost* of job $\tau_{i,j}$ are denoted by $r_{i,j}, d_{i,j}, f_{i,j}$, and $C_{i,j}$, respectively. The jobs of each task are sequential, i.e., $\tau_{i,j+1}$ cannot start execution before $\tau_{i,j}$ completes even if $\tau_{i,j}$ misses its deadline. The *response time* of $\tau_{i,j}$ is denoted by $R_{i,j} = f_{i,j} - r_{i,j}$. The *tardiness* of a job $\tau_{i,j}$ is defined as $\max\{0, f_{i,j} - d_{i,j}\}$. The tardiness of task τ_i is the maximum tardiness among any of its jobs. The following definitions closely follow from material in [9, 13].

Def. 1. A job $\tau_{i,j}$ is active at time t in a schedule $\mathcal{S}$ if $r_{i,j} \leq t < d_{i,j}$. If a task τ_i has an active job at t , then τ_i is active at t .

Def. 2. A job $\tau_{i,j}$ is pending at time t in a schedule $\mathcal{S}$ if $r_{i,j} \leq t$ and $\tau_{i,j}$ has not completed execution by t in $\mathcal{S}$.

Def. 3. Let C^ℓ (resp., U^ℓ) denote the sum of the highest ℓ execution costs (resp., utilizations) of tasks in τ .

Allocation. The amount of time allocated to a task τ_i in a schedule $\mathcal{S}$ over an interval $[t_1, t_2)$ is denoted by $A(\tau_i, t_1, t_2, \mathcal{S})$. Similarly, the amount of time allocated to τ in a schedule $\mathcal{S}$ over the interval $[t_1, t_2)$ is denoted by $A(\tau, t_1, t_2, \mathcal{S})$. Thus,

$$A(\tau, t_1, t_2, \mathcal{S}) = \sum_{\tau_i \in \tau} A(\tau_i, t_1, t_2, \mathcal{S}). \quad (1)$$

Ideal schedule. Let $\hat{\pi}_1, \hat{\pi}_2, \dots, \hat{\pi}_n$ be n processors having speeds $u_1, u_2, \dots, u_n$, respectively. In an *ideal schedule* $\mathcal{I}$, each task τ_i is partitioned to execute on processor $\hat{\pi}_i$. Each job starts execution as soon as it is released in $\mathcal{I}$. Executing

at speed u_i , the response time of each job of τ_i is at most T_i in $\mathcal{I}$. For task τ_i , $A(\tau_i, t_1, t_2, \mathcal{I}) \leq u_i(t_2 - t_1)$. For the task system τ , $A(\tau, t_1, t_2, \mathcal{I}) \leq U(t_2 - t_1)$.

LAG. The lag of a task τ_i in a schedule $\mathcal{S}$ at time t is the difference between its allocation in $\mathcal{S}$ and $\mathcal{I}$, respectively, over the time interval $[0, t)$. Thus, the lag of τ_i at time t in $\mathcal{S}$ is

$$\text{lag}(\tau_i, t, \mathcal{S}) = A(\tau_i, 0, t, \mathcal{I}) - A(\tau_i, 0, t, \mathcal{S}). \quad (2)$$

Similarly, the LAG of a task system τ in a schedule $\mathcal{S}$ at time t is defined as

$$\text{LAG}(\tau, t, \mathcal{S}) = \sum_{\tau_i \in \tau} \text{lag}(\tau_i, t, \mathcal{S}) = A(\tau, 0, t, \mathcal{I}) - A(\tau, 0, t, \mathcal{S}). \quad (3)$$

Since $\text{LAG}(\tau, 0, \mathcal{S}) = 0$, for $t_2 \geq t_1$ we have

$$\text{LAG}(\tau, t_2, \mathcal{S}) = \text{LAG}(\tau, t_1, \mathcal{S}) + A(\tau, t_1, t_2, \mathcal{I}) - A(\tau, t_1, t_2, \mathcal{S}). \quad (4)$$

Concrete and non-concrete tasks system. A task system is *concrete* if the release time and actual execution time of every job of each task is known, and *non-concrete*, otherwise. Infinitely many concrete task systems can be specified for a non-concrete task system and we call each such concrete task system a *concrete instantiation* of the non-concrete system.

Minimum-parallelism form. *Minimum-parallelism (MP) form* allows different SRT components or clusters to be scheduled on a multiprocessor platform without utilization loss. In MP-form, at most one processor allocated to each cluster is partially available [12]. To analyze tardiness assuming processor supply is in MP-form, restricted processor capacity must be considered. Chakroborty et al. introduced *service functions* to specify available processor capacity [8]. Leontyev and Anderson defined the service function of a processor with restricted supply as follows [13].

Def. 4. The service function $\beta_k(\Delta)$ for processor π_k lower bounds the available capacity $H_k(t, t + \Delta)$ over any interval $[t, t + \Delta)$ [13], where

$$\beta_k(\Delta) = \max\{0, \hat{u}_k(\Delta - \sigma_k)\}, \quad (5)$$

$\hat{u}_k$ is the long-term utilization available on π_k and σ_k is the x -intercept necessary for $\beta_k(\Delta)$ to lower bound $H_k(t, t + \Delta)$.

Ex. 1. Consider a processor π_k whose availability pattern is shown in Fig. 1(a) where intervals of availability are shown as shaded regions. The availability pattern repeats every eight time units, within which π_k is available for four time units. Thus, $\hat{u}_k$ is 0.5. The thick curve in Fig. 1(b) represents the piecewise linear available processor capacity. The service function $\beta_k(\Delta) = \max\{0, 0.5(\Delta - 2)\}$ is shown by the dashed curve. Here $\sigma_k = 2$. $\diamond$

III. SC-EDF

In this section, we present algorithm SC-EDF which schedules a set of sporadic tasks on a multiprocessor platform. Our goal in designing SC-EDF is to develop a scheduling algorithm that has a tardiness bound in the form $c \cdot C_{max}$

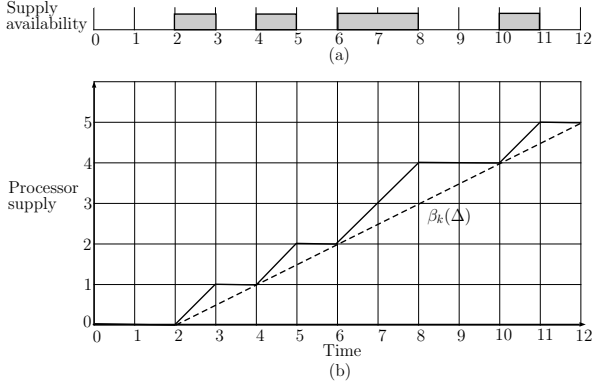


Fig. 1: (a) Availability of processor π_k , and (b) the service function of π_k .

without any utilization loss or excessive job preemptions, where c is a relatively small constant.

A. Scheduling

SC-EDF consists of two parts: an offline method for assigning tasks to processors and an online scheduler. In the offline method, SC-EDF partitions tasks into some disjoint subsets $G_1, G_2, \dots, G_\ell$. We call each subset a *cluster*. The utilization of G_i , denoted by U_i , is the sum of the utilizations of the tasks in G_i . We refer to the utilization of a cluster as the *size* of the cluster, and use these terms interchangeably. The clusters are constructed according to the following rule.

- R.** Each cluster has a size within $[1, p+1)$ where p is an integer such that $p \geq 2$.

We allocate $\lfloor U_i \rfloor$ fully available processors to each cluster G_i . To schedule the fractional part of each cluster G_i , we construct a synchronous (i.e., starts execution at time 0) periodic server task S_i of utilization $u_i^S = U_i - \lfloor U_i \rfloor$. If the utilization of S_i is $u_i^S = \frac{a}{b}$, then we set the period and execution cost of S_i to be $b \cdot q$ and $a \cdot q$, respectively, where q is the *quantum size*, which is a tunable parameter applicable to the scheduling of server tasks. The total utilization of the server tasks is $U^S = \sum_{i=1}^\ell u_i^S$. The server tasks are scheduled on $\lceil U^S \rceil$ processors by a Pfair scheduler. The tasks of G_i are in turn scheduled on the processors fully allocated to G_i and the periodic server S_i by a G-EDF scheduler. Therefore, each cluster G_i has processor supply in MP-form, i.e., $\lfloor U_i \rfloor$ fully available processors and at most one partially available processor.

Choice of p . We will later derive a tardiness bound under SC-EDF that will depend on p . To keep the tardiness bound constant, we propose p to be a small constant, i.e., $p \leq 4$.

Choice of quantum size. We propose the quantum size q to be some number within $[C_{min}, C_{max}]$. The reason behind such a choice is to reduce the large preemption overhead incurred by Pfair schedulers when the quantum size is small. Intuitively, a cluster's server is available to its tasks for at least the amount of time required to complete the job with the smallest execution cost. However, the interval of time over

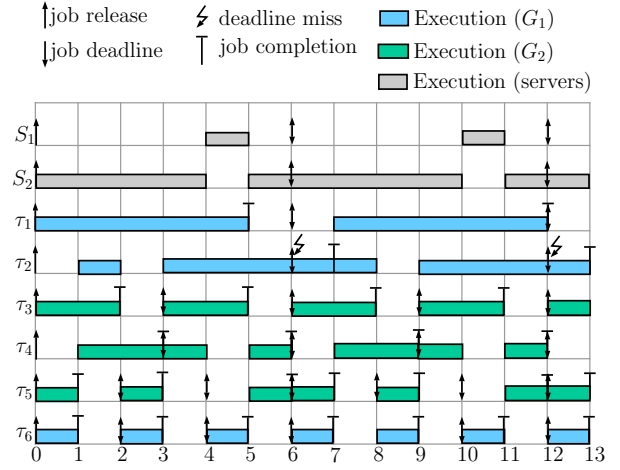


Fig. 2: (a) An SC-EDF schedule for the task system in Ex. 2.

which the server is unavailable to the cluster can be large for such a choice, causing larger tardiness compared to the choice of a smaller quantum size.

Utilizing unallocated processors. If $U \leq m - 1$, then some processors will be unallocated. To improve tardiness, we schedule the pending tasks of all clusters that are not scheduled on the processors and servers allocated to the clusters on the unallocated processors in a G-EDF manner. Later, we will derive a tardiness bound assuming the unallocated processors are idle. Note that, scheduling tasks on the unallocated processors does not increase tardiness.

Ex. 2. Consider a task system τ consisting of $\tau_1 = (5, 6)$, $\tau_2 = (5, 6)$, $\tau_3 = (3, 4)$, $\tau_4 = (3, 4)$, $\tau_5 = (1, 2)$, and $\tau_6 = (1, 2)$ to be scheduled on four processors by SC-EDF. Assume $p = 2$, and τ is partitioned into two clusters $G_1 = \{\tau_1, \tau_2, \tau_6\}$ and $G_2 = \{\tau_3, \tau_4, \tau_5\}$ according to rule R. The utilization of G_1 is $U_1 = \frac{5}{6} + \frac{5}{6} + \frac{1}{2} = \frac{13}{6}$, and the utilization of G_2 is $U_2 = \frac{2}{3} + \frac{2}{3} + \frac{1}{2} = \frac{11}{6}$. There are two server tasks $S_1 = (1, 6)$ and $S_2 = (5, 6)$ for G_1 and G_2 , respectively. G_1 is scheduled on two fully available processors and the server S_1 , while G_2 is scheduled on one fully available processor and the server S_2 . S_1 and S_2 are scheduled on a processor by a Pfair scheduler. Fig. 2 shows the SC-EDF schedule of τ up to 13 time units. Since S_2 executes over the interval $[0, 4)$, G_2 executes on two processors during this interval. On the other hand, G_1 executes on three processors over the interval $[4, 5)$, since S_1 executes during this interval. $\diamond$

B. Partitioning

Although SC-EDF works for any partitioning method that satisfies rule R, it is desirable to maximize the number of fully available processors allocated to the clusters. Instead of devising a potentially sophisticated procedure, we propose a simple greedy algorithm, as shown in Alg. 1, to partition tasks onto clusters that does not guarantee to maximize the number of fully available processors. The greedy algorithm creates an initial partitioning of τ , corrects any cluster that violates rule R, creates periodic servers, and allocates processors to the clusters and servers (lines 27–32). During the initial

Algorithm 1 Greedy Partitioning

```

1: procedure InitialPartitioning()
2:   Index tasks in descending order of utilization
3:    $i, j, k \leftarrow 1, n, 1$ 
4:   while  $i \leq j$  do ▷ Until all tasks are assigned
5:     Create an empty cluster  $G_k$ 
6:     while  $i \leq j$  and  $u_i + U_k \leq p$  do
7:       Add  $\tau_i$  to  $G_k$  ▷ Available task with highest utilization
8:        $i \leftarrow i + 1$ 
9:     while  $i \leq j$  and  $U_k < p$  do
10:      Add  $\tau_j$  to  $G_k$  ▷ Available task with smallest utilization
11:       $j \leftarrow j - 1$ 
12:       $k \leftarrow k + 1$ 
13:
14:
15: procedure Refine()
16:    $G_\ell, G_{\ell-1} \leftarrow$  last and second-to-the-last cluster by
17:   InitialPartitioning()
18:   if  $U_\ell < 1.0$  then
19:     if  $U_\ell + U_{\ell-1} < p + 1$  then
20:       Add all tasks of  $G_\ell$  to  $G_{\ell-1}$ , and
21:       remove  $G_\ell$ 
22:     else
23:       while  $U_\ell < 1.0$  do
24:         Remove a task  $\tau_u$  from  $G_{\ell-1}$ , and
25:         add  $\tau_u$  to  $G_\ell$ 
26:
27: procedure Partitioning()
28:   InitialPartitioning()
29:   Refine()
30:   for each cluster  $G_k$  do
31:     Create a server  $S_k$  of utilization  $U_k - \lfloor U_k \rfloor$ 
32:     Allocate  $\lfloor U_k \rfloor$  processors to  $G_k$ 
33:     Allocate  $\lceil U^S \rceil$  processors to the servers

```

partitioning, the algorithm iteratively constructs clusters by adding the available task with highest utilization to a cluster while the size of the cluster is at most p (lines 6–9). To keep the fractional part of each cluster small, the algorithm then repeatedly adds the unassigned task with smallest utilization until the size of the cluster is at least p (lines 10–13). Only the last cluster constructed by procedure InitialPartitioning can have size less than p . In case such a cluster is formed, procedure Refine corrects this by either merging the final two clusters (line 19–21) or moving some tasks from the second-to-the-last cluster to the last cluster (lines 22–25).

Ex. 3. Consider a task system τ consisting of $\tau_1 = (4, 5)$, $\tau_2 = (4, 5)$, $\tau_3 = (4, 5)$, $\tau_4 = (3, 5)$, $\tau_5 = (3, 5)$, $\tau_6 = (1, 2)$, $\tau_7 = (1, 2)$, and $\tau_8 = (1, 2)$. For $p = 2$, the initial partitioning under the greedy algorithm is shown in Fig. 3(a). To construct G_1 , the greedy algorithm considers the tasks in decreasing order of utilization. It adds τ_1 and τ_2 to G_1 , but the addition of τ_3 causes the size of G_1 to exceed 2.0. Instead of adding τ_3 , the algorithm then considers tasks in increasing order of utilization and adds τ_8 to G_1 . Since after adding τ_8 , the size of G_1 exceeds 2.0, no further tasks are added to G_1 . Similarly, G_2 and G_3 are formed. Since the size of G_3 is $\frac{2}{3} < 1.0$, the algorithm refines the partitioning as illustrated in Fig. 3(b). The sum of the size of G_2 and G_3 is $2 + \frac{2}{5} + \frac{2}{3} = \frac{46}{15} \geq 3$.

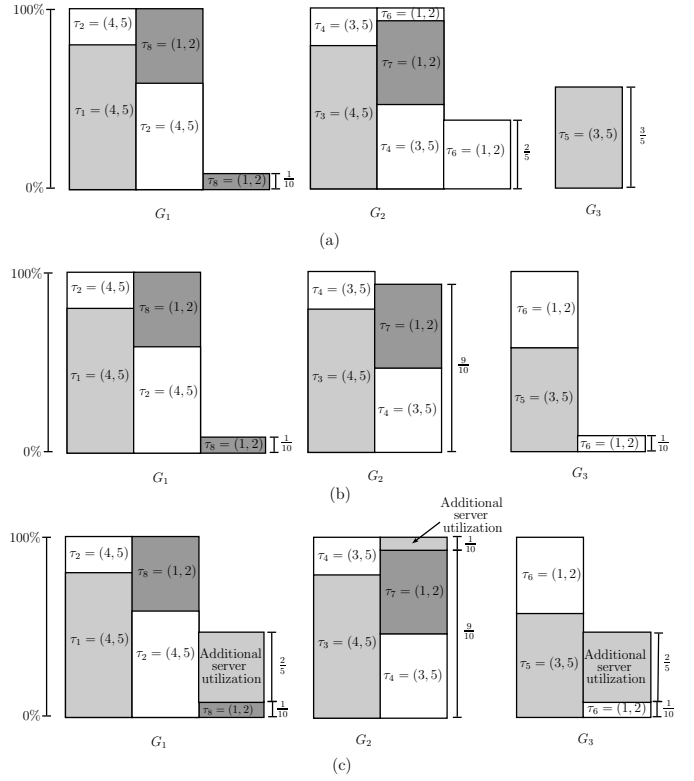


Fig. 3: (a) Output of InitialPartitioning and (b) Refine procedure for the task system in Ex. 3, and (c) the result of increasing server utilizations for the task system in Ex. 4.

Hence, τ_6 is removed from G_2 , and added to G_3 . $\diamond$

Increasing server utilizations. If U is not integral, then the sum of the utilizations of the servers U^S is also not integral. This can cause capacity loss, since the servers are scheduled on $\lceil U^S \rceil$ processors. We can remedy this by increasing the utilizations of the servers by distributing the residual utilization $\lceil U^S \rceil - U^S$. To distribute the residual utilization, we increase the utilization of the server of each cluster evenly to make $U^S = \lceil U^S \rceil$. If increasing the utilization evenly causes some server's utilization to exceed 1.0, we set its utilization to 1.0 and redistribute the remaining utilization to the other clusters.

Ex. 4. Consider the task system and the partitioning illustrated in Ex. 3. Since the total utilization of the servers is now 1.1 and they are scheduled on two processors, the available utilization $2.0 - 1.1 = 0.9$ is distributed among the servers as illustrated in Fig. 3(c). The server utilization of G_2 is increased by 0.1 because increasing its utilization by 0.3 would cause its utilization to exceed 1.0. The remaining available utilization 0.8 is divided evenly between the servers of G_1 and G_3 . $\diamond$

IV. TARDINESS BOUND

In this section we derive a tardiness bound under SC-EDF. Our approach consists of three steps: firstly, we derive a tardiness bound under G-EDF in MP-form; secondly, we derive a supply function for the server tasks scheduled by Pfair; and finally, we combine the tardiness bound obtained

in the first step and the supply function obtained in the second step.

A. Tardiness Bound for Tasks in MP-Form under G-EDF

In this section, we derive a tardiness bound for a set of implicit-deadline periodic tasks $\tau^N = \{\tau_1, \tau_2, \dots, \tau_n\}$ on m processors in MP-form under G-EDF. As shown in [15], any tardiness bound derived for periodic task systems scheduled by G-EDF also holds for sporadic task systems under G-EDF. Therefore, without loss of generality, we limit our attention to periodic task sets in deriving our bound. Note that [15] holds for uniform heterogeneous platforms, but an identical platform is a special case of a uniform heterogeneous one. For the sake of generality, we use n and m to denote the number of tasks and the number of processors, respectively, although when we apply the results of this section later, we will be considering subsets of tasks and processors. We also use the relevant definitions and notation from Sec. II. Our tardiness bound under G-EDF in MP-form is an improvement compared to the tardiness bound given in [13]. We improve that bound by extending techniques used to improve the tardiness bound under G-EDF when all processors are fully available [9] to the best currently known analysis of tardiness under G-EDF in MP-form [13]. Let $\beta(\Delta) = \max\{0, \hat{u}(\Delta - \sigma)\}$ be the service function of the restricted processor. By analysis from [13], the tardiness of a task τ_k is at most $y + C_k$, where

$$y = \frac{C^{m-1} + 2\hat{u} \cdot \sigma - \hat{u}C_k}{m-1 + \hat{u} - U^{m-1}}. \quad (6)$$

We show in contrast that the tardiness of a task τ_k is at most $x + C_k$ provided that the total utilization does not exceed $m - 1 + \hat{u}$, where

$$x \geq \frac{C^{U^+-1} + 2\hat{u} \cdot \sigma - \hat{u}C_k}{m-1 + \hat{u} - U^{U^+-2}}. \quad (7)$$

We prove the tardiness bound in (7) by contradiction. To this end, we assume that there exists a concrete instantiation τ of non-concrete task system τ^N such that the following conditions hold.

- A1.** The tardiness of a job $\tau_{k,\ell}$ exceeds $x + C_k$ and the tardiness of every job with deadline earlier than $t_d = d_{k,\ell}$ is at most $x + C_k$.
- A2.** No concrete instantiation of τ^N satisfying A1 releases fewer jobs than τ .
- A3.** No concrete instantiation of τ^N satisfying A1 and A2 has a smaller sum of the execution costs of all released jobs than τ .

By A2, τ consists of only jobs with deadlines at or before t_d . Let S be the corresponding G-EDF schedule of τ .

Deriving an upper bound of LAG. We now derive an upper bound of $\text{LAG}(\tau, t_d, S)$. Defs. 5–8 are adapted from [9, 13].

Def. 5. A time instant t is called busy if at least U^+ tasks have pending jobs at t , and non-busy otherwise. A time interval $[t_1, t_2)$ is called busy (non-busy) if each instant in the interval is busy (non-busy).

Def. 6. An interval $[t, t')$ is a maximal non-busy interval if $[t, t')$ is a non-busy, and there is no interval $[\tilde{t}, \tilde{t}')$ with either (i) $\tilde{t} < t$ and $\tilde{t}' \geq t'$ or (ii) $\tilde{t} \leq t$ and $\tilde{t}' > t'$ that is a non-busy interval (see Fig. 4).

Def. 7. An interval $[t_1, t_2)$ is called LAG-increasing if $\text{LAG}(\tau, t^-, S) < \text{LAG}(\tau, t, S)$ holds for any $t \in [t_1, t_2)$. An interval $[t_1, t_2)$ is called LAG-non-increasing if $\text{LAG}(\tau, t^-, S) \geq \text{LAG}(\tau, t, S)$ holds for any $t \in [t_1, t_2)$.

Def. 8. An interval $[t, t')$ is a maximal non-busy LAG-increasing interval if $[t, t')$ is both a non-busy and a LAG-increasing interval, and there is no interval $[\tilde{t}, \tilde{t}')$ with either (i) $\tilde{t} < t$ and $\tilde{t}' \geq t'$ or (ii) $\tilde{t} \leq t$ and $\tilde{t}' > t'$ that is both non-busy and LAG-increasing.

Def. 9. Let ϕ be the latest time instant such that at most $U^+ - 1$ tasks release their first jobs by ϕ^- .

Lemma 1, given next, was initially proved in [9] assuming processors are fully available, and later extended in [13] to the case when one or more processors can have limited supply.

Lemma 1. [9, 13] Let t be a time instant at or before t_d . Let $\tau_{i,j}$ be the earliest pending job of task τ_i at t . If $d_{i,j} < t$, then $\text{lag}(\tau_i, t, S) \leq x \cdot u_i + C_i$, otherwise $\text{lag}(\tau_i, t, S) \leq C_i$.

Lemmas 2, 3, and 4, given next, were initially proved in [9] for sporadic tasks under G-EDF assuming processors are fully available in the context of maximal non-busy LAG-increasing intervals. Using similar techniques, we prove them for the case of maximal non-busy intervals for periodic tasks scheduled by G-EDF with processor supply in MP-form.

Lemma 2. Let $[t_n, t_b)$ be a maximal non-busy interval such that $t_n > \phi$ and $t_b \leq t_d$. Then, there exists a job $\tau_{i,j}$ such that $f_{i,j} \leq t_n$ and $d_{i,j} \geq t_b$.

Proof. Assume for a contradiction that $\tau_{i,j}$ does not exist. We first show that there exists a job $\tau_{i,j}$ such that $f_{i,j} < t_b$ and $d_{i,j} \geq t_b$. Since $[t_n, t_b)$ is a maximal non-busy interval, there exists a task τ_i having no pending job at t_b^- . Let $\tau_{i,j}$ be the latest job of τ_i completed before t_b^- . Since τ_i has no pending job at t_b^- , the release of the next job $\tau_{i,j+1}$ must be at or after t_b . Hence, since τ_i is periodic, the deadline of $\tau_{i,j}$ is at or after t_b . Therefore, $f_{i,j} < t_b$ and $d_{i,j} \geq t_b$ holds.

Next, we show that $\tau_{i,j}$ completes execution at or before t_n . For a contradiction, assume that $\tau_{i,j}$ executes after t_n in S as illustrated in Fig. 4. Let δ be the amount of execution of $\tau_{i,j}$ completed at or before t_n , and let τ' be a concrete instantiation of τ^N obtained by reducing the execution cost of $\tau_{i,j}$ to δ . Let S' be a G-EDF schedule of τ' such that ties are resolved identically in both S and S' . Hence, $\tau_{i,j}$ does not execute after t_n in S' . Clearly, S and S' are identical at or before t_n . Since there are at most $U^+ - 1$ tasks with pending jobs in $[t_n, t_b)$, no job of τ' is scheduled on the additional available processor in S' due to $\tau_{i,j}$ not executing in $[t_n, t_b)$. Hence, the completion time of every job except $\tau_{i,j}$ is the same in both S and S' . Thus τ' is a concrete instantiation of

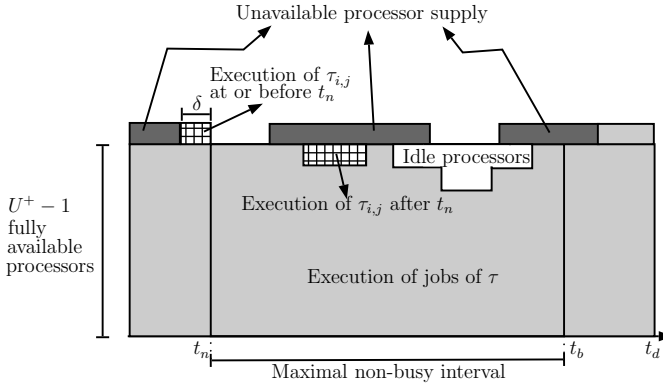


Fig. 4: Scenario considered in the proof of Lemma 2.

τ^N satisfying A1 and A2 with less execution time for $\tau_{i,j}$, contradicting A3. $\square$

Def. 10. A job fragment J of a job is a portion of the job that executes continuously in S .

Def. 11. The removal of a job from τ can cause one or more job fragments to shift earlier in S . Such shifts of job fragments can be viewed as a set of one or more displacement chains where a displacement chain consists of one or more equal-length job fragments. We denote the u^{th} displacement chain by $\Delta_u = (J_{u,1}, J_{u,2}, \dots, J_{u,n_u})$, and the time instant when $J_{u,v}$ starts execution in S by $t_{u,v}$.

Lemma 3. Let $[t_n, t_b)$ be a maximal non-busy interval such that $t_n > \phi$ and $t_b \leq t_d$. Then, there exists a job $\tau_{g,h}$ such that $\tau_{g,h}$ executes at t_b^- and $d_{g,h} \geq t_b$.

Proof. Assume for a contradiction that

B. $\tau_{g,h}$ as described in the lemma statement does not exist. By Lemma 2, there exists a job $\tau_{i,j}$ such that $f_{i,j} \leq t_n$ and $d_{i,j} \geq t_b$. Since the tardiness of $\tau_{i,j}$ is 0, $\tau_{i,j}$ cannot be $\tau_{k,\ell}$. We now consider the concrete instantiation τ' of τ^N by removing a job fragment of length ϵ from $\tau_{i,j}$. Let S' be the G-EDF schedule of τ' such that ties are resolved identically in both S and S' . We will show that no job fragment scheduled at or after t_b shifts left, which implies by A1 that the tardiness of $\tau_{k,\ell}$ is more than $x + C_k$ in S' , contradicting A3. Assume to the contrary that there is a job fragment scheduled at or after t_b that undergoes a left shift. Let Δ_u be a displacement chain, as illustrated in Fig. 5, caused by the removal of a job fragment of $\tau_{i,j}$, i.e., $J_{u,1}$ in the figure is a job fragment of $\tau_{i,j}$. By the definition of a displacement chain, for any two consecutive job fragments $J_{u,v}$ and $J_{u,v+1}$, we have $t_{u,v} < t_{u,v+1}$. Since G-EDF prioritizes $J_{u,v}$ over $J_{u,v+1}$ and the deadline of $\tau_{i,j}$ is at or after t_b , each job fragment of Δ_u has a deadline at or after t_b .

Let $J_{u,v}$ and $J_{u,v+1}$ be two job fragments of Δ_u such that $J_{u,v}$ starts its execution before t_b , and $J_{u,v+1}$ ends its execution after t_b in S . If $t_{u,v+1} < t_b$, then there is a job $\tau_{g,h}$ as described in the statement of the lemma, contradicting B. Now consider the remaining possibility, i.e., $t_{u,v+1} \geq t_b$ holds. Since $J_{u,v+1}$ starts execution before t_b in S' (after shifting),

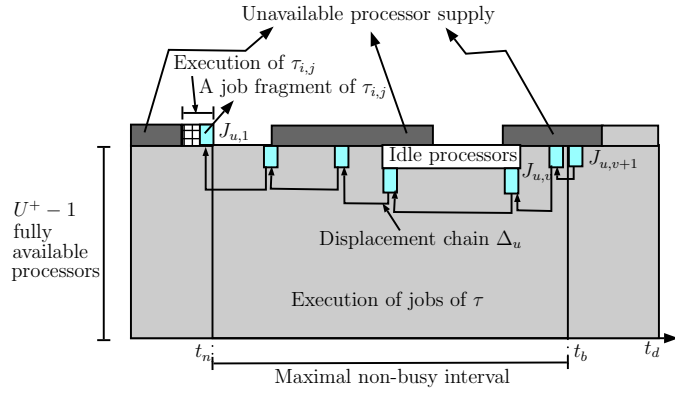


Fig. 5: Scenario considered in the proof of Lemma 3.

the job containing $J_{u,v+1}$ is active at t_b^- in S (before shifting). As $[t_n, t_b)$ is a non-busy interval, there must be a job fragment J' of the same task executing at t_b^- causing $J_{u,v+1}$ to not execute at t_b^- in S . It follows that J' is in the displacement chain Δ_u , which implies that J' has a deadline at or after t_b , contradicting B. Hence, if B holds, then no job fragment scheduled at or after t_b in S undergoes a left shift in obtaining S' , which implies that the schedule after t_b is the same in both S and S' . Thus, τ' is a concrete instantiation of τ^N satisfying A1 and A2, but has smaller sum of the execution costs of all released jobs, contradicting A3. $\square$

Def. 12. Let t_c be the latest non-busy time instant at or before t_d if there is any, otherwise let $t_c = 0$.

Lemma 4. $\text{LAG}(\tau, t_c, S) \leq x \cdot U^{U^+-2} + C^{U^+-1}$.

Proof. If t_c is in $[0, \phi]$, then LAG at t_c is at most 0, because of the work-conserving nature of G-EDF. We now consider the remaining case where $t_c > \phi$. Since t_c is a non-busy instant, there must be a non-busy interval $[t'_c, t_c)$ with $t'_c > \phi$. There are at most $U^+ - 1$ tasks with pending jobs at t_c . By Lemma 3, among the tasks with pending jobs at t_c , at least one task has only one pending job at t_c having a deadline at or after t_c . Thus, by (3) and Lemma 1, the lemma follows. $\square$

The following lemma gives an upper bound on LAG at t_d in terms of LAG at t_c . Since $[t_c, t_d)$ is a busy interval, LAG can only increase due to unavailability of the processor with restricted supply. A more general version of Lemma 5 was proved in [13] in the context of window-constrained schedulers when more than one processor can have restricted supply. A proof of Lemma 5 as stated for our context is provided in an online appendix [1].

Lemma 5. [13] $\text{LAG}(\tau, t_d, S) \leq \text{LAG}(\tau, t_c, S) + \hat{u} \cdot \sigma$.

Lemma 6. $\text{LAG}(\tau, t_d, S) \leq x \cdot U^{U^+-2} + C^{U^+-1} + \hat{u} \cdot \sigma$.

Proof. Immediate from Lemmas 4 and 5. $\square$

Necessary condition for tardiness to exceed $x + C_k$. We now give a necessary condition for the tardiness of $\tau_{k,\ell}$ to exceed $x + C_k$.

Def. 13. Let W be the total allocation of τ after t_d in $\mathcal{S}$.

Def. 14. Let R be the amount of unavailable processor time over the interval $[t_d, t_d + x + C_k)$ of the processor with restricted supply.

The following lemma gives a lower bound of $W + R$ for the tardiness of $\tau_{k,\ell}$ to exceed $x + C_k$. A more general version of Lemma 7 where more than one processor can have restricted supply was proved in [13].

Lemma 7. [13] If the tardiness of $\tau_{k,\ell}$ exceeds $x + C_k$, then

$$W + R > mx + C_k. \quad (8)$$

Deriving tardiness bound. We now derive a tardiness bound using similar techniques from [13]. By the definition of LAG , t_d , and W ,

$$W = \text{LAG}(\tau, t_d, \mathcal{S}). \quad (9)$$

Let $H(t_d, t_d + x + C_k)$ be the amount of time the processor with restricted supply is available over the interval $[t_d, t_d + x + C_k)$. We now derive an upper bound of R .

$$\begin{aligned} R &= \{\text{By Def. 14}\} \\ &\quad x + C_k - H(t_d, t_d + x + C_k) \\ &\leq \{\text{By Def. 4}\} \\ &\quad x + C_k - \beta(x + C_k) \\ &= \{\text{By (5)}\} \\ &\quad x + C_k - \max\{0, \hat{u}(x + C_k - \sigma)\} \\ &\leq \{\text{Since } \hat{u}(x + C_k - \sigma) \leq \max\{0, \hat{u}(x + C_k - \sigma)\}\} \\ &\quad x + C_k - \hat{u}(x + C_k - \sigma) \\ &= \{\text{Rearranging}\} \\ &\quad (1 - \hat{u})(x + C_k) + \hat{u} \cdot \sigma \end{aligned} \quad (10)$$

Therefore, $W + R$ can be upper bounded as follows.

$$\begin{aligned} W + R &\leq \{\text{By (9), (10), and Lemma 6}\} \\ &\quad xU^{U^+-2} + C^{U^+-1} + 2\hat{u} \cdot \sigma \\ &\quad + (1 - \hat{u})(x + C_k) \end{aligned} \quad (11)$$

Since the tardiness of $\tau_{k,\ell}$ exceeds $x + C_k$, by (8) and (11),

$$xU^{U^+-2} + C^{U^+-1} + 2\hat{u} \cdot \sigma + (1 - \hat{u})(x + C_k) > mx + C_k,$$

which implies

$$x < \frac{C^{U^+-1} + 2\hat{u} \cdot \sigma - \hat{u}C_k}{m - 1 + \hat{u} - U^{U^+-2}} \quad (12)$$

However, (12) contradicts (7). Therefore, the tardiness of $\tau_{k,\ell}$ is at most $x + C_k$ where x is defined as (7). Thus, we have the following theorem.

Theorem 1. The tardiness of any task τ_k under $G\text{-EDF}$ in $MP\text{-form}$ is at most $x + C_k$, where x is as defined in (7).

Note that, the denominator of (7) is always positive. Hence, no additional utilization restriction is required for Theorem 1 to hold. In Sec. IV-B below, we will derive the supply function

for the periodic server of each cluster in $SC\text{-EDF}$.

B. Deriving Supply Function

We now consider the synchronous periodic servers $\tau^s = \{S_1, S_2, \dots, S_\ell\}$ to be scheduled by a **Pfair** scheduler as described in Sec. III. Our goal is to derive the supply function corresponding to the available processor time to an arbitrary periodic server task S_i . Let $\beta_i(\Delta) = \max\{0, \hat{u}_i(\Delta - \sigma_i)\}$ be the service function corresponding to S_i . To determine $\beta_i(\Delta)$, we need to find appropriate values of $\hat{u}_i$ and σ_i . Let u_i^S be the utilization of S_i . By the definition of $\hat{u}_i$, $\hat{u}_i = u_i^S$. Next, we determine σ_i .

Let $\mathcal{P}$ be a **Pfair** schedule of τ^s . For notational convenience, we initially assume the **Pfair** scheduler uses a quantum size of 1.0 time unit, with integral server periods and execution costs. Later, we will show how to reinterpret σ_i for an arbitrary quantum size.

Lemma 8. [4] For any task S_i and time instant t , $-1 < \text{lag}(S_i, t, \mathcal{P}) < 1$.

Lemma 9. For any interval $[t, t + \Delta)$,

$$A(S_i, t, t + \Delta, \mathcal{P}) > \Delta \hat{u}_i - 2. \quad (13)$$

Proof. We first lower bound the difference between the lag of S_i at t and $t + \Delta$. This difference is minimized when the lag of S_i at t is minimum and the lag of S_i at $t + \Delta$ is maximum. Thus,

$$\begin{aligned} \text{lag}(S_i, t, \mathcal{P}) - \text{lag}(S_i, t + \Delta, \mathcal{P}) &> \{\text{By Lemma 8}\} \\ &\quad -1 - 1 \\ &= -2. \end{aligned} \quad (14)$$

By (4), we have

$$\begin{aligned} \text{lag}(S_i, t + \Delta, \mathcal{P}) &= \text{lag}(S_i, t, \mathcal{P}) + A(S_i, t, t + \Delta, \mathcal{I}) \\ &\quad - A(S_i, t, t + \Delta, \mathcal{P}). \end{aligned} \quad (15)$$

Rearranging (15), we get

$$\begin{aligned} A(S_i, t, t + \Delta, \mathcal{P}) &= A(S_i, t, t + \Delta, \mathcal{I}) + \text{lag}(S_i, t, \mathcal{P}) \\ &\quad - \text{lag}(S_i, t + \Delta, \mathcal{P}) \\ &> \{\text{By the definition of } \mathcal{I} \text{ and (14)}\} \\ &\quad \Delta \hat{u}_i - 2. \end{aligned}$$

□

Lemma 10. For any Δ , $\beta_i(\Delta) = \max\{0, \hat{u}_i(\Delta - \sigma_i)\}$ lower bounds the actual allocation of S_i in $\mathcal{P}$ where $\sigma_i = \frac{2}{\hat{u}_i}$.

Proof. We prove this lemma by showing $\beta_i(\Delta) < A(S_i, t, t + \Delta, \mathcal{P})$ for any interval $[t, t + \Delta)$. We consider two cases.

Case 1. $\Delta \leq \sigma_i$. In this case, $\beta_i(\Delta) = 0$. Since $A(S_i, t, t + \Delta, \mathcal{P})$ is non-negative, the lemma holds.

Case 2. $\Delta > \sigma_i$. In this case,

$$\beta_i(\Delta) = \{\text{By (5)}\}$$

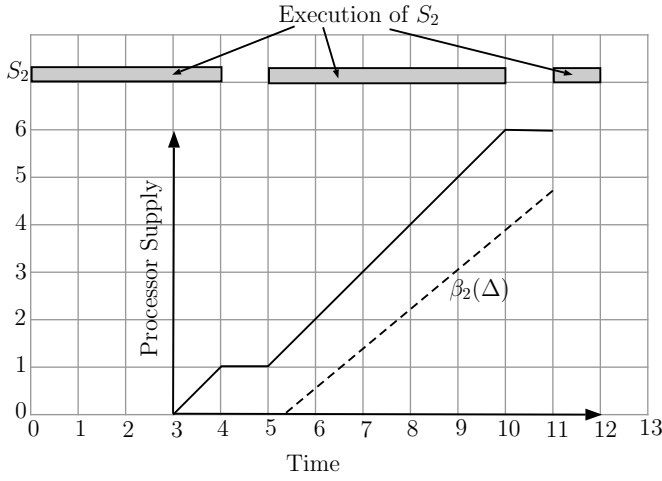


Fig. 6: The service function of the server task S_2 in Ex. 5.

$$\begin{aligned}
& \hat{u}_i(\Delta - \sigma_i) \\
&= \{\text{By the choice of } \sigma_i\} \\
& \hat{u}_i(\Delta - \frac{2}{\hat{u}_i}) \\
&= \{\text{Rearranging}\} \\
& \Delta \hat{u}_i - 2.
\end{aligned} \tag{16}$$

By Lemma 9 and (16), $\beta_i(\Delta) < A(S_i, t, t + \Delta, \mathcal{P})$. $\square$

Corollary 1. *If the quantum size is q and the server parameters are integer multiples of q , then $\beta_i(\Delta)$ lower bounds the actual allocation of S_i in $\mathcal{P}$ for $\sigma_i = \frac{2}{\hat{u}_i}q$.*

Ex. 5. Consider the server task S_2 of Ex. 2. The utilization of S_2 is $\frac{5}{6}$. Hence, $\hat{u}_2 = \frac{5}{6}$. Since the quantum size is 1.0 in the schedule shown in Ex. 2, $\sigma_2 = \frac{2}{5/6} \cdot 1 = \frac{12}{5}$. Thus, the service function corresponding to S_2 is $\beta_2(\Delta) = \max\{0, \frac{5}{6}(\Delta - \frac{12}{5})\}$. Fig. 6 illustrates $\beta_2(\Delta)$ over the interval $[3, 10)$. Since, σ_2 is $\frac{12}{5}$, $\beta_2(\Delta)$ is 0 from time 3 to $3 + \frac{12}{5} = \frac{27}{5}$. $\diamond$

C. Tardiness bound under SC-EDF

We now derive a tardiness bound under SC-EDF using the results from Secs. IV-A and IV-B.

Theorem 2. *The tardiness of a task $\tau_k \in \tau$ under SC-EDF is at most $x + C_k$, where*

$$x = \frac{C^p + 4q - \min\{\hat{u}_i\}C_{min}}{1 + \min\{\hat{u}_i\}} \tag{17}$$

and q is the quantum size used in scheduling the server tasks.

Proof. Suppose τ_k is assigned to cluster G_i of size U_i where G_i is allocated h fully available processors and a periodic server of utilization $\hat{u}_i$. Note that, if $U_i = 1$, then τ_k does not miss a deadline. We thus consider $U_i > 1$. We have two cases.

Case 1. $\hat{u}_i = 0$. In this case, U_i is integral and G_i is scheduled on $h = U_i$ fully available processors. By [9], tardiness of τ_k is at most $x' + C_k$ where

$$x' \geq \frac{C^{U_i+1} - C_k}{h - U_i^{U_i-2}} \tag{18}$$

Let y denotes the right-hand side of (18). Then, we have

$$\begin{aligned}
y &= \frac{C^{U_i+1} - C_k}{h - U_i^{U_i-2}} \\
&\leq \{\text{Since } h = U_i = U_i^+ \text{ and by Rule R}\} \\
& \frac{C^{p+1-1} - C_k}{U_i - U_i^{U_i-2}} \\
&\leq \{\text{Since per-task utilizations are at most one, } U_i > 1, \text{ and by Def. 3}\} \\
& \frac{C^p - C_k}{U_i - (U_i - 2) \cdot 1} \\
&\leq \{\text{Simplifying and by the definition of } C_{min}\} \\
& \frac{C^p - C_{min}}{2}.
\end{aligned} \tag{19}$$

By (17), (18), and (19), the theorem holds.

Case 2. $\hat{u}_i > 0$. In this case, G_i is scheduled on $h + 1$ processors where one processor is partially available. By Theorem 1 interpreted in the context of G_i , the tardiness of τ_k is at most $x' + C_k$ where

$$x' \geq \frac{C^{U_i+1} + 2\hat{u}_i \cdot \sigma_i - \hat{u}_i C_k}{h + 1 - 1 + \hat{u}_i - U_i^{U_i-2}}. \tag{20}$$

Let z denotes the right-hand side of (20). Then, we have

$$\begin{aligned}
z &= \frac{C^{U_i+1} + 2\hat{u}_i \cdot \sigma_i - \hat{u}_i C_k}{h + \hat{u}_i - U_i^{U_i-2}} \\
&\leq \{\text{Since } h = \lfloor U_i \rfloor \geq U_i^+ - 1\} \\
& \frac{C^{U_i+1} + 2\hat{u}_i \cdot \sigma_i - \hat{u}_i C_k}{U_i^+ - 1 + \hat{u}_i - U_i^{U_i-2}} \\
&\leq \{\text{Since per-task utilizations are at most one, } U_i > 1, \text{ and by Def. 3}\} \\
& \frac{C^{U_i+1} + 2\hat{u}_i \cdot \sigma_i - \hat{u}_i C_k}{U_i^+ - 1 + \hat{u}_i - (U_i^+ - 2) \cdot 1} \\
&= \{\text{Simplifying}\} \\
& \frac{C^{U_i+1} + 2\hat{u}_i \cdot \sigma_i - \hat{u}_i C_k}{1 + \hat{u}_i} \\
&\leq \{\text{By rule R}\} \\
& \frac{C^{p+1-1} + 2\hat{u}_i \cdot \sigma_i - \hat{u}_i C_k}{1 + \hat{u}_i} \\
&= \{\text{By the definition of } \sigma_i \text{ as in Corollary 1}\} \\
& \frac{C^p + 2\hat{u}_i \cdot \frac{2}{\hat{u}_i}q - \hat{u}_i C_k}{1 + \hat{u}_i} \\
&= \{\text{Simplifying}\} \\
& \frac{C^p + 4q - \hat{u}_i C_k}{1 + \hat{u}_i} \\
&\leq \{\text{By the definition of } \min\{\hat{u}_i\} \text{ and } C_{min}\} \\
& \frac{C^p + 4q - \min\{\hat{u}_i\}C_{min}}{1 + \min\{\hat{u}_i\}}.
\end{aligned} \tag{21}$$

The theorem follows by (20) and (21). $\square$

Corollary 2. If the quantum size is C_{min} , then x as defined in (17) is at most $p \cdot C_{max} + (4 - \min\{\hat{u}_i\})C_{min}$.

Proof. By (17),

$$\begin{aligned}
x &= \frac{C^p + 4q - \min\{\hat{u}_i\}C_{min}}{1 + \min\{\hat{u}_i\}} \\
&\leq \frac{\{\text{By Def. 3, } C^p \leq p \cdot C_{max}\}}{1 + \min\{\hat{u}_i\}} \\
&\leq \frac{p \cdot C_{max} + 4q - \min\{\hat{u}_i\}C_{min}}{1 + \min\{\hat{u}_i\}} \\
&\leq \{\text{Since } \min\{\hat{u}_i\} \geq 0\} \\
&= \frac{p \cdot C_{max} + 4q - \min\{\hat{u}_i\}C_{min}}{1} \\
&= \{ \text{By the choice of } q \} \\
&= p \cdot C_{max} + (4 - \min\{\hat{u}_i\})C_{min}. \tag{22}
\end{aligned}$$

□

V. EXPERIMENTS

In this section, we present the results of experimental evaluations we performed to compare SC-EDF with G-EDF. We evaluated maximum observed tardiness, the analytical tardiness bound, and the number of preemptions under SC-EDF compared to those under G-EDF for randomly generated periodic task sets. To compare the tardiness bounds, we used the closed-form G-EDF tardiness bound from [9]. We also evaluated the effect of the quantum size on tardiness and the number of preemptions in SC-EDF.

We generated task sets by a similar method of [7]. Task-system utilizations were chosen to be *medium*, *heavy*, *very heavy*, or *wide*, which correspond to per-task utilizations being uniformly distributed in $[0.1, 0.5]$, $[0.5, 1]$, $[0.8, 1]$, or $[0.1, 1]$, respectively. For each choice of task-system utilization, periods were chosen to be integers uniformly distributed between $[3, 33]$ ms, $[10, 100]$ ms, or $[50, 250]$ ms, which we refer to using the terminology *short*, *medium*, and *long*, respectively. The number of processors was chosen to be 32. Task sets were generated for utilization caps within $[24, 32]$ with a step size of 0.5. For each combination of utilization distribution, periods, number of processors, and utilization cap, 50 task sets were generated by adding randomly generated tasks until five consecutive attempts to add a next task would cause the utilization cap to be exceeded. For each task set, p was set to be 2 and the quantum size was chosen from C_{min} to C_{max} with a step size of $\frac{1}{4}(C_{max} - C_{min})$. The observed tardiness and the number of preemptions were measured by scheduling each task set for 10,000 time units. Due to space constraints, we present a small representative selection of our results—other results can be found in an online appendix [1].

Obs. 1. The average observed maximum tardiness of SC-EDF was larger than that of G-EDF for task sets with high total utilizations.

This can be seen in Figs. 7 and 8. This is likely because each cluster has some time intervals in SC-EDF when the number of processors available to the cluster is less than the size of the cluster. Utilizing the unallocated processors to execute pending

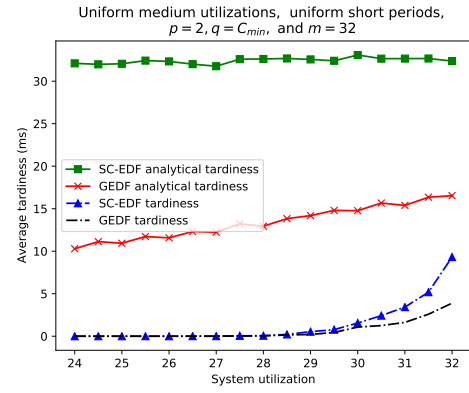


Fig. 7: Average observed and analytical tardiness with respect to system utilization for uniform medium utilizations, uniform short periods, $p = 2$, $q = C_{min}$, and $m = 32$.

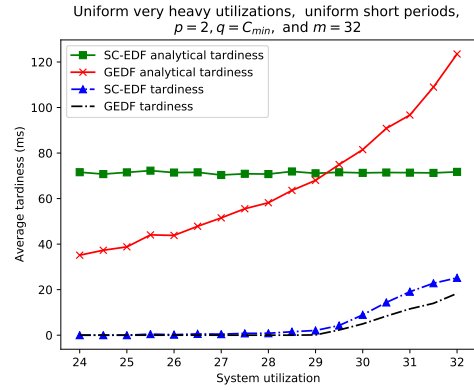


Fig. 8: Average observed and analytical tardiness with respect to system utilization for uniform very heavy utilizations, uniform short periods, $p = 2$, $q = C_{min}$, and $m = 32$.

tasks of the clusters reduces observed tardiness for task sets with relatively smaller total utilizations.

Obs. 2. For heavy and very heavy task sets with high total utilizations and smaller quantum sizes, the average tardiness bound of SC-EDF is smaller than that of G-EDF. The converse is true for task sets with low total utilizations. For medium task sets, the average tardiness bound of SC-EDF is larger than that of G-EDF.

This can be observed in Figs. 7 and 8. For task sets with smaller total utilizations, the denominator of the tardiness bound under G-EDF in [9] is larger, which results in relatively smaller tardiness bound under G-EDF. Moreover, the tardiness bound under SC-EDF has an additional term due to the restricted processor supply, which is also derived from relatively pessimistic analysis.

Obs. 3. The average number of preemptions in SC-EDF is typically smaller than that of G-EDF for task sets with high total utilizations, and the converse is true for task sets with low total utilizations.

This can be seen in Fig. 9. Only tasks within the same cluster and the availability pattern of the periodic server can cause preemptions in SC-EDF. For low total utilizations,

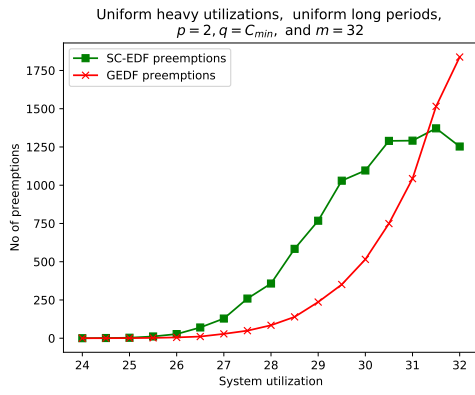


Fig. 9: Average number of preemptions over a schedule with respect to system utilization for uniform heavy utilizations, uniform long periods, $p = 2, q = C_{min}$, and $m = 32$.

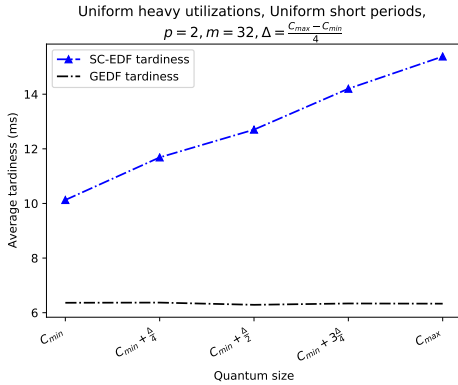


Fig. 10: Average observed tardiness with respect to the quantum size for uniform heavy utilizations, uniform short periods, $p = 2$, and $m = 32$.

additional preemptions occur due to utilizing the unallocated processors to schedule pending tasks.

Obs. 4. The average observed tardiness under SC-EDF increases with respect to the quantum size. The converse is true for the average number of preemptions.

Increasing the quantum size causes the periodic server to be unavailable for a longer period of time in SC-EDF, resulting in increased tardiness as shown in Fig. 10. In contrast, increasing the quantum size results in fewer preemptions of the periodic servers in SC-EDF as shown in Fig. 11.

VI. CONCLUSION

In this paper, we have presented a semi-clustered scheduler SC-EDF that has a tardiness bound of the form of $c \cdot C_{max}$. It is the first scheduler known to have such a tardiness bound without introducing frequent task preemptions. We have also demonstrated the competitive performance of SC-EDF compared to G-EDF, especially for task sets with high utilizations, by an experimental evaluation.

This work opens up several directions for future work. For example, it is not known whether any semi-partitioned or global scheduler with job-level fixed priorities has a constant tardiness bound; we plan to investigate this issue. We also plan to devise other semi-clustered schedulers that have a constant

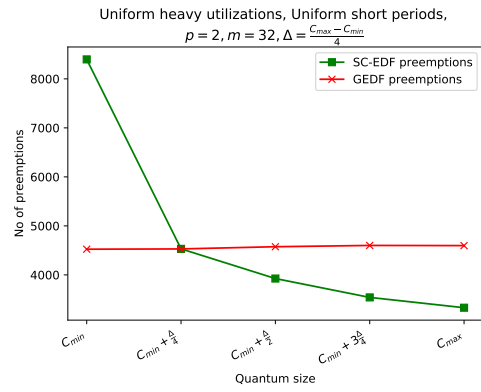


Fig. 11: Average number of preemptions over a schedule with respect to the quantum size for uniform heavy utilizations, uniform short periods, $p = 2$, and $m = 32$.

tardiness bound but less preemption and migration overheads than SC-EDF.

VII. ACKNOWLEDGEMENTS

We thank Stephen Tang and Sergey Voronov for their comments on an earlier draft of this paper.

REFERENCES

- [1] S. Ahmed and J. H. Anderson. (2020) A soft-real-time optimal semi-clustered scheduler with a constant tardiness bound (longer version with additional material). [Online]. Available: <http://jamesanderson.web.unc.edu/papers/>
- [2] J. H. Anderson, V. Bud, and U. C. Devi, "An EDF-based scheduling algorithm for multiprocessor soft real-time systems," in *ECRTS*, 2005, pp. 199–208.
- [3] J. H. Anderson, J. P. Erickson, U. C. Devi, and B. N. Casses, "Optimal semi-partitioned scheduling in soft real-time systems," *J. Signal Process. Syst.*, vol. 84, no. 1, pp. 3–23, 2016.
- [4] J. H. Anderson and A. Srinivasan, "Mixed pfair/erfair scheduling of asynchronous periodic tasks," *J. Comput. Syst. Sci.*, vol. 68, no. 1, pp. 157–204, 2004.
- [5] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.
- [6] S. K. Baruah, J. Gehrke, and C. G. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *IPSS*, 1995, pp. 280–288.
- [7] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "Is semi-partitioned scheduling practical?" in *ECRTS*, 2011, pp. 125–135.
- [8] S. Chakraborty, S. Kunzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *DATE*, 2003, pp. 190–195.
- [9] U. Devi and J. H. Anderson, "Tardiness bounds under global EDF scheduling on a multiprocessor," *Real-Time Systems*, vol. 38, no. 2, pp. 133–189, 2008.
- [10] J. P. Erickson, J. H. Anderson, and B. C. Ward, "Fair lateness scheduling: reducing maximum lateness in G-EDF-like scheduling," *Real-Time Systems*, vol. 50, no. 1, pp. 5–47, 2014.
- [11] C. Hobbs, Z. Tong, and J. H. Anderson, "Optimal soft real-time semi-partitioned scheduling made simple (and dynamic)," in *RTNS*, 2019, pp. 112–122.
- [12] H. Leontyev and J. H. Anderson, "A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees," *Real-Time Systems*, vol. 43, no. 1, pp. 60–92, 2009.
- [13] H. Leontyev and J. H. Anderson, "Generalized tardiness bounds for global multiprocessor scheduling," *Real-Time Systems*, vol. 44, no. 1-3, pp. 26–71, 2010.
- [14] P. Valente, "Using a lag-balance property to tighten tardiness bounds for global EDF," *Real-Time Systems*, vol. 52, no. 4, pp. 486–561, 2016.
- [15] K. Yang and J. H. Anderson, "On the soft real-time optimality of global EDF on uniform multiprocessors," in *RTSS*, 2017, pp. 319–330.